


EBook-AstroInformatics Series: IEEE CS Connect-An Initiative of IEEE Computer Society Bangalore Chapter

# A MANUAL ON MACHINE LEARNING AND ASTRONOMY:

AUTHORED, EDITED AND COMPILED BY

SNEHANSHU SAHA

June 15, 2019

Chapter contributions from: Suryoday Basak, Rahul Yedida, Kakoli Bora
Archana Mathur, Surbhi Agrawal, Margarita Safonova
Nithin Nagaraj, Gowri Srinivasa, Jayant Murthy

PES University
University of Texas at Arlington
North Carolina State University
Indian Statistical Institute
National Institute for Advanced Studies
Indian Institute of Astrophysics

June 15, 2019

# Preface

The E-book is dedicated to the new field of Astroinformatics: an interdisciplinary area of research where astronomers, mathematicians and computer scientists collaborate to solve problems in astronomy through the application of techniques developed in data science. Classical problems in astronomy now involve the accumulation of large volumes of complex data with different formats and characteristic and cannot be addressed using classical techniques. As a result, machine learning (ML) algorithms and data analytic techniques have exploded in importance, often without a mature understanding of the pitfalls in such studies.

This E-book aims to capture the baseline, set the tempo for future research in India and abroad, and prepare a scholastic primer that would serve as a standard document for future research. The E-book should serve as a primer for young astronomers willing to apply ML in astronomy, a way that could rightfully be called "Machine Learning Done Right", borrowing the phrase from Sheldon Axler ("Linear Algebra Done Right")! The motivation of this handbook has two specific objectives:

- develop efficient models for complex computer experiments and data analytic techniques which can be used in astronomical data analysis in the short term, and various related branches in physical, statistical, computational sciences much later (larger goal as far as memetic algorithm is concerned).

- develop a set of fundamentally correct thumb rules and experiments, backed by solid mathematical theory, and render the marriage of astronomy and Machine Learning stability for far reaching impact. We will do this in the context of specific science problems of interest to the proposers: the classification of exoplanets, classification of nova, separation of stars, galaxies and quasars in the survey catalogs, and the classification of multi-wavelength sources.

We hope the E-book serves its purpose and inspires scientists across communities to collaborate and develop a very promising field. We gratefully acknowledge the grant (File Number: EMR/2016/005687) received from SCIENCE & ENGINEERING RESEARCH BOARD (SERB), under the scheme- Extra Mural Research (EMR), a division of DST.

**********

Sincerely,
Authors

# Contents

# 1 INTRODUCTION

While developing methodologies for of Astroinformatics, during the next three to five years we anticipate a number of applied research problems to be addressed. These include:

- Decision-theoretical model addressing exoplanet habitability using the power of convex optimization and algorithmic machine learning: we will focus here on the applicability and efficacy of various machine learning algorithms to the investigation of planetary habitability. There are several different methods available, namely, K-Nearest Neighbor (KNN), Decision Tree (DT), Random Forest (RF), Support Vector Machine (SVM), Naïve Bayes, and Linear Discriminant Analysis (LDA). We plan to evaluate their performance in the determination of the habitability of exoplanets. PHL's Exoplanet Catalog (PHL-EC) is one of the most complete catalogs which contains observed and estimated stellar and planetary parameters for a total of 3415 (July 2016) currently confirmed exoplanets, where the estimates of the surface temperature are given for 1586 planets. We will test the machine learning algorithms on this and other catalogs to derive the Habitability Index for each planet. Through this, we expect to develop a unified scheme to determine the habitability index of an exoplanet. We will implement a standalone or web-based software package to be applied to any new planets found. Exoplanets are one of the most exciting subjects in astrophysics, and we expect large volumes of new data to become available with Gaia and the next generation of dedicated planet hunting missions, including WFIRST and JWST.

- Variational Approaches to eccentricity estimation: The problem of determining optimal eccentricity as a feature in computing the habitability score – Variational Calculus and the theory of Optimal control ("Variational Methods in Optimization" by Donald R. Smith) will be used.

- Star-Galaxy Classification using Machine Learning: Using the data from the Super COSMOS Sky Survey (SSS), we intend to demonstrate the efficiency of gradient boosted methods, particularly that of the XGBoost algorithm, to be able to produce results which can compete with those of other ensemble-based machine learning methods in the task of star-galaxy classification. Extensive experiments involving cost-sensitive learning and variable subset selection shall be carried out which, in turn, should help resolve some intrinsic problems with the data. The improvisations are expected to work well in handling the complexity of the data set which otherwise have not been attempted in the literature.

- ML Driven Mining and Automatic Labeling of the Habitability Catalog: Classical problems in astronomy are compounded by accumulation of large volume of complex data, rendering the task of classification and interpretation incredibly laborious. The presence of noise in the data makes analysis and interpretation even more arduous. Machine learning algorithms and data analytic techniques provide the right platform for the challenges posed by these problems. Novel Meta-heuristic (Cross culture Evolution based) clustering and probabilistic herding based clustering algorithms will be proposed to investigate the potential habitability of exoplanets by using information from PHL's Exoplanet Catalog (PHL-EC). Accuracy of such predictions is evaluated. The machine learning algorithms are integrated to analyze data from PHL's Exoplanet Catalog (PHL-EC) with specific examples being presented and discussed. Exoplanet, a software for analyzing data obtained from PHL's Exoplanet Catalog via machine learning, will also be developed and deployed in public domain.

- Nova Classification using Machine Learning: We propose a completely novel and never attempted before classification scheme, based on the shape of the light curves obtained from the AAVSO database. Nova eruptions discovered by Payne-Gaposhkin in 1964 occur on the surface of white dwarf stars in interacting binaries, generically called cataclysmic variables by Warner in 1995. They usually have a red dwarf companion star, where material accumulates on the surface until the pressure and temperature become high enough for a thermonuclear runaway reaction to occur. We obtain the light curves for the V-band from the AAVSO database. The AAVSO database has photometric magnitude estimates from amateur and professional astronomers all around the world. This database has magnitudes for roughly 200 novas. A catalog of 93 very well-observed nova light curves has been formed, in which light curves were constructed from numerous individually measured magnitudes, and 26 of the light curves following the eruption all the way to quiescent. An automatic classification scheme of nova, not attempted before, is a fundamental contribution beyond reasonable doubt.

This E-book is one of the outcomes of the extended research efforts in AstroInformatics. We would like to thank the Science and Engineering research Board (SERB), Department of Science and Technology, Government of India for supporting our research by providing us with resources to conduct our experiments. The project reference number is: EMR/2016/005687.

# 2 PROS AND CONS OF CLASSIFICATION OF EXOPLANETS: IN SEARCH FOR THE RIGHT HABITABILITY METRIC

Since time immemorial, humanity was looking at cosmos and believing other worlds being out there, inhabited with other or, may be same, beings like us. Indian ancient texts talk about travelling to other worlds in 'bodily form' (as says inscription on the iron pillar at Qutub Minar, probably left in 4th century BC). Ancient Greeks also believed in the existence of other planetary bodies with beings living on them (with mentions dating as far back as 6th century BC: Thales of Miletus and Pythagoras). With our technological advances, we are continuing this same quest, the quest for the habitable planet, ultimately, the second Earth; or at the least, for the answer to the question of whether we are alone in the Universe. Currently, we already know about the existence of thousands of exoplanets, and the estimates of the actual number of planets exceed the number of stars in our Galaxy alone by orders of magnitude (both bound and free-floating); small rocky planets, super-Earths, the most abundant type. Our interest in exoplanets lies in the fact that, anthropically, we believe that life can only originate and exist on planets, therefore, the most fundamental interest is in finding a habitable planet - the planet with life on it. This quest can be broadly classified into the following: looking for Earth-like conditions or the planets similar to the Earth (Earth similarity), and looking for the possibility of life in a form known or unknown to us (habitability). But what is habitability? Is it the ability of a planet to beget life – a potential habitability? Or is it our ability to detect it: a planet may host life as we know it, in other words, be inhabited, but we will not detect it unless it evolved sufficiently to change the environment on a planetary scale. In both cases, the only comparison for recognition is our planet, therefore, we are looking for the terrestrial likeness in exoplanets.

With a constantly increasing number of discovered exoplanets and the possibility that stars with planets are a rule rather than an exception, it became possible to begin characterizing exoplanets in terms of planetary parameters, types, populations and, ultimately, in the habitability potential. This is also important in understanding the formation pathways of exoplanets. But since complete appraisal of the potential habitability needs the knowledge of multiple planetary parameters which, in turn, requires hours of expensive telescope time, it became necessary to prioritize the planets to look at, to develop some sort of a quick screening tool for evaluating habitability perspectives from observed properties. Here, the quick selection is needed for a long painstaking spectroscopic follow-up to look for the tell-tales of life, the bio-signatures-atmospheric gases that only living organisms produce in abundance. It can be oxygen, ozone, methane, carbon dioxide or, better, their combinations

(see e.g. Safonova et al. 2016; Krissansen-Totton et al. 2018, and references therein). For all the upcoming space missions dedicated to the search of life in the Universe: PLATO, Euclid, JWST, etc., we need to make a list of our preferred candidates, so that this quest hopefully can be completed within our lifetimes. It is estimated that one in five solar-type stars and approximately half of all M-dwarf stars may host an Earth-like planet in the habitable zone (HZ). Extrapolation of Kepler data shows that in our Galaxy alone there could be as many as 40 billion such planets (e.g. Borucki et al. 2010; Batalha et al. 2013; Petigura et al. 2013). And it is quite possible that soon we may actually detect most of them. But with the ultimate goal of a discovery of life, astronomers do not have millennia to quietly sit and sift through more information than even petabytes of data. Obtaining the spectra of a small planet around a small star is difficult, and even a large-scale expensive space mission (such as e.g. JWST) may be able to observe only about a hundred stars over its lifetime (e.g., Turnbull et al. 2012).

For that purpose, several assessment scales have been introduced: a concept of the Habitable Zone (HZ) – a range of orbital distances from the host star that allows the preservation of the water in liquid state on the surface of a planet (Kasting et al., 1993); Earth Similarity Index – an ensemble of planetary physical parameters with Earth as reference frame for habitability, or Planetary Habitability Index (PHI), based on the biological requirements such as water or a substrate (Schulze-Makuch et al. 2011); habitability index for transiting exoplanets (HITE) based on the certain limit of planetary insolation at the surface (Barnes et al. 2015). Our group has developed an index applicable to small planets – Mars Similarity Index (MSI) – as potential planets to host extremophile life forms (Kashyap et al. 2017). Habitability may also be viewed as probabilistic measure, in contrast to the binary definition of, say, being in the HZ or not, and such approach requires optimization classification methods that are part of machine learning (ML) techniques. Thus, we have introduced a Cobb-Douglas Habitability Score – an index based on Cobb-Douglas habitability production function (CD-HPF), which computes the habitability score by using measured and estimated planetary parameters (Bora et al. 2016), and recently extended it to include a statistical ML classification method (XGBoost) used for supervised learning problems, where the training data with multiple features are used to predict a target variable (Saha et al. 2018).

However, all classification strategies have caveats, and some (e.g. Tasker et al. 2017) reject the exercise entirely on the basis of impossibility to quantitatively compare habitability, and on the idea that pretending otherwise can risk damaging the field in the eyes of the public community. In addition, some researchers believe that the priority for the exoplanet and planetary science community is to explore the diversity of exoplanets, and not to concentrate on exclusions.

The first qualitative scale for habitability was the concept of a HZ – it assumes once a planet is in the HZ, it is potentially habitable. However, such criterion is binary, and we know that e.g. our Moon is inside the HZ and is a rocky planetary body, but definitely not potentially habitable for our kind of life. Earth itself is located on the very edge of the HZ (making it marginally habitable) and will get out of it in the next 1-3 billion years. Mars is technically inside the HZ, and Venus once was. Titan, on the other hand, is totally outside the HZ but may host a life, albeit dissimilar to ours. Besides, recent discoveries of free-floating planets (planets without the host star where the concept of a HZ cannot apply) brought back the interest in their potential habitability that was first addressed in 1999 (Stevenson, 1999).

Coming to more quantitative assessments, HITE predicted that planets that receive between 60-90% of same amount of insolation as Earth are likely to be habitable. It however assumes only circular orbits and the location inside HZ, which again refers back to mostly Earth similarity; besides our Solar System has a unique feature of very low ellipticities. It was earlier proposed that low eccentricity favours multiple planetary systems which, in turn, favours habitability (Limbach and Turner, 2015). However, most know exoplanets have high eccentricities; the most potentially habitable planets now, TRAPPIST-1 and Proxima b, have high ellipticities, and it was estimated recently (Wang et al. 2017) that though eccentricity shrinks HZ, even high ellipticity orbits can have low effect on planetary climate provided they are in a certain spin-orbit resonances. For $e = 0.4$, if $p = 0.1$ (where $p$ is the ratio of orbital period to spin period), the HZ is the widest and the climate is most stable.

The ESI is based on the well-known statistical Bray-Curtis scale of quantifying the difference between samples, frequently used by ecologists to quantify differences between samples based on count data. However, most multivariate community analyses are about understanding a complex dataset and not finding the 'truth', meant in a sense of 'significance'. Thus, it may not be enough to understand a complex hierarchy of classification. But since all we know is the Earth-based habitability, our search for habitable exoplanets (an Earth-like life clearly favoured by the Earth-like conditions) has to be by necessity anthropocentric, and any such indexing has to be centred around finding Earth-like planets, at least initially. But Earth may not be the ideal place for life, and the concept of a super-habitability was introduced in 2014 (Heller and Armstrong, 2014). Though this concept got rid of a HZ limits admitting the tidal heating as a possible heat source, it still assumed the necessity of liquid water on the surface as a prerequisite for life, preferably as a shallow ocean with no large continuous land masses. Recent simulations, however, has shown that too much water is not good for the detectability of exo-life (Desch et al. 2018). Exoplanets without land would have life with much slower biogeochemical cycles and oxygen in the atmosphere would

be indistinguishable from the one produced abiogenically. The question now shifts to the definition of habitability as our ability to detect it if we cannot get to the planets which may have life not on the surface, they are as good as uninhabited.

The search for life on the planets outside the Solar System can be broadly classified into the following: looking for Earth-like conditions or the planets similar to the Earth (Earth similarity), and looking for the possibility of life in a form known or unknown to us (habitability). The two frequently used indices, ESI and PHI, describe heuristic methods to score similarity/habitability in the efforts to categorize different exoplanets or exomoons. ESI, in particular, considers Earth as the reference frame for habitability and is a quick screening tool to categorize and measure physical similarity of any planetary body with the Earth. The PHI assesses the probability that life in some form may exist on any given world, and is based on the essential requirements of known life: a stable and protected substrate, energy, appropriate chemistry and a liquid medium. Bora et. al (2016), [**?**] proposed a different metric, a Cobb-Douglas Habitability Score (CDHS), based on Cobb-Douglas habitability production function (CD-HPF), convex optimization techniques and constrained behavior of the optimizing model drawing inspiration from the earlier work (Ginde et. al. [Ginde2016], 2016 and Saha et.al, 2016 [3]) which computes the habitability score by using measured and calculated planetary input parameters. The metric, with exponents accounting for metric elasticity, is endowed with verifiable analytical properties that ensure global optima, and is scalable to accommodate finitely many input parameters. The model is elastic, does not suffer from curvature violations and, as the authors discovered, the standard PHI is a special case of CDHS. Computed CDHS scores are fed to K-NN (K-Nearest Neighbour) classification algorithm with probabilistic herding that facilitates the assignment of exoplanets to appropriate classes via supervised feature learning methods, producing granular clusters of habitability. The proposed work describes a decision-theoretical model using the power of convex optimization and algorithmic machine learning. Saha et al. (2018) expanded previous work by Bora et al. (2016) on using Machine Learning algorithm to construct and test planetary habitability functions with exoplanet data. This time they analyzed the elasticity of their Cobb-Douglas Habitability Score (CDHS) and compared its performance with other machine learning algorithms. They demonstrate the robustness of their methods to identify potentially habitable planets from exoplanet dataset. Given our little knowledge on exoplanets and habitability, the results have limited value now. However, their methods provide one important step toward automatically identifying objects of interest from large datasets by future ground and space observatories. Therefore, our work provides a logical evolution from the previous work by Bora et.al (2017). CDHS contributes to the Earth similarity concepts where the scores

have been used to classify exoplanets based on their degree of similarity to Earth. However Earth similarity is not equivalent to exoplanetary habitability. Therefore, Saha et. al. (2018) adopted another approach where machine classification algorithms have been exploited to classify exoplanets into three classes: nonhabitable, mesoplanets and psychroplanets (originally adopted by the Planetary Habitability Laboratory (PHL), http://phl.upr.edu). While this classification was performed, CDHS was not used at all, rather discriminating features from the PHL-EC were used. This is fundamentally different from CDHS-based Earth similarity approach where explicit scores were computed. Therefore, it was pertinent and remarkable that the outcome of these two fundamentally distinct exercises reconcile. This reconciliation approach is the first of its kind and fortifies CDHS, more than anything else. Moreover, this convergence between the two approaches is not accidental. Essentially, the ESI score gives non-dynamic weights to all the different planetary (with no trade-off between the weights) observables or calculated features considered, which in practice may not be the best approach, or at least, the only way of indicating habitability. It might be reasonable to say that for different exoplanets, the various planetary observables may weigh each other out to create a unique kind of favorable condition. For instance, in one planet, the mass may be optimal, but the temperature may be higher than the average of the Earth, but still within permissible limits (like Venus); in another planet, the temperature may be similar to that of the Earth, but the mass may be much lower. By discovering the best combination of the weights (or, as we call it,elasticities) to maximize the resultant score, to the different planetary observables, we are creating the metric which presents the best case scenario for the habitability of a planet.

The essence of the CD-HPF, and consequently, that of the CDHS is indeed orthogonal to the essence of the ESI or BCI. The argument is not in favor of the superiority of our metric, but for the new approach that have been developed. There should actually be various metrics arising from different schools of thought so that the habitability of an exoplanet may be collectively determined from all these. Such a kind of adaptive modeling has not been used in the context of planetary habitability prior to the CD-HPF.

The CD-HPF reconciles with the machine learning methods that have been used to automatically classify exoplanets. It is not easy to propose two fundamentally different approaches (one of which is CDHS) that lead to a similar conclusion about an exoplanet. While the CDHS provides a numerical indicator (in fact, the existence of one global optima shouldn't be a concern at all but rather a vantage point of the model that thus eliminates the possibility of computing scores arbitrarily), the machine classification bolsters the proposition by telling us automatically which class of habitability an exoplanet belongs to. Bora et al. explain that a CDHS close to 1 indicates a greater chance of habitability. The performance of machine clas-

sification is evaluated by class-wise accuracy. The accuracies achieved are remarkably high, and at the same time, it is observed that the values of the CDHS for the sample of potentially habitable exoplanets which are considered are also close to 1. Therefore, the computational approaches map Earth similarity to habitability. This is remarkable and non-trivial.

Despite recent criticism of the whole idea of exoplanetary ranking, we are sure that this field has to continue and evolve to use all available machinery of astroinformatics and machine learning. It might actually develop into a sort of same scale as stellar types in astronomy. It can be used as a quick tool of screening planets in important characteristics in search for potentially habitable planets for the follow-up investigations.

## 2.1 References:


- Bora, K. et. al., 2016. CD-HPF: New Habitability Score Via Data Analytic Modeling. Astronomy and Computing, 17, 129-143
- Ginde, G. et.al, 2016. ScientoBASE: A Framework and Model for Computing Scholastic Indicators of Non-Local Influence of Journals via Native Data Acquisition Algorithms. J. Scientometrics, 107:1, 1-51
- Heller R. & Armstrong J. 2014. Superhabitable Worlds. Astrobiology, 14, 50-66.
- Kasting, J. F., Whitmire, D. P., and Reynolds, R. T.,1993. Habitable zones around main sequence stars. Icarus, 101, 108-108.
- Krissansen-Totton, J. el al., Disequilibrium biosignatures over Earth history and implications for detecting exoplanet life, Science Advances (2018), DOI: 10.1126/sciadv.aao5747
- Limbach, M. A., & Turner, E. L. 2015. Exoplanet orbital eccentricity: Multiplicity relation and the Solar System. Proceedings of the National Academy of Science, 112, 20
- Safonova, M., Murthy, J., & Shchekinov, Y. A. 2016. Age aspects of habitability. International Journal of Astrobiology, 15, 93
- Saha, S. et. al., 2016. A novel revenue optimization model to address the operation and maintenance cost of a data center. Journal of Cloud Computing, 5:1, 1-23
- Saha et.al., 2018. Theoretical Validation of Potential Habitability via Analytical and Boosted Tree Methods: An Optimistic Study on Recently Discovered Exoplanets, arXiv:1712.01040 [astro-ph.EP]

- Schulze-Makuch, D., MÃl'ndez, A., FairÃl'n, A. G., von Paris, P., Turse, C., Boyer, G., Davila, A. F., Resendes de Sousa AntÃşnio, M., Irwin, L. N., and Catling, D., 2011. A Two-Tiered Approach to Assess the Habitability of Exoplanets.
- Stevenson, D. J. 1999, Life-sustaining planets in interstellar space? Nature, 400, 32

# 3 A Comparative Study in Classification Methods of Exoplanets: Machine Learning Exploration via Mining and Automatic Labeling of the Habitability Catalog

## 3.1 Introduction

For centuries, astronomers, philosophers and other scientists have considered possibilities of the existence of other planets that could support life, as it is on Earth or in different forms. The fundamental question that remains unanswered is: are other extrasolar planets (exoplanets) or moons (exomoons) capable of supporting life? Exoplanet research is one of the newest and most active areas in astrophysics and astroinformatics. In the last decade, thousands of planets were discovered in our Galaxy alone. The inference is that stars with planets are a rule rather than exception, with estimates of the actual number of planet exceeding the number of stars in our Galaxy by orders of magnitude [Strigari et al.(2012)].

Led by the NASA Kepler Mission [Batalha 2014], more than 3500 planets have been confirmed, and 4900+ celestial objects remain as candidates, yet to be confirmed as planets. **The discovery and characterization of exoplanets requires both extremely accurate instrumentation and sophisticated statistical methods to extract the weak planetary signals from the dominant starlight or from very large samples. Stars and galaxies can be seen directly in telescopes, but exoplanets can be observed only after advanced statistical analysis of the data. Consequently, statistical methodology is at the heart of almost every exoplanet science result. - from https://www.iau.org/science/events/1135/ - IAU Focus Meetings (GA) FM 8: Statistics and Exoplanets.** Different exoplanet detection methods [Fischer et al.2014] include radial velocity based detection, astrometry, transits, direct imaging, and microlensing. Each of these methods possesses its own advantages and difficulties [Danielski2014]. For example, detection through radial velocity cannot determine accurately the mass of a distant object [Ridden-Harper et al.2016] but only estimate the minimum mass of a planet, whereas mass is the primary criterion for exoplanet confirmation. Similarly, each method entails its own disadvantages for detection, confirmations and analysis. This requires a careful study and analysis of light curves.

Characterization of Kepler's different planets is important to judge their habitability [Swift et al.2013]. Detailed modeling of planetary signals to extract information of the orbital or atmospheric properties is even more challenging. Moreover, there is also the challenge of inferring the properties of the underlying planet population from incomplete and biased samples. In previous work, measurements have been performed in order to estimate habitability

or *earth similarity* such as Earth Similarity Index (ESI) [Schulze-Makuch et al.2011], Biological Complexity Index (BCI) [Irwin et al.2014], Planetary Habitability Index (PHI) [Schulze-Makuch et al.2011] and Cobb-Douglas Habitability Score (CDHS) [Bora et al.2016]. The increased importance of statistical methodology is a trend that extends across the domain of astronomy. Naturally, a need for software to process complex astronomical data and collaborative engagement among astronomers, astrostatisticians and computer scientists emerges. These problems fall into the new field of *astroinformatics*: an interdisciplinary area of research where astronomers, mathematicians and computer scientists collaborate to solve problems in astronomy through the application of techniques developed in data science. Classical problems in astronomy now involve the accumulation of large volumes of complex data with different formats and characteristics and cannot now be addressed using classical techniques. As a result, machine learning algorithms and data analytic techniques have exploded in importance, often without a mature understanding of the pitfalls in such studies (for example, [Peng, Zhang & Zhao2013] reported remarkable accuracy but accomplished on unbalanced data thereby handicapped by the inherent bias in the data, unfortunately)

Planetary Habitability Laboratory's (University of Puerto Rico) Exoplanet Catalog (PHL-EC) [Méndez2015, Méndez2016] contains several features which may be analyzed in the process of detection and classification of exoplanets [Fischer et al.2014] based on habitability. These include the composition of the planets (*P. Composition Class*), the climate of the planets (*P. Zone Class*) and the surface pressure of the planets (*P. Surf Press*) among others. The ecological conditions in any exoplanet must be suitable in order for life (like on Earth) to exist. Hence, in the data set, the *classes* of planets broadly include planets which are habitable, and planets which are not habitable. A typical data set is derived from either photometry or spectroscopy, which is calibrated and analyzed in the form of light curves [Soutter2012]. Classification of these light curves and identification of the source producing dips in the light curve are essential for detection through the transit method. A dip in the light curve represents the presence of an exoplanet but this phenomenon may also be due to the presence of eclipsing binaries, pulsating stars, red giants etc. Similarly, significant challenges to the process of classification is posed by varying intensities of light curves, presence of noise, etc.

The purpose of this research is to understand the following: given the features of habitable planets, whether it may be feasible to automate the task of determining the habitability of a planet that has not previously been classified. The planet's ecological conditions such as presence of water, pressure, gravitational force, magnetic field etc have to be studied in detail [Heller & Armstrong2014] in order to adequately determine the na-

ture of habitability of a planet. For example, the presence of water may increase the likelihood for an exoplanet to be a potential habitable candidate [Irwin et al.2014], but this cannot be affirmed until other parameters are considered. These factors not only explain the existence of life on a planet, but also its evolution such that life can be sustained [Irwin & Schulze-Makuch2011, Irwin et al.2014]. The goal of the current work is to classify the exoplanets into the different categories of habitability on the basis of their atmospheric, physical, and chemical conditions [Gonzalez, Brownlee & Ward2001], or more aptly, based on whether the respective planet is located in the *comfortable habitable zone* (CHZ) of planet's parent star. If a planet resides in the CHZ of their parent star, it is considered to be a potential candidate for being habitable as the atmospheric conditions in these zones are more likely to support life [Kaltenegger et al.2011]. A planet's atmosphere is the key to establishing its identity, allowing us to guess the formation, development and sustainability of life [Kasting1993]. Numerous features such as planet's composition, habitable zone , atmosphere class, mass, radius, density, orbital period, radial velocity, just to name a few, have to be considered for a complete atmospheric study of an exoplanet.

*Machine learning* (ML) is a field of data analysis that evolved from studies of classical *pattern recognition* techniques. Statistics is at the heart of ML algorithms, which is why it is generally treated differently from related fields such as artificial intelligence (AI). The areas of data analytics, pattern recognition, artificial intelligence and machine learning have a lot in common; ML stems out as a convergence of statistical methods and computer science. The phrase *machine learning* almost literally signifies its purpose: to enable machines to learn trends and features in data. In the current study, existing machine learning techniques have been used to explore solutions to the problem of habitability. ML techniques have proved to be effective for the task of classification in important data sets and extracting necessary information and interesting patterns from large amount of data. ML algorithms are classified into *supervised* and *unsupervised* methods, also known as *predictive* and *descriptive* methods respectively. According to [Ball & Brunner2010], supervised methods rely on a training set of objects (with both features and labels) for which the target property is known with confidence. An algorithm is *trained* on this set of objects; *training* refers to the process of discerning between classes (for tasks of classification) of data based on the feature set (in astrophysics, a *feature* should be considered the same as an *observable*). The mapping resulting from training is applied to other objects for which the target property, or the *class label* is not available, in order to estimate which class of data they belong to. In contrast, unsupervised methods do not label data into classes; the task of an unsupervised ML technique is to generally find underlying trends in data, which are not explicitly stated or mentioned in the respective data

set. Unsupervised algorithms usually require an initial input to one or more of the adjustable parameters and the solution obtained depends on this input [Waldmann & Tinetti2012].

In the atmosphere of exoplanets, the desired accuracy of flux is $10^{-4}$ to $10^{-5}$, which is difficult to achieve. An improved version of independent component analysis (ICA) has been proposed [Waldmann & Tinetti2012], where the noise due to instrumental, systematic and other stellar sources was filtered using an unsupervised learning approach; a wavelet filter was used to remove noise even in low signal-to-noise (SNR) conditions. This is achieved for HD189733b spectrum obtained through Hubble/NICMOS. In another supervised ML approach [Debray & Wu2013], stellar light curves were used to determine the existence of an exoplanet; this was accomplished by representing light curves as time series data, which was then combined with feature selection to obtain the appropriate outcome. Through a dynamic time warping algorithm, each light curve was compared to a baseline light curve, elucidating the similarity between the two. Other models which utilize alternate sequential minimal optimization (SMO) and multi-layer perceptron (MLP) have been implemented with the accuracy of 83% and 82.2% respectively. In [Abraham2014], a data set based on light curves, obtained from Kepler observatory was used for classification of stars as potentially harboring exoplanets or not. The pre-processing of the large data set of Kepler light curves removed the initial noise from the light curves and strong peaks (most likely transiting planets) were identified by calculating standard deviations and means for certain threshold values; these thresholds were selected from the percentage change metric. Next, feature extraction was performed to help capture the information regarding consistency of the peaks and transit time, which are otherwise relatively short. Principal component analysis (PCA) on these extracted features was used as a measure towards dimensionality reduction. Furthermore, four different supervised learning algorithms: k-nearest neighbor classifier (K-NN), logistic regression, softmax regression, and support vector machine (SVM) were applied. Softmax regression produced the best result for the training data set. The overall accuracy was boosted by applying k-means clustering and further application of softmax regression and PCA to 85% on the test data. NASA's catalog provides recent information about the planetary data, where certain celestial bodies are considered as Kepler's Object of Interest (KOI). Analysis and classification of KOIs is done in [McCauliff et al.2014], via a supervised machine learning approach that automates the categorization of the raw threshold crossing events (TCE) into a set of three classes namely planet candidate (PC), astrophysical false positive (AFP) and non-transiting phenomena (NTP), otherwise carried out manually by NASA's Kepler TCE Review (TCERT) team. Random forest classifier was proposed and the classification function was decided based on the statistical distribution of the attributes of each TCE like SNR, angular

offset, etc. The labels of training data were obtained by matching ephemeris contained in KOI to TCE catalog. Data imputation was carried out by using sentinel values to fill in missing attribute values; sensitivity analysis was carried out for the same operations. The precision of 95% for PC, 93% for AFP and 99% for NTP was observed. Further analysis with different classification algorithms (naïve Bayes, K-NN) was carried out, which proves greater effectiveness of Random forests.

In the process of conducting experiments, the authors developed a software called *ExoPlanet* [Theophilus, Reddy & Basak2016]. ExoPlanet served as a platform for conducting the experiments and is an open source software. In all future works, the authors will use it as a platform for analyzing data and testing algorithms.

## 3.2 Motivation

Today, many observatories all over the world survey and catalog astronomical data. For any newly discovered exoplanet, or a planet for which data is more recently collected, many attributes must be carefully considered before it may be appropriately classified. Manually completing this task is extremely cumbersome. Recently, the Kepler Habitable Zone Working Group submitted their *Catalog of Kepler Habitable Zone Exoplanet Candidates* for publication. Notably absent from this initial list are any true *Earth twins*: Earth-size planets with Earth-like orbits around Sun-like stars. While the search for Earth-twins continues as increasingly sophisticated software searches through Kepler's huge database, extrapolations from earlier statistical studies suggest that maybe one-in-ten Sun-like stars have Earth-size rocky planets orbiting inside the Habitable Zone [Dayal et al.2015]. Several Earth-twins could still be awaiting discovery in Kepler's data. A method that could rapidly find Earth-twins from Kepler's database is desirable. There are some salient features of the PHL-EC data set which make it an attractive option for machine learning based analysis. Eccentricity is assumed to be 0 when unknown; the attributes of equilibrium and surface temperature for non-gaseous planets show a linear relationship: this makes PHL-EC remarkably different from other data sets. Further, the data set exhibits a huge *bias* towards one of the classes (the *non-habitable* class of exoplanets: this poses significant challenges which needed to be properly addressed by using appropriate machine learning approaches, in order to prevent *over-fitting* and to avoid the problem of false positives.

ML techniques for analyzing data have become popular over the past two decades due to an increase in computational power. Despite this, ML techniques are not known to be applied to automate the task of classifying exoplanets. This prompted the authors to explore

the data set with various ML algorithms. Several mathematical techniques were explored and improvisations are proposed and implemented to check reliability of the classification methods. Much later in the manuscript, the effectiveness of the algorithms to accurately classify the exoplanets considered as *most likely habitable* in the optimistic and conservative lists of habitable planets of PHL-EC have been verified. The authors were keen to test the goodness of different classification algorithms and reconcile the data driven approach with the discovery and subsequent physics based inferences about habitability. This has been a very strong motivation and helped the authors go through painstaking and elaborate experimentation. Later in the paper, the results of classification performed on artificially augmented data (based on the samples naturally present in the catalog) have been presented to demonstrate that ML can be effectively used to handle large volumes of data. The authors believe that using machine learning as a *black box* should be strongly discouraged and instead, its treatment should be rigorous. The word *exploration* in the title indicates a thorough surveying of appropriate methods to solve the problem of exoplanet classification. This paper presents the results of SVM, K-NN, LDA, Gaussian naïve Bayes, decision trees, random forests and XGBoost. The performance of each classifier is examined and is correlated with the nature of the data.

## 3.3 Methods

The advancement of technology and sophisticated data acquisition methods generates a plethora of moderately complex to very complex data exist in the field of astronomy. Statistical analysis of this data is hence a very challenging and important task [Saha et al.2016]. Machine learning based approaches can carry out this analysis effectively [Ball & Brunner2010]. ML based approaches are broadly categorized into two main types: supervised and unsupervised techniques. The authors studied some of the most important work which used ML techniques on astronomical data. This motivated them to revisit important machine learning techniques and to discover their potential in the field of astronomical data analysis. The goal of the current work is to determine whether a given exoplanet can be classified as potentially habitable or not. These will be elaborated in detail later. Different algorithms were investigated in this context using data obtained from PHL-EC.

Classification techniques may also be classified into *metric* and *non-metric* classifiers, based on their working principles. Metric classifiers generally apply measures of geometric similarity and distances of feature vectors, whereas non-metric classifiers should be applied in scenarios where there are no definitive notions of similarity between feature vectors.

The results from metric and non-metric methods of classification have been enunciated separately for better understanding of suitability of these approaches in the context of the given data set. The classifier whose performance is considered as a threshold is naïve Bayes, considered as the gold standard in data analytics.

### 3.3.1 Naïve Bayes

Naïve Bayes classifier is based on Bayes' theorem. It can perform the classification of an arbitrary number of independent variables and is generally used when data has many attributes. Consequently, this method is of interest since the data set used, PHL-EC, has a large number of attributes. The data to be classified may be either *categorical*, such as P.Zone Class or P.Mass Class, or *numerical*, such as P.Gravity or P.Density. A small amount of training data is sufficient to estimate necessary parameters [Rish2001]. The method assumes independent distribution for attributes and thus estimates class conditional probability as in Equation.

$$P(X \mid Y_i) = P(X_1 \mid Y_i) \times P(X_2 \mid Y_i) \times ... \times P(X_n \mid Y_i) \tag{1}$$

As an example from the data set used in this work, consider two attributes: P. Sem Major Axis and P. Esc Vel. Assuming *independent* distribution between these attributes implies that the distribution of P. Sem Major Axis does not depend on the distribution of P. Esc Vel, and vice versa (albeit this assumption is often violated in practice; regardless, this algorithm is used and is known to produce good results). The naïve Bayes algorithm can be expressed as:

**Step 1**: Let $X = \{x_1, x_2, ..., x_d\}$ and $C = \{c_1, c_2, ..., c_d\}$ be the set of feature vectors corresponding to each entity in the data set, and the set of corresponding class labels (each class label can have one of three unique values here: *mesoplanet*, *psychroplanet*, and *non-habitable planets*, discussed) respectively. Reiterating, the attributes in the PHL-EC data set are mass of the planet, surface temperature, pressure etc., of each cataloged planet.

**Step 2**: Using Bayes' rule and applying naïve Bayes' assumption of *class conditional independence*, the *likelihood* that a given feature vector $x$ belongs to a class $c_j$ to a product of terms as in Equation.

$$p(x \mid c_j) = \prod_{k=1}^{d} p(x_k \mid c_j) \tag{2}$$

*Class conditional independence* in this context means that the output of the classifiers are independent of the classes. For example, if a data set has two classes $c_a$ and $c_b$, then

for a feature vector $x$, the outcomes $p(c_a|x)$ and $p(c_b|x)$ are independent of each other, that is, there is no relationship between the classes.

**Step 3**: Recompute the posterior probability as shown in Equation .

$$p(c_j \mid x) = p(c_j) \prod_{k=1}^{d} p(x_k \mid c_j) \tag{3}$$

The *posterior probability* is the conditional probability that a given feature vector $x$ belongs to the class $c_j$.

**Step 4**: Using Bayes' rule, the class label $c_j$ which achieves highest probability is assigned to a new pattern $x$. Since the pattern in this context refers to the feature vector of a planet, the class labels mesoplanet, psychroplanet, or non-habitable will be assigned as the class label of the sample being classified based on whichever class has highest probability score for a particular planet.

### 3.3.2 Metric Classifiers

1. *Linear Discriminant Analysis* (LDA): The LDA classifier attempts to find a linear boundary that best separates the different classes in the data. This yields the optimal Bayes' classification (i.e. under the rule of assigning the class having highest *a posteriori* probability) under the assumption that the covariance is the same for all classes. The authors implemented an enhanced version of LDA (often called regularized discriminant analysis). This involves eigen decomposition of the sample covariance matrices and transformation of the data and class centroid. Finally, the classification is performed using the nearest centroid in the transformed space considering prior probabilities into account [Welling2005]. The algorithm is expressed as:

   **Step 1**: Compute mean vectors, $\mu_i$ for $i = 1, 2, ..., c$ classes from the data set, where each mean vector is $d$-dimensional; $d$ is the number of attributes in the data. This aspect is similar to what has been stated in the subsection on Naïve Bayes'. Hence, $\mu_i$ is the mean vector of class $i$, where an element at position $j$ in $\mu_i$ is the average of all the values of the $j^{th}$ attribute for the class $i$.

   **Step 2**: Compute scatter matrices between classes and within class as shown in Equations .

$$S_B = \sum_{i=1}^{c} M_i(\mu_i - m)(\mu_i - m)^T \tag{4}$$

where $S_B$ represents scatter matrix between classes, $M_i$ is size of the respective class, $m$ is the overall mean, considering values from all the classes for each attribute, and $\mu_i$ is the sample mean.

$$S_w = \sum_{i=1}^{c} S_i \tag{5}$$

where $S_w$ is the scatter matrix within class $w$, and $S_i$ is the scatter matrix for the $i^{th}$ class and is given as

$$S_i = \sum_{x \in D_i}^{n} (x_i - \mu_i)(x_i - \mu_i)^t \tag{6}$$

**Step 3**: Compute eigen vectors and eigen values corresponding to scatter matrices.

**Step 4**: Select $k$ eigen vectors that corresponds to largest eigen values and frame a matrix $M$ whose dimensions are $d \times k$.

**Step 5**: Apply transformation $X \times M$, where the dimensions of $X$ are $n \times d$, and $i^{th}$ row is the $i^{th}$ sample. Every row in the matrix $X$ corresponds to an entity in the data set.

This method is found to be unsuitable for the classification problem. The reasons are explained in nezt section.

2. *Support Vector Machine* (SVM): SVM classifiers are effective for binary class discrimination [Hsu, Chang & Lin2016]. The basic formulation is designed for the linear classification problem; the algorithm yields an optimal hyperplane i.e. one that maintains the largest minimum distance from all the training data; it is defined as the margin for separating entities from different classes. For instance, if the two classes are the ones belonging to habitable and non-habitable planets respectively, the problem is a binary classification problem and the hyper-plane must maintain the largest possible distance from the data-points of either class. It can also perform non-linear classification by using *kernels*, which involves the computation of inner products of all pairs of data in the feature space. This implicitly transforms the data into a different space where

a separating hyperplane may be found. The algorithm for classification using SVM, stated briefly, is as follows:

**Step 1**: Create a support vector set $S$ using a pair of points from different classes.

**Step 2**: Add the points to $S$ using Kuhn-Tucker conditions, while there are violating points, add every violating point $V$ to $S$.

$$S = S \cup V \tag{7}$$

If any of the coefficients, $a_p$ is negative due to addition of $V$ to $S$ then prune all such points.

3. *K-Nearest Neighbor* (K-NN): K-nearest neighbors is an instance-based classifier that compares new incoming instance with the data stored in memory [Cai, Duo & Cai2010]. K-NN uses a suitable distance or similarity function and relates new problem instances to the existing ones in the memory. $K$ neighbors are located and majority vote outcome decides the class. For example, let us assume $K$ to be 7. Suppose the test entity has 4 out of the nearest 7 entities belonging to class habitable and the remaining 3 out of the 7 nearest entities belonging to class non-habitable. In such a scenario, the test entity is classified as habitable. However, if the choice of $K$ is 9 and the number of nearest neighbors belonging to class non-habitable is 5, instead of 3, then the test entity will be classified as non-habitable. Occasionally, the high degree of local sensitivity makes the method susceptible to noise in the training data. If $K = 1$, then the object is assigned to the class of that single nearest neighbor. A shortcoming of the K-NN algorithm is its sensitivity to the local structure of the data. K-NN can be understood in an algorithmic way as:

   **Step 1**: Let $X$ is the set of training data, $Y$ be the set of class labels for $X$, and $x$ be the new pattern to be classified.

   **Step 2**: Compute the Euclidean distance between $x$ and all other points in $X$.

   **Step 3**: Create a set $S$ containing $K$ smallest distances.

   **Step 4**: Return majority label for $Y_i$, where $i \in S$.

Surveying various machine learning algorithms was a key motivation even though some methods and algorithms could easily suffice. This explains the reason for describing methods such as SVM, K-NN or LDA even though the results are not very promising for obvious

reasons explained. We reiterate that any learning method is as good as the data and without a balanced data set, there could not exist any reasonable scrutiny of the efficiency of the methods used in the manuscript or elsewhere. In the next subsection, non-metric classifiers which include decision trees, random forests, and extreme gradient boosted trees (XGBoost), would bolster the logic behind discouraging *black box* approaches in data analytics in the context of this problem or otherwise. Readers are advised to pay special attention to the following section.

### 3.3.3 Non-Metric Classifiers

1. *Decision Tree*: A decision tree constructs a tree data structure that can be used for classification or regression [Quinlan1986]. Each of the nodes in the tree splits the training set based on a feature; the first node is called the *root node*, which is based on the feature considered to be the best predictor. Every other node of the tree is then split into child nodes based on a certain splitting criteria or decision rule which determines the allegiance of the particular object (data) to the feature class. A node is said to be more *pure* if the likelihood to classify a given feature vector belonging to class $c_i$ in comparison with any other class $c_j$, for $i \neq j$ is greater. The leaf nodes must be pure nodes, i.e., whenever any data sample that is to be classified reaches a leaf node, it should be classified into one of the classes of the data with a very high accuracy. Typically, an *impurity measure* is defined for each node and the criterion for splitting a node is based on the increase in the *purity* of child nodes as compared to the parent node. In other words, splits that produce child nodes having significantly less impurity as compared to the parent node are favored. The *Gini index* and *entropy* are two popular impurity measures. Gini index interprets the reduction of error at each node, whereas entropy is used to interpret the information gained at a node. One significant advantage of decision trees is that both categorical and numerical data can be dealt with. However, decision trees tend to over-fit the training data. The algorithm used to explain the working of decision trees is as follows:

   **Step 1**: Begin tree construction by creating a node $T$. Since this is the first node of the tree, it is the root node. Classification is of interest only in cases with multiple classes and a root node may not be sufficient for the task of classification. At the root node, all the entities in the training set are considered and a single attribute which results in the least error when used to discern between classes is utilized to *split* the entity set into subsets.

**Step 2**: Before the node is split, the number of child nodes needs be determined. Let us considering a binary valued attribute such as P. Habitable, the resulting number of child nodes after a split will simply be two. In the case of discrete valued attributes, if the number of possible values is more than two, then the number of child nodes may be more than two depending on the DT algorithm used. In the case of continuous valued attributes, a threshold needs to be determined such that minimum error in classification is effected.

**Step 3**: In each of the child nodes, the steps 1 and 2 should be repeated, and the tree should be subsequently grown, until it provides for a satisfactory classification accuracy. An impurity measure such as the Gini impurity index or entropy must be used to determine the best attribute on which the split should be based between any two subsequent levels in the decision tree.

**Step 4**: *Pruning* may be done, while constructing the tree or after the tree is constructed, in order to prevent over-fitting.

**Step 5**: For the task of classification, a test entity is traced to an appropriate leaf node from the root node of the tree.

It is important to observe here and in the later part of the manuscript that DT and other tree based algorithms yield significantly better results for balanced as well as biased data.

2. *Random Forest*: A random forest is an ensemble of multiple decision trees. Each tree is constructed by selecting a random subset of attributes from the data set. Each tree in turn performs a regression or a classification and a decision is taken based on mean prediction (regression) or majority voting (classification) [Breiman2001]. The task of classifying a new object from the data set is accomplished using randomly constructed trees. Classification requires a *tree voting* for a class i.e. the test entity is classified as class $c_i$ if a majority of the decision trees in the forest classified the entity into class $c_i$. For example, if a random forest consists of ten decision trees, out of which six trees classified a feature vector $x$ as belonging to the class of psychroplanets, and the remaining four trees classified $x$ as being non-habitable, then we may conclude that the random forest classified $x$ as a psychroplanet.

   Random forests work efficiently with large data sets. The training algorithm for random forests applies the general technique of bootstrap aggregation or *bagging* to tree learn-

ers. Given a training set $X = \{x_1, x_2, ..., x_n\}$ with class labels $Y = \{y_1, y_2, ..., y_n\}$, bagging selects random samples from the training set with iterative replacement and fits trees to these samples subsequently. The algorithm for classification may be described as:

**Step 1**: For $a = 1, ..., N$ and for $b = 1, 2, ..., M$ sample with replacement, $n$ training samples from $X$ with the corresponding set of $Y_m$ features from $Y$; let this subset of samples be denoted as $X_a$, $Y_b$.

**Step 2**: Next, the $i^{th}$ decision tree is trained: $f_i$ on $X_a$, $Y_b$. Steps 1 and 2 are repeated for as many trees as desired in the random forest.

**Step 3**: After training, predictions for unseen samples $x'$ can be made by considering the majority votes from all the decision trees in the forest.

The brief primer on non-metric classifiers is terminated by including a recently developed boosted-tree machine learning algorithm, XGBoost.

3. *XGBoost*: XGBoost [Chen & Guestrin2016] is another method of classification that is similar to random forests: it uses an ensemble of decision trees. The major departure from random forest lies in how the trees in XGBoost are trained. XGBoost uses *gradient boosting*. Unlike random forests, an objective function is minimized and each leaf has an associated score which determines the class membership of any test entity. Subsequent trees constructed in a forest of XGBoosted trees must minimize the chosen objective function so that there is measured improvement in classification accuracy as more trees are constructed. The steps in XGBoosted trees are as follows:

   **Step 1**: For $a = 1, ..., N$ and for $b = 1, 2, ..., M$ sample with replacement, $n$ training samples from $X$ with the corresponding set of $Y_m$ features from $Y$; let this subset of samples be denoted as $X_a$, $Y_b$.

   **Step 2**: Next, the $i^{th}$ decision tree is trained: $f_i$ on $X_a$, $Y_b$. Steps 1 and 2 are repeated for as many trees as desired in the random forest.

   **Step 3**: Steps 1 and 2 are repeated by considering more trees. Subsequent trees must be chosen carefully so as to minimize the value of a chosen objective function. The results from each tree are added, that is each tree then contributes to the decision.

   **Step 4**: Once the model is trained, the prediction can be done in a way similar to random forests, but by making use of structure scores.

For more details on the working principles of XGBoost and a brief illustrative example, the reader should refer to Appendix.

## 3.4 Framework and Experimental Set Up

### 3.4.1 Data Acquisition: Web Scraping

The data is retrieved from Planetary Habitability Laboratory, University of Puerto Rico which is regularly updated with new data and discovery. Therefore, web scraping helps to easily update any local repository on a remote computer. Web scraping is a method of extracting data from web pages, given the structure of the web page is known a prior. The positioning of HTML tags and meta-data in a web page may be used for developing a scraper.

Figure 1 presents the outline of the scraper used to retrieve data from the website of the Planetary Habitability Laboratory. Modern web browsers are equipped with utilities for exploring structures of web pages. By using such inspection tools to understand the structure of web pages, scrapers may be developed to retrieve data present in HTML pages already. The steps in developing a scraper are explained below:

1. **Explore Website Structure**

   The first step in the process of developing a scraper is to understand the structure of the web pages. The position of HTML tags is carefully studied and patterns are discovered that may help define the placement of desired data.

2. **Create Scraping Template**

   Based on the knowledge gained in the first step, a template is designed that allows a program to extract data from a web page. In essence, a web page is a long string of characters. A set of web pages which display similar data may have similar characteristic structure and hence a single template may be used to extract data from similar web pages.

3. **Automate Navigation and Extraction**

   Once a template is developed, a scraper may be deployed to automatically collect data from web pages. It may be scheduled to update a local catalog or may be run as and when required. The authors did not schedule the scraper to run at regular intervals since that was not needed. The necessity may arise in future and scheduling may be acted upon.

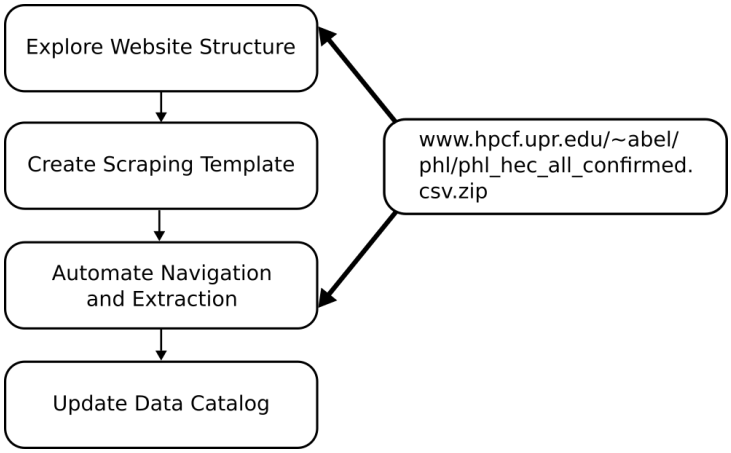



**Figure 1:** Steps in scraper

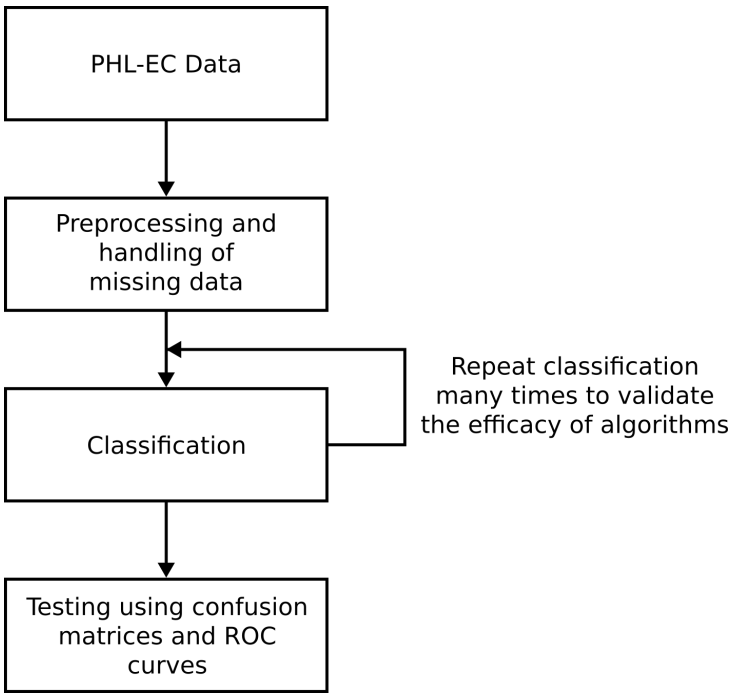



**Figure 2:** Overview of the steps in the analysis of data.

4. **Update Data Catalog**

   As a good practice, most scrapers update a local catalog. As the scraping process progresses, newly added or altered data should be updated thus avoiding redundant data handling. As and when a new or altered element is discovered, it should be immediately updated in the local catalog before retrieving the next element in sequence.

### 3.4.2 Classification of Data

PHL-EC has been derived from the Hipparcose catalog which contains 118,219 stars. PHL-EC was created from the Hipparcose Catalog by examining the information on distances, stellar variability, multiplicity, kinematics, and spectral classification for the stars contained therein. In this study, PHL-EC has been used because it provides an expanded target list for use in the search for extraterrestrial intelligence by Project Phoenix of the SETI Institute. PHL-EC data set consists of a total of 68 features and about 3500 confirmed exoplanets (at the time of writing of this paper). The reason behind selecting PHL-EC as the source of data is that it combines measures and modeled parameters from various sources. Hence, it provides a good metric for visualization and statistical analysis [Méndez2011]. Statistical machine learning approaches have not been applied on this data set, to the best of the authors' knowledge, providing good reasons to explore and exploit accuracy of different machine learning algorithms.

The PHL-EC data set possesses 13 categorical features and 55 continuous features. There are three classes in the data set, namely non-habitable, mesoplanets, and psychroplanets on which the ML methods have been tried (there do exist other classes in the data set on which the methods cannot be tried the reasons for which has been explained in Section 3.6.1. These three labels or classes or types of planets (for the purpose of classification) can be defined on the basis of their thermal properties as follows:

1. **Mesoplanets** [Asimov1989]: The planetary bodies whose sizes lie between Mercury and Ceres falls under this category (smaller than Mercury and larger than Ceres). These are also referred to as M-planets [Méndez2011]. These planets have mean global surface temperature between 0°C to 50°C, a necessary condition for complex terrestrial life. These are generally referred as Earth-like planets.

2. **Psychroplanets** [Méndez2011]: These planets have mean global surface temperature between -50°C to 0°C. Hence, the temperature is colder than optimal for sustenance of terrestrial life.

3. **Non-Habitable**: Planets other than mesoplanets and psychroplanets do not have thermal properties required to sustain life.

The catalog includes features like atmospheric type, mass, radius, surface temperature, escape velocity, earth's similarity index, flux, orbital velocity etc. Online data source for the current work is available at http://phl.upr.edu/projects/habitable-exoplanets-catalog/data/database.

The data flow diagram of the entire system is depicted in Figure 2. As a first step, data from PHL-EC is pre-processed (the authors have tried to tackle the missing values by taking mean for continuous valued attribute and mode for categorical attributes). Certain attributes from the database namely P.NameKepler (planet's name), Sname HD and Sname Hid (name of parent star), S.constellation (name of constellation), Stype (type of parent star), P.SPH (planet standard primary habitability), P.interior ESI (interior earth similarity index), P.surface ESI (surface earth similarity index), P.disc method (method of discovery of planet), P.disc year (year of discovery of planet), P. Max Mass, P. Min Mass, P.inclination and P.Hab Moon (flag indicating planet's potential as a habitable exomoons) were removed as these attributes do not contribute to the nature of classification of habitability of a planet. Interior ESI and surface ESI, however, together contribute to habitability, but since the data set directly provides P.ESI, these two features were neglected. Following this, classification algorithms were applied on the processed data set. In all, 51 features are used.

Initially, a ten-fold cross-validation procedure was carried out, that is, the entire data set was divided into ten bins in which one of the bins was considered as test-bin while the remaining 9 bins were taken as training data. In this method the data is sampled without replacement. Later, upon careful exploration of the data, more robust *artificial balancing* methods were used. The details are enunciated in Sections 3.5.2 and 3.5.3.

Scikit-learn [Pedregosa et al.2011] was used to perform these experiments. A brief overview of the classifiers used and their respective settings (in Scikit-learn) are provided below:

1. *Gaussian Naïve Bayes* evaluates the classification labels based on class conditional probabilities with class apriori probabilities, class count, mean and variance set to default values.

2. The *k-nearest neighbor* classifier was used with the $k$ value being set to 3 while the weights are assigned uniform values and the algorithm was set to auto.

3. *Support vector machines*, a binary classifier was used with a penalty parameter $C$ of the error term, initialized to default 1.0 while the kernel used was that of a radial basis function (RBF) [Powell1977] and the gamma parameter (kernel coefficient) was assigned to 0.0 and coefficient of the kernel was set to 0.0 as well.

4. The parameters setup for *linear discriminant analysis* classifier was implemented by the decomposition strategy similar to *SVM* [Eckart & Young1936, Hestenes1958]. No shrinkage metric was specified and no class prior probabilities were assigned.

5. *Decision trees* build tree based structures by using a split criterion namely Gini impurity, with measure of split being selected as best split and no max-depth and min-depth were specified whereas a *random forest* is an ensemble of decision trees with estimator value set up to 100 trees; the remaining parameters were set to the same as the decision tree.

6. *XGBoost* is a recent ensemble tree-based method which optimizes the tree being built. For this algorithm, the maximum number of estimators chosen to develop a classifier was 1000 and the maximum permissible depth of each tree bound at 8. The objective function used was that of a multinomial softmax.

## 3.5 Complexity of the data set used and Results

### 3.5.1 Classification performed on an unbalanced and smaller Data Set

Initially, 664 planets were considered, as their surface temperature was known out of which 9 planets were mesoplanets and 7 planets were psychroplanets, from the data set scraped in June 2015 [Méndez2015]. These planets selected for classification at this stage were rocky planets, deemed more habitable than planets of other terrain. The accuracy of all classifiers are documented in Table 1.

The PHL-EC data set is too complex for an immediate application of classifiers. The cause of the initial high accuracy is due to the *data bias* of a single class: The non-habitable class dominates over all the other classes. The sensitivity and specificity using this method were both very close to 1, for all classifiers.

**Table 1:** Accuracy of each algorithm executed on unbalanced PHL-EC data set

| Algorithm | Accuracy (%) |
|---|---|
| Naïve Bayes | 98.7 |
| Decision Tree | 98.61 |
| LDA | 93.23 |
| K-NN | 97.84 |
| Random Forest | 98.7 |
| SVM | 97.84 |

### 3.5.2 Classification performed on a balanced and smaller data set

Unbelievably high accuracy for all methods, metric and non-metric in the unbalanced data set and unreasonable sensitivity and specificity values were recorded. Bias towards a particular class, as evident from the number of samples across the different classes in the data set, was responsible for this. The efficacy of ML algorithms can not be judged when such a bias is present. Therefore, the data set needed to be balanced artificially so that dominance of one particular class samples is removed from the and the real picture emerges regarding the appropriateness of a particular machine learning algorithm, metric or otherwise.

To counter the problems faced in the first phase of research with regard to data bias, smaller data sets were constructed by selecting all planets belonging to mesoplanet and psycroplanet classes and selecting 10 planets which belonged to the non-habitable class at random, resulting in 26 planets in a smaller, artificially balanced data set. Classification and testing was then performed on each artificially balanced data set. In every iteration of testing on a smaller data set, the test data was formed by selecting one entity from mesoplanet, one from psychroplanet and two from non-habitable; the remaining entities from all classes were used as training data for that respective training-testing cycle. All possible combinations of training and test data were used, resulting in $\binom{8}{1} \times \binom{7}{1} \times \binom{10}{2} = 2835$ training and testing cycles for each smaller data set. Five hundred iterations, or artificially balanced data sets, were formed and tested for each classifier. The data set used by the authors can be obtained by clicking on the link: $https://github.com/SuryodayBasak/exoplanets\_data$. It should be noted that this method is not the same as that of blatant *undersampling* to counter the effects of bias. Rather, after the artificial balancing is done, a large number of iterations of the experiments are performed. As the process of selecting random non-habitable samples is stochastic in nature, by increasing the number of training-testing iterations and averaging the classification accuracy, the test results become more reliable and representative of the performance of the ML classifiers.

**Table 2:** Accuracy of each algorithm executed on pre-processed and artificially balanced PHL-EC data set.

| Algorithm | Curve Color | Accuracy(%) |
|---|---|---|
| Random Forest | Green | 96.667 |
| Decision Tree | Red | 96.697 |
| Naïve Bayes | Cyan | 91.037 |
| LDA | Magenta | 84.251 |
| K-NN | Blue | 72.191 |
| SVM | Yellow | 79.055 |

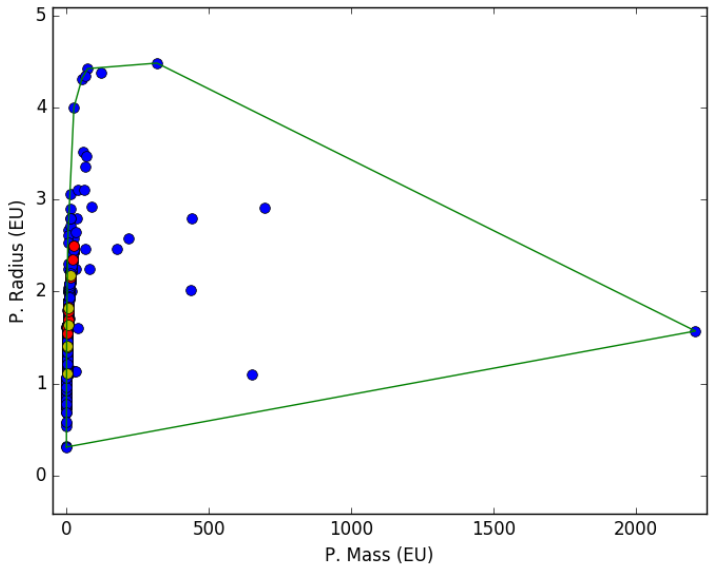



**Figure 3:** Convex hull shown across two dimensions.

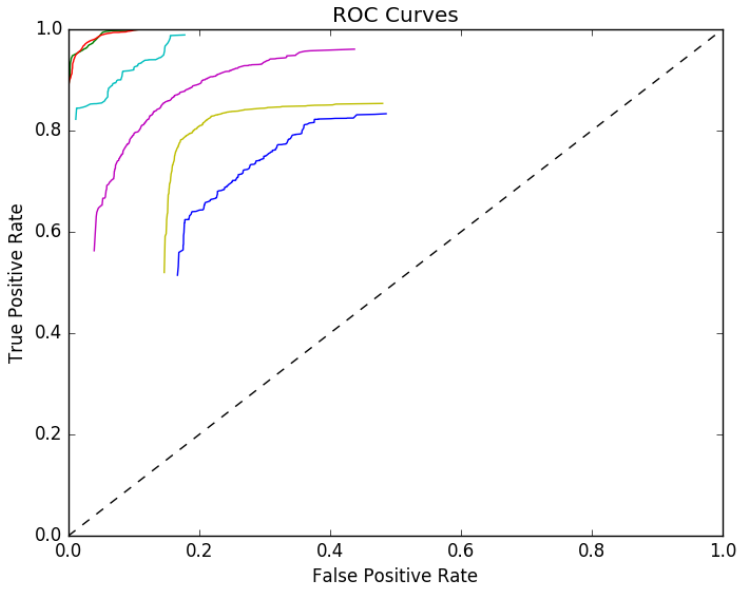



**Figure 4:** ROC curves for each method used on artificially balanced data sets.

A *separability test* was also performed on the data in order to determine if the data set is linearly separable or not. If the different classes in data are not linearly separable, certain classifiers may not work well or may not be appropriate for the respective application. The *convex hull* of different classes in the data provides us with an indication of separability: the convex hull of a given set of points is the smallest $n$ dimensional polygon which can adequately envelop all the points in the respective set, where $n$ is the number of attributes of the points. If the convex hull of any two or more classes intersect or overlap, then it may be concluded that the classes of data are not linearly separable. Figure 3 depicts the convex hull of data across two dimensions (P. Mass vs P. Radius). Although only the convex hull test considering all the dimensions of the data is completely representative of separability, a graph across two dimensions is depicted for simplicity as it is difficult to plot the convex hull for all pairs of features for a data set with many dimensions. The data points in blue represent the entities belonging to the non-habitable class, red represents mesoplanets and yellow represents psychroplanets. It is observed that the data belonging to the classes of mesoplanet and psychroplanet are present within the convex hull of the class non-habitable. Thus, the three classes in the data set are linearly inseparable.

The accuracy and ROC curves of the different classifiers after artificially balancing the data set are shown in Table 2 and Figure 4 respectively. The receiver operating characteristics (ROC) curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR). The ROC curve is a useful tool for visualizing and analyzing the performance of a classifier and selecting the classifier with the best performance for a given data set. Simply stated, the closer the points in a curve are towards the top left corner, the better the performance of a classifier is, and vice-versa.

### 3.5.3 Classification performed on a balanced and larger data set

An updated version of the data set was scraped on 20th May 2016. This data set had 3411 entries: 24 mesoplanets, 13 psychroplanets, and 3374 non-habitable planets. At this stage, after the preliminary explorations described in Sections 3.5.1 and 3.5.2, the authors decided not to leave out any of the planets from the ML analysis: all of the 3411 entities were considered for determining the habitability. The number of items in this data set was significantly more than the older data set used to have. Hence, the artificial balancing method was modified. In the new balancing method, all 13 psychroplanets were considered in a smaller data set and 13 random and unique entities from each of the other two classes were also considered. Thus, in this case, the number of entities in a smaller, artificially balanced data set was 39. Following this, each smaller data set was divided in the ratio of 9:4 (training:testing) and 500

**Table 3:** Accuracy of each algorithm executed on pre-processed, artificially balanced and updated PHL-EC data set without seven attributes

| Algorithm | Accuracy(%) |
|---|---|
| Random Forest | 96.466 |
| Decision Tree | 95.1376 |
| Naïve Bayes' | 91.3 |
| LDA | 84.251 |
| K-NN | 59.581 |
| SVM | 39.7792 |

iterations of training and testing were performed on each such data set. 500 such data sets were framed for analysis. To sum it up, 2,50,000 iterations of training-testing were performed for each classifier.

1. **First Iteration of the Experiment**

   Initial analysis using the updated data set did not include the attributes such as P.SFlux Min, P.SFlux Max, P.Teq Min, P.Teq Max, P.Ts Min, P.Ts Max, and P.Omega. For temperature and flux, the corresponding average values were considered as the authors tried to make do with a lesser number of attributes by considering just the mean of the equilibrium and surface temperatures. The accuracy observed at this phase is recorded in Table 3.

2. **Second Iteration of the Experiment**

   In the next step, all the seven attributes initially not considered were included in the data set for analysis. This is considered to be a complete analysis and the accuracy achieved at this stage is reported in three separate sub-subsections.

   (a) **Naïve Bayes**

   The accuracy achieved using the Gaussian naïve Bayes Classifier is 92.583%. The ROC curve for this is given in Figure 5. The results of naïve Bayes is given in Table 4.

   (b) **Metric Classifiers**

   The accuracy using metric classifiers is given in Table 5. The corresponding color of curves in the ROC is present as a column. The ROC curve for all the metric classifiers is given in Figure 6.

**Table 4:** Sensitivity, accuracy, precision, and specificity achieved using naïve Bayes

| Class | Sensitivity | Accuracy | Precision | Specificity |
|---|---|---|---|---|
| Non-Habitable | 0.9999 | 0.9883 | 0.9999 | 0.9649 |
| Psychroplanet | 0.8981 | 0.9173 | 0.8243 | 0.9558 |
| Mesoplanet | 0.9691 | 0.9173 | 0.9295 | 0.8136 |

**Table 5:** Accuracy and ROC curve colors for metric classifiers

| Algorithm | Curve Color | Accuracy(%) |
|---|---|---|
| SVM | Blue | 36.489 |
| K-NN | Green | 68.607 |
| LDA | Red | 77.396 |

(c) **Non-Metric Classifiers** The accuracy using non metric classifiers is given in Table 9. The corresponding color of curves in the ROC is present as a column. The ROC curve for all the non metric classifiers is given in Figure **??**.

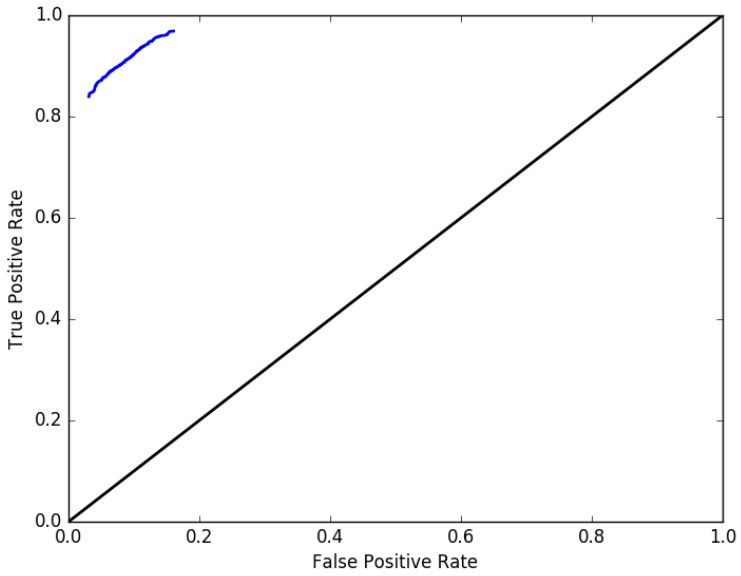



**Figure 5:** ROC for Gaussian naïve Bayes Classifier

As the data is linearly inseparable, classifiers utilizing the separability of data naturally performed less efficiently. Such classifiers are metric classifiers and include SVM, LDA, and K-NN as discussed before. SVM and LDA both work by constructing a hyperplane between classes of data: LDA constructs a hyper-plane by assuming the data from each

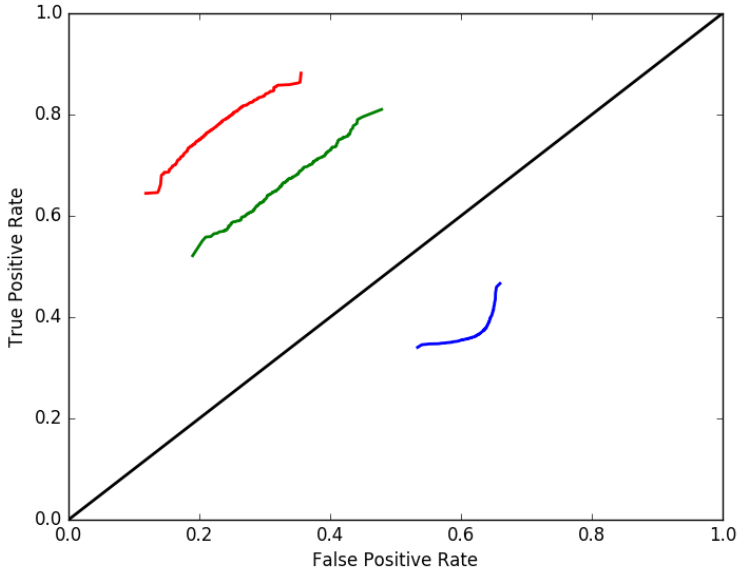



**Figure 6:** ROC curves for metric classifiers

**Table 6:** Sensitivity, accuracy, precision, and specificity achieved using SVM classifier

| Class | Sensitivity | Accuracy | Precision | Specificity |
|---|---|---|---|---|
| Non-Habitable | 0.8216 | 0.6151 | 0.3617 | 0.2022 |
| Psychroplanet | 0.7517 | 0.6268 | 0.4316 | 0.3771 |
| Mesoplanet | 0.4869 | 0.5050 | 0.3453 | 0.5412 |

**Table 7:** Sensitivity, accuracy, precision, and specificity achieved using K-NN classifier

| Class | Sensitivity | Accuracy | Precision | Specificity |
|---|---|---|---|---|
| Non-Habitable | 0.9998 | 0.9585 | 0.9996 | 0.8759 |
| Psychroplanet | 0.7200 | 0.6962 | 0.5366 | 0.6486 |
| Mesoplanet | 0.7797 | 0.6779 | 0.5184 | 0.4744 |

**Table 8:** Sensitivity, accuracy, precision, and specificity achieved using LDA classifier

| Class | Sensitivity | Accuracy | Precision | Specificity |
|---|---|---|---|---|
| Non-Habitable | 0.9935 | 0.9417 | 0.9847 | 0.8382 |
| Psychroplanet | 0.8520 | 0.8030 | 0.7042 | 0.7050 |
| Mesoplanet | 0.8155 | 0.8032 | 0.6785 | 0.7787 |

**Table 9:** Accuracy and ROC curve colors for non-metric classifiers

| Algorithm | Curve Color | Accuracy(%) |
|---|---|---|
| Decision Tree | Blue | 94.542 |
| Random Forests | Green | 96.311 |
| XGBoost | Red | 93.960 |

**Table 10:** Sensitivity, accuracy, precision, and specificity achieved using decision tree classifier

| Class | Sensitivity | Accuracy | Precision | Specificity |
|---|---|---|---|---|
| Non-Habitable | 0.9926 | 0.9691 | 0.9843 | 0.9220 |
| Psychroplanet | 0.9610 | 0.9578 | 0.9242 | 0.9512 |
| Mesoplanet | 0.9479 | 0.9419 | 0.8993 | 0.9299 |

class to be normally distributed with the parameters of mean and co-variance for the respective classes; SVM is a relatively recent kernel based method. Considering binary classification, in both cases, the hyperplane defines a threshold and the classes are assigned based on the response of a function $g(x)$, which may be higher or lower than the threshold. For example, if the output of $g(x)$ is greater than the corresponding threshold for any data point $x_1$, then the associated class may be *Class-1* and if it is lower than the threshold, then the associated class may be *Class-0*. For tasks involving multi-class classification, an appropriate set of thresholds is defined (based on the number of required hyperplanes) and the function $g(x)$ then has to find the class to which the data corresponds best by considering appropriate conditions for membership to each class. If data is linearly inseparable, it becomes nearly impossible to appropriately define a hyperplane which may adequately separate the different classes of data in a vector space. The K-NN classifier works on the basis of similarity to nearest neighbors. Even in this case, chances of error increases as the method works based on geometric similarity to singular regions in a vector space corresponding to each class.

Decision trees, random forests, and XGBoost, on the other hand, are non-metric classifiers. These classifiers do not work by constructing hyperplanes or consider kernels. These classifiers are able to divide the feature space into multiple regions corresponding

**Table 11:** Sensitivity, accuracy, precision, and specificity achieved using random forest classifier

| Class | Sensitivity | Accuracy | Precision | Specificity |
|---|---|---|---|---|
| Non-Habitable | 0.9990 | 0.9811 | 0.9978 | 0.9452 |
| Psychroplanet | 0.9617 | 0.9681 | 0.9276 | 0.9809 |
| Mesoplanet | 0.9757 | 0.9661 | 0.9513 | 0.9468 |

**Table 12:** Sensitivity, accuracy, precision, and specificity achieved using XGBoost classifier

| Class | Sensitivity | Accuracy | Precision | Specificity |
|---|---|---|---|---|
| Non-Habitable | 0.9993 | 0.9677 | 0.9984 | 0.9046 |
| Psychroplanet | 0.9613 | 0.9599 | 0.9252 | 0.9572 |
| Mesoplanet | 0.9489 | 0.9515 | 0.9034 | 0.9569 |

to a single class and the same is done for all the classes. This is a measure to overcome the limitation of the requirement of separability among classes of data in a classifier. Hence, the results from these classifiers are the best. A probabilistic classifier, Gaussian naïve Bayes performs better than the metric classifiers due to the strong independence assumptions between the features.

Different specificity and sensitivity values, along with the precision are given in Tables 4, 6, 7, 8, 10, 11 and 12 for the classification algorithms.

## 3.6 Discussion

### 3.6.1 Note on new classes in PHL-EC

Two new classes appeared in the augmented data set scraped on 28th May 2016. These two new classes are:

1. **Thermoplanet**: A class of planets, which has a temperature in the range of 50°C-100°C. This is warmer than the temperature range suited for most terrestrial life [Méndez2011].

2. **Hypopsychroplanets**: A class of planets whose temperature is below −50°C. Planets belonging to this category are too cold for the survival of most terrestrial life [Méndez2011].

   The above two classes have two data entities each in the augmented data set used. This number is inadequate for the task of classification, and hence the total of four entities were excluded from the experiment.

### 3.6.2 Missing attributes

It can be expected in any data set for feature values to be missing. The PHL-EC too has attributes with missing values.

1. The attributes of P. Max Mass and P. SPH were dropped as they had too few values for some algorithms to consider them as strong features.

2. All other named attributes such as the name of the parent star, planet name, the name in Kepler's database, etc. were not included as they do not have much relevance in a data analytic sense.

3. For the remaining, if a data sample had a missing value for a continuous valued attribute, then the mean of the values of all the available attributes for the corresponding class was substituted. In the case of discrete valued attributes, the same was done, but using the mode of the values.

After the said features were dropped, about 1% of all the values of the the data set used for analysis were missing. Most algorithms require the optimization of an objective function. Tree based algorithms also need to determine the importance of features as they have to optimally split every node till the leaf nodes are reached. Hence, ML algorithms are equipped with mechanisms to deal with important and non-important attributes.

### 3.6.3 Reason for extremely high accuracy of classifiers before artificial balancing of data set

Since the data set is dominated by the non-habitable planets class, it is essential that the training sets used for training the algorithms be artificially balanced. The initial set of results achieved were not based on artificial balancing and are described in Table 1. Most of the classifiers resulted in an accuracy between 97% and 99%.

In the data set, the number of entities in the non-habitable class is greater than 1000 times the number of entities in both the other classes put together. In such a case, voting for the dominating class naturally increases as the number of entities belonging to this class is greater: the number test entities classified as non-habitable are far greater than the number of test entities classified into the other two classes. The extremely high accuracies depicted in Table 1 is because of the dominance of one class and not because the classes are correctly identified. In such a case, the sensitivity and specificity are also close to 1. Artificial balancing is thus a necessity unless a learning method is designed specifically which auto corrects the imbalance in the data set. Performing classification on the given data set straightaway is not an appropriate methodology and artificial balancing is a must. Artificial balancing was done by selecting 13 entities from each class. This number corresponds to the number of total entities in the class of psychroplanets.

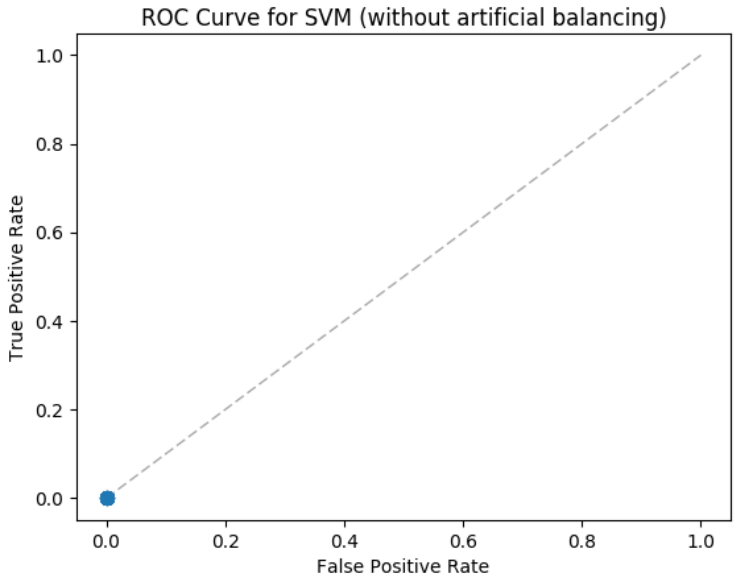



**Figure 7:** ROC for SVM without artificial balancing

### 3.6.4 Demonstration of the necessity for artificial balancing

Predominantly in the case of metric classifiers, an imbalanced training set can lead to misclassifications. The classes which are underrepresented in the training set might not be classified as well as the dominating class. This can be easily analyzed by considering the *area under the curve* (AUC) of the ROC of the metric classifiers in the case of balanced and imbalanced training sets. As an illustration: the AUC of SVM for the unbalanced training set (tested using a a balanced test set) is 0%, but after artificial balancing, it comes up to approximately 37%. The ROC for the unbalanced case is shown in Figure 7. The marker at (0,0) shows the only point in the plot; the FPR and TPR are observed to be constantly zero for SVM without artificial balancing. Similarly, in the case of other metric classifiers, classification biases can be eliminated using artificial balancing.

### 3.6.5 Order of importance of features

In any large data set, it is natural for certain features to contribute more towards defining the characteristics of the entities in that set. In other words, certain features contribute more towards class belongingness than certain others. As a part of the experiments, the authors wanted to observe which features are more important. The ranks of features and the percentage importance for random forests and for XGBoosted trees are presented in Tables

13 and 14 respectively. Every classifier uses the features in a data set in different ways. That is why the ranks and percentage importances observed using random forests and XGBoosted trees are different. The feature importances were determined using artificially balanced data sets.

### 3.6.6 Why are the results from SVM, K-NN and LDA relatively poor?

As the data set has been improving since the first iteration of the classification experiments, the authors were able to understand the nature of the data set better with time. With the continuous augmentation of the data set, it is easier to understand why some methods work poorer compared to others.

As mentioned in Section 3.5.2, the data entities from different classes are not linearly separable. This is proved by finding the data points from the classes of mesoplanets and psychroplanets within the convex hull of the non-habitable class. Classifiers such as SVM and LDA rely on data to be separable in order to optimally classify test entities. Since this condition is not satisfied by the data set, SVM and LDA have not performed as well as other classifiers such as random forest or decision trees.

SVM with radial basis kernels performed poorly as well. The poor performance of LDA and SVM may be attributed to the similar trends that entities from all the classes follow as observed from Figure 3. Apart from a few outliers, most of the data points follow a logarithmic trend and classes are geometrically difficult to discern.

K-NN also classifies based on geometric similarity and has a similar reason for poor performance: the nearest entities to a test entity may not be from same class as the test class. K-NN is observed to perform best when the value of $k$ is between 7 and 11 [Hassanat et al.2014]. In our data set, these numbers almost correspond to the number of entities present in the classes of mesoplanets and psychroplanets; the number of entities belonging to mesoplanets and psychroplanets are inadequate for the best performance.

### 3.6.7 Reason for better performance of decision trees

A decision tree algorithm can detect the most relevant features for splitting the feature space. A decision tree may consequently be pruned while growing or after it is fully grown. This prevents over-fitting of the data and yields good classification results.

In decision trees, an $n$-dimensional space is partitioned into multiple parts corresponding to a single class. Unlike SVM or LDA, there isn't a single portion of the $n$-dimensional space corresponding to a single class. The advantage of this is that this can handle non-linear trends.

**Table 13:** Ranks of features based on random forests

| Rank | Attribute | Percent Importance |
|---|---|---|
| 1 | P. Ts Mean (K) | 6.731 |
| 2 | P. Ts Min (K) | 6.662 |
| 3 | P. Teq Min (K) | 6.628 |
| 4 | P. Teq Max (K) | 6.548 |
| 5 | P. Ts Max (K) | 6.49 |
| 6 | S. Mag from Planet | 6.399 |
| 7 | P. Teq Mean (K) | 6.393 |
| 8 | P. SFlux Mean (EU) | 6.366 |
| 9 | P. SFlux Max (EU) | 6.292 |
| 10 | P. SFlux Min (EU) | 6.264 |
| 11 | P. Mag | 4.216 |
| 12 | P. HZD | 3.822 |
| 13 | P. Inclination (deg) | 3.732 |
| 14 | P. Min Mass (EU) | 3.571 |
| 15 | P. ESI | 3.177 |
| 16 | S. No. Planets HZ | 3.014 |
| 17 | P. Habitable | 3.005 |
| 18 | P. Zone Class | 2.82 |
| 19 | P. HZI | 1.627 |
| 20 | S. Size from Planet (deg) | 1.376 |
| 21 | P. Period (days) | 1.034 |
| 22 | S. Distance (pc) | 0.54 |
| 23 | S. [Fe/H] | 0.42 |
| 24 | P. Mean Distance (AU) | 0.379 |
| 25 | S. Teff (K) | 0.251 |
| 26 | P. Sem Major Axis (AU) | 0.227 |
| 27 | S. Age (Gyrs) | 0.17 |
| 28 | S. Luminosity (SU) | 0.156 |
| 29 | S. Appar Mag | 0.145 |
| 30 | S. Mass (SU) | 0.134 |
| 31 | S. Hab Zone Max (AU) | 0.128 |
| 32 | P. Appar Size (deg) | 0.12 |
| 33 | S. Hab Zone Min (AU) | 0.118 |
| 34 | S. Radius (SU) | 0.097 |
| 35 | P. Radius (EU) | 0.095 |
| 36 | P. Eccentricity | 0.089 |
| 37 | P. HZC | 0.088 |
| 38 | P. Density (EU) | 0.083 |
| 39 | S. No. Planets | 0.08 |
| 40 | P. Gravity (EU) | 0.078 |
| 41 | P. Mass (EU) | 0.076 |
| 42 | P. HZA | 0.074 |
| 43 | S. DEC (deg) | 0.066 |
| 44 | P. Surf Press (EU) | 0.065 |
| 45 | P. Mass Class | 0.055 |
| 46 | P. Esc Vel (EU) | 0.049 |
| 47 | S. RA (hrs) | 0.045 |
| 48 | P. Omega (deg) | 0.018 |

**Table 14:** Ranks of features based on XGBoost

| Rank | Attribute | Percent Importance |
|---|---|---|
| 1 | S. HabCat | 25.0 |
| 2 | P. Ts Mean (K) | 25.0 |
| 3 | P. Mass (EU) | 25.0 |
| 4 | P. SFlux Mean (EU) | 25.0 |

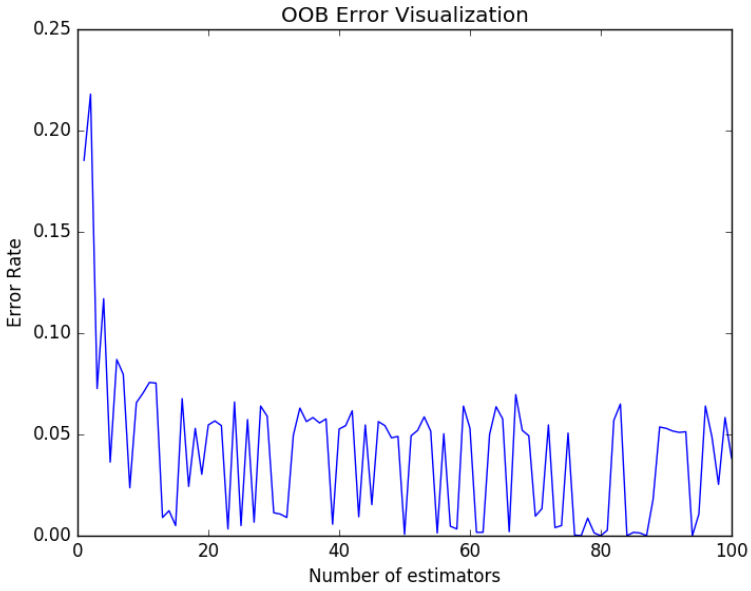



**Figure 8:** Decrease in OOB error with increase in number of trees in RF

This kind of an approach is appropriate for the PHL-EC data set as the trends in the data are not linear and classes are difficult to discern. Hence, multiple partitions of the feature space can greatly improve classification accuracy.

### 3.6.8 Explanation of OOB error visualization

Figure 8 shows the decrease in error rate as the number of tree estimators increase. After a point, the error rate fluctuates between approximately 0% and 6%. The decrease in the error rate with an increase in the number of trees to a smaller range of error testifies convergence in random forests.

### 3.6.9 What is remarkable about random forests?

Decision trees are often encountered with the problem of over-fitting i.e. ignorance of a variable in case of small sample size and large p-value (however, in the context of the work presented in this paper, this is not observed since unnecessary predictor variables are pruned). In contrast, random forests bootstrap aggregation or *bagging* [Breiman2001] which is particularly well-suited to problems with small sample size and large p-value. The PHL-EC data set is not large by any means. Random forest, unlike decision trees, do not require split sampling method to assess accuracy of the model. Self-testing is possible even if all the data is used for training as $2/3^{rd}$ of available training data is used to grow any one tree and the remaining one-third portion of the training data is used to calculate out-of-bag error. This helps assess model performance.

### 3.6.10 Random forest: mathematical representation of binomial distribution and an example

In random forests, approximately $2/3^{rd}$ of the total training data is used for growing each tree, and the remaining $1/3^{rd}$ of the cases are left out and not used in the construction of trees. Each tree returns a classification result or a *vote* for a class corresponding to each sample to be classified. The forest chooses the classification having the majority votes over all the trees in the forest. For a binary dependent variable, the vote will be *yes* or *no*; the number of affirmative votes is counted: this is the RF score and the percentage of affirmative votes received is the predicted probability of the outcome being correct. In the case of regression, it is the average of the responses from each tree.

In any DT which is a part of a random forest, an attribute $x_a$ may or may not be included. The inclusion of an attribute in a Decision Tree is of the $yes/no$ form. The binary nature of dependent variables is easily associated with *binomial distribution*. This implies that the probability of inclusion of $x_a$ is binomially distributed. As an example, consider that a random forest consists of 10 trees, and the probability of correct classification due to an attribute $x_a$ is 0.6. The probability mass function of the binomial distribution is given by Equation (8).

$$Pr(X = k) = \binom{n}{k} p^k (1-p)^{n-k} \tag{8}$$

It is easy to note that $n = 10$ and $p = 0.6$. The value of $k$ indicates the number of times an attribute $x_a$ is included in a DT in the forest. Since $n = 10$, the values of $k$ may be $0, 1, 2, ..., 10$.

$k = 0$ implies that the attribute is never accounted for in the forest, and $k = 10$ implies that $x_a$ is considered in all the trees.

The cumulative distribution function (CDF) for the binomial distribution is given by Equation (9).

$$Pr(X \leq m+1) = \sum_{k=0}^{n} \binom{n}{i} p^i (1-p)^{n-k} \tag{9}$$

For $n = 10$, $p = 0.6$ and $m = 10$, the probability for *s*uccess in Equation (10):

$$Pr(X \leq m+1) = \sum_{k=0}^{10} \binom{10}{k} (0.6)^k (0.4)^{n-k} = 1.0 \tag{10}$$

As $k$ assumes a larger value, the value of the Cumulative Distribution approaches 1. This indicates a greater probability of success or correct classification. It follows that, increasing the number of decision trees consequently reduces the effect of noise and if the features are generally robust, the classification accuracy gets reinforced.

## 3.7 Binomial distribution based confidence splitting criteria

The binomially distributed probability of correct classification of an entity may be used as a node-splitting criteria in the constituent DT of an RF. From the cumulative binomial distribution function, the probability of $k$ or more entities of class $i$ occurring in a partition $A$ of $n$ observations with probability greater than or equal to $p$ is given by the binomial random variable as in Equation (11).

$$X(n,p) = P[X(n,p_i) \geq k] = 1 - B(k;n,p_i) \tag{11}$$

As the value of $X(n,p)$ tends to zero, the probability of the partition $A$ being a pure node, with entities belonging to only class $i$, increases. However, an extremely low value of $X(n,p)$ may lead to an over-fitting of of data, in turn reducing classification accuracy. A way to prevent this is to use a *confidence threshold*: the corresponding partition or node is considered to be a *pure* node if the value of $X(n,p)$ exceeds a certain threshold.

Let $c$ be the number of classes in the data and $N$ be the number of *outputs*, or *branches* from a particular node $j$. If $n_j$ is the number of entities in the respective node, $k_i$ the number of entities of class $i$, and $p_i$ the minimum probability of occurrence of $k_i$ entities in a child node, then the model for the confidence based node splitting criteria as used by the authors may be formulated as Equation (12).

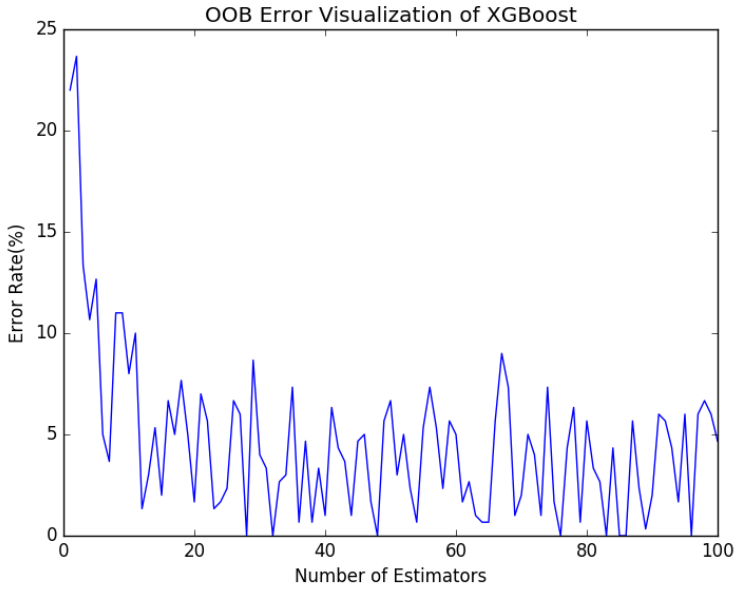



**Figure 9:** OOB error rate as the number of trees increases (Confidence Split)

$$var = \prod_{j=1}^{N} min\{1 - B(k_{ij}; n_j, p_j)\}$$

$$\mathscr{I} = \begin{cases} 0, & \text{if } var < \text{confidence threshold} \\ var, & \text{otherwise} \end{cases} \tag{12}$$

subject to the conditions, $c \geq 1$, $p = [0,1]$, confidence threshold $= [0,1)$, $k_{ij} \leq n_j$, $i = \{1,2,...,c\}$, $j = \{1,2,...,N\}$. Here, the $i$ subscript represents the class of data, and the $j$ subscript represents the output branch. So, $k_{ij}$ represents the number of expected entities of class $i$ in the child node $j$.

From the OOB error plot (Figure 9), it is observed that the classification error decreases as the number of trees increases. This is akin to the OOB plot of random forests using Gini split (Figure 8), which validates our *confidence-based* approach as a splitting criteria. For the current data set, the results obtained by using this criteria is comparable to the results obtained by using Gini impurity splitting criteria (the results are analyzed in Section 3.5.3). In the current data set, balanced data sets with 39 entities equally distributed among three

classes were used. A closer look at this method, however, reveals that it could be a difficult function to deal with as the number of samples in the data sets go on increasing, as it is in the multiplicative form. Nonetheless, it is a method worth exploring and can be considered a good method for small data sets. Hence, this method is of interest for the PHL-EC dataset. This is observed later, in the case of Proxima b (Table 23). Even otherwise, the results presented in Tables 19 and 20 indicate a comparable performance to the other tree-based classification algorithms. In the future, further work on this this method may enable it to scale up and work on large data sets.

### 3.7.1 Margins and convergence in random forests

The *margin* in a random forest measures the extent to which the average number of votes for $\mathbf{X}, Y$ for the right class exceeds the average vote for any other class. A larger margin thus implies a greater accuracy in classification [Breiman2001].

The generalization error in random forests converges almost surely as the number of trees increases. A convergence in the generalization error is important. It shows that the increase in the number of tree classifiers we tends to move the accuracy of classification towards near perfect (refer to Section **??** of Appendix **??**).

### 3.7.2 Upper bound of error and Chebyshev inequality

Accuracy is an important measure for any classification or approximation function. It is indeed an important question to be asked: *what is the error incurred by a certain classifier?* In the case of a classifier, the lower the error, the greater the probability of correct classification. It is critical that the upper bound of error be at least finite. *Chebyshev Inequality* can be related to the error bound of the random forest learner. The generalization error is bounded above by the inequality as defined by Equation 13 (refer to Section **??** of Appendix **??**).

$$Error \leq \frac{var(margin_{RF}(x,y))}{s^2} \tag{13}$$

### 3.7.3 Gradient tree boosting and XGBoosted trees

Boosting refers to the method of combining the results from a set of *weak learners* to produce a *strong* prediction. Generally, a weak learner's performance is only slightly better than that of a random guess. The idea is to divide the job of a single predictor across many weak predictor functions and to optimally combine the votes from all the smaller predictors. This helps enhance the overall prediction accuracy.

XGBoost [Chen & Guestrin2016] is a tool developed by utilizing these boosting principles. The word XGBoost stands for *eXtreme Gradient Boosting* as coined by the authors. XGBoost combines a large number of regression trees with a small learning rate. Subsequent trees in the forest of XGBoosted trees are grown by minimizing an objective function. Here, the word *regression* may refer to logistic or soft-max regression for the task of classification, albeit these trees may be used to solve linear regression problems as well. The boosting method used in XGBoost considers trees added early to be significant and trees added later to be inconsequential (refer to Section **??**).

XGBoosted trees [Chen & Guestrin2016] may be understood by considering four central concepts.

*7.15.1*: *Additive Learning*

For additive learning, functions $f_i$ must be learned which contain the tree structure and leaf scores [Chen & Guestrin2016]. This is more difficult compared to traditional optimization problems as there are multiple functions to be considered, and it is not sufficient to optimize every tree by considering its gradient. Another overhead is with respect to implementation in a computer: it is difficult to train all the trees all at once. Thus, the training phase is divided into a sequence of steps. For $t$ steps, the prediction value from each step, $\hat{y}_i^{(t)}$ are added as:

$$\begin{aligned}
\hat{y}_i^{(0)} &= 0 \\
\hat{y}_i^{(1)} &= f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i) \\
\hat{y}_i^{(2)} &= f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i) \\
&\cdots \\
\hat{y}_i^{(t)} &= \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)
\end{aligned} \tag{14}$$

Central to any optimization method is an objective function which needs to be optimized. In each step. the selected tree is the one that optimizes the objective function of the learning algorithm. The objective function is formulated as:

$$\begin{aligned}
\text{obj}^{(t)} &= \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^{t} \Omega(f_i) \\
&= \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + constant
\end{aligned} \tag{15}$$

Mean squared error (MSE) is used as its mathematical formulation is convenient. Logistic loss, for example, has a more complicated form. Since error needs to be minimized, the gradient of the error must be calculated: for MSE, calculating the gradient and finding a minima is not difficult, but in the case of logistic loss, the process becomes more cumbersome. In the general case, the Taylor expansion of the loss function is considered up to the term of second order.

$$\text{obj}^{(t)} = \sum_{i=1}^{n} [l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) + constant \tag{16}$$

where $g_i$ and $h_i$ are defined as:

$$\begin{aligned} g_i &= \partial_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)}) \\ h_i &= \partial^2_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)}) \end{aligned} \tag{17}$$

By removing the lower order terms from Equation (16), the objective function becomes:

$$\sum_{i=1}^{n} [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) \tag{18}$$

which is the optimization equation of XGBoost.

### *7.15.2*: *Model Complexity and Regularized Learning Objective*

The definition of the tree $f(x)$ may be refined as:

$$f_t(x) = w_{q(x)}, w \in R^T, q : R^d \rightarrow \{1, 2, \cdots, T\}. \tag{19}$$

where $w$ is the vector of scores on leaves, $q$ is a function assigning each data point to the corresponding leaf and $T$ is the number of leaves. In XGBoost, the model complexity may be given as:

$$\Omega(f) = \gamma T + \frac{1}{2} \lambda \sum_{j=1}^{T} w_j^2 \tag{20}$$

A regularized objective is minimized for the algorithm to learn the set of functions given in the model. It is given by:

$$\mathscr{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k) \tag{21}$$

*7.15.3*: *Structure Score*

After the objective value has been re-formalized, the objective value of the $t^{th}$ tree may be calculated as:

$$\begin{aligned} Obj^{(t)} &\approx \sum_{i=1}^{n} [g_i w_{q(x_i)} + \frac{1}{2} h_i w^2_{q(x_i)}] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \\ &= \sum_{j=1}^{T} [(\sum_{i \in I_j} g_i) w_j + \frac{1}{2} (\sum_{i \in I_j} h_i + \lambda) w_j^2] + \gamma T \end{aligned} \tag{22}$$

where $I_j = \{i | q(x_i) = j\}$ is the set of indices of data points assigned to the $j^{th}$ leaf. In the second line of the Equation (22), the index of the summation has been changed because all the data points in the same leaf must have the same score. Let $G_j = \sum_{i \in I_j} g_i$ and $H_j = \sum_{i \in I_j} h_i$. The equation can hence be further by substituting for $G$ and $H$ as:

$$\text{obj}^{(t)} = \sum_{j=1}^{T} [G_j w_j + \frac{1}{2}(H_j + \lambda) w_j^2] + \gamma T \tag{23}$$

In Equation (23), every $w_j$ is independent of each other. The form $G_j w_j + \frac{1}{2}(H_j + \lambda) w_j^2$ is quadratic and the best $w_j$ for a given structure $q(x)$ and the best objective reduction which measures goodness of tree is:

$$w_j^* = -\frac{G_j}{H_j + \lambda} \text{obj}^* = -\frac{1}{2} \sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda} + \gamma T \tag{24}$$

*7.15.4*: *Learning Structure Score*

XGBoost learns tree the tree level during learning based on the equation:

$$Gain = \frac{1}{2} \left[ \frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} \right] - \gamma \tag{25}$$

The equation comprises of four main parts:

- The score on the new left leaf

- The score on the new right leaf
- The score on the original leaf
- Regularization on the additional leaf

The value of $Gain$ should as high as possible for learning to take place effectively. Hence, if the value of gain is greater than $\gamma$, the corresponding branch should not be added.

A working principle of XGBoost in the context of the problem is illustrated using Figure **??**, Figure **??** and Table **??** of Appendix **??**.

### 3.7.4 Classification of conservative and optimistic samples of potentially habitable planets

The end objective of any machine learning pursuit is to be able to correctly analyze data as it increases with time. In the case of classifying exoplanets, the number of exoplanets in the catalog increase with time. In February 2015, the PHL-EC had about 1800 samples, whereas in January 2017, it has more than 3500 samples! This is twice the number of exoplanets as the time the authors started the current work.

The project home page of the Exoplanets Catalog of PHL (http://phl.upr.edu/projects/habitable-exoplanets-catalog) provides two lists of potentially habitable planets: the *conservative* list and the *optimistic* list. The conservative list contains those exoplanets that are more likely to have a rocky composition and maintain surface liquid water i.e. planets with 0.5 < Planet Radius ≤ 1.5 Earth radii or 0.1 < Planet Minimum Mass ≤ 5 Earth masses, and the planet is orbiting within the conservative habitable zone. The optimistic list contains those exoplanets that are less likely to have a rocky composition or maintain surface liquid water i.e. planets with 1.5 < Planet Radius ≤ 2.5 Earth radii or 5 < Planet Minimum Mass ≤ 10 Earth masses, or the planet is orbiting within the optimistic habitable zone. The tree based classification algorithms were tested on the planets in both the conservative samples' list as well as the optimistic samples' list. Out of the planets listed, Kepler-186 f is a hypopsychroplanet and was not included in the test sample (refer to Section 3.6.1). The experiment was conducted on the listed planets (except Kepler-186 f). The samples were individually isolated from the data set and were treated as the test set. The remainder of the data set was treated as the training set. The test results are presented in tables 15, 16, 21, 22, 17, 18, 19 and 20.

## 3.8 Habitability Classification System applied to Proxima b

On 24th August 2016, [Anglada-EscudÃl'2016] published the discovery of an apparently rocky planet( or believed to be one) orbiting Proxima Centauri, the nearest star to the Sun, named *Proxima b*. The discovery was made by Guillem Anglada-Escude, an astronomer at Queen Mary University of London along with a team. Proxima b is 1.3 times heavier than Earth. According to the PHL-EC, its radius is 1.12 EU, density is 0.9 EU, surface temperature is 262.1 K and escape velocity is 1.06 EU. These attributes are close to those of Earth. Hence, there are plausible reasons to believe that Proxima b may be a habitable planet (refer to row #3389 in the data set, *Confirmed Exoplanets: phl_hec_all_confirmed.csv* hosted at http://phl.upr.edu/projects/habitable-exoplanets-catalog/data/database).

In the PHL-EC data set, Proxima b is classified as a psychroplanet. The classification models which earlier resulted in high classification accuracy were used to classify Proxima b separately. The results are enunciated in Table 23. The results provide evidence of the strength of the system to automatically label and classify newly discovered exoplanets.

## 3.9 Data Synthesis and Artificial Augmentation

As mentioned earlier, the focus of the manuscript is to track the performance of the classifiers to scale. The reliability of the semi-automatic process depends on the efficacy of the classifiers if the data set grew rapidly with many entities. Since the required data is not naturally available, the authors have simulated a data generation process, albeit briefly, and performed classification experiments on the artificially generated data. The strategy has a two-fold objective: to devise a preemptive measure to check scalability of the classifiers, and to tackle classes of exoplanets with insufficient data. We elaborate the concept, theory, and model in this section and establish the equivalence of both premises.

Different kinds of simulations are seen in astrophysics. [Sale2015] modeled the extinction of stars by placing them into spatial bins and applying the Poisson point process to estimate the posterior probability of various 3D extinction maps; in this specific example, the assertion is that photometric catalogs are subject to bright and faint magnitude limits based on the instruments used for observations. This work, however, is more focused on the method of Poisson point processes instead of the specific application of it. [Sale2015] mentions that one of the assumptions is that the number of objects in a region of space (a statistical bin) follows a Poisson distribution. However, there are no quantitative metrics provided in support of this claim, such as a goodness-of-fit test, etc. But as the purpose of this work is to estimate point extinctions in a catalog that *largely* conforms to the actual physical extinctions, the

assumption that the data conforms to a Poisson distribution might be a reasonable one. [Green et al.2015] used a Markov Chain Monte Carlo (MCMC) method to create a dust map along *sightlines* (bins or discrete columns). In this work, they have assumed a Gaussian prior probability to model the dust distribution in every column. The idea of dividing the field of observations into bins is common in [Sale2015] and [Green et al.2015], however, the fundamental difference is that the posterior probability in [Sale2015] is modeled using an assumed distribution, whereas the prior probability in [Green et al.2015] is taken as Gaussian, possibly allowing the nature of the analysis to be more empirical. Thus, two methods of synthetic oversampling are explored:

1. By assuming a Poisson distribution in the data.
2. By estimating an empirical distribution from the data.

The strengths and weaknesses of each of these methods are mentioned in their respective subsections, albeit the authors would insist on the usage of empirical distribution estimation over the assumption of a distribution. Nonetheless, the first method paved way to the next, more robust method.

26 planets (data samples) belonged to the mesoplanet class and 16 samples belonged to the psychroplanet class, as of the day the analysis was done; these samples have been used for the classification experiments described in Sections 3.5 and 3.6. The naturally occurring data points are relatively less in order to describe the distribution of data by a known distribution (such as Poisson, or Gaussian). If a known distribution is estimated using this data, chances are that the distribution thus determined is not representative of the actual density of the data. As this fact is almost impossible to establish at this point in time, two separate methods of synthesizing data have been developed and implemented to gauge the efficacy of ML algorithms.

### 3.9.1 Generating Data by Assuming a Distribution

### 3.9.2 Artificially Augmenting Data in a Bounded Manner

The challenge with artificially oversampling data in PHL-EC is that the original data available is too less to estimate a reliable probability distribution which is satisfactorily representative of the probability density of the naturally occurring data. For this, a *bounding mechanism* should be used so that while augmenting the data set artificially, the values of each feature or observable does not exceed the physical limits of the respective observable, and the physical limits are analyzed from the naturally occurring data.

**Table 15:** Results of using decision trees (Gini impurity) to classify the planets in the conservative sample

| Name | True Class | Classification Accuracy | non-habitable | psychroplanet | mesoplane |
|---|---|---|---|---|---|
| Proxima Cen b | psychroplanet | 84.5 | 0.1 | 84.5 | 15.4 |
| GJ 667 C c | mesoplanet | 91.7 | 0.0 | 8.3 | 91.7 |
| Kepler-442 b | psychroplanet | 56.9 | 0.1 | 56.9 | 43.0 |
| GJ 667 C f | psychroplanet | 100.0 | 0.0 | 100.0 | 0.0 |
| Wolf 1061 c | psychroplanet | 100.0 | 0.0 | 100.0 | 0.0 |
| Kepler-1229 b | psychroplanet | 100.0 | 0.0 | 100.0 | 0.0 |
| Kapteyn b | psychroplanet | 100.0 | 0.0 | 100.0 | 0.0 |
| Kepler-62 f | psychroplanet | 100.0 | 0.0 | 100.0 | 0.0 |
| GJ 667 C e | psychroplanet | 100.0 | 0.0 | 100.0 | 0.0 |

For this purpose, we use a hybrid of SVM and K-NN to set the limits for the observables. The steps in the SVM-KNN algorithm are summarized below:

**Step 1**: The best boundary between the psychroplanets and mesoplanets are found using SVM with a linear kernel.

**Step 2**: By analyzing the distribution of either class, data points are artificially created.

**Step 3**: Using the boundary determined in Step 1, an artificial data point is analyzed to determine if it satisfies the boundary conditions: if a data point generated for one class falls within the boundary of the respective class, the data point is kept in it's labeled class in the artificial data set.

**Step 4**: If a data point crosses the boundary of its respective class, then a K-NN based verification is applied. If 3 out of the nearest 5 neighbors belongs to the class to which the data point is supposed to belong, then the data point is kept in the artificially augmented data set.

**Step 5**: If the conditions in Steps 3 and 4 both fail, then the respective data point's class label is changed so that it belongs to the class whose properties it corresponds to better.

**Step 6**: Steps 3, 4 and 5 are repeated for all the artificial data points generated, in sequence.

It is important to note that in Section **??**, the K-NN and SVM algorithms have been explained as classification algorithms; however, they are not used as classifiers in this over-sampling simulation. Rather, they are used, along with density estimation, to rectify the class-belongingness (class labels) of artificially generated random samples. If an artificially

**Table 16:** Results of using decision trees (Gini impurity) to classify the planets in the optimistic sample

| Name | True Class | Classification Accuracy | non-habitable | psychroplanet | mesoplanet |
|---|---|---|---|---|---|
| Kepler-438 b | mesoplanet | 92.5 | 0.2 | 7.3 | 92.5 |
| Kepler-296 e | mesoplanet | 99.8 | 0.2 | 0.0 | 99.8 |
| Kepler-62 e | mesoplanet | 100.0 | 0.0 | 0.0 | 100.0 |
| Kepler-452 b | mesoplanet | 99.6 | 0.4 | 0.0 | 99.6 |
| K2-72 e | mesoplanet | 99.7 | 0.3 | 0.0 | 99.7 |
| GJ 832 c | mesoplanet | 99.0 | 0.0 | 1.0 | 99.0 |
| K2-3 d | non-habitable | 0.8 | 0.8 | 0.0 | 99.2 |
| Kepler-1544 b | mesoplanet | 99.9 | 0.1 | 0.0 | 99.9 |
| Kepler-283 c | mesoplanet | 100.0 | 0.0 | 0.0 | 100.0 |
| Kepler-1410 b | mesoplanet | 99.9 | 0.1 | 0.0 | 99.9 |
| GJ 180 c | mesoplanet | 79.7 | 0.0 | 20.3 | 79.7 |
| Kepler-1638 b | mesoplanet | 99.4 | 0.6 | 0.0 | 99.4 |
| Kepler-440 b | mesoplanet | 94.8 | 5.2 | 0.0 | 94.8 |
| GJ 180 b | mesoplanet | 99.7 | 0.3 | 0.0 | 99.7 |
| Kepler-705 b | mesoplanet | 100.0 | 0.0 | 0.0 | 100.0 |
| HD 40307 g | psychroplanet | 87.7 | 0.0 | 87.7 | 12.3 |
| GJ 163 c | psychroplanet | 100.0 | 0.0 | 100.0 | 0.0 |
| Kepler-61 b | mesoplanet | 96.9 | 3.1 | 0.0 | 96.9 |
| K2-18 b | mesoplanet | 100.0 | 0.0 | 0.0 | 100.0 |
| Kepler-1090 b | mesoplanet | 99.7 | 0.3 | 0.0 | 99.7 |
| Kepler-443 b | mesoplanet | 99.5 | 0.3 | 0.2 | 99.5 |
| Kepler-22 b | mesoplanet | 98.4 | 1.6 | 0.0 | 98.4 |
| GJ 422 b | mesoplanet | 17.1 | 0.9 | 82.0 | 17.1 |
| Kepler-1552 b | mesoplanet | 97.3 | 2.7 | 0.0 | 97.3 |
| GJ 3293 c | psychroplanet | 100.0 | 0.0 | 100.0 | 0.0 |
| Kepler-1540 b | mesoplanet | 98.5 | 1.5 | 0.0 | 98.5 |
| Kepler-298 d | mesoplanet | 95.6 | 4.4 | 0.0 | 95.6 |
| Kepler-174 d | psychroplanet | 99.9 | 0.1 | 99.9 | 0.0 |
| Kepler-296 f | psychroplanet | 100.0 | 0.0 | 100.0 | 0.0 |
| GJ 682 c | psychroplanet | 99.2 | 0.8 | 99.2 | 0.0 |
| tau Cet e | mesoplanet | 99.4 | 0.6 | 0.0 | 99.4 |

**Table 17:** Results of using random forests (Gini impurity) to classify the planets in the conservative sample

| Name | True Class | Classification Accuracy | non-habitable | psychroplanet | mesoplane |
|---|---|---|---|---|---|
| Proxima Cen b | psychroplanet | 100.0 | 0.0 | 100.0 | 0.0 |
| GJ 667 C c | mesoplanet | 100.0 | 0.0 | 0.0 | 100.0 |
| Kepler-442 b | psychroplanet | 94.1 | 0.0 | 94.1 | 5.9 |
| GJ 667 C f | psychroplanet | 100.0 | 0.0 | 100.0 | 0.0 |
| Wolf 1061 c | psychroplanet | 100.0 | 0.0 | 100.0 | 0.0 |
| Kepler-1229 b | psychroplanet | 100.0 | 0.0 | 100.0 | 0.0 |
| Kapteyn b | psychroplanet | 100.0 | 0.0 | 100.0 | 0.0 |
| Kepler-62 f | psychroplanet | 100.0 | 0.0 | 100.0 | 0.0 |
| GJ 667 C e | psychroplanet | 100.0 | 0.0 | 100.0 | 0.0 |

generated random sample is generated such that it does not conform to the general properties of the respective class (which can be either mesoplanets or psychroplanets), the class label of the respective sample is simply changed such that it may belong to the class of habitability whose properties it exhibits better. The strength of using this as a rectification mechanism lies in the fact that artificially generated points which are near the boundary of the classes stand a chance to be rectified so that they might belong to the class they better represent. Moreover, due to the density estimation, points can be generated over an entire region of the feature space, rather than augmenting based on individual samples. This aspect of the simulation is the cornerstone of the novelty of this approach: in comparison to existing approaches as SMOTE (Synthetic Minority Oversampling Technique) [Chawla et al.2002], the oversampling does not depend on individual samples in the data. In simple terms, SMOTE augments data by geometrically inserting samples between existing samples; this is suitable for experiments for which there is already exist an appreciable amount of data in a data set, but reiterating, as PHL-EC has less data already (for the classes of mesoplanets and psychroplanets), an oversampling based on individual samples is not a good way to proceed. Here, it is best to estimate the probability density of the data and proceed with the oversampling in a bounded manner. For large-scale simulation tasks similar in nature to this, thus, ML-based approaches can go a long way to save time and automate the process of discovery of knowledge.

### 3.9.3 Fitting a Distribution to the Data Points

In this method, the mean surface temperature was selected as the core discriminating feature since it emerged as the most important feature amongst the classes in the catalog (Tables 13 and 14). The mean surface temperature for different classes of planets falls in different

**Table 18:** Results of using random forests (Gini impurity) to classify the planets in the optimistic sample

| Name | True Class | Classification Accuracy | non-habitable | psychroplanet | mesoplanet |
|---|---|---|---|---|---|
| Kepler-438 b | mesoplanet | 100.0 | 0.0 | 0.0 | 100.0 |
| Kepler-296 e | mesoplanet | 100.0 | 0.0 | 0.0 | 100.0 |
| Kepler-62 e | mesoplanet | 100.0 | 0.0 | 0.0 | 100.0 |
| Kepler-452 b | mesoplanet | 99.9 | 0.1 | 0.0 | 99.9 |
| K2-72 e | mesoplanet | 100.0 | 0.0 | 0.0 | 100.0 |
| GJ 832 c | mesoplanet | 100.0 | 0.0 | 0.0 | 100.0 |
| K2-3 d | non-habitable | 0.0 | 0.0 | 0.0 | 100.0 |
| Kepler-1544 b | mesoplanet | 100.0 | 0.0 | 0.0 | 100.0 |
| Kepler-283 c | mesoplanet | 100.0 | 0.0 | 0.0 | 100.0 |
| Kepler-1410 b | mesoplanet | 100.0 | 0.0 | 0.0 | 100.0 |
| GJ 180 c | mesoplanet | 96.2 | 0.0 | 3.8 | 96.2 |
| Kepler-1638 b | mesoplanet | 100.0 | 0.0 | 0.0 | 100.0 |
| Kepler-440 b | mesoplanet | 100.0 | 0.0 | 0.0 | 100.0 |
| GJ 180 b | mesoplanet | 100.0 | 0.0 | 0.0 | 100.0 |
| Kepler-705 b | mesoplanet | 100.0 | 0.0 | 0.0 | 100.0 |
| HD 40307 g | psychroplanet | 100.0 | 0.0 | 100.0 | 0.0 |
| GJ 163 c | psychroplanet | 100.0 | 0.0 | 100.0 | 0.0 |
| Kepler-61 b | mesoplanet | 100.0 | 0.0 | 0.0 | 100.0 |
| K2-18 b | mesoplanet | 100.0 | 0.0 | 0.0 | 100.0 |
| Kepler-1090 b | mesoplanet | 100.0 | 0.0 | 0.0 | 100.0 |
| Kepler-443 b | mesoplanet | 100.0 | 0.0 | 0.0 | 100.0 |
| Kepler-22 b | mesoplanet | 100.0 | 0.0 | 0.0 | 100.0 |
| GJ 422 b | mesoplanet | 0.0 | 0.0 | 100.0 | 0.0 |
| Kepler-1552 b | mesoplanet | 99.9 | 0.1 | 0.0 | 99.9 |
| GJ 3293 c | psychroplanet | 100.0 | 0.0 | 100.0 | 0.0 |
| Kepler-1540 b | mesoplanet | 100.0 | 0.0 | 0.0 | 100.0 |
| Kepler-298 d | mesoplanet | 100.0 | 0.0 | 0.0 | 100.0 |
| Kepler-174 d | psychroplanet | 100.0 | 0.0 | 100.0 | 0.0 |
| Kepler-296 f | psychroplanet | 100.0 | 0.0 | 100.0 | 0.0 |
| GJ 682 c | psychroplanet | 100.0 | 0.0 | 100.0 | 0.0 |
| tau Cet e | mesoplanet | 100.0 | 0.0 | 0.0 | 100.0 |

**Table 19:** Results of using random forests (binomial confidence split) to classify the planets in the conservative sample

| Name | True Class | Classification Accuracy | non-habitable | psychroplanet | mesoplane |
|---|---|---|---|---|---|
| Proxima Cen b | psychroplanet | 99.8 | 0.0 | 99.8 | 0.2 |
| GJ 667 C c | mesoplanet | 88.1 | 0.0 | 11.9 | 88.1 |
| Kepler-442 b | psychroplanet | 98.5 | 0.0 | 98.5 | 1.5 |
| GJ 667 C f | psychroplanet | 100.0 | 0.0 | 100.0 | 0.0 |
| Wolf 1061 c | psychroplanet | 100.0 | 0.0 | 100.0 | 0.0 |
| Kepler-1229 b | psychroplanet | 100.0 | 0.0 | 100.0 | 0.0 |
| Kapteyn b | psychroplanet | 100.0 | 0.0 | 100.0 | 0.0 |
| Kepler-62 f | psychroplanet | 100.0 | 0.0 | 100.0 | 0.0 |
| GJ 667 C e | psychroplanet | 100.0 | 0.0 | 100.0 | 0.0 |

ranges [Méndez2011]. The mean surface temperature was fit to a Poisson distribution; the vector of remaining features was randomly mapped to these randomly generated values of S. Temp. The resulting vectors of artificial samples may be considered to be a vector $S = (Temp_{Surface}, X)$, where $X$ is any naturally occurring sample in the PHL-EC data set without its corresponding value of the S. Temp feature. The set of the pairs $(S, c)$ thus becomes an entire artificial catalog, where $c$ is the class label. The following are the steps to generate artificial data set for the mesoplanet class:

**Step 1**: For the original set of values pertinent to the mean surface temperature of mesoplanets, a Poisson distribution is fit. The surface temperature of the planets assumed to be randomly distributed, following a Poisson distribution. Here, an approximation may be made that the surface temperatures occur in discrete bins or intervals, without a loss of generality. As the number of samples is naturally less, a Poisson distribution may be fit to the S. Temp features after rounding off the values to the nearest decimal.

**Step 2**: Then, using the average value of the mesoplanets' S. Temp data, 1000 new values are generated, using the Poisson distribution:

$$Pr(X) = \frac{e^{-\lambda}\lambda^{x}}{x!} \tag{26}$$

where $\lambda$ is the mean of the values of the S. Temp feature of the mesoplanet class.

**Step 3**: For every planet in the original data set, duplicate the data sample 40 times and replace the surface temperature value of these (total of 1000 samples) with new values of the mean surface temperature randomly, as generated in Step 2.

**Table 20:** Results of using random forests (binomial confidence split) to classify the planets in the optimistic sample

| Name | True Class | Classification Accuracy | non-habitable | psychroplanet | mesoplanet |
|---|---|---|---|---|---|
| Kepler-438 b | mesoplanet | 100.0 | 0.0 | 0.0 | 100.0 |
| Kepler-296 e | mesoplanet | 100.0 | 0.0 | 0.0 | 100.0 |
| Kepler-62 e | mesoplanet | 100.0 | 0.0 | 0.0 | 100.0 |
| Kepler-452 b | mesoplanet | 100.0 | 0.0 | 0.0 | 100.0 |
| K2-72 e | mesoplanet | 100.0 | 0.0 | 0.0 | 100.0 |
| GJ 832 c | mesoplanet | 99.9 | 0.0 | 0.1 | 99.9 |
| K2-3 d | non-habitable | 0.1 | 0.1 | 0.0 | 99.9 |
| Kepler-1544 b | mesoplanet | 99.7 | 0.0 | 0.3 | 99.7 |
| Kepler-283 c | mesoplanet | 99.8 | 0.0 | 0.2 | 99.8 |
| Kepler-1410 b | mesoplanet | 100.0 | 0.0 | 0.0 | 100.0 |
| GJ 180 c | mesoplanet | 65.10 | 0.0 | 34.9 | 65.1 |
| Kepler-1638 b | mesoplanet | 99.1 | 0.9 | 0.0 | 99.1 |
| Kepler-440 b | mesoplanet | 100.0 | 0.0 | 0.0 | 100.0 |
| GJ 180 b | mesoplanet | 100.0 | 0.0 | 0.0 | 100.0 |
| Kepler-705 b | mesoplanet | 98.6 | 0.0 | 1.4 | 98.6 |
| HD 40307 g | psychroplanet | 96.39 | 0.0 | 96.4 | 3.6 |
| GJ 163 c | psychroplanet | 100.0 | 0.0 | 100.0 | 0.0 |
| Kepler-61 b | mesoplanet | 100.0 | 0.0 | 0.0 | 100.0 |
| K2-18 b | mesoplanet | 100.0 | 0.0 | 0.0 | 100.0 |
| Kepler-1090 b | mesoplanet | 99.9 | 0.1 | 0.0 | 99.9 |
| Kepler-443 b | mesoplanet | 99.9 | 0.0 | 0.1 | 99.9 |
| Kepler-22 b | mesoplanet | 100.0 | 0.0 | 0.0 | 100.0 |
| GJ 422 b | mesoplanet | 0.0 | 0.0 | 100.0 | 0.0 |
| Kepler-1552 b | mesoplanet | 99.8 | 0.2 | 0.0 | 99.8 |
| GJ 3293 c | psychroplanet | 100.0 | 0.0 | 100.0 | 0.0 |
| Kepler-1540 b | mesoplanet | 100.0 | 0.0 | 0.0 | 100.0 |
| Kepler-298 d | mesoplanet | 99.7 | 0.3 | 0.0 | 99.7 |
| Kepler-174 d | psychroplanet | 100.0 | 0.0 | 100.0 | 0.0 |
| Kepler-296 f | psychroplanet | 100.0 | 0.0 | 100.0 | 0.0 |
| GJ 682 c | psychroplanet | 100.0 | 0.0 | 100.0 | 0.0 |
| tau Cet e | mesoplanet | 99.9 | 0.1 | 0.0 | 99.9 |

**Table 21:** Results of using XGBoost to classify the planets in the conservative sample

| Name | True Class | Classification Accuracy | non-habitable | psychroplanet | mesoplanet |
|---|---|---|---|---|---|
| Proxima Cen b | psychroplanet | 100.0 | 0.0 | 100.0 | 0.0 |
| GJ 667 C c | mesoplanet | 100.0 | 0.0 | 0.0 | 100.0 |
| Kepler-442 b | psychroplanet | 6.8 | 0.2 | 6.8 | 93.0 |
| GJ 667 C f | psychroplanet | 100.0 | 0.0 | 100.0 | 0.0 |
| Wolf 1061 c | psychroplanet | 100.0 | 0.0 | 100.0 | 0.0 |
| Kepler-1229 b | psychroplanet | 100.0 | 0.0 | 100.0 | 0.0 |
| Kapteyn b | psychroplanet | 100.0 | 0.0 | 100.0 | 0.0 |
| Kepler-62 f | psychroplanet | 100.0 | 0.0 | 100.0 | 0.0 |
| GJ 667 C e | psychroplanet | 100.0 | 0.0 | 100.0 | 0.0 |

This exercise is repeated for the psychroplanet class separately. Once the probability densities of both the classes were developed, the rectification mechanism using the algorithm described in Section 3.9.2 was used to retain only those samples in either class which conformed to the properties of the respective class. Using this method, 1000 artificial samples were generated for the mesoplanet and psychroplanet classes.

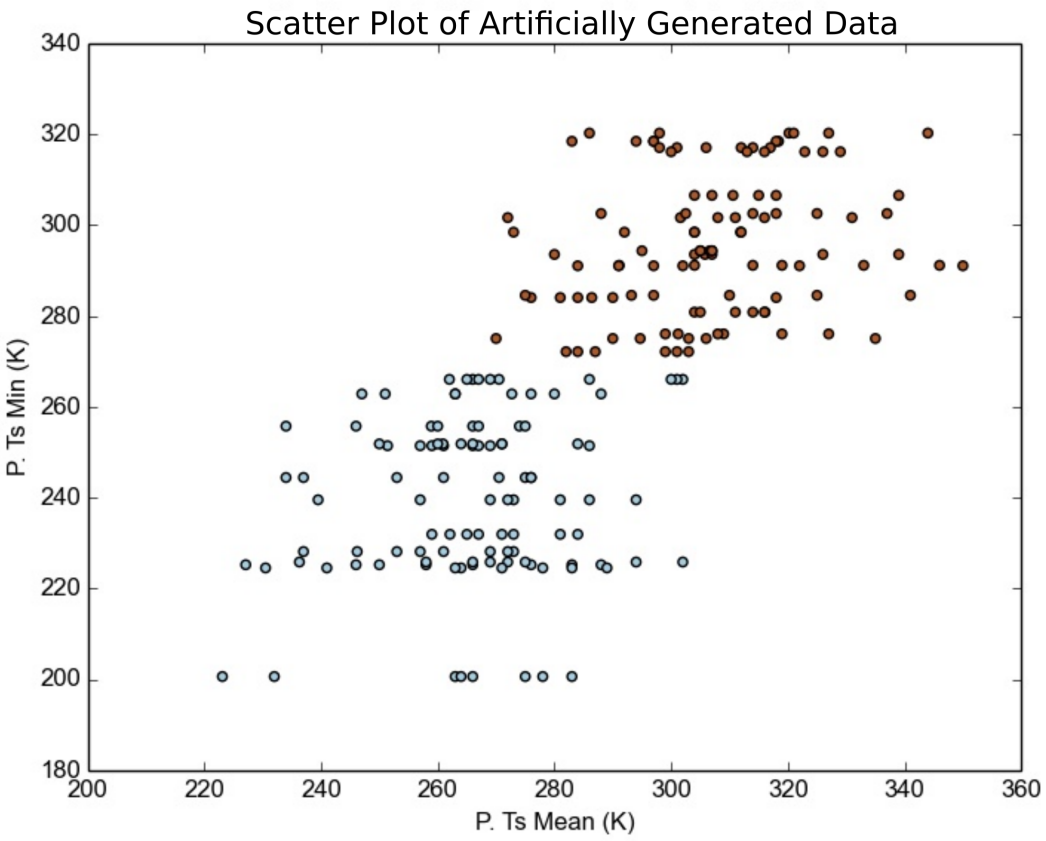
In order to generate 1000 samples for the classes with less number of samples (mesoplanets and psychroplanets), the hybrid SVM-KNN algorithm as described in Section 3.9.2 is used to rectify the class-belongingness of any non-conforming random samples. Only the top four features of the data sets from Table 13, i.e. P. Ts Mean, P. Ts Min, P. Teq Min, and P. Teq Max are considered in this rectification mechanism. This method of acceptance-rectification is self-contained in itself: the artificially generated data set is iteratively split into training and testing sets (in a ratio of 70:30). If any artificially generated sample in any iteration fails to be accepted by the SVM-KNN algorithm, its class-belongingness in the data set is changed; as this simulation is done only on two classes in the data, non-conformance to one class could only indicate the belongingness to the other class. The process of artificially generating and labeling data is illustrated using Figure 10. In Figure 10(a), a new set of data points generated randomly from the estimated Poisson distributions of both classes are plotted. The points in red depict artificial points belonging to the class of psychroplanets and the points in blue depict artificial points belonging to the class of mesoplanets. The physical limits as per the Planetary Habitability Catalog are incorporated into the data synthesis scheme and hence, in general, the number of non-conforming points generated are less. Figure 10(b) depicts three points (encircled) that should belong to the psychroplanet class but belongs to the mesoplanet class: note that these three points cross the boundary between the two classes as set by an SVM. The blue portion may contain points which belong to only

**Table 22:** Results of using XGBoost to classify the planets in the optimistic sample

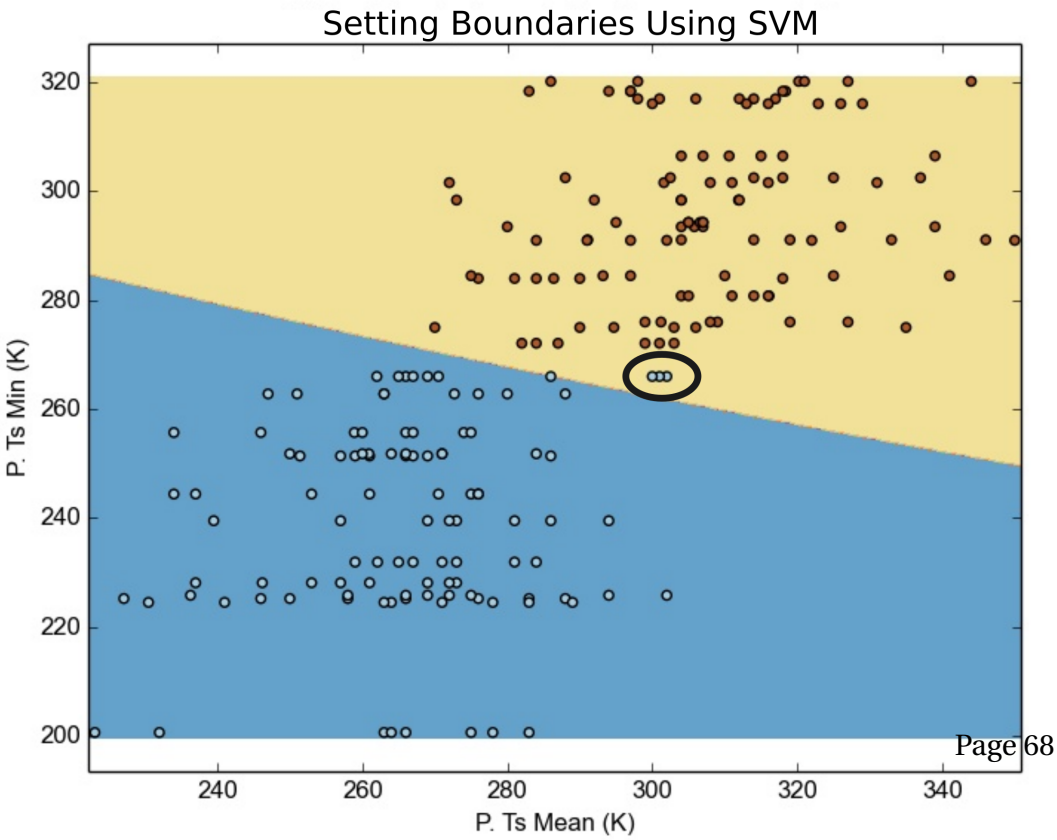
| Name | True Class | Classification Accuracy | non-habitable | psychroplanet | mesoplanet |
|---|---|---|---|---|---|
| Kepler-438 b | mesoplanet | 99.9 | 0.1 | 0 | 99.9 |
| Kepler-296 e | mesoplanet | 100 | 0 | 0 | 100 |
| Kepler-62 e | mesoplanet | 100 | 0 | 0 | 100 |
| Kepler-452 b | mesoplanet | 100 | 0 | 0 | 100 |
| K2-72 e | mesoplanet | 99.2 | 0.8 | 0 | 99.2 |
| GJ 832 c | mesoplanet | 98.1 | 0 | 1.9 | 98.1 |
| K2-3 d | non-habitable | 1.2 | 1.2 | 0 | 98.8 |
| Kepler-1544 b | mesoplanet | 100 | 0 | 0 | 100 |
| Kepler-283 c | mesoplanet | 100 | 0 | 0 | 100 |
| Kepler-1410 b | mesoplanet | 99.6 | 0.4 | 0 | 99.6 |
| GJ 180 c | mesoplanet | 74 | 0 | 26 | 74 |
| Kepler-1638 b | mesoplanet | 99.2 | 0.8 | 0 | 99.2 |
| Kepler-440 b | mesoplanet | 97.9 | 2.1 | 0 | 97.9 |
| GJ 180 b | mesoplanet | 100 | 0 | 0 | 100 |
| Kepler-705 b | mesoplanet | 100 | 0 | 0 | 100 |
| HD 40307 g | psychroplanet | 99 | 0 | 99 | 1 |
| GJ 163 c | psychroplanet | 100 | 0 | 100 | 0 |
| Kepler-61 b | mesoplanet | 99 | 1 | 0 | 99 |
| K2-18 b | mesoplanet | 100 | 0 | 0 | 100 |
| Kepler-1090 b | mesoplanet | 100 | 0 | 0 | 100 |
| Kepler-443 b | mesoplanet | 99.9 | 0 | 0.1 | 99.9 |
| Kepler-22 b | mesoplanet | 99.9 | 0.1 | 0 | 99.9 |
| GJ 422 b | mesoplanet | 57.2 | 1.3 | 41.5 | 57.2 |
| Kepler-1552 b | mesoplanet | 99.9 | 0.1 | 0 | 99.9 |
| GJ 3293 c | psychroplanet | 100 | 0 | 100 | 0 |
| Kepler-1540 b | mesoplanet | 99.7 | 0.3 | 0 | 99.7 |
| Kepler-298 d | mesoplanet | 99 | 1 | 0 | 99 |
| Kepler-174 d | psychroplanet | 100 | 0 | 100 | 0 |
| Kepler-296 f | psychroplanet | 100 | 0 | 100 | 0 |
| GJ 682 c | psychroplanet | 99.7 | 0.3 | 99.7 | 0 |
| tau Cet e | mesoplanet | 99.9 | 0.1 | 0 | 99.9 |

**Table 23:** Accuracy of algorithms used to classify Proxima b

| Algorithm | Accuracy(%) |
|---|---|
| Decision Tree | 84.5 |
| Random Forest (Gini Split) | 100.0 |
| Random Forest (Conf. Split) | 100.0 |
| XGBoost | 100.0 |




**(a)** Scatter plot of newly generated artificial data points in two dimensions.






**(b)** Best boundaries between two classes set using SVM. Here, there are three non-conforming data points (encircled) belonging to the mesoplanets' class.

the mesoplanet class and the yellow portion may contain points which belong only to the psychroplanet class, but these three points are non-conforming according to the boundary imposed. Hence, in order to ascertain the correct labels, these three points are subjected to a K-NN based rectification. In Figure 10(c), the points in the data set are plotted after being subjected to K-NN with $k = 5$ and class labels are modified as required. The three previously non-conforming points are determined to actually belong to the class of psychroplanets, and hence their class-belongingness is changed. Figure 10(d) shows that the boundary between the two classes is altered by incorporating the rectified class-belongingness of the previously non-conforming points. In this figure, it is to be noted that all the points are conforming, and there are no points which belong to the region of the wrong class. This procedure was run many times on the artificially generated data to estimate the number of iterations and the time required for each iteration until the resulting data set was devoid of any non-conforming data points. As the process is inherently stochastic, each new run of the SVM-KNN algorithm might result in a different number of iterations (and different amounts of execution time for each iteration) required until zero non-conforming samples are achieved. However, a general trend may be analyzed for the purpose of ascertaining that the algorithm will complete in a finite amount of time. Figure 11 is a plot of the $i^{th}$ iteration against the time required for the algorithm to execute the respective iteration (to rectify the points in the synthesized data set). From this figure, it should be noted that each successive iteration requires a smaller amount of time to complete: the red curve (a quadratic fit of the points) represents a decline in the time required for the SVM-KNN method to complete execution in successive iterations of a run. The number of iterations required for the complete execution of the SVM-KNN method ranges from one to six, with a generally declining execution time of successive iterations, proving the stability of the hybrid algorithm. Any algorithm is required to *converge*: a point beyond which the execution of the algorithm ceases. In this case, convergence must ensure that every artificially generated data point conforms to the general properties of the class to which it is labeled to belong.

The advantage of this method is that there is enough evidence of work done previously that makes use of standard probability distributions to model the occurrence of various stellar objects. This current simulation only has the added dimension of class-label rectification by using the hybrid SVM-KNN method. The method is easy to interpret. However, as the amount of data in the PHL-EC catalog are less, fitting a distribution to the data may lead to an over-fitting of the probability density estimate. To counter this point, an empirical multivariate distribution estimation was performed as a follow-through to this piece of work.

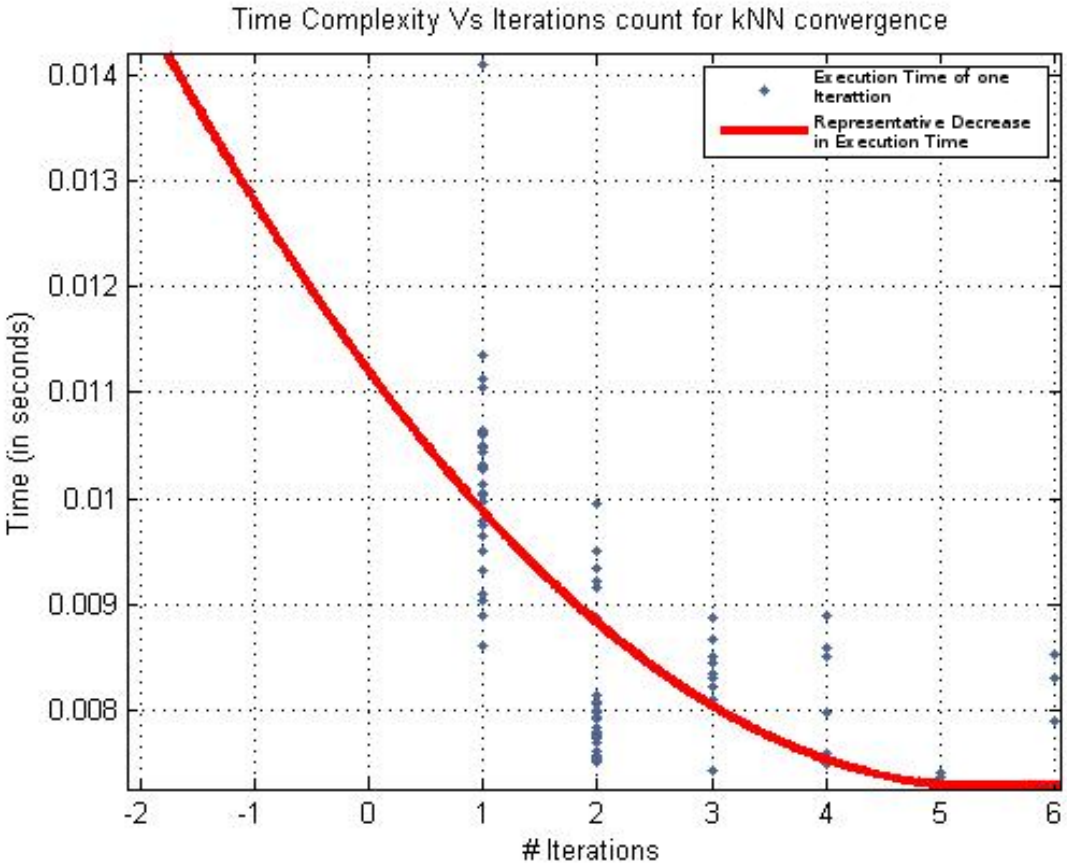



**Figure 11:** A quadratic curve has been fit to the execution times of successive iterations in a run of the SVM-KNN method. The time required to converge to the perfect labeling of class-belongingness of the synthetic data points reduces with each successive iteration resulting in the dip exhibited in successive iterations. This fortifies the efficiency of the proposed hybrid SVM-KNN algorithm. Accuracy is not traded with the speed of convergence.

### 3.9.4 Generating Data by Analyzing the Distribution of Existing Data Empirically: Window Estimation Approach

In this method of synthesizing data samples, the density of the data distribution is approximated by a numeric mathematical model, instead of relying on an established analytical model (such as Poisson, or Gaussian distributions). As the sample distribution here is sporadic, the density function itself should be approximated. The process outlined for this estimation of the population density function was described independently by [Roesnblatt1956] and [Parzen1962] and is termed Kernel Density Estimation (KDE). KDE, as a non-parametric technique, requires no assumptions on the structure of the data and further, with slight alterations to the kernel function, may also be extended to multivariate random variables.

### 3.9.5 Estimating Density

Let $X = x_1, x_2, \ldots, x_n$ be a sequence of independent and identically distributed multivariate random variables having $d$ dimensions. The window function used is a variation of the uniform kernel defined on the set $R^d$ as follows:

$$\phi(u) = \begin{cases} 1 & u_j \leq \frac{1}{2} \quad \forall\, j \in \{1,2,\ldots,d\} \\ 0 & otherwise \end{cases} \tag{27}$$

Additionally, another parameter, the edge length vector $h = \{h_1, h_2, \ldots h_d\}$, is defined, where each component of $h$ is set on a heuristic that considers the values of the corresponding feature in the original data. If $f_j$ is the column vector representing some feature $j \in X$ and

$$\begin{aligned} l_j &= min\{(a-b)^2 \quad \forall\ a,b \in f_j\} \\ u_j &= max\{(a-b)^2 \quad \forall\ a,b \in f_j\}, \end{aligned} \tag{28}$$

the edge length $h_j$ is given by,

$$h_j = c\left(\frac{u_j + 2l_j}{3}\right) \tag{29}$$

where $c$ is a scale factor.

Let $x' \in R^d$ be a random variable at which the density needs to be estimated. For the estimate, another vector $u$ is generated whose elements are given by:

$$u_j = \frac{x_j{}' - x_{ij}}{h_j} \qquad \forall\, j \in \{1,2,\ldots,d\} \tag{30}$$

The density estimate is then given by the following equation:

$$p(x') = \frac{1}{n\prod_{i=1}^{d} h_i} \sum_{i=1}^{n} \phi(u) \tag{31}$$

### 3.9.6 Generating Synthetic Samples

Traditionally, random numbers are generated from an analytic density function by inversion sampling. However, this would not work on a numeric density function unless the quantile function is numerically approximated by the density function. In order to avoid this, a form of rejection sampling has been used.

Let $r$ be a $d$-dimensional random vector with each component drawn from a uniform distribution between the minimum and maximum value of that component in the original data. Once the density, $p(r)$ is estimated by Equation (31), the probability is approximated to:

$$Pr(r) = p(r) \prod_{j=1}^{d} h_j \tag{32}$$

To either accept or reject the sample $r$, another random number is generated from a uniform distribution within the range $[0,1)$. If this number is greater than the probability estimated by Equation (32), then the sample is accepted. Otherwise, it is rejected.

Data synthesis using KDE and rejection sampling (refer to Appendix **??** for visual details) was used to generate a synthetic data set. For the PHL-EC data set, synthetic data was generated for the mesoplanet and psychroplanet classes by estimating their density by Equation (31) taking $c = 4$ for mesoplanets and $c = 3$ for psychroplanets. 1000 samples were then generated for each class using rejection sampling on the density estimate. In this method, the bounding mechanism was not used and the samples were drawn out of the estimated density. Here, the top 16 features (top 85% of the features by importance, Table 13) were considered to estimate the probability density, and hence the boundary between the two classes using SVM was not constructed. The values of the remaining features were copied from the naturally occurring data points and shuffled between the artificially augmented data points in the same way as in the method described in Section 3.9.3). The advantage of using this method is that it may be used to estimate a distribution which resembles more closely the actual distribution of the data. However, this process is more complex and takes a longer time to execute. Nonetheless, the authors would assert this as a method of synthetic oversampling than the method described in Section 3.9.2 as it is inherently unassuming and can accommodate distributions in data which are otherwise difficult to describe using the

**Table 24:** Results on artificially augmented data sets by assuming a distribution and augmenting in a bounded manner.

| Algorithm | Class | Sensitivity | Specificity | Precision | Accuracy |
|---|---|---|---|---|---|
| Decision Trees | Non-Habitable | 0.9977333333 | 1 | 1 | 0.9992735043 |
| | Mesoplanet | 1 | 0.9994227809 | 0.9988474837 | 0.9996152531 |
| | Psychroplanet | 1 | 0.9994768164 | 0.9990133202 | 0.9996579881 |
| Random Forests | Non-Habitable | 0.9997333333 | 1 | 1 | 0.9999145299 |
| | Mesoplanet | 1 | 0.9998717949 | 0.9997436555 | 0.9999145299 |
| | Psychroplanet | 1 | 1 | 1 | 1 |
| XGBoost | Non-Habitable | 0.9989333333 | 1 | 1 | 0.9996581197 |
| | Mesoplanet | 1 | 1 | 1 | 1 |
| | Psychroplanet | 1 | 0.9994771242 | 0.9990133202 | 0.9996581197 |

commonly used methods for describing the density of data.

## 3.10 Results of Classification on Artificially Augmented Data Sets

The results of classification experimented on the data sets generated by the methods described in Sections 3.9.1 and 3.9.4 are shown in Tables 24 and 25 respectively.

In both methods, 1000 samples were then generated for mesoplanet and psychroplanet classes; in each iteration of testing the classifiers, 1000 samples were randomly drawn from the non-habitable class. The original mesoplanet and psychroplanet data, the synthetic samples, and the samples drawn from the non-habitable class together form an augmented data set. This data set was then subjected to the non-metric classifiers to test their performance. For each iteration applied to evaluating classification accuracy, the augmented data set was split into a training and test set by randomly sampling the records of each class into the two sets at a ratio of 7:3. The classifiers were trained and their accuracy of classification estimated through the test set. The training and test sets were then re-sampled for the next iteration. This was 100 times, following which, a new set of 1000 samples were drawn from the non-habitable class to replace the previous samples. The whole process of drawing non-habitable samples and subjecting the augmented data set to 100 iterations of testing has been repeated 20 times with the averaged results presented in Tables 24 and 25. The commonly available methods available in most open source toolkits have been tried to classify the artificially augmented samples. These results indicate that the classifiers are capable of handling large data sets without any signs of diminishing performance.

The whole exercise of artificially augmenting the data set may be considered to be nothing but a simulation of the natural increase of the data points in the catalog. The methods

**Table 25:** Results on artificially augmented data sets by empirical analysis.

| Algorithm | Class | Sensitivity | Specificity | Precision | Accuracy |
|---|---|---|---|---|---|
| Decision Trees | Non-Habitable | 0.9977333333 | 1 | 1 | 0.9992552026 |
| | Mesoplanet | 1 | 0.9992102665 | 0.9984287024 | 0.9994741455 |
| | Psychroplanet | 1 | 0.999669159 | 0.9993510707 | 0.9997808267 |
| Random Forests | Non-Habitable | 0.9992 | 1 | 1 | 0.9997371188 |
| | Mesoplanet | 1 | 0.9999341845 | 0.9998688697 | 0.9999561769 |
| | 0.9998701299 | 0.9996033058 | 0.9992212849 | 0.9996933187 | |
| XGBoost | Non-Habitable | 0.9986666667 | 1 | 1 | 0.9995618839 |
| | Mesoplanet | 1 | 0.9998025406 | 0.9996067121 | 0.9998685248 |
| | Psychroplanet | 1 | 0.9995370983 | 0.9990917348 | 0.9996932784 |

described are a combination of data synthesis by density estimation as well as oversampling with replacement: the most important features were modeled using probability distributions and the values of the less important features were sampled by replacement. By incorporating the physical limits, bounding the nature of growing data points (representing planets), fitting probability densities and classifying, the authors have emulated the application of classification algorithms to the data set with a considerable growth in the number of points. Exoplanets are being discovered at a fast rate, with recent hypes on Proxima b and the TRAPPIST-1 systems. This simulation shows that even with the growing number of discovered exoplanets, machine learning classifiers can do well to segregate planets into the correct classes of habitability.

The synthetic datasets which were generated and on which classification algorithms were tried can be found at: https://github.com/SuryodayBasak/ExoplanetsSyntheticData.

## 3.11 Conclusion

This paper has presented statistical techniques used on the PHL-EC data set in order to explore the capability of Machine learning algorithms in determining the habitability of an exoplanet. The potential of many algorithms, namely naïve Bayes, LDA, SVM, K-NN, decision trees, random forests, and XGBoost was explored to classify exoplanets based on their habitability. Naturally, questions are bound to arise regarding the choice and use of so many classifiers. Machine learning as an emerging area has its limitations and it's not surprising that scores of manuscripts are available in the public domain. However, many of these papers have applied different machine learning algorithms without adequately justifying the motivation and limitations of these algorithms. A method is as good as the data! The authors, throughout the manuscript, wanted to highlight this and endeavored to

construct and present their work as a primer in machine learning with respect to the data set used. The goal was to discover intrinsic limitations of each learning method and document those for the benefit of readers and young researchers who wish to apply machine learning in astronomy. The performance of a classifier depends on the nature of data, the size of data etc; however, there is no guarantee that a classifier which works well on one data set will work equally well on another data set, even if both data sets are from the same domain of astronomy. The separability of data is a major factor in deciding kind of appropriate classifiers for the corresponding data set. This fact is validated in the work presented. Hence, it is imperative for any exploratory data analysis to present a comparative study of different methods used.

The novelty of the current work lies in the selection of an appropriate data set, HEC (PHL-EC catalog) that was hitherto not investigated in the existing literature. The most important difference between NASA's catalog and PHL-EC is that the former makes data available for only those planets which are Kepler's Objects of interest whereas the latter contains data for all discovered planets, KOI or not, confirmed or unconfirmed. NASA's catalog for exoplanets has around 25 features whereas PHL-EC has 68 features, including but not limited to planet's mass, radius, orbital period, planet type, flux, density, distance from star, habitable zone, Earth similarity index (ESI) [Schulze-Makuch et al.2011], habitable class, composition class, eccentricity, etc. The website of the University of Puerto Rico, Planetary Habitability Laboratory lists the number of potentially habitable exoplanets: the results of our classification correlate remarkably well with what PHL has already stated in the conservative and optimistic samples of habitable planets.

It is important to point out that the PHL-EC [Méndez2016] data set assumes circular orbits (zero eccentricity) for planets with unknown eccentricities. This could raise questions if our predictions are accurate enough to describe real systems given the initial data set. Similarly, the equilibrium temperature is measured for Earth-like planets or mainly non-gaseous planets by considering the albedo of Earth (0.367), which again may not be close to their actual equilibrium temperature. In other words, the selected data set contains several estimated stellar and planetary parameters and they have also claimed many corrections [Méndez2016] which make them different from other exoplanet databases. This is the main reason we have selected PHL-EC: since it poses such challenges. Notwithstanding these limitations in the data set, the methods and improvisations as enunciated in Section 3.6.9, have worked remarkably well and the theoretical justifications of the efficacy of those methods have been well understood and documented by the authors.

In the process of exploring the data set and the various classifiers, a software called Ex-

oPlanet [Theophilus, Reddy & Basak2016] was developed (refer to Appendix **??**). This is a follow-up to the ASTROMLS KIT [Saha et al.2015]. The goal of the software is to reduce programming overheads in research involving data analytics. The software provides a graphical user interface (GUI) to select a data set, and then a method (classification, regression, and clustering) of choice can be selected by a *point-and-click mechanism*. The results (accuracy, sensitivity, specificity, etc) and all necessary graphs (ROC, etc.) are displayed in the same window. The software is currently in its infancy. However, the authors plan to extend the functionality by including more analysis, pre-processing and post-processing methods. A cloud-based web application is on the anvil.

The accuracy of various machine learning algorithms used on the PHL-EC data set has been computed and tabulated. Random forest, decision trees, and XGBoost rank best with the highest accuracy closely followed by naïve Bayes. A separate section has been dedicated to the classification of the recently discovered exoplanet Proxima b, where the current classification system has achieved accuracy as high as 100%. However, a lot of data is not available in the PHL-EC catalog: there exists a tremendously high bias towards the non-habitable class of planets, where the number of entities is 1000 times more than that of the other classes. The number of entities available in the psychroplanet and mesoplanet classes might be deemed as insufficient for an effective classification. Despite this bias, we were able to achieve remarkable accuracy with ML algorithms by performing artificial balancing on the data. This also goes to show that deep learning (which has unfortunately grown to become a cottage industry) is not necessary for every difficult classification scenario. A careful study of the nature of the data and trends is a must and simple solutions may often suffice.

In another effort to counter the effects of bias in the data set, the underrepresented classes of mesoplanets and psychroplanets were artificially augmented using assumptions of distributions as well as empirical analysis. Though many different methods of data synthesis may be adopted, the two most common and reasonable paradigms were tried. The accuracies achieved for this were near perfect (Tables 24 and 25). From this simulation exercise, it may be expected that with the natural extension of the data set in the future, the learning algorithms will continue to infer the data better and the accuracy will remain very high.

Exoplanets are frequently discovered and categorizing them manually is an arduous task. However, the work presented here may be translated into a simple automated system. A crawler, simple enough to design, may target the major databases and can append the catalog with discovered but non-categorized exoplanets. The suite of machine learning algorithms could then perform the task of classification, as demonstrated earlier, with reasonably acceptable accuracy. A significant portion of time, otherwise invested in studying parameters and

manual labeling, could thus be saved. In future, a continuation of the present work would be directed towards achieving a sustainable and automated discrimination system for efficient and accurate analysis of different exoplanet databases.

Coupled with web scraping methods and the suite of learning algorithms, automatic labeling of newly discovered exoplanets could thus be facilitated in a fairly easy and accurate manner. In summary, the work is a detailed primer on exploratory data analysis involving algorithmic improvisations and machine learning methods applied to a very complex data set, bolstered by a comprehensive understanding of these methods as documented in the appendices. The inferences drawn fortify these methods and the effort and time invested. The software ExoPlanet is designed to achieve the ultimate goal of classifying exoplanets with the aid of a limited manual or human intervention.

# 4 CD-HPF: New Habitability Score Via Data Analytic Modeling

## 4.1 Introduction

In the last decade, thousands of planets are discovered in our Galaxy alone. The inference is that stars with planets are a rule rather than exception [Cassan et al.2012], with estimates of the actual number of planet exceeding the number of stars in our Galaxy by orders of magnitude [Strigari et al.(2012)]. The same line of reasoning suggests a staggering number of at least $10^{24}$ planets in the observable Universe. The biggest question posed therefore is whether there are other life-harbouring planets. The most fundamental interest is in finding the Earth's twin. In fact, *Kepler* space telescope (`http://kepler.nasa.gov/`) was designed specifically to look for Earth's analogs – Earth-size planets in the habitable zones (HZ) of G-type stars [Batalha 2014]. More and more evidence accumulated in the last few years suggests that, in astrophysical context, Earth is an average planet, with average chemistry, existing in many other places in the Galaxy, average mass and **size**. Moreover, recent discovery of the rich organic content in the protoplanetary disk of newly formed star MWC 480 [Öberg et al.2015] has shown that neither is our Solar System unique in the abundance of the key components for life. Yet the only habitable planet in the Universe known to us is our Earth.

The question of habitability is of such interest and importance that the theoretical work has expanded from just the stellar HZ concept to the Galactic HZ (Gonzales et al. 2001) and, recently, to the Universe HZ — asking a question which galaxies are more habitable than others (Dayal et al. 2015). However, the simpler question — which of thousands detected planets are, or can be, habitable is still not answered. Life on other planets, if exists, may be similar to what we have on our planet, or may be in some other unknown form. The answer to this question may depend on understanding how different physical planetary parameters, such as planet's orbital properties, its chemical composition, mass, radius, density, surface and interior temperature, distance from it's parent star, even parent star's temperature or mass, combine to provide habitable conditions. With currently more than 1800 confirmed and more than 4000 unconfirmed discoveries[1], there is already enormous amount of accumulated data, where the challenge lies in the selection of how much to study about each planet, and which parameters are of the higher priority to evaluate.

Several important characteristics were introduced to address the habitability question. [Schulze-Makuch et al.2011] first addressed this issue through two indices, the Planetary

[1] Extrasolar Planets Encyclopedia, http://exoplanet.eu/catalog/

Habitability Index (PHI) and the Earth Similarity Index (ESI), where maximum, by definition, is set as 1 for the Earth, PHI=ESI=1.

ESI represents a quantitative measure with which to assess the similarity of a planet with the Earth on the basis of mass, size and temperature. But ESI alone is insufficient to conclude about the habitability, as planets like Mars have ESI close to 0.8 but we cannot still categorize it as habitable. There is also a possibility that a planet with ESI value slightly less than 1 may harbor life in some form which is not there on Earth, i.e. unknown to us. PHI was quantitatively defined as a measure of the ability of a planet to develop and sustain life. However, evaluating PHI values for large number of planets is not an easy task. In [Irwin et al.2014], another parameter was introduced to account for the chemical composition of exoplanets and some biology-related features such as substrate, energy, geophysics, temperature and age of the planet — the Biological Complexity Index (BCI). Here, we briefly describe the mathematical forms of these parameters.

**Earth Similarity Index (ESI)** ESI was designed to indicate how Earth-like an exoplanet might be [Schulze-Makuch et al.2011] and is an important factor to initially assess the habitability measure. Its value lies between 0 (no similarity) and 1, where 1 is the reference value, i.e. the ESI value of the Earth, and a general rule is that any planetary body with an ESI over 0.8 can be considered an Earth-like. It was proposed in the form

$$ESI_x = \left(1 - \left|\frac{x - x_0}{x + x_0}\right|\right)^w , \tag{33}$$

where $\mathrm{ESI}_x$ is the ESI value of a planet for $x$ property, and $x_0$ is the Earth's value for that property. The final ESI value of the planet is obtained by combining the geometric means of individual values, where $w$ is the weighting component through which the sensitivity of scale is adjusted. Four parameters: surface temperature $T_s$, density $D$, escape velocity $V_e$ and radius $R$, are used in ESI calculation. This index is split into interior $ESI_i$ (calculated from radius and density), and surface $ESI_s$ (calculated from escape velocity and surface temperature). Their geometric means are taken to represent the final $ESI$ of a planet. However, ESI in the form (69) was not introduced to define habitability, it only describes the similarity to the Earth in regard to some planetary parameters. For example, it is relatively high for the Moon.

**Planetary Habitability Index (PHI)** To actually address the habitability of a planet, [Schulze-Makuch et al.2
defined the PHI as

$$PHI = (S \cdot E \cdot C \cdot L)^{1/4} , \tag{34}$$

where $S$ defines a substrate, $E$ – the available energy, $C$ – the appropriate chemistry and $L$ – the liquid medium; all the variables here are in general vectors, while the corresponding scalars represent the norms of these vectors. For each of these categories, the PHI value is divided by the maximum PHI to provide the normalized PHI in the scale between 0 to 1. However, PHI in the form (34) lacks some other properties of a planet which may be necessary for determining its present habitability. For example, in Shchekinov et al. (2013) it was suggested to complement the original PHI with the explicit inclusion of the age of the planet (see their Eq. 6).

**4.1.0.1 Biological Complexity Index (BCI)** To come even closer to defining habitability, yet another index was introduced, comprising the above mentioned four parameters of the PHI and three extra parameters, such as geophysical complexity $G$, appropriate temperature $T$ and age $A$ [Irwin et al.2014]. Therefore, the total of seven parameters were initially considered to be important for the BCI. However, due to the lack of information on chemical composition and the existence of liquid water on exoplanets, only five were retained in the final formulation,

$$BCI = (S \cdot E \cdot T \cdot G \cdot A)^{1/5} . \tag{35}$$

It was found in [Irwin et al.2014] that for 5 exoplanets the BCI value is higher than for Mars, and that planets with high BCI values may have low values of ESI.

All previous indicators for habitability assume a planet to reside within in a classical HZ of a star, which is conservatively defined as a region where a planet can support liquid water on the surface [Huang1959, Kasting1993]. The concept of an HZ is, however, a constantly evolving one, and it has have been since suggested that a planet may exist beyond the classical HZ and still be a good candidate for habitability [Irwin & Schulze-Makuch2011, Heller & Armstrong2014]. Though presently all efforts are in search for the Earth's twin where the ESI is an essential parameter, it never tells that a planet with ESI close to 1 is habitable. Much advertised recent hype in press about finding the best bet for life-supporting planet – Gliese 832c with ESI = 0.81 [Wittenmyer et al.2014], was thwarted by the realization that the planet is more likely to be a super-Venus, with large thick atmosphere, hot surface and probably tidally locked with its star.

We present here the novel approach to determine the habitability score of all confirmed exoplanets analytically. Our goal is to determine the likelihood of an exoplanet to be habitable using the newly defined habitability score (CDHS) based on Cobb-Douglas habitability production function (CD-HPF), which computes the habitability score by using measured

and calculated planetary input parameters. Here, the PHI in its original form turned out to be a special case. We are looking for a feasible solution that maximizes habitability scores using CD-HPF with some defined constraints. In the following sections, the proposed model and motivations behind our work are discussed along with the results and applicability of the method. We conclude by listing key takeaways and robustness of the method. The related derivations and proofs are included in the appendices.

## 4.2 CD-HPF: Cobb-Douglas Habitability Production Function

We first present key definitions and terminologies that are utilized in this paper. These terms play critical roles in understanding the method and the algorithm adopted to accomplish our goal of validating the habitability score, **CDHS**, by using CD-HPF eventually.

**Key Definitions**

- **Mathematical Optimization**

  Optimization is one of the procedures to select the best element from a set of available alternatives in the field of mathematics, computer science, economics, or management science [Hájková & Hurnik2007]. An optimization problem can be represented in various ways. Below is the representation of an optimization problem. Given a function $f : A \to R$ from a set $A$ to the real numbers $R$. If an element $x_0$ in $A$ is such that $f(x_0) \leq f(x)$ for all $x$ in $A$, this ensures minimization. The case $f(x_0) \geq f(x)$ for all $x$ in $A$ is the specific case of maximization. The optimization technique is particularly useful for modeling the habitability score in our case. In the above formulation, the domain $A$ is called a search space of the function $f$, CD-HPF in our case, and elements of $A$ are called the candidate solutions, or feasible solutions. The function as defined by us is a utility function, yielding the habitability score CDHS. It is a feasible solution that maximizes the objective function, and is called an optimal solution under the constraints known as **Returns to scale**.

- **Returns to scale** measure the extent of an additional output obtained when all input factors change proportionally. There are three types of returns to scale:

  1. **Increasing returns to scale (IRS)**. In this case, the output increases by a larger proportion than the increase in inputs during the production process. For example, when we multiply the amount of every input by the number $N$, the factor by which output increases is more than $N$. This change occurs as

     [(i)]

(a) Greater application of the variable factor ensures better utilization of the fixed factor.

(b) Better division of the variable factor.

(c) It improves coordination between the factors.

The 3-D plots obtained in this case are neither concave nor convex.

2. **Decreasing returns to scale (DRS)**. Here, the proportion of increase in input increases the output, but in lower ratio, during the production process. For example, when we multiply the amount of every input by the number $N$, the factor by which output increases is less than $N$. This happens because:
   [(i)]

   (a) As more and more units of a variable factor are combined with the fixed factor, the latter gets over-utilized. Hence, the rate of corresponding growth of output goes on diminishing.

   (b) Factors of production are imperfect substitutes of each other. The divisibility of their units is not comparable.

   (c) The coordination between factors get distorted so that marginal product of the variable factor declines.

   The 3-D plots obtained in this case are concave.

3. **Constant returns to scale (CRS)**. Here, the proportion of increase in input increases output in the same ratio, during the production process. For example, when we multiply the amount of every input by a number $N$, the resulting output is multiplied by $N$. This phase happens for a negligible period of time and can be considered as a passing phase between IRS and DRS. The 3-D plots obtained in this case are concave.

- **Computational Techniques in Optimization**. There exist several well-known techniques including Simplex, Newton-like and Interior point-based techniques [Nemirovski & Todd2008]. One such technique is implemented via MATLAB's optimization toolbox using the function **fmincon**. This function helps find the global optima of a constrained optimization problem which is relevant to the model proposed and implemented by the authors. Illustration of the function and its syntax are provided in Appendix D.

- **Concavity**. Concavity ensures global maxima. The implication of this fact in our case is that if CD-HPF is proved to be concave under some constraints (this will be elaborated

later in the paper), we are guaranteed to have maximum habitability score for each exoplanet in the global search space.

- **Machine Learning**. Classification of patterns based on data is a prominent and critical component of machine learning and will be highlighted in subsequent part of our work where we made use of a standard K-NN algorithm. The algorithm is modified to tailor to the complexity and efficacy of the proposed solution. Optimization, as mentioned above, is the art of finding maximum and minimum of surfaces that arise in models utilized in science and engineering. More often than not, the optimum has to be found in an efficient manner, i.e. both the speed of convergence and the order of accuracy should be appreciably good. Machines are trained to do this job as, most of the times, the learning process is iterative. Machine learning is a set of methods and techniques that are intertwined with optimization techniques. The learning rate could be accelerated as well, making optimization problems deeply relevant and complementary to machine learning.

## 4.3 Cobb-Douglas Habitability Production Function CD-HPF

The general form of the Cobb-Douglas production function CD-PF is

$$Y = k \cdot (x_1)^{\alpha} \cdot (x_2)^{\beta}\,, \tag{36}$$

where $k$ is a constant that can be set arbitrarily according to the requirement, $Y$ is the total production, i.e. output, which is homogeneous with the degree 1; $x_1$ and $x_2$ are the input parameters (or factors); $\alpha$ and $\beta$ are the real fixed factors, called the elasticity coefficients. The sum of elasticities determines returns to scale conditions in the CDPF. This value can be less than 1, equal to 1, or greater than 1.

What motivates us to use the Cobb-Douglas production function is its properties. Cobb-Douglas production function (Cobb & Douglas, 1928) was originally introduced for modeling the growth of the American economy during the period of 1899–1922, and is currently widely used in economics and industry to optimize the production while minimizing the costs [Wu2001, Hossain et al.2012, Hassani2012, Saha et al.2016]. Cobb-Douglas production function is concave if the sum of the elasticities is not greater than one (see the proof in Bergstrom 2010). This gives global extremum in a closed interval which is handled by constraints in elasticity (Felipe & Adams, 2005). The physical parameters used in the Cobb-Douglas model may change over time and, as such, may be modeled as continuous entities. A functional

representation, i.e response, $Y$, is thus a continuous function, and may increase or decrease in maximum or minimum value as these parameters change (Hossain et al., 2012). Our formulation serves this purpose, where elasticities may be adjusted via *fmincon* or fitting algorithms, in conjunction with the intrinsic property of the CD-HPF that ensures global maxima for concavity. Our simulations, that include animation and graphs, support this trend (see Figures 1 and 2 in Section 3). As the physical parameters change in value, so do the function values and its maximum for all the exoplanets in the catalog, and this might rearrange the CDHS pattern with possible changes in the parameters, while maintaining consistency with the database.

The most important properties of this function that make it flexible to be used in various applications are:

- It can be transformed to the log-linear form from its multiplicative form (non-linear) which makes it simple to handle, and hence, linear regression techniques can be used for estimation of missing data.
- Any proportional change in any input parameter can be represented easily as the change in the output.
- The ratio of relative inputs $x_1$ and $x_2$ to the total output $Y$ is represented by the elasticities $\alpha$ and $\beta$.

The analytical properties of the CDPF motivated us to check the applicability in our problem, where the four parameters considered to estimate the habitability score are surface temperature, escape velocity, radius and density. Here, the production function $Y$ is the habitability score of the exoplanet, where the aim is to maximize $Y$, subject to the constraint that the sum of all elasticity coefficients shall be less than or equal to 1. Computational optimization is relevant for elasticity computation in our problem. Elasticity is the percentage change in the output $Y$ (Eq. 4), given one percent change in the input parameter, $x_1$ or $x_2$. We assume $k$ is constant. In other words, we compute the rate of change of output $Y$, the CDPF, with respect to one unit of change in input, such as $x_1$ or $x_2$. As the quantity of $x_1$ or $x_2$ increases by one percent, output increases by $\alpha$ or $\beta$ percent. This is known as the elasticity of output with respect to an input parameter. As it is, values of the elasticity, $\alpha$ and $\beta$ are not ad-hoc and need to be approximated for optimization purpose by some computational technique. The method, *fmincon* with interior point search, is used to compute the elasticity values for CRS, DRS and IRS. The outcome is quick and accurate. We elaborate the significance of the scales and elasticity in the context of CDPF and CDHS below.

- **Increasing returns to scale (IRS):** In Cobb-Douglas model, if $\alpha + \beta > 1$, the case is called an IRS. It improves the coordination among the factors. This is indicative of boosting the habitability score following the model with one unit of change in respective predictor variables.

- **Decreasing returns to scale (DRS):** In Cobb-Douglas model, if $\alpha + \beta < 1$, the case is called a DRS, where the deployment of an additional input may affect the output with diminishing rate. This implies the habitability score following the model may decrease with the one unit of change in respective predictor variables.

- **Constant returns to scale (CRS):** In Cobb-Douglas model, if $\alpha + \beta = 1$, this case is called a CRS, where increase in $\alpha$ or/and $\beta$ increases the output in the same proportion. The habitability score, i.e the response variable in the Cobb-Douglas model, grows proportionately with changes in input or predictor variables.

The range of elasticity constants is between 0 and 1 for DRS and CRS. This will be exploited during the simulation phase (Section 3). It is proved in Appendices B and C that the habitability score (CDHS) maximization is accomplished in this phase for **DRS and CRS**, respectively.

The impact of change in the habitability score according to each of the above constraints will be elaborated in Sections 4 and 5. Our aim is to optimize elasticity coefficients to maximize the habitability score of the confirmed exoplanets using the CD-HPF .

## 4.4 Cobb-Douglas Habitability Score estimation

We have considered the same four parameters used in the ESI metric (Eq. 69), i.e. surface temperature, escape velocity, radius and density, to calculate the Cobb-Douglas Habitability Score (CDHS). Analogous to the method used in ESI, two types of Cobb-Douglas Habitability Scores are calculated – the interior $\text{CDHS}_i$ and the surface $\text{CDHS}_i$. The final score is computed by a linear convex combination of these two, since it is well known that a convex combination of convex/concave function is also convex/concave. The interior $\text{CDHS}_i$, denoted by $Y1$, is calculated using radius $R$ and density $D$,

$$Y1 = CDHS_i = (D)^{\alpha} \cdot (R)^{\beta}. \tag{37}$$

The surface $\text{CDHS}_s$, denoted by $Y2$, is calculated using surface temperature $T_s$ and escape velocity $V_e$,

$$Y2 = CDHS_s = (T_s)^{\gamma} \cdot (V_e)^{\delta}. \tag{38}$$

The final CDHS $Y$, which is a convex combination of $Y1$ and $Y2$, is determined by

$$Y = w' \cdot Y1 + w'' \cdot Y2\,, \tag{39}$$

where the sum of $w'$ and $w''$ equals 1. The values of $w'$ and $w''$ are the weights of the interior CDHS$_i$ and surface CDHS$_s$, respectively. These weights depend on the importance of individual parameters of each exoplanet. The $Y1$ and $Y2$ are obtained by applying CDPF (Eq. 36) with $k = 1$. Finally, the Cobb-Douglas habitability production function (CD-HPF) can be formally written as

$$\mathbb{Y} = f\,(R, D, T_s, V_e) = (R)^{\alpha} \cdot (D)^{\beta} \cdot (T_s)^{\gamma} \cdot (V_e)^{\delta}\,. \tag{40}$$

For a 3-D interpretation of the CDPF model with elasticities $\alpha$ and $\beta$, Appendix A contains brief discussion on manipulating $\alpha$ and $\beta$ algebraically. The goal is to maximize $Y$, iff $\alpha + \beta + \gamma + \delta < 1$. It is possible to calculate the CDHS by using both Eqs. (39) and (50), however there is hardly any difference in the final value. Equation (50) is impossible to visualize since it is a 5-dimensional entity. Whereas, Eq. (39) has 3-dimensional structure.The ease of visualization is the reason **CDHS** is computed by splitting into two parts $Y1$ and $Y2$ and combining by using the weights $w'$ and $w''$. Individually, each of $Y1$ and $Y2$ are sample 3-D models and, as such, are easily comprehensible via surface plots as demonstrated later (see Figs. 1 and 2 in Section 3). The authors would like to emphasize that instead of splitting and computing CDHS as a convex combination of $Y1$ and $Y2$, a direct calculation of CDHS through Eq. (50) is possible, which does not alter the final outcome. It is avoided here, since using the product of all four parameters with corresponding elasticities $\alpha, \beta, \gamma$ and $\delta$ would make rendering the plots impossible for the simple reason of dimensionality being too high, 5 instead of 3. We reiterate that the scalability of the model from $\alpha, \beta$ to $\alpha, \beta, \gamma$ and $\delta$ does not suffer due to this scheme. The proof presented in Appendix B bears testimony to our claim.

## 4.5 The Theorem for Maximization of Cobb-Douglas habitability production function

**Statement:** CD-HPF attains global maxima in the phase of DRS or CRS [Saha et al.2016].
*Sketch of proof*: Generally profit of a firm can be defined as

$$\text{profit} = \text{revenue} - \text{cost} = \big(\text{price of output} \times \text{output}\big) - \big(\text{price of input} \times \text{input}\big)\,.$$

Let $p_1, p_2, \ldots, p_n$ be a vector of prices for outputs, or products, and $w_1, w_2, \ldots, w_m$ be a vector of prices for inputs of the firm, which are always constants; and let the input levels be $x_1, x_2, \ldots, x_m$, and the output levels be $y_1, y_2, \ldots, y_n$. The profit, generated by the production plan, $(x_1, \ldots, x_m, y_1, \ldots, y_n)$ is

$$\pi = \left(p_1 \cdot y_1 + \ldots + p_n \cdot y_n - w_1 \cdot x_1 - \ldots - w_m \cdot x_m\right).$$

Suppose the production function for $m$ inputs is

$$Y = f\left(x_1, x_2, ..., x_m\right),$$

and its profit function is

$$\pi = p \cdot Y - w_1 \cdot x_1 - \ldots - w_m \cdot x_m.$$

A single output function needs $p$ as the price, while multiple output functions will require multiple prices $p_1, p_2, \ldots, p_n$. The profit function in our case, which is a single-output multiple-inputs case, is given by

$$\pi = p f\left(R, D, T_s, V_e\right) - w_1 R - w_2 D - w_3 T_s - w_4 V_e, \tag{41}$$

where $w_1, w_2, w_3, w_4$ are the weights chosen according to the importance for habitability for each planet. Maximization of CD-HPF is achieved when

$$(1)\ p\frac{\partial f}{\partial R} = w_1, \quad (2)\ p\frac{\partial f}{\partial D} = w_2, \quad (3)\ p\frac{\partial f}{\partial T_s} = w_3, \quad (4)\ p\frac{\partial f}{\partial V_e} = w_4. \tag{42}$$

The habitability score is conceptualized as a profit function where the cost component is introduced as a penalty function to check unbridled growth of CD-HPF. This bounding framework is elaborated in the proofs of concavity, the global maxima and computational optimization technique, and function *fmincon* in Appendices B, C and D, respectively.

**Remark**: If we consider the case of CRS, where all the elasticities of different cost components are equal, the output is $Y = \prod_{i=1}^{n} x_i^{\alpha_i}$, where all $\alpha_i$ are equal and $\sum \alpha_i = 1$. In such scenario, $Y \equiv G.M.$ (Geometric Mean) of the cost inputs. Further scrutiny reveals that the geometric mean formalization is nothing but the representation of the PHI, thus establishing our framework of CD-HPF as a broader model, with the PHI being a corollary for the CRS case.

Once we compute the habitability score, $Y$, the next step is to perform clustering of the Y

values. We have used K-nearest neighbor (K-NN) classification algorithm and introduced probabilistic herding and thresholding to group the exoplanets according to their $Y$ values. The algorithm finds the exoplanets for which $Y$ values are very close to each other and keeps them in the same group, or cluster. Each CDHS value is compared with its $K$ (specified by the user) nearest exoplanet's (closer $Y$ values) CDHS value, and the class which contains maximum nearest to the new one is allotted as a class for it.

## 4.6 Implementation of the Model

We applied the CD-HPF to calculate the Cobb-Douglas habitability score (CDHS) of exoplanets. A total of 664 confirmed exoplanets are taken from the Planetary Habitability Laboratory Exoplanets Catalog (PHL-EC)[2]. The catalog contains observed and estimated stellar and planetary parameters for a total of 3415 (July 2016) currently confirmed exoplanets, where the estimates of the surface temperature are given for 1586 planets. However, there are only 586 rocky planets where the surface temperature is estimated, using the correction factor of 30-33 K added to the calculated equilibrium temperature, based on the Earth's greenhouse effect (Schulze-Makuch et al. 2011a; Volokin & ReLlez 2016). For our dataset, we have taken all rocky planets plus several non-rocky samples to check the algorithm. In machine learning, such random samples are usually used to check for the robustness of the designed algorithm and to add variations in the training and test samples. Otherwise, the train and test samples would become heavily biased towards one particular trend. As mentioned above, the CDHS of exoplanets are computed from the interior $\text{CDHS}_i$ and the surface $\text{CDHS}_s$. The input parameters radius $R$ and density $D$ are used to compute the values of the elasticities $\alpha$ and $\beta$. Similarly, the input parameters surface temperature $T_S$ and escape velocity $V_e$ are used to compute the elasticities $\gamma$ and $\delta$. These parameters, except the surface temperature, are given in Earth Units (EU) in the PHL-EC catalog. We have normalized the surface temperatures $T_s$ of exoplanets to the EU, by dividing each of them with Earth's mean surface temperature, 288 K.

The Cobb-Douglas function is applied on varying elasticities to find the CDHS close to Earth's value, which is considered as 1. As all the input parameters are represented in EU, we are looking for the exoplanets whose CDHS is closer to Earth's CDHS. For each exoplanet, we obtain the optimal elasticity and the maximum CDHS value. The results are demonstrated graphically using 3-D plot. All simulations were conducted using the MATLAB software for the cases of DRS and CRS. From Eq. (B.38), we can see that for CRS $Y$ will grow asymptotically,

[2] provided by the Planetary Habitability Laboratory @ UPR Arecibo, accessible at `http://phl.upr.edu/projects/habitable-exoplanets-catalog/data/database`

if

$$\alpha + \beta + \gamma + \delta = 1\,. \tag{43}$$

Let us set

$$\alpha = \beta = \gamma = \delta = 1/4\,. \tag{44}$$

In general, the values of elasticities may not be equal but the sum may still be 1. As we know already, this is CRS. A special case of CRS, where the elasticity values are made to be equal to each other in Eq. (12), turns out to be structurally analogous to the PHI and BCI formulations. Simply stated, the CD-HPF function satisfying this special condition may be written as

$$Y = f = k\,(R \cdot D \cdot T_s \cdot V_e)^{1/4}\,. \tag{45}$$

The function is concave for CRS and DRS (Appendices B and C).

## 4.7 Computation of CDHS in DRS phase

We have computed elasticities separately for interior $\text{CDHS}_i$ and surface $\text{CDHS}_s$ in the DRS phase. These values were obtained using function *fmincon*, a computational optimization technique explained in Appendix D. Tables 1 through 3 show a sample of computed values. Table 26 shows the computed elasticities $\alpha, \beta$ and $\text{CDHS}_i$. The optimal interior $\text{CDHS}_i$ for most exoplanets are obtained at $\alpha = 0.8$ and $\beta = 0.1$. Table 2 shows the computed elasticities $\gamma, \delta$ and $\text{CDHS}_s$. The optimal surface CDHS are obtained at $\gamma = 0.8$ and $\delta = 0.1$. Using these results, 3-D graphs are generated and are shown in Figure 1. The $X$ and $Y$ axes represent elasticities and $Z$-axis represents CDHS of exoplanets. The final CDHS, $Y$, calculated using Eq. (7) with $w' = 0.99$ and $w'' = 0.01$, is presented in Table 3.

## 4.8 Computation of CDHS in CRS phase

The same calculations were carried out for the CRS phase. Tables 4, 5 and 6 show the sample of computed elasticities and habitability scores in CRS. The convex combination of $\text{CDHS}_i$ and $\text{CDHS}_s$ gives the final CDHS (Eq. 7) with $w' = 0.99$ and $w'' = 0.01$. The optimal interior $\text{CDHS}_i$ for most exoplanets were obtained at $\alpha = 0.9$ and $\beta = 0.1$, and the optimal surface $\text{CDHS}_s$ were obtained at $\gamma = 0.9$ and $\delta = 0.1$. Using these results, 3-D graphs were generated and are shown in Figure 2.

Tables 1, 2 and 3 represent CDHS for DRS, where the corresponding values of elasticities were found by *fmincon* to be 0.8 and 0.1, and the sum= $0.9 < 1$ (The theoretical proof is

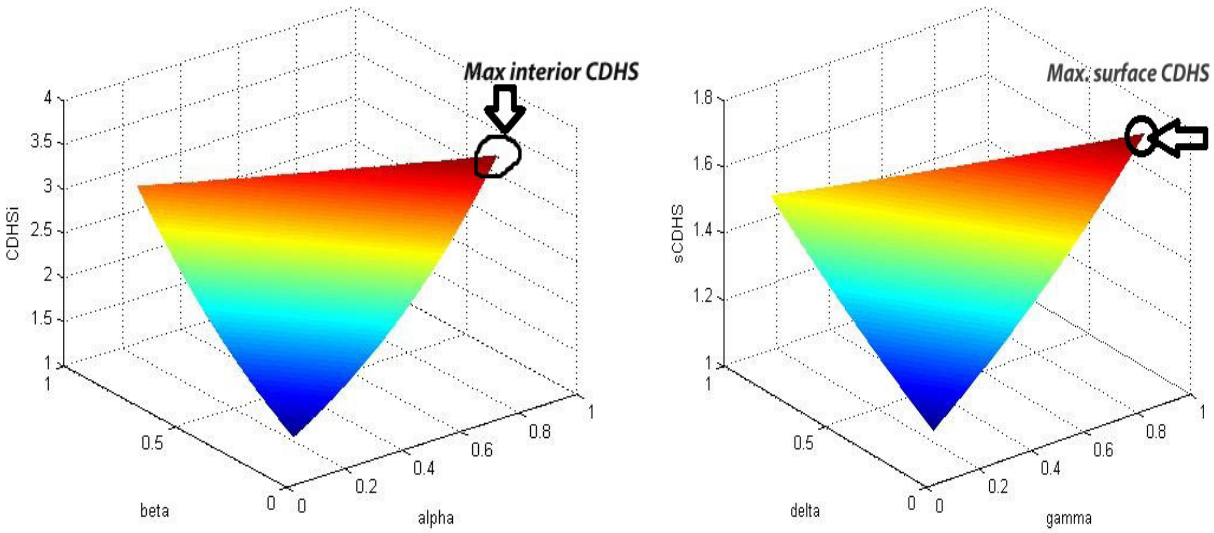



**Figure 12:** Plot of interior CDHS$_i$ (*Left*) and surface CDHS$_s$ (*Right*) for DRS

given in Appendix B). Tables 4, 5 and 6 show results for CRS, where the sum of the elasticities = 1 (The theoretical proof is given in Appendix C). The approximation algorithm *fmincon* initiates the search for the optima by starting from a random initial guess, and then it applies a step increment or decrements based on the gradient of the function based on which our modeling is done. It terminates when it cannot find elasticities any better for the maximum

**Table 26:** Sample simulation output of interior CDHS$_i$ of exoplanets calculated from radius and density for DRS

| Exoplanet | Radius | Density | Elasticity($\alpha$) | Elasticity ($\beta$) | CDHS$_i$ |
|---|---|---|---|---|---|
| GJ 163 c | 1.83 | 1.19 | 0.8 | 0.1 | 1.65012 |
| GJ 176 b | 1.9 | 1.23 | 0.8 | 0.1 | 1.706056 |
| GJ 667C b | 1.71 | 1.12 | 0.8 | 0.1 | 1.553527 |
| GJ 667C c | 1.54 | 1.05 | 0.8 | 0.1 | 1.4195 |
| GJ 667C d | 1.67 | 1.1 | 0.8 | 0.1 | 1.521642 |
| GJ 667C e | 1.4 | 0.99 | 0.8 | 0.1 | 1.307573 |
| GJ 667C f | 1.4 | 0.99 | 0.8 | 0.1 | 1.307573 |
| GJ 3634 b | 1.81 | 1.18 | 0.8 | 0.1 | 1.634297 |
| Kepler-186 f | 1.11 | 0.9 | 0.8 | 0.1 | 1.075679 |
| Gl 15 A b | 1.69 | 1.11 | 0.8 | 0.1 | 1.537594 |
| HD 20794 c | 1.35 | 0.98 | 0.8 | 0.1 | 1.26879 |
| HD 40307 e | 1.5 | 1.03 | 0.8 | 0.1 | 1.387256 |
| HD 40307 f | 1.68 | 1.11 | 0.8 | 0.1 | 1.530311 |
| HD 40307 g | 1.82 | 1.18 | 0.8 | 0.1 | 1.641517 |

**Table 27:** Sample simulation output of surface CDHS of exoplanets calculated from escape velocity and surface temperature for DRS

| Exoplanet | Escape Velocity | Surface temperature | Elasticity ($\gamma$) | Elasticity ($\delta$) | $CDHS_s$ |
|---|---|---|---|---|---|
| GJ 163 c | 1.99 | 1.11146 | 0.8 | 0.1 | 1.752555 |
| GJ 176 b | 2.11 | 1.67986 | 0.8 | 0.1 | 1.91405 |
| GJ 667C b | 1.81 | 1.49063 | 0.8 | 0.1 | 1.672937 |
| GJ 667C c | 1.57 | 0.994 | 0.8 | 0.1 | 1.433764 |
| GJ 667C d | 1.75 | 0.71979 | 0.8 | 0.1 | 1.51409 |
| GJ 667C e | 1.39 | 0.78854 | 0.8 | 0.1 | 1.27085 |
| GJ 667C f | 1.39 | 0.898958 | 0.8 | 0.1 | 1.287614 |
| GJ 3634 b | 1.97 | 2.1125 | 0.8 | 0.1 | 1.946633 |
| Kepler-186 f | 1.05 | 0.7871 | 0.8 | 0.1 | 1.015213 |
| Gl 15 A b | 1.78 | 1.412153 | 0.8 | 0.1 | 1.641815 |
| HD 40307 e | 1.53 | 1.550694 | 0.8 | 0.1 | 1.482143 |
| HD 40307 f | 1.76 | 1.38125 | 0.8 | 0.1 | 1.623444 |
| HD 40307 g | 1.98 | 0.939236 | 0.8 | 0.1 | 1.716365 |
| HD 20794 c | 1.34 | 1.89791667 | 0.8 | 0.1 | 1.719223 |

**Table 28:** Sample simulation output of CDHS with $w' = 0.99$ and $w'' = 0.01$ for DRS

| Exoplanet | $CDHS_i$ | $CDHS_s$ | CDHS |
|---|---|---|---|
| GJ 163 c | 1.65012 | 1.752555 | 1.651144 |
| GJ 176 b | 1.706056 | 1.91405 | 1.708136 |
| GJ 667C b | 1.553527 | 1.672937 | 1.554721 |
| GJ 667C c | 1.4195 | 1.433764 | 1.419643 |
| GJ 667C d | 1.521642 | 1.514088 | 1.521566 |
| GJ 667C e | 1.307573 | 1.27085 | 1.307206 |
| GJ 667C f | 1.307573 | 1.287614 | 1.307373 |
| GJ 3634 b | 1.634297 | 1.946633 | 1.63742 |
| Gl 15 A b | 1.537594 | 1.641815 | 1.538636 |
| Kepler-186 f | 1.075679 | 1.015213 | 1.075074 |
| HD 20794 c | 1.26879 | 1.719223 | 1.273294 |
| HD 40307 e | 1.387256 | 1.482143 | 1.388205 |
| HD 40307 f | 1.530311 | 1.623444 | 1.531242 |
| HD 40307 g | 1.641517 7 | 1.716365 | 1.642265 |

CDHS. The plots in Figures 1 and 2 show all the elasticities for which *fmincon* searches for the global maximum in CDHS, indicated by a black circle. Those values are read off from the code (given in Appendix E) and printed as 0.8 and 0.1, or whichever the case may be. A minimalist web page is designed to host all relevant data and results: sets, figures, animation video and a graphical abstract. It is available at `https://habitabilitypes.wordpress.com/`.

**Table 29:** Sample simulation output of interior $CDHS_i$ of exoplanets calculated from radius and density for CRS

| Exoplanet | Radius | Density | Elasticity($\alpha$) | Elasticity ($\beta$) | $CDHS_i$ |
|---|---|---|---|---|---|
| GJ 163 c | 1.83 | 1.19 | 0.9 | 0.1 | 1.752914 |
| GJ 176 b | 1.9 | 1.23 | 0.9 | 0.1 | 1.819151 |
| GJ 667C b | 1.71 | 1.12 | 0.9 | 0.1 | 1.639149 |
| GJ 667C c | 1.54 | 1.05 | 0.9 | 0.1 | 1.482134 |
| GJ 667C d | 1.67 | 1.1 | 0.9 | 0.1 | 1.601711 |
| GJ 667C e | 1.4 | 0.99 | 0.9 | 0.1 | 1.352318 |
| GJ 667C f | 1.4 | 0.99 | 0.9 | 0.1 | 1.352318 |
| GJ 3634 b | 1.81 | 1.18 | 0.9 | 0.1 | 1.734199 |
| Kepler-186 f | 1.11 | 0.9 | 0.9 | 0.1 | 1.086963 |
| Gl 15 A b | 1.69 | 1.11 | 0.9 | 0.1 | 1.62043 |
| HD 20794 c | 1.35 | 0.98 | 0.9 | 0.1 | 1.307444 |
| HD 40307 e | 1.5 | 1.03 | 0.9 | 0.1 | 1.444661 |
| HD 40307 f | 1.68 | 1.11 | 0.9 | 0.1 | 1.611798 |
| HD 40307 g | 1.82 | 1.18 | 0.9 | 0.1 | 1.74282 |

**Table 30:** Sample simulation output of surface CDHS of exoplanets calculated from escape velocity and surface temperature for CRS

| Exoplanet | Escape Velocity | Surface temperature | Elasticity ($\gamma$) | Elasticity ($\delta$) | $CDHS_s$ |
|---|---|---|---|---|---|
| GJ 163 c | 1.99 | 1.11146 | 0.9 | 0.1 | 1.877401 |
| GJ 176 b | 2.11 | 1.67986 | 0.9 | 0.1 | 2.062441 |
| GJ 667C b | 1.81 | 1.49063 | 0.9 | 0.1 | 1.775201 |
| GJ 667C c | 1.57 | 0.994 | 0.9 | 0.1 | 1.499919 |
| GJ 667C d | 1.75 | 0.71979 | 0.9 | 0.1 | 1.601234 |
| GJ 667C e | 1.39 | 0.78854 | 0.9 | 0.1 | 1.313396 |
| GJ 667C f | 1.39 | 0.898958 | 0.9 | 0.1 | 1.330722 |
| GJ 3634 b | 1.97 | 2.1125 | 0.9 | 0.1 | 2.097798 |
| Kepler-186 f | 1.05 | 0.7871 | 0.9 | 0.1 | 1.020179 |
| Gl 15 A b | 1.78 | 1.412153 | 0.9 | 0.1 | 1.739267 |
| HD 40307 e | 1.53 | 1.550694 | 0.9 | 0.1 | 1.548612 |
| HD 40307 f | 1.76 | 1.38125 | 0.9 | 0.1 | 1.717863 |
| HD 40307 g | 1.98 | 0 .939236 | 0.9 | 0.1 | 1.837706 |
| HD 20794 c | 1.34 | 1.89791667 | 0.9 | 0.1 | 1.832989 |

The animation video, available at the website, demonstrates the concavity property of CD-HPF and CDHS. The animation comprises 664 frames (each frame is a surface plot essentially), corresponding to 664 exoplanets under consideration. Each frame is a visual representation of the outcome of CD-HPF and CDHS applied to each exoplanet. The $X$ and $Y$ axes of the 3-D plots represent elasticity constants and $Z$-axis represents the CDHS. Simply

**Table 31:** Sample simulation output of CDHS with $w' = 0.99$ and $w'' = 0.01$ for CRS

| Exoplanet | $CDHS_i$ | $CDHS_s$ | CDHS |
|---|---|---|---|
| GJ 163 c | 1.752914 | 1.877401 | 1.754159 |
| GJ 176 b | 1.819151 | 2.062441 | 1.821584 |
| GJ 667C b | 1.639149 | 1.775201 | 1.64051 |
| GJ 667C c | 1.482134 | 1.499919 | 1.482312 |
| GJ 667C d | 1.601711 | 1.601234 | 1.601706 |
| GJ 667C e | 1.352318 | 1.313396 | 1.351929 |
| GJ 667C f | 1.352318 | 1.330722 | 1.352102 |
| GJ 3634 b | 1.734199 | 2.097798 | 1.737835 |
| Kepler-186 f | 1.086963 | 1.020179 | 1.086295 |
| GI 15 A b | 1.62043 | 1.739267 | 1.621618 |
| HD 40307 e | 1.444661 | 1.548612 | 1.445701 |
| HD 40307 f | 1.611798 | 1.717863 | 1.612859 |
| HD 40307 g | 1.74282 | 1.837706 | 1.743769 |
| HD 20794 c | 1.307444 | 1.832989 | 1.312699 |

stated, each frame, demonstrated as snapshots of the animation in Figs. 1 and 2, is endowed with a maximum CDHS and the cumulative effect of all such frames is elegantly captured in the animation.

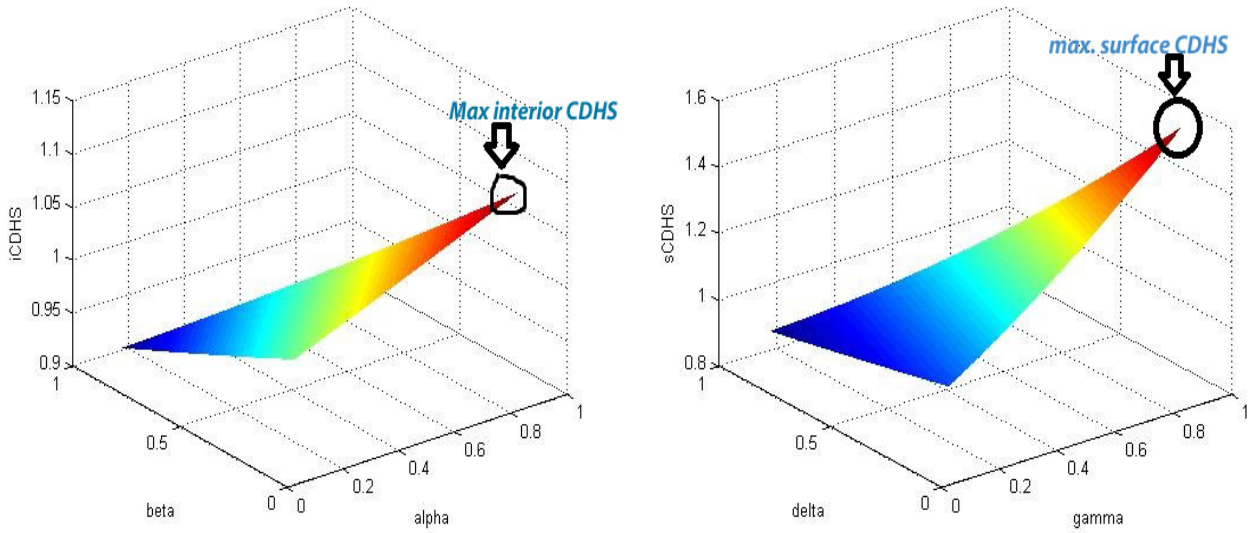



**Figure 13:** Plot of interior $CDHS_i$ (*Left*) and surface $CDHS_s$ (*Right*) for CRS

## 4.9 Attribute Enhanced K-NN Algorithm: A Machine learning approach

K-NN, or K-nearest neighbor, is a well-known machine learning algorithm. Attribute-enhanced K-NN algorithm is used to classify the exoplanets into different classes based on the computed CDHS values. 80% of data from the Habitable Exoplanets Catalog (HEC)[3]) are used for training, and remaining 20% for testing. Training–testing process is integral to machine learning, where the machine is trained to recognize patterns by assimilating a lot of data and, upon applying the learned patterns, identifies new data with a reasonable degree of accuracy. The efficacy of a learning algorithm is reflected in the accuracy with which the test data is identified. The training data set is uniformly distributed between 5 classes, known as balancing the data, so that bias in the training sample is eliminated. The algorithm produces 6 classes, wherein each class carries exoplanets with CDHS values close to each other, a first condition for being called as "neighbours". Initially, each class holds one fifth of the training data and a new class, i.e. Class 6, defined as Earth's Class (or "Earth-League"), is derived by the proposed algorithm from first 5 classes where it contains data based on the two conditions.
The two conditions that our algorithm uses to select exoplanets into Class 6 are defined as:

1. Thresholding: Exoplanets with their CDHS minus Earth's CDHS being less than or equal to the specified boundary value, called threshold. We have set a threshold in such a way that the exoplanets with CDHS values within the threshold of 1 (closer to Earth) fall in Earth's class. The threshold is chosen to capture proximal planets as the CDHS of all exoplanets considered vary greatly However, this proximity alone does not determine habitability.

2. Probabilistic Herding: if exoplanet is in the HZ of its star, it implies probability of membership to the Earth-League, Class 6, to be high; probability is low otherwise. Elements in each class in K-NN get re-assigned during the run time. This automatic re-assignment of exoplanets to different classes is based on a weighted likelihood concept applied on the members of the initial class assignment.

Consider $K$ as the desired number of nearest neighbors and let $S := p_1, \ldots, p_n$ be the set of training samples in the form $p_i = (x_i, c_i)$, where $x_i$ is the $d$-dimensional feature vector of the point $p_i$ and $c_i$ is the class that $p_i$ belongs to. In our case, dimension, $d = 1$. We fix $S' := p_{1'}, \ldots, p_{m'}$ to be the set of testing samples. For every sample, the difference in CDHS

[3]The Habitable Exoplanets Catalog (HEC) is an online database of potentially habitable planets, total 32 as on January 16, 2016; maintained by the Planetary Habitability LaboratoryUPR Arecibo, and available at `http://phl.upr.edu/projects/habitable-exoplanets-catalog`

between Earth and the sample is computed by looping through the entire dataset containing the 5 classes. Class 6 is the offspring of these 5 classes and is created by the algorithmic logic to store the selected exoplanets which satisfy the conditions of the K-NN and the two conditions – thresholding and probabilistic herding defined above. We train the system for 80% of the data-points based on the two constraints, prob($habitability_i$) = 'high' and CDHS($p_i$)−CDHS(**Earth**)≤ threshold. These attributes enhance the standard K-NN and help the re-organization of $exoplanet_i$ to Class 6.

If CDHS of $exoplanet_i$ falls with a certain range, K-NN classifies it accordingly into one of the remaining 5 classes. For each $p' = (x', c')$, we compute the distance $d(x', x_i)$ between $p'$ and all $p_i$ for the dataset of 664 exoplanets from the PHL-EC, $S$. Next, the algorithm selects the $K$ nearest points to $p'$ from the list computed above. The classification algorithm, K-NN, assigns a class $c'$ to $p'$ based on the condition prob($habitability_i$) = 'high' plus the thresholding condition mentioned above. Otherwise, K-NN assigns $p'$ to the class according to the range set for each class. Once the "Earth-League" class is created after the algorithm has finished its run, the list is cross-validated with the habitable exoplanet catalog HEC. It must be noted that Class 6 not only contains exoplanets that are similar to Earth, but also the ones which are most likely to be habitable. The algorithmic representation of K-NN is presented in Appendix E.

## 4.10 Results and Discussion

The K-NN classification method has resulted in "Earth-league", Class 6, having 14 and 12 potentially habitable exoplanets by DRS and CRS computations, respectively. The outcome of the classification algorithm is shown in Tables 7 and 8.

There are 12 common exoplanets in Tables 7 and 8. We have cross-checked these planets with the Habitable Exoplanets Catalog and found that they are indeed listed as potentially habitable planets. Class 6 includes all the exoplanets whose CDHS is proximal to Earth. As explained above, classes 1 to 6 are generated by the machine learning technique and classification method. Class 5 includes the exoplanets which are likely to be habitable, and planets in Classes 1, 2, 3 & 4 are less likely to be habitable, with Class 1 being the least likely to be habitable. Accuracy achieved here is 92% for $K = 1$, implying 1-nearest neighbor, and is 94% for $K = 7$, indicating 7 nearest neighbors.

In Figure 3 we show the plots of K-NN algorithm applied on the results in DRS (top plot) and CRS (bottom plot) cases. The $X$-axis represents CDHS and $Y$-axis – the 6 dif-

**Table 32:** Potentially habitable exoplanets in Earth's class using DRS: Outcome of CDHS and K-NN

| Exoplanet | CDH Score |
|---|---|
| GJ 667C e | 1.307206 |
| GJ 667C f | 1.307373 |
| GJ 832 c | 1.539553 |
| HD 40307 g | 1.642265 |
| Kapteyn's b | 1.498503 |
| Kepler-61 b | 1.908765 |
| Kepler-62 e | 1.475502 |
| Kepler-62 f | 1.316121 |
| Kepler-174 d | 1.933823 |
| Kepler-186 f | 1.07507 |
| Kepler-283 c | 1.63517 |
| Kepler-296 f | 1.619423 |
| GJ 667C c | 1.419643 |
| GJ 163 c | 1.651144 |

**Table 33:** Potentially habitable exoplanets in Earth's class using CRS: Outcome of CDHS and K-NN

| Exoplanet | CDH Score |
|---|---|
| GJ 667C e | 1.351929 |
| GJ 667C f | 1.352102 |
| GJ 832 c | 1.622592 |
| HD 40307 g | 1.743769 |
| Kapteyn's b | 1.574564 |
| Kepler-62 e | 1.547538 |
| Kepler-62 f | 1.362128 |
| Kepler-186 f | 1.086295 |
| Kepler-283 c | 1.735285 |
| Kepler-296 f | 1.716655 |
| GJ 667C c | 1.482312 |
| GJ 163 c | 1.754159 |

ferent classes assigned to each exoplanet. The figure is a schematic representation of the outcome of our algorithm. The color points, shown in circles and boxes to indicate the membership in respective classes, are representative of membership only and do not indicate a quantitative equivalence. The numerical data on the number of the exoplanets in each class is provided in Appendix F. A quantitative representation of the figures may be found at `https://habitabilitypes.wordpress.com/`.

We also normalized CDHS of each exoplanet, dividing by the maximum score in each category, for both CRS and DRS cases (with Earth's normalized score for CRS = 0.003176 and DRS = 0.005993). This resulted in CDHS of all 664 exoplanets ranging from 0 to 1. Analogous to the case of non-normalized CDHS, these exoplanets have been assigned equally to 5 classes. K-NN algorithm was then applied to all the exoplanets' CDHS for both CRS and DRS cases. Similar to the method followed in non-normalized CDHS for CRS and DRS, K-NN has been applied to "dump" exoplanets which satisfy the criteria of being members of Class 6. Table 9 shows the potentially habitable exoplanets obtained from classification on normalized data for both CRS and DRS. This result is illustrated in Figs. 3c and 3d. In this figure, Class 6 contains 16 exoplanets generated by K-NN and which are considered to be potentially habitable according to the PHL-EC. The description of the remaining classes is

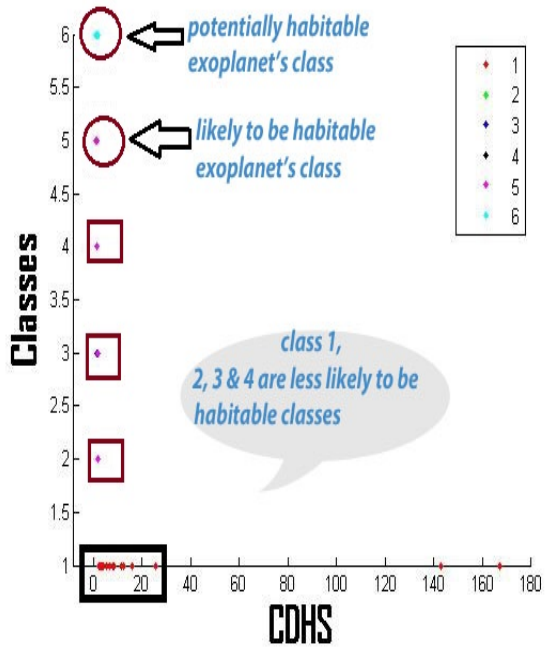
the same as in Figs. 3a and 3b.

**Table 34:** The outcome of K-NN on normalized dataset: potentially habitable exoplanets in Class 6 (Earth-League).

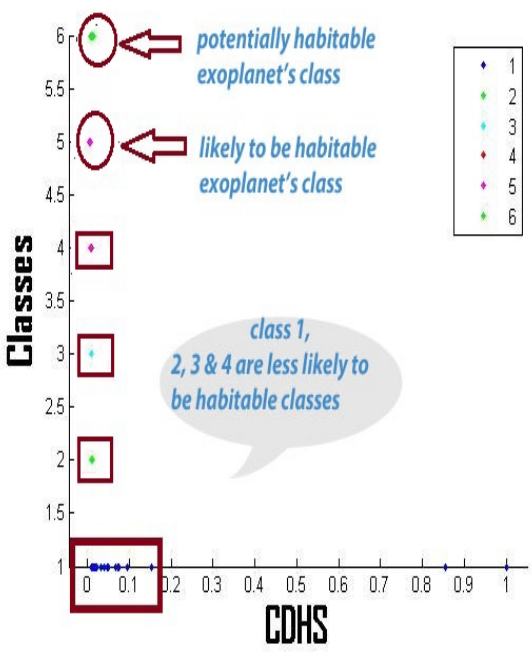
| Exoplanet | DRSnormCDHS | CRSnormCDHS |
|---|---|---|
| GJ 667C e | 0.007833698 | 0.004294092 |
| GJ 667C f | 0.007834698 | 0.004294642 |
| GJ 832 c | 0.009226084 | 0.005153791 |
| HD 40307 g | 0.009841607 | 0.005538682 |
| Kapteyn's b | 0.008980084 | 0.00500124 |
| Kepler-22 b | 0.01243731 | 0.007181929 |
| Kepler-61 b | 0.011438662 | 0.006546287 |
| Kepler-62 e | 0.008842245 | 0.004915399 |
| Kepler-62 f | 0.007887122 | 0.004326487 |
| Kepler-174 d | 0.011588827 | 0.006641471 |
| Kepler-186 f | 0.006442599 | 0.003450367 |
| Kepler-283 c | 0.009799112 | 0.005511735 |
| Kepler-296 f | 0.009704721 | 0.005452561 |
| Kepler-298 d | 0.013193284 | 0.007666263 |
| GJ 667C c | 0.007028218 | 0.00775173 |
| GJ 163 c | 0.022843579 | 0.005571684 |

As observed, the results of classification are almost similar for non-normalized (Figs. 3a & 3b) and normalized (Figs. 3c & 3d) CDHS. Both methods have identified the exoplanets that were previously assumed as potentially habitable (listed in the HEC database) with comparable accuracy. However, after normalization, the accuracy increases from 94% for $K = 1$ to above 99% for $K = 7$. All our results for confirmed exoplanets from PHL-EC, including DRS and CRS habitability CDHS scores and classes assignations, are presented in the catalog at `https://habitabilitypes.wordpress.com/`. CRS gave better results compared to DRS case in the non-normalized dataset, therefore, the final habitability score is considered to be the CDHS obtained in the CRS phase.

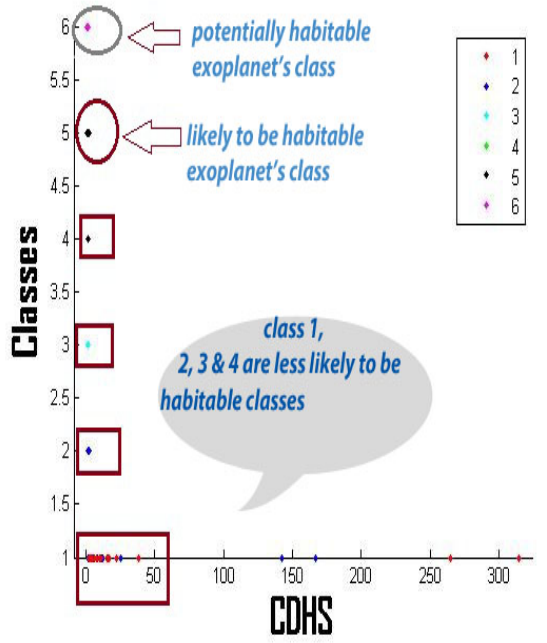
**Remark:** Normalized and non-normalized CDHS are obtained by two different methods. After applying the K-NN on the non-normalized CDHS, the method produced 12 and 14 habitable exoplanets in CRS and DRS cases, respectively, from a list of 664 exoplanets. The "Earth-League", Class 6, is the class where the algorithm "dumps" those exoplanets which satisfy the conditions of K-NN and threshold and probabilistic herding as explained in Sections 3.1, 3.2 and 3.3. We applied this algorithm again to the normalized CDHS of 664 exoplanets under the same conditions. It is observed that the output was 16 exoplanets that



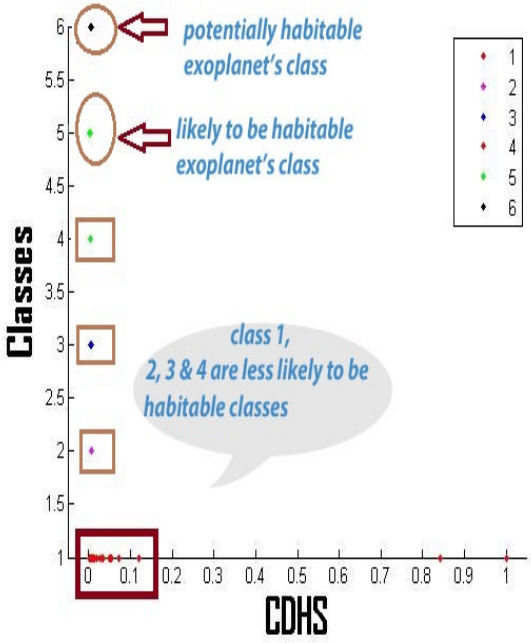

**(a)** for DRS on non-normalized data set




**(c)** for DRS on normalized data set




**(b)** for CRS on non-normalized data set




**(d)** for CRS on normalized data set

**Figure 14:** Results of attribute enhanced K-NN algorithm. The $X$-axis represents the Cobb-Douglas habitability score and $Y$-axis – the 6 classes: schematic representation of the outcome of our algorithm. The points in circles and boxes indicate membership in respective classes. These points are representative of membership only and do not indicate a quantitative equivalence of the exact representation. Full catalog is available at our website `https://habitabilitypes.wordpress.com/`

satisfied the conditions of being in Class 6, the "Earth-league", irrespective of CRS or DRS conditions. The reason is that the normalized scores are tighter and much closer to each other compared to the non-normalized CDHS, and that yielded a few more exoplanets in Class 6.

ESI is a metric that tells us whether an exoplanet is similar to Earth in some parameters. However, it may have nothing to do with habitability, and a planet with an ESI of 0.5 can be as habitable as a planet with an ESI of 0.99, since essentially only three Earth comparison points enter the ESI index: mass, radius and surface temperature (both density and escape velocity are calculated from mass and radius). Another metric, PHI, also cannot be used as a single benchmark for habitability since many other physical conditions have to be checked before a conclusion is drawn, such as existence of a magnetic field as a protector of all known forms of life, or stellar host variability, among others. Our proposed novel method of computing habitability by CD-HPF and CDHS, coupled with K-NN with probabilistic herding, estimates the habitability index of exoplanets in a statistically robust way, where optimization method is used for calculation. K-NN algorithm has been modified as an attribute-enhanced voting scheme, and the probabilistic proximity is used as a checkpoint for final class distribution. For large enough data samples, there are theoretical guarantees that the algorithm will converge to a finite number of discriminating classes. The members of the “Earth-League" are cross-validated with the list of potentially habitable exoplanets in the HEC database. The results (Table 9) render the proposed metric CDHS to behave with a reasonable degree of reliability.

Currently existing habitability indices ESI and PHI are restricted to only few parameters. At any rate, any one single benchmark for habitability may sound ambitious at present state of the field, given also the perpetual complexity of the problem. It is possible that developing the metric flexible enough to include any finite number of other planetary parameters, such as, e.g. orbital period, eccentricity, planetary rotation, magnetic field etc. may help in singling out the planets with large enough probability of potential habitability to concentrate the observational efforts. This is where the CD-HPF model has an advantage. The model generates 12 potentially habitable exoplanets in Class 6, which is considered to be a class where Earth-like planets reside. We have added several non-rocky samples to the dataset so that we could validate the algorithm. In machine learning, such random samples are usually used to check for the robustness of the designed algorithm. For example, if a non-rocky planet were classified by our algorithm as a member of the Earth-class, it would mean that the algorithm (and model) is wrong. However, it has not happened, and all the results of the Earth-league were verified to be rocky and potentially habitable. All these 12 exoplanets are identified as potentially habitable by the PHL.

The score generated by our model is a single metric which could be used to classify habitability of exoplanets as members of the "Earth League", unlike ESI and PHI. Attribute-enhanced K-NN algorithm, implemented in the paper, helps achieve this goal and the assignment of exoplanets to different classes of habitability may change as the input parameters of Cobb-Douglas model change values.

We would like to note that throughout the paper we equate habitability with Earth-likeness. We are searching for life as we know it (as we do not know any other), hence, the concept of an HZ and the “follow the water" directive. It is quite possible that this concept of habitability is too anthropocentric, and can be challenged, but not at present when we have not yet found any extraterrestrial life. At least, being anthropocentric allows us to know exactly what we can expect as habitable conditions, to know what we are looking for (e.g. biomarkers). In this process, we certainly will come across “exotic” and unexpected finds, but the start has to be anthropocentric.

## 4.11 Conclusion and Future Work

CD-HPF is a novel metric of defining habitability score for exoplanets. It needs to be noted that the authors perceive habitability as a probabilistic measure, or a measure with varying degrees of certainty. Therefore, the construction of different classes of habitability 1 to 6 is contemplated, corresponding to measures as “most likely to be habitable" as Class 6, to “least likely to be habitable" as Class 1. As a further illustration, classes 6 and 5 seem to represent the identical patterns in habitability, but they do not! Class 6 – the "Earth-League", is different from Class 5 in the sense that it satisfies the additional conditions of thresholding and probabilistic herding and, therefore, ranks higher on the habitability score. This is in stark contrast to the binary definition of exoplanets being “habitable or non-habitable", and a deterministic perception of the problem itself. The approach therefore required classification methods that are part of machine learning techniques and convex optimization — a sub-domain, strongly coupled with machine learning. Cobb-Douglas function and CDHS are used to determine habitability and the maximum habitability score of all exoplanets with confirmed surface temperatures in the PHL-EC. Global maxima is calculated theoretically and algorithmically for each exoplanet, exploiting intrinsic concavity of CD-HPF and ensuring "no curvature violation". Computed scores are fed to the attribute enhanced K-NN algorithm — a novel classification method, used to classify the planets into different classes to determine how similar an exoplanet is to Earth. The authors would like to emphasize that, by using classical K-NN algorithm and not exploiting the probability of habitability

criteria, the results obtained were pretty good, having 12 confirmed potentially habitable exoplanets in the "Earth-League". We have created a web page for this project to host all relevant data and results: sets, figures, animation video and a graphical abstract. It is available at `https://habitabilitypes.wordpress.com/`. This page contains the full customized catalog of all confirmed exoplanets with class annotations and computed habitability scores. This catalog is built with the intention of further use in designing statistical experiments for the analysis of the correlation between habitability and the abundance of elements (this work is briefly outlined in Safonova et al., 2016). It is a very important observation that our algorithm and method give rise to a score metric, CDHS, which is structurally similar to the PHI as a corollary in the CRS case (when the elasticities are assumed to be equal to each other). Both are the geometric means of the input parameters considered for the respective models.

CD-HPF uses four parameters (radius, density, escape velocity and surface temperature) to compute habitability score, which by themselves are not sufficient to determine habitability of exoplanets. Other parameters, such as e.g. orbital period, stellar flux, distance of the planet from host star, etc. may be equally important to determine the habitability. Since our model is scalable, additional parameters can be added to achieve better and granular habitability score. In addition, out of all confirmed exoplanets in PHL-EC, only about half have their surface temperatures estimated. For many expolanets, the surface temperature, which is an important parameter in this problem, is not known or not defined. The unknown surface temperatures can be estimated using various statistical models. Future work may include incorporating more input parameters, such as orbital velocity, orbital eccentricity, etc. to the Cobb-Douglas function, coupled with tweaking the attribute enhanced K-NN algorithm by checking an additional condition such as, e.g. distance to the host star. Cobb-Douglas, as proved, is a scalable model and doesn't violate curvature with additional predictor variables. However, it is pertinent to check for the dominant parameters that contribute more towards the habitability score. This can be accomplished by computing percentage contributions to the response variable – the habitability score. We would like to conclude by stressing on the efficacy of the method of using a few of the parameters rather than sweeping through a host of properties listed in the catalogs, effectively reducing the dimensionality of the problem. To sum up, CD-HPF and CDHS turn out to be self-contained metrics for habitability.

# 5 THEORETICAL VALIDATION OF POTENTIAL HABITABILITY VIA ANALYTICAL AND BOOSTED TREE METHODS: AN OPTIMISTIC STUDY ON RECENTLY DISCOVERED EXOPLANETS

## 5.1 Introduction

With discoveries of exoplanets pouring in hundreds, it is becoming necessary to develop some sort of a quick screening tool – a ranking scale – for evaluating habitability perspectives for the follow-up targets. One such scheme was proposed recently by us – the Cobb-Douglas Habitability Score (CDHS; [Bora et al.2016]). While our paper "CD-HPF: New Habitability Score Via Data Analytic Modeling" was in production, the discovery of an exoplanet Proxima b orbiting the nearest star (Proxima Centauri) to the Sun was announced [Anglada-EscudÃl'2016]. This planet generated a lot of stir in the news [Witze2016] because it is located in the habitable zone and its mass is in the Earth's mass range: $1.27-3\ M_{\oplus}$, making it a potentially habitable planet (PHP) and an immediate destination for the Breakthrough Starshot initiative [Starshot].

This work is motivated by testing the efficacy of the suggested model, CDHS, in determining the habitability score of Proxima b. The habitability score model has been found to work well in classifying exoplanets in terms of potential habitability. Therefore it was natural to test whether the model can also classify Proxima b as potentially habitable by computing its habitability score. This could indicate whether the model may be extended for a quick check of the potential habitability of newly discovered exoplanets in general. As we discover in ***Section VI***, this is indeed the case with the newly announced TRAPPIST-1 system [Trappist-1].

CDHS does encounter problems commonly found in convex functional modeling, such as scalability and curvature violation. Scalability is defined as the condition on the global maximum of the function; the global maximum is adjusted as the number of parameters entering the function (elasticity) increases, i.e if a global maximum is ensured for $n$ parameters, it will continue to hold for $n+1$ parameters. The flowchart in Fig. 1 summarizes our approach to the habitability investigation of Proxima b. A novel inductive approach inspired by the Cobb-Douglas model from production economics [cobb-douglas] was proposed to verify theoretical conditions of global optima of the functional form used to model and compute the habitability score of exoplanets in [1]. The outcome of classification of exoplanets based on the score (Method 1) is then tallied with another classification method which discriminates samples (exoplanets) into classes based on the features/attributes of the samples (Method 2).

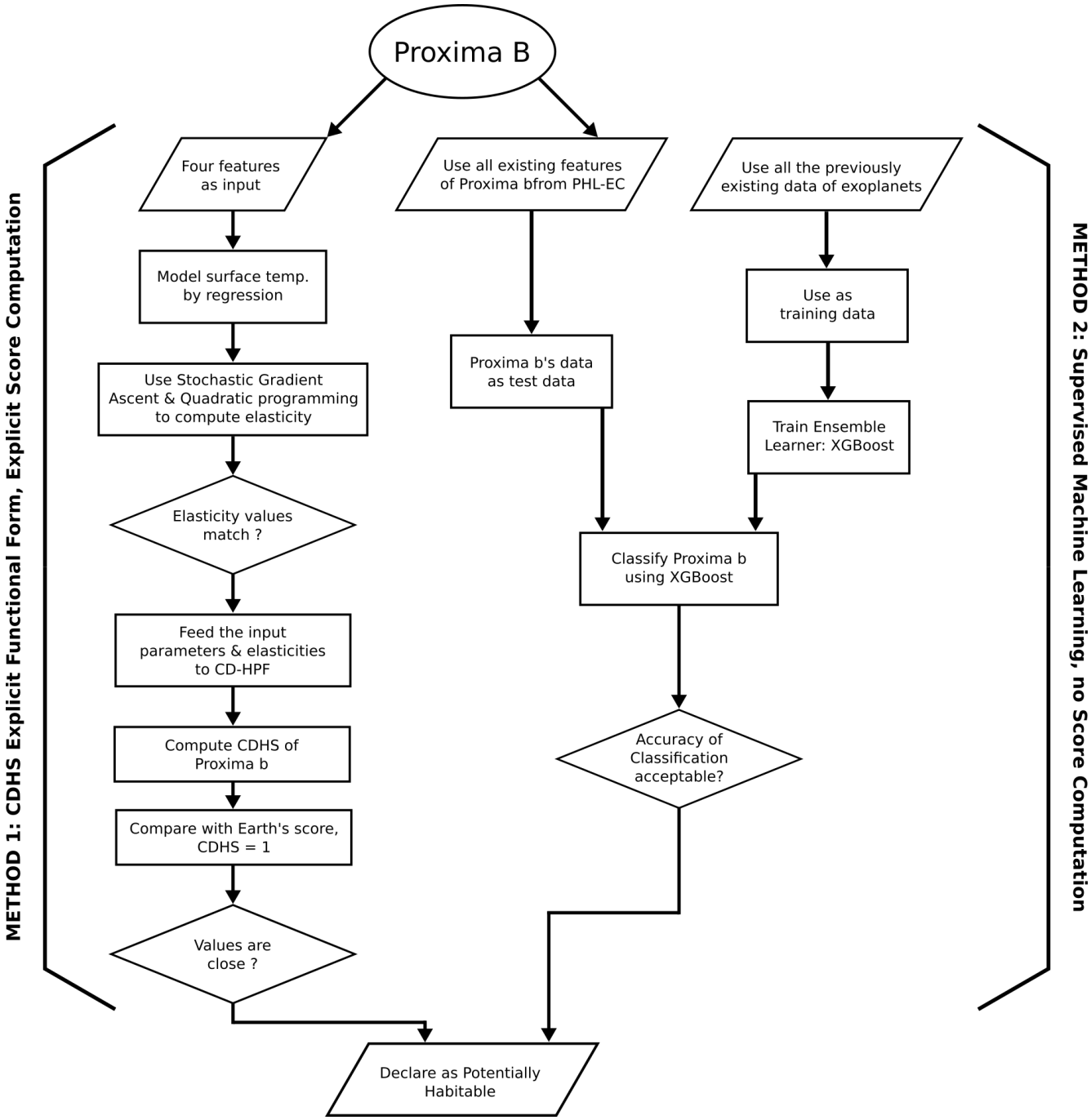



**Figure 15:** The convergence of two different approaches in investigation of the potential habitability of Proxima b. The outcome of the explicit scoring scheme for Proxima b places it in the “Earth-League", which is synonymous to being classified as potentially habitable.

The similar outcome from both approaches (the exoplanets are classified in the same habitability class), markedly different in structure and methodology, fortifies the growing advocacy of using machine learning in astronomy.

The habitability score model considers four parameters/features, namely mass, radius, density and surface temperature of a planet, extracted from the PHL-EC (Exoplanet Catalog hosted by the Planetary Habitability Laboratory (PHL); http://phl.upr.edu/projects). Though the catalog contains 68 observed and derived stellar and planetary parameters, we have currently considered only four for the CDHS model. However, we show here that the CDHS model is scalable, i.e. capable of accommodating more parameters (see Section IV on model scalability, and Appendix I for the proof of the theorem). Therefore, we may use more parameters in future to compute the CDHS. The problem of curvature violation in tackled in Sec. II.A later in the paper.

PHL classifies all discovered exoplanets into five categories based on their thermal characteristics: non-habitable, and potentially habitable: psychroplanet, mesoplanet, thermoplanet and hypopsychroplanet. Proxima b is one of the recent additions to the catalog with recorded features. Here, we employ a non-metric classifier to predict the class label of Proxima b. We compute the accuracy of our classification method, and aim to reconcile the result with the habitabilty score of Proxima b, which may suggest its proximity to "Earth-League". We call this an investigation in the optimistic determination of habitabilty. The hypothesis is the following: a machine learning-based classification method, known as boosted trees, classifies exoplanets and returns some with the class by mining the features present in the PHL-EC (Method 2 in Fig. 1). This process is independent of computing an explicit habitability score for Proxima b (aka Method 1 in Fig. 1), and indicates habitability class by learning attributes from the catalog. This implicit method should match the outcome suggested by the CDHS, i.e. that Proxima b score should be close to the Earth's CDHS habitability score (with good precision), computed explicitly.

The second approach is based on XGBoost – a statistical machine-learning classification method used for supervised learning problems, where the training data with multiple features is used to predict a target variable. Authors intend to test whether the two different approaches to investigate the habitability of Proxima b, analytical and statistical, converge with a reasonable degree of confidence.

## 5.2 Analytical Approach via CDHS: Explicit Score Computation of Proxima b

We begin by discussing the key elements of the analytical approach. The parameters of the planet (Entry #3389 in the dataset) for this purpose were extracted from the PHL-EC: minimal mass 1.27 EU, radius 1.12 EU, density 0.9 EU, surface temperature 262.1 K, and escape velocity 1.06 EU, where EU is the Earth Units. The Earth Similarity Index (ESI) for this new planet, estimated using a simplified version of ESI[4], is 0.87. By definition, ESI ranges from 0 (totally dissimilar to Earth) to 1 (identical to Earth), and a planet with ESI ≥ 0.8 is considered an Earth-like.

### 5.2.1 Earth Similarity Index

In general, the ESI value of any exoplanet's planetary property is calculated using the following expression [Schulze-Makuch et al.2011],

$$ESI_x = \left(1 - \left|\frac{x - x_0}{x + x_0}\right|\right)^{w_x}, \tag{46}$$

where $x$ is a planetary property – radius, surface temperature, density, or escape velocity, $x_0$ is the Earth's reference value for that parameter – 1 EU, 288 K, 1 EU and 1 EU, respectively, and $w_x$ is the weighted exponent for that parameter. After calculating ESI for each parameter by Eq. (1), the global ESI is found by taking the geometric mean (G.M.) of all four $ESI_x$,

$$ESI = \left(\prod_{x=1}^{n} ESI_x\right)^{\frac{1}{n}}. \tag{47}$$

The problem in using Eq. (2) to obtain the global ESI is that sometimes there no available data to obtain all input parameters, such as in the case of Proxima b – only its mass and the distance from the star are known. Due to that, a simplified expression was proposed by the PHL for ESI calculation in terms of only radius and stellar flux,

$$ESI = 1 - \sqrt{\frac{1}{2}\left(\frac{R - R_0}{R + R_0}\right)^2 + \left(\frac{S - S_0}{S + S_0}\right)^2}, \tag{48}$$

where $R$ and $S$ represent radius and stellar flux of a planet, and $R_0$ and $S_0$ are the reference values for the Earth. Using 1.12 EU for the radius and 0.700522 EU for the stellar flux, we

[4] http://phl.upr.edu/projects/earth-similarity-index-esi

obtain ESI = 0.8692. It is worth mentioning that once we know one observable – the mass – other planetary parameters used in the ESI computation (radius, density and escape velocity) can be calculate based on certain assumptions. For example, the small mass of Proxima b suggests a rocky composition. However, since 1.27 EU is only a low limit on mass, it is still possible that its radius exceeds 1.5 – 1.6 EU, which would make Proxima b not rocky [rogers2014]. In the PHL-EC, its radius is estimated using the mass-radius relationship -

$$R = \begin{cases} M^{0.3} & M \leq 1 \\ M^{0.5} & 1 \leq M < 200 \\ (22.6)\, M^{(-0.0886)} & M \geq 200 \end{cases} \tag{49}$$

Since Proxima b mass is 1.27 EU, the radius is $R = M^{0.5} \equiv 1.12$ EU. Accordingly, the escape velocity was calculated by $V_e = \sqrt{2GM/R} \equiv 1.065$ (EU), and the density by the usual $D = 3M/4\pi R^3 \equiv 0.904$ (EU) formula. If we use all four parameters provided in the catalog, the global ESI becomes 0.9088.

### 5.2.2 Cobb Douglas Habitability Score (CDHS)

We have proposed the new model of the habitability score in [Bora et al.2016] using a convex optimization approach [Saha et al.2016]. In this model, the Cobb Douglas function is reformulated as Cobb-Douglas habitability production function (CD-HPF) to compute the habitability score of an exoplanet,

$$\mathbb{Y} = f(R, D, T_s, V_e) = (R)^{\alpha} \cdot (D)^{\beta} \cdot (T_s)^{\gamma} \cdot (V_e)^{\delta}\,, \tag{50}$$

where the same planetary parameters are used – radius $R$, density $D$, surface temperature $T_s$ and escape velocity $V_e$. $\mathbb{Y}$ is the habitability score CDHS, and $f$ is defined as CD-HPF. The goal is to maximize the score, $\mathbb{Y}$, where the elasticity values are subject to the condition $\alpha + \beta + \gamma + \delta < 1$. These values are obtained by a computationally fast algorithm Stochastic Gradient Ascent (SGA) described in Section 3. We calculate CDHS score for the constraints known as returns to scale: Constant Return to Scale (CRS) and Decreasing Return to Scale (DRS) cases; for more details please refer to [Bora et al.2016].

As Proxima b is considered an Earth-like planet, we endeavored to cross-match the observation via the method explained in the previous section. The analysis of CDHS will help to explore how this method can be effectively used for newly discovered planets. The eventual classification of any exoplanet is accomplished by using the proximity of CDHS

of that planet to the Earth, with additional constraints imposed on the algorithm termed "probabilistic herding". The algorithm works by taking a set of values in the neighborhood of 1 (CDHS of Earth). A threshold of 1 implies that CDHS value between 1 and 2 is acceptable for membership in "Earth-League", pending fulfillment of further conditions. For example, the CDHS of the most potentially habitable planet before Proxima b, Kepler-186 f, is 1.086 (the closest to the Earth's value), though its ESI is only 0.64. While another PHP GJ-163 c has the farthest score (1.754) from 1; and though its ESI is 0.72, it may not be even a rocky planet as its radius can be between 1.8 to 2.4 EU, which is not good for a rocky composition theory, see e.g. [rogers2014].

### 5.2.3 CDHS calculation using radius, density, escape velocity and surface temperature

Using the estimated values of the parameters from the PHL-EC, we calculated CDHS score for the CRS and DRS cases, and obtained optimal elasticity and maximum CDHS value. The CDHS values in CRS and DRS cases were 1.083 and 1.095, respectively. The degree/extent of closeness is explained in [Bora et al.2016] in great detail.

**Table 35:** Rocky planets with unknown surface temperature:*Oversampling, attribute mining and using association rules for missing value imputation: cf. subsection 1*

| *P.Name* | *P.Composition Class* |
|---|---|
| Kepler-132 e | rocky-iron |
| Kepler-157 d | rocky-iron |
| Kepler-166 d | rocky-iron |
| Kepler-176 e | rocky-iron |
| Kepler-192 d | rocky-iron |
| Kepler-217 d | rocky-iron |
| Kepler-271 d | rocky-iron |
| Kepler-398 d | rocky-iron |
| Kepler-401 d | rocky-iron |
| Kepler-403 d | rocky-iron |
| WD 1145+017 b | rocky-iron |

### 5.2.4 Missing attribute values: Surface Temperature of 11 rocky planets (Table I)

We observed missing values of surface temperature in Table I. The values of equilibrium temperature of those entries are also unknown. Imputation of missing values is commonly done by filling in the blanks by computing the mean of continuous valued variables in the same column, using other entries of the same type, rocky planets in this case. However, this

method has demerits. We propose the following method to achieve the task of imputing missing surface temperature values.

**Data imputation using Association Rules:** A more sophisticated method of data imputation is that of *rule based learning*. Popularized by Agrawal et al. [Agrawal1993] through their seminal paper in 1993, it is a robust approach in Data Mining and Big Data Analytics for the purpose of filling in missing values. The original approach was inspired by unexpected correlations in items being purchased by customers in markets. An illustrative example making use of samples and features from the PHL-EC dataset is presented here.

Any dataset has samples and features. Say, we have n samples $S = \{s_1, s_2, ..., s_n\}$ and $m$ features, $X = \{X_1, X_2, ..., X_m\}$, such that each sample is considered to be a $1 \times m$ vector of features, $s_i = \{x_{i1}, x_{i2}, ..., x_{im}\}$. Here, we would like to find out if the presence of any feature set $A$ amongst all the samples in $S$ implies the presence of a feature set $B$.

**Table 36:** Table of features used to construct the association rule for missing value imputation

| *P.Name* | *P.Zone Class* | *P.Mass Class* | *P.Composition Class* | *P.Atmosphere Class* | Class Label |
|---|---|---|---|---|---|
| 8 Umi b | Hot | Jovian | gas | hydrogen-rich | non-habitable |
| GJ 163 c | Warm | Superterran | rocky-iron | metals-rich | psychroplanet |
| GJ 180 b | Warm | Superterran | rocky-iron | metals-rich | mesoplanet |
| GJ 180 c | Warm | Superterran | rocky-iron | metals-rich | mesoplanet |
| 14 Her b | Cold | Jovian | gas | hydrogen-rich | non-habitable |

Consider Table 36. An interesting observation is that all the planets with *P.Zone Class* = *Warm, P.Mass Class* = *Superterrain* and *P. Composition Class* = *rocky-iron* (the planets GJ 163 c, GJ 180 b and GJ 180 c) also have *P. Atmosphere Class* as *metals-rich.* Here, if we consider conditions $A$ = *{P.Zone Class = Warm, P.Mass Class = Superterran, P.Composition Class = rocky-iron}* and $B$ = *{P.Atmosphere Class = metals-rich}*, then $A \Rightarrow B$ holds true. But what does it mean in for data imputation? If there exists a sample $s_k$ in the dataset such that condition A holds good for $s_k$ but the value of *P.Atmosphere Class* is missing, then by the association rule $A \Rightarrow B$, we can impute the value of *P.Atmosphere Class* for $s_k$ as metals rich. Similarly, if $A'$ = *{P.Mass Class = Jovian, P.Composition Class = gas}* and $B'$ = *{P.Atmosphere Class = hydrogen-rich}*, then $A' \Rightarrow B'$ becomes another association rule which may be used to impute vales of *P.Atmosphere Class.* Note here the exclusion of the variable *P.Zone Class.* In the two samples which satisfy $A'$, the value of *P.Zone Class* are not the same and hence they do not make for a strong association with $B'$.

In Table 36, we have mentioned the class labels alongside the samples. However this is just indicative; the class labels should not be used to form associations (if they are used, then

some resulting associations might become similar to a traditional classification problem!) Different metrics are used to judge how interesting a rule is, i.e., the goodness of a rule. Two of the fundamental metrics are:

1. **Support**: It is an indicator of how frequently a condition A appears in the database. Here, $t$ is the set of samples in $S$ which exhibit the condition $A$.

$$supp(A) = \frac{|t \in S; A \subseteq t|}{|S|} \tag{51}$$

   In the example considered, $A$ = *{P.Zone Class = Warm, P.Mass Class = Superterran, P.Composition Class = rocky-iron}* has a support of $3/5 = 0.6$.

2. **Confidence**: It is an indication of how often the rule was found to be true. For the rule $A \Rightarrow B$ in S, the confidence is defined as:

$$conf(A \Rightarrow B) = \frac{supp(A \cup B)}{supp(A)} \tag{52}$$

   For example, the rule $A \Rightarrow B$ considered in our example has a confidence of $0.6/0.6 = 1$, which means 100% of the samples satisfying $A$ = *{P.Zone Class = Warm, P.Mass Class = Superterran, P.Composition Class = rocky-iron}* will also satisfy $B$ = *{P.Atmosphere Class = metals-rich}*.

Association rules must satisfy thresholds set for support and confidence in order to be accepted as rules for data imputation. The example illustrated is a very simple one. In practice, association rules need to be considered over thousands or millions of samples. From one dataset, millions of association rules may arise. Hence, the support and confidence thresholds must be carefully considered. The example makes use of only categorical variables for the sake of simplicity. However, association rules may be determined for continuous variables by considering bins of values. Algorithms exist that are used for discovering association rules, amongst which *a priori* [Agrawal 1994] is the most popular.

In the original text, the features considered here are called *items* and each sample is called a *transaction*.

### 5.2.5 CDHS calculation using stellar flux and radius

Following the simplified version of the ESI on the PHL website, we repeated the CDHS computation using only radius and stellar flux (1.12 EU and 0.700522 EU, respectively). Using

the scaled down version of Eq. (5), we obtain $\text{CDHS}_{DRS}$ and $\text{CDHS}_{CRS}$ as 1.168 and 1.196, respectively. These values confirm the robustness of the method used to compute CDHS and validate the claim that Proxima b falls into the "Earth-League" category.

### 5.2.6 CDHS calculation using stellar flux and mass

The habitability score requires the use of available physical parameters, such as radius, or mass, and temperature, and the number of parameters is not extremely restrictive. As long as we have the measure of the interior similarity – the extent to which a planet has a rocky interior, and exterior similarity – the location in the HZ, or the favorable range of surface temperatures, we can reduce the number of parameters (or increase). Since radius is calculated from an observable parameter – mass, we decided to use the mass directly in the calculation. We obtained $\text{CDHS}_{DRS}$ as 1.168 and $\text{CDHS}_{CRS}$ as 1.196. The CDHS achieved using radius and stellar flux (Section 2.3) and the CDHS achieved using mass and stellar flux have the same values.

***Remark:*** *Does this imply that the surface temperature and radius are enough to compute the habitability score as defined by our model? It cannot be confirmed until enough number of clean data samples are obtained containing the four parameters used in the original ESI and CDHS formulation. We plan to perform a full-scale dimensionality analysis as future work*

The values of ESI and CDHS using different methods as discussed above are summarized in Table 37.

**Table 37:** ESI and CDHS values calculated for different parameters

| Parameters Used | ESI | $\text{CDHS}_{CRS}$ | $\text{CDHS}_{DRS}$ |
|---|---|---|---|
| $R$, $D$, $T_s$, $V_e$ | 0.9088 | 1.083 | 1.095 |
| Stellar Flux, $R$ | 0.869 | 1.196 | 1.168 |
| Stellar Flux, $M$ | 0.849 | 1.196 | 1.167 |

NOTE: The nicety in the result, i.e. little difference in the values of CDHS, is due to the flexibility of the functional form in the model proposed in [Ginde2016], and the computation of the elasticities by the Stochastic Gradient Ascent method. Using this method led to the fast convergence of the elasticities $\alpha$ and $\beta$. Proxima b passed the scrutiny and is classified as a member of the "Earth-league".

## 5.3 Elasticity computation: Stochastic Gradient Ascent (SGA)

[Bora et al.2016] used a library function **fmincon** to compute the elasticity values. Here, we have implemented a more efficient algorithm to perform the same task. This was done for two reasons: to be able to break free from the in-built library functions, and to devise a sensitive method which would mitigate oscillatory nature of Newton-like methods around the local minima/maxima. There are many methods which use gradient search including the one proposed by Newton. Although theoretically sound, algorithmic implementations of most of these methods faces convergence issues in real time due to the oscillatory nature. Stochastic Gradient Descent was used to find the minimal value of a multivariate function, when the input parameters are known. We tried to identify the elasticities for mass, radius, density and escape velocity. We do this separately for interior CDHS and surface CDHS, and use a convex combination to compute the final CDHS (for detail, see [Bora et al.2016]) for which the maximum value is attained under certain constraints. Our objective is to maximize the final CDHS. We have employed a modified version of the descent, a Stochastic Gradient Ascent algorithm, to calculate the optimum CDHS and the elasticity values $\alpha$, $\beta$, etc. As opposed to the conventional Gradient Ascent/Descent method, where the gradient is computed only once, stochastic version recomputes the gradient for each iteration and updates the elasticity values. Theoretical convergence, guaranteed otherwise in the conventional method, is though sometimes slow to achieve. Stochastic variant of the method speeds up the convergence, justifying its use in the context of the problem (the size of data, i.e. the number of discovered exoplanets, is increasing every day).

Output elasticity of Cobb-Douglas habitability function is the accentual change in the output in response to a change in the levels any of the inputs. $\alpha$ and $\beta$ are the output elasticity of density and radius respectively. Accuracy of $\alpha$ and $\beta$ values is crucial in deciding the right combination for the optimal CDHS, where different approaches are analyzed before arriving at final decision.

### 5.3.1 Computing Elasticity via Gradient Ascent

Gradient Ascent algorithm is used to find the values of $\alpha$ and $\beta$. Gradient Ascent is an optimization algorithm used for finding the local maximum of a function. Given a scalar function $F(x)$, gradient ascent finds the $\max_x F(x)$ by following the slope of the function. This algorithm selects initial values for the parameter $x$ and iterates to find the new values of $x$ which maximizes $F(x)$ (here CDHS). Maximum of a function $F(x)$ is computed by iterating

through the following step,

$$x_{n+1} \leftarrow x_n + \chi \frac{\partial F}{\partial x}, \tag{53}$$

where $x_n$ is an initial value of $x$, $x_{n+1}$ the new value of $x$, $\frac{\partial F}{\partial x}$ is the slope of function $Y = F(x)$ and $\chi$ denotes the step size, **which is greater than** 0 **and forces the algorithm make a small jump (descent or ascent algorithms are trained to make small jumps in the direction of the new update)** . Note that the interior $\text{CDHS}_i$, denoted by $Y1$, is calculated using radius $R$ and density $D$, while the surface $\text{CDHS}_s$, denoted by $Y2$, is calculated using surface temperature $T_s$ and escape velocity $V_e$. The objective is to find elasticity value that produces the optimal habitability score for the exoplanet, i.e. to find $Y_1 = \max_{\alpha,\beta} Y(R,D)$ such that, $\alpha > 0$, $\beta > 0$ and $\alpha + \beta \leq 1$. (Please note that $\alpha + \beta < 1$ is the DRS condition for elasticity which may be scaled to $\alpha_1 + \alpha_2 + \ldots + \alpha_n < 1$). Similarly, we need to find $Y_2 = \max_{\gamma,\delta} Y(T, V_e)$ such that $\gamma > 0$, $\delta > 0$ and $\delta + \gamma \leq 1$. (Analogously, $\delta + \gamma < 1$ is the DRS condition for elasticity which may be scaled to $\delta_1 + \delta_2 + \ldots + \delta_n < 1$).

Stochastic variant thus mitigates the oscillating nature of the global optima – a frequent malaise in the conventional Gradient Ascent/Descent and Newton-like methods, such as **fmincon** used in [1]. At this point of time, without further evidence of recorded/measured parameters, it may not be prudent to scale up the CD-HPF model by including more parameters other than the ones used by either ESI or our model. But if it ever becomes a necessity (to utilize more than the four parameters), the algorithm will come in handy and multiple optimal elasticity values may be computed fairly easily.

### 5.3.2 Computing Elasticity via Constrained Optimization

Let the assumed parametric form be $\log(y) = \log(K) + \alpha \log(S) + \beta \log(P)$. Consider a set of data points,

$$\begin{array}{ccccccc} \ln(y_1) & = & K' & + & \alpha S'_1 & + & \beta P'_1 \\ \vdots & & \vdots & & \vdots & & \vdots \\ \ln(y_N) & = & K' & + & \alpha S'_N & + & \beta P'_N \end{array} \tag{54}$$

where

$$\begin{aligned} K' &= \log(K), \\ S'_i &= \log(S'_i), \\ P'_i &= \log(P'_i). \end{aligned}$$

If $N > 3$, this is an over-determined system, where one possibility to solve it is to apply a least squares method. Additionally, if there are constraints on the variables (the parameters to be solved for), this can be posed as a constrained optimization problem. These two cases are discussed below.

**No constraints:** This is an ordinary least squares solution. The system is in the form $y = Ax$, where

$$x = \begin{bmatrix} K' & \alpha & \beta \end{bmatrix}^T, \tag{55}$$

$$y = \begin{bmatrix} y_1 \\ . \\ . \\ y_N \end{bmatrix} \tag{56}$$

and

$$A = \begin{bmatrix} 1 & S'_1 & P'_1 \\ & \dots & \\ 1 & S'_N & P'_N \end{bmatrix}. \tag{57}$$

The least squares solution for $x$ is the solution that minimizes

$$(y - Ax)^T (y - Ax). \tag{58}$$

It is well known that the least squares solution to Eq. (54) is the solution to the system

$$A^T y = A^T A x, \tag{59}$$

i.e.

$$x = (A^T A)^{-1} A^T y. \tag{60}$$

In *Matlab*, the least squares solution to the overdetermined system $y = Ax$ can be obtained by $x = A \backslash y$. Table II presents the results of least squares (**No constraints**) obtained for the elasticity values after performing the least square fitting, while Table III displays the results obtained for the elasticity values after performing the constrained least square fitting.

**Table 38:** Elasticity values for IRS, CRS & DRS cases after performing the least square test (**No constraints**): elasticity values $\alpha$ and $\beta$ satisfy the theorem $\alpha+\beta<1, \alpha+\beta=1$, and $\alpha+\beta>1$ for DRS, CRS and IRS, respectively, and match the values reported previously [Bora et al.2016].

| | IRS | CRS | DRS |
|---|---|---|---|
| $\alpha$ | 1.799998 | 0.900000 | 0.799998 |
| $\beta$ | 0.100001 | 0.100000 | 0.099999 |

**Constraints on parameters:** this results in a constrained optimization problem. The objective function to be minimized (maximized) is still the same, namely,

$$(y-Ax)^T(y-Ax). \tag{61}$$

This is a quadratic form in $x$. If the constraints are linear in $x$, then the resulting constrained optimization problem is a quadratic program (QP). A standard form of a QP is

$$\max x^T Hx + f^T x, \tag{62}$$

such that

Suppose the constraints are $\alpha, \beta > 0$ and $\alpha+\beta \leq 1$. The QP can be written as (neglecting the constant term $y^T y$)

$$\max x^T(A^T A)x - 2y^T Ax, \tag{63}$$

such that

$$\begin{aligned} &\alpha > 0, \\ &\beta > 0, \\ &\alpha+\beta \leq 1. \end{aligned} \tag{64}$$

For the standard form as given in Eq. (16), Eqs. (63)-(64) can be represented by rewriting the objective function as

$$x^T Hx + f^T x, \tag{65}$$

where

$$H = A^T A \text{ and } f = -2A^T y. \tag{66}$$

The inequality constraints can be specified as

$$C = \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 1 \end{bmatrix}, \tag{67}$$

and

$$b = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}. \tag{68}$$

# 6 COMPARING HABITABILITY METRICS

A sequence of recent explorations by Saha et al. [4] expanding previous work by Bora et al.[1] on using Machine Learning algorithm to construct and test planetary habitability functions with exoplanet data raises important questions. The 2018 paper analyzed the elasticity of their Cobb-Douglas Habitability Score (CDHS) and compared its performance with other machine learning algorithms. They demonstrated the robustness of their methods to identify potentially habitable planets from exoplanet dataset. Given our little knowledge on exoplanets and habitability, these results and methods provide one important step toward automatically identifying objects of interest from large datasets by future ground and space observatories. The paper provides a logical evolution from their previous work. Additionally, model and the methods yielding a new metric for "Earth-similarity" paves the way for comparison with its predecessor, ESI [Schulze-Makuch et al.2011]. Even though, CDHS proposed in [1] and extended in [4] consider identical planetary parameters such as Mass, Radius, Escape Velocity and Surface Temperature; the contrasts are overwhelming compared to the similarities. The note aims to bring out the differences and investigates the contrasts between the two metrics, both from machine learning and modeling perspectives.

It is worth mentioning that once we know one observable – the mass – other planetary parameters used in the ESI computation (radius, density and escape velocity) can be calculated based on certain assumptions. For example, the small mass of Proxima b suggests a rocky composition. However, since 1.27 EU is only a low limit on mass, it is still possible that its radius exceeds 1.5 – 1.6 EU, which would make Proxima b not rocky [rogers2014]. Since Proxima b mass is 1.27 EU, the radius is $R = M^{0.5} \equiv 1.12$ EU[5]. Accordingly, the escape velocity was calculated by $V_e = \sqrt{2GM/R} \equiv 1.065$ (EU), and the density by the usual $D = 3M/4\pi R^3 \equiv 0.904$ (EU) formula.

The manuscript, [4] consists of three related analyses: (i) computation and comparison of ESI and CDHS habitability scores for Proxima-b and the Trappist-1 system, (ii) some considerations on the computational methods for computing the CDHS score, and (iii) a machine learning exercise to estimate temperature-based habitability classes. The analysis is carefully conducted in each case, and the depth of the contribution to the literature helped unfold the differences and approaches to CDHS and ESI.

Several important characteristics were introduced to address the habitability question. [Schulze-Makuch et al.2011] first addressed this issue through two indices, the Planetary

[5]http://phl.upr.edu/library/notes/standardmass-radiusrelationforexoplanets, Standard Mass-Radius Relation for Exoplanets, Abel Mendez, June 30, 2012.

Habitability Index (PHI) and the Earth Similarity Index (ESI), where maximum, by definition, is set as 1 for the Earth, PHI=ESI=1.

ESI represents a quantitative measure with which to assess the similarity of a planet with the Earth on the basis of mass, size and temperature. But ESI alone is insufficient to conclude about the habitability, as planets like Mars have ESI close to 0.8 but we cannot still categorize it as habitable. There is also a possibility that a planet with ESI value slightly less than 1 may harbor life in some form which is not there on Earth, i.e. unknown to us. PHI was quantitatively defined as a measure of the ability of a planet to develop and sustain life. However, evaluating PHI values for large number of planets is not an easy task. In [Irwin et al.2014], another parameter was introduced to account for the chemical composition of exoplanets and some biology-related features such as substrate, energy, geophysics, temperature and age of the planet — the Biological Complexity Index (BCI). Here, we briefly describe the mathematical forms of these parameters.

## 6.1 Earth Similarity Index (ESI)

ESI was designed to indicate how Earth-like an exoplanet might be [Schulze-Makuch et al.2011] and is an important factor to initially assess the habitability measure. Its value lies between 0 (no similarity) and 1, where 1 is the reference value, i.e. the ESI value of the Earth, and a general rule is that any planetary body with an ESI over 0.8 can be considered an Earth-like. It was proposed in the form

$$ESI_x = \left(1 - \left|\frac{x - x_0}{x + x_0}\right|\right)^w, \tag{69}$$

where $\mathrm{ESI}_x$ is the ESI value of a planet for $x$ property, and $x_0$ is the Earth's value for that property. The final ESI value of the planet is obtained by combining the geometric means of individual values, where $w$ is the weighting component through which the sensitivity of scale is adjusted. Four parameters: surface temperature $T_s$, density $D$, escape velocity $V_e$ and radius $R$, are used in ESI calculation. This index is split into interior $ESI_i$ (calculated from radius and density), and surface $ESI_s$ (calculated from escape velocity and surface temperature). Their geometric means are taken to represent the final $ESI$ of a planet. However, ESI in the form (69) was not introduced to define habitability, it only describes the similarity to the Earth in regard to some planetary parameters. For example, it is relatively high for the Moon.

*What's the paradox?*

The ESI and CDHS scores should be similar in the sense of being computed from essentially the same ingredients. However, similarity in numerical values may be accidental also unless more informative picture could be provided with the additional exploration of the general

relationship between ESI and CDHS scores. We explore relationship and establish the contrasts in the following section. It will also be established that ESI and CDHS are not related and there exists no causal relationship between the two. We note that the only similarity is both ESI and CDHS use identical input parameters.
We have not argued that the CD-HPF is a superior metric. However, we do argue that the foundations of the CD-HPF, in both a mathematical and a philosophical sense, are solidly grounded, and substantiated by analytical proofs.

- The CDHS is not derived from ESI. CDHS is not a classifier and neither is ESI – this is because we do not (yet) categorize planets based on the values of either ESI or CDHS. Even though we were categorical in emphasizing the contrast and reconciliation of two approaches (one being CDHS), we reiterate our baseline argument for the benefit of the reviewer. CDHS contributes to the Earth-Similarity concepts where the scores have been used to classify exoplanets based on their degree of similarity to Earth. However Earth-Similarity is not equivalent to exoplanetary habitability. Therefore, we adopted another approach where machine classification algorithms have been exploited to classify exoplanets in to three classes, non-habitable, mesoplanets and psychroplanets. While this classification was performed, CDHS was not used at all, rather discriminating features from the PHL-EC were used. This is fundamentally different from CDHS based Earth-Similarity approach where explicit scores were computed. Therefore, it was pertinent and remarkable that the outcome of these two fundamentally distinct exercises reconcile. We achieved that. This reconciliation approach is the first of its kind and fortifies CDHS, more than anything else. Fig.1 captures this spirit and portrays the hallmark of the manuscript. We maintain that this convergence between the two approaches is not accidental. We have been constantly watching the catalog, PHL-EC and scientific investigations in habitability of exoplanets. Please refer to the discussion section of the revised manuscript (last bulleted item) for further details. We urge the referee to revisit the schematic flow elucidated in Fig. 1 of the manuscript. We made minor modifications to Fig. 1 for a better understanding.

- As an exercise, we tried to find the optimal point of the ESI in a same fashion as we found it for CD-HPF – the finding was that a minima or a maxima cannot be guaranteed by the functional form of the CD-HPF.

- As ESI is not based on maximizing a score, when we go on increasing the number of constituent parameters, the numerical result of the ESI might go on decreasing: there

is no guarantee of stability. The reason for this is that each constituent term in the ESI is a number between 0 and 1 – now as we add more terms to this, the value of the score tends to zero. This is not the case with our model, CD-HPF, as we have shown in the supplementary file of [4] that global optima is unaffected under additional input parameters (finitely many). However this throws a computational challenge of local oscillations which has been tackled by Stochastic Gradient Ascent[].

- It is easy to misconstrue CDHS with ESI as being one since the parameters of the CD-HPF are the same as that of the ESI, and the functional form in either metric is multiplicative in nature. However, that is not the case. We have laid a little emphasis on how our metric is similar to ESI in saying that the ESI is a special case of the CD-HPF. The crux of the philosophy behind the CD-HPF is essentially that of *adaptive modeling*, i.e., the score generated is based on the best combination of the factors and not by static weights as followed by ESI.

  That, essentially, the ESI score gives non-dynamic weights to all the different planetary (with no trade-off between the weights) observables or calculated features considered, which in practice may not be the best approach, or at least, the only way of indicating habitability. It might be reasonable to say that for different exoplanets, the various planetary observables may weigh each other out to create a unique kind of favorable condition. For instance, in one planet, the mass may be optimal, but the temperature may be higher than the average of the Earth, but still within permissible limits; in another planet, the temperature may be similar to that of the Earth, but the mass may be lower. By discovering the best combination of the weights (or, as we call it, *elasticities*) to maximize the resultant score, to the different planetary observables, we are creating a score which presents the best case scenario for the habitability of a planet.

- The essence of the CD-HPF, and consequently, that of the CDHS is indeed orthogonal to the essence of the ESI or BCI. We argue not in favor of the superiority of our metric, but for the new approach that is developed. We believe that there should actually be various metrics arising from different schools of thought so that the habitability of an exoplanet may be collectively determined from all these. Such a kind of adaptive modeling has not been used in the context of planetary habitability prior to the CD-HPF. While not in agreement with the structure of ESI, CDHS does complement ESI.

**Key Differences between ESI and CDHS:** In the case of the ESI, there is no evidence of a rigorous functional analysis. Schulze-Makuch et al. have not reported the existence of

**Table 39:** Differences between ESI and CDHS at a glance

| S. No. | ESI | CDHS |
|---|---|---|
| 1. | Derived for four input parameters only. It is unclear how the ESI will behave if additional parameters such as eccentricity, flux, radial velocity, etc. are added. | On the contrary, the CDHS is solidly grounded in optimization theory as we have shown (in the supplementary file). |
| 2. | The exponents of each term in the CDHS, $w_i/n$ is predetermined. | The exponents $\alpha$, $\beta$, $\gamma$ and $\delta$ are not predetermined; computing them is a part of the optimization problem. |
| 3. | In all likelihood, the inclusion of additional input parameters will diminish the ESI since all input parameters are scaled between 0 and 1, and the weights are fixed. | This does not happen with the CDHS even if we include additional parameters. |

extrema for the ESI. We tried to find the maximum for the ESI in a manner similar to that of CD-HPF and we were unable to do so with the appropriate mathematical tools. This means that if we were to draw parallels between the CDHS and the ESI, the ESI would have no guarantee of having a maxima. The CD-HPF is based on an adaptive modeling, that is, when the CDHS is computed, the response is a maximum, which is based on the functional form of the CD-HPF. We reported the proof in order to substantiate the fact that a maximum does exist of the functional form that we have used. The highlights of the CD-HPF are: – **this paragraph seems a little redundant**

- The exponents (or, the *elasticities*) of each observable in the function, $R$, $D$, $T_s$, $V_e$ are denoted using $\alpha$, $\beta$, $\gamma$, and $\delta$ respectively. The constraints on the permissible values of the elasticities are: – **we've also mentioned this in the paper, so even this is a little redundant**

  $$\alpha + \beta + \gamma + \delta \leq 1$$

  and

  $$\alpha, \beta, \gamma, \delta \geq 0$$

  Thus, the computation of the CDHS of a planet is essentially a *constrained optimization problem.*

- In (**insert page number**), we have also shown that the number of components in the

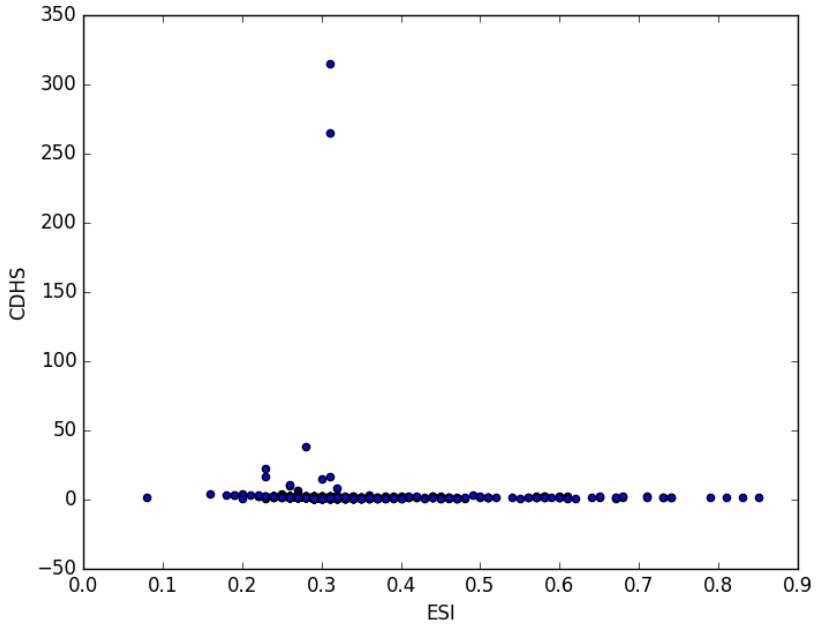



**Figure 16:** ESI vs CDHS (CRS)

CDHS can be countably infinite. Of course, the components being considered for each planet should be the same and the scoring system becomes different, but the possibility still exists that $n$ number of observables and/or input parameters can be accommodated into the function, while keeping the constraints on the elasticities intact. Moreover, we have proved a very important theorem that even if $n$ number of observables and/or input parameters make the functional form extremely complicated, a global optima is still guaranteed. – **this probably can be combined with a point we previously made**

The reason why CDHS computation is a challenging problem (and ESI is not!) is

- Local oscillations about the optima [3], [Ginde2016], [8] are difficult to mitigate even though we have shown there exists a theoretical guarantee for the same.
- Extrema doesn't occur at the corner points and is therefore the location of such is difficult to predict.
- We have emphasized enough on how the CD-HPF reconciles with the machine learning methods that we have used to automatically classify exoplanets. It is not easy two to propose two fundamentally different approaches (one of which is CDHS) that lead

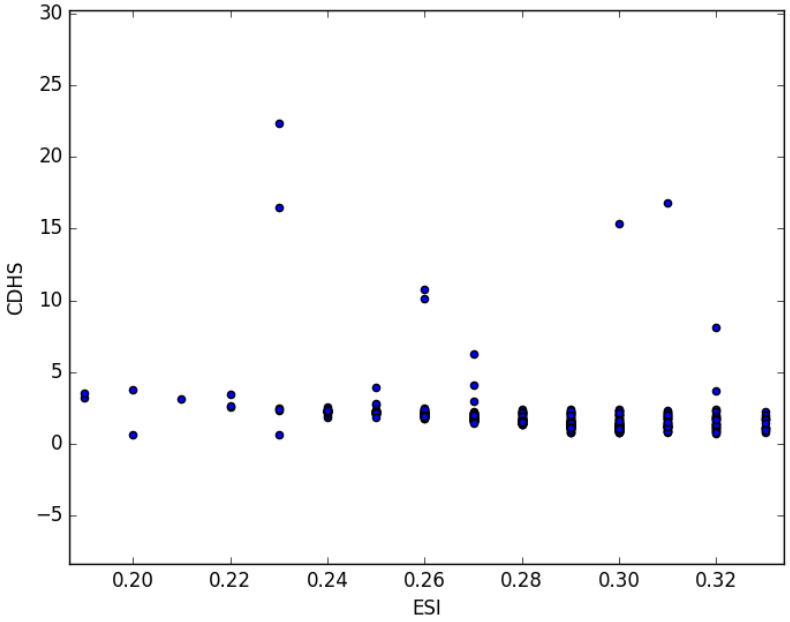



**Figure 17:** ESI vs CDHS (CRS) – a closeup

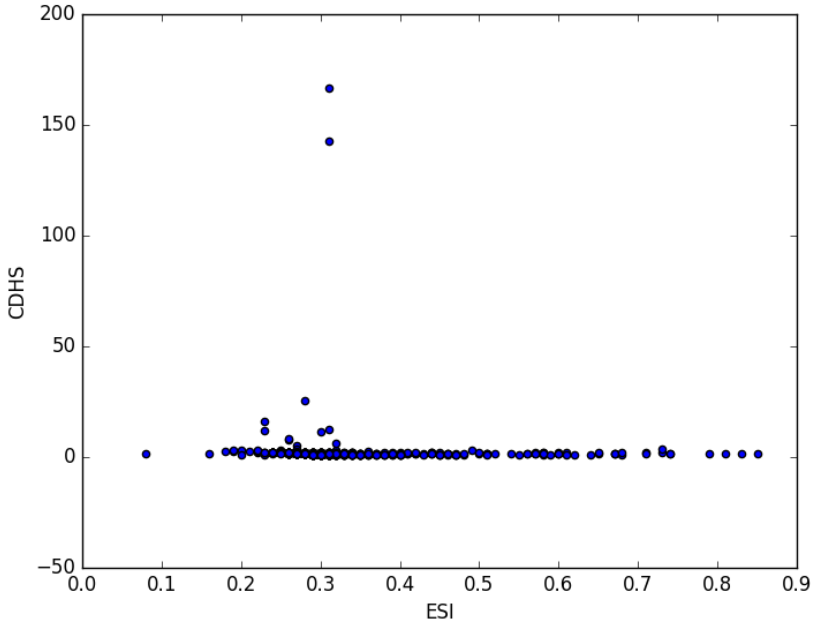



**Figure 18:** ESI vs CDHS (DRS)

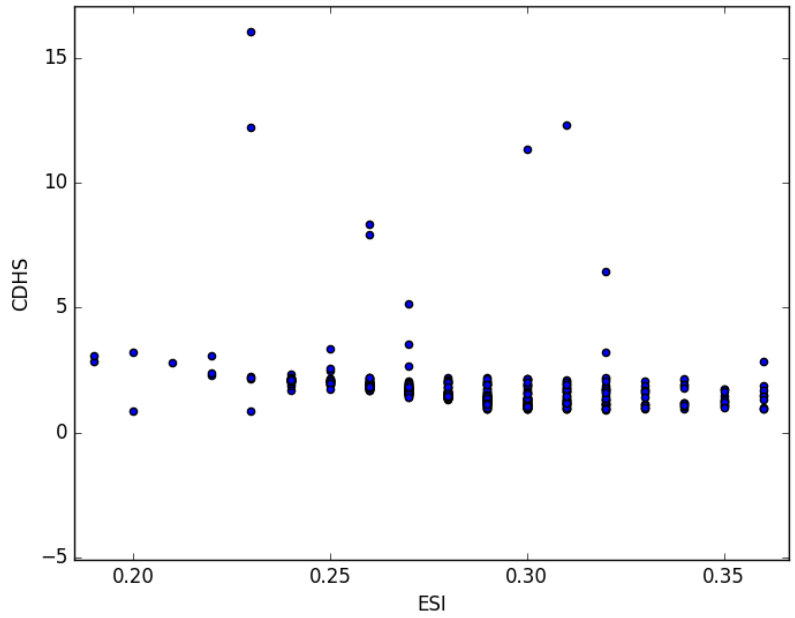



**Figure 19:** ESI vs CDHS (DRS) – a closeup

to a similar conclusion about an exoplanet. While the CDHS provides a numerical indicator (in fact the existence of one global optima shouldn't be a concern at all, rather a vantage point of the model that this eliminates the possibility of computing scores arbitrarily), the machine classification bolsters our proposition by telling us automatically which class of habitability an exoplanet belongs to (**more on the concerns of the reviewer on the ML methods in a later section in this document**). The performance of machine classification is evaluated by class-wise accuracy. The accuracies achieved are remarkably high, and at the same time, we see that the values of the CDHS for the sample of potentially habitable exoplanets which we have considered are also close to 1. Therefore, the computational approaches map earth similarity to habitability. This is remarkable and non-trivial. This degree of computational difficulty is non-existent in ESI.

- **NOTE:** One should not miss the crucial point of computation of CDHS which is only a part of the exercise. The greater challenge is the vindication of the CDHS metric in the classification of habitability of exoplanets by reconciliation of the modeling approach with the machine classification approach (where explicit scores were not used to classify habitability of exoplanets but the outcome validates the metric). The greatest strength

of CDHS is its flexibility in functional form helping forge the unification paradigm.

To be specific, even under the constrained optimization problem, the search space is countably infinite making the problem of finding the global optima computationally intractable until some intervention is made. We have mitigated the problem by employing stochastic gradient ascent. If it's a simplex problem, we know the optima is at one of the corner points (theoretically) and therefore don't bother looking for it in the entire search space (please note even in that case, the computational complexity is hard to ignore and people develop different kinds of approximation algorithms to improve upon the complexity). The functional form in our case is non-linear, convex and is bounded by constraints (which is typically a geometric region, contour/curved) making the search space complex enough to find the optima. The only positive thing in our favor is the theoretical existence of global optima which ensures, once our algorithm finds the optima, it terminates! There is the other issue of handling oscillation around the optima, coupled with finding "the optima". It is, indeed the question of finding the optima efficiently that distinguishes CDHS from ESI. As an exploratory investigation, we rendered ESI a functional form, similar to CDHS with the express objective of finding elasticity (dynamic weights to each of the parameters) that may compute optimum Earth similarity of any exoplanet. What we found was such theoretical guarantee is non-existent for ESI and it may ensure global optima only in the case of IRS, which is infeasible (Please refer to [**?**]). Therefore, our attempt to render theoretical credence to ESI fails as the new metric holds for IRS condition only (neither maxima nor minima). The new ESI input structure, despite a functional form very similar to CDHS is not able to reproduce the dynamic and flexible behavior of the Earth Similarity Score reported by CDHS. The details are documented in Appendix A.

## 6.2 Discussion

While each of these analyses is conducted independently, the results reconcile in a way that is sensible. The earth similarity indices, usually a solution that doesn't address the habitability classification problem has been reconciled with machine classification approach. In the process of doing so, we developed CDHS which is more representative (as opposed to ESI) of the mapping between these two completely different approaches.

A different perspective of the inference sought from the results of the ML algorithms is in the values of the Cobb-Douglas Habitability Scores (CDHS). We see that the values of CDHS of Proxima-b and TRAPPIST-1 c, d, and e (on which the probability of life is deemed to be more probable) are closer to the CDHS of Earth than those of the remaining planetary

samples which we have specifically explored in our study.

The exposition of the efficacy of our methods are concurrently being made by astrophysicists. In a recent article in Astrobiology, it has been said that two planets in the TRAPPIST-1 system are likely to be habitable: http://astrobiology.com/2018/01/two-trappist-1-system-planets-are-potentially-habitable.html. TRAPPIST-1 b and c are likely to have molten-core mantles and rocky surfaces, and hence, moderate surface temperatures and modest amounts of tidal heating. The expected favorable conditions are also reflected in the CDHS values of these planets **(insert)**.

The computational aspect is also worth mentioning here so as to provide the reader a thorough understanding of the method we have explored, their advantages over existing methods, and notwithstanding, their limitations.

The rate at which exoplanets are being discovered is rapidly increasing. Given the current situation and how technology is rapidly improving in astronomy, we firmly believe that eventually, problems such as this in observational astronomy will require a thorough dealing of data science to be handled efficiently. As the number of confirmed exoplanets grow, it might be wise to use an indexing and classification method which can do the job automatically, with little need for human intervention. This is where both ESI and CDHS shall play useful roles, we believe.

**Appendix A: Constraint Conditions for elasticity; $\alpha$, $\beta$, $\gamma$ and $\delta$: ESI with dynamic input elasticity fails to be an optimizer.**

In the function,

$$Y = k.\left(1-\frac{R_e-R}{R_e+R}\right)^{\alpha}.\left(1-\frac{D_e-D}{D_e+D}\right)^{\beta}.\left(1-\frac{T_e-T}{T_e+T}\right)^{\gamma}.\left(1-\frac{V_e-V}{V_e+V}\right)^{\delta} \tag{70}$$

where $R_e$, $D_e$,$T_e$ and $V_e$ are the radius, density, surface temperature and escape velocity of Earth and are constant terms. R,D.T and V are radius , density ,surface temperature and escape velocity for the planet under study.k is also a constant parameter

Differentiating eq.1 above partially w.r.t. R

$$\frac{\partial Y}{\partial R} = \alpha.\left(\frac{2R}{R_e+R}\right)^{\alpha-1}.\frac{2R_e}{(R_e+R)^2} \tag{71}$$

finding second derivative from eq.2

$$\begin{aligned}\frac{\partial^2 Y}{\partial R^2} &= \alpha.(\alpha-1).\left(\frac{2R}{R_e+R}\right)^{\alpha-2}.\left(\frac{2R_e}{(R_e+R)^2}\right).\left(\frac{2R_e}{(R_e+R)^2}\right)-\alpha.\left(\frac{2R}{R_e+R}\right)^{\alpha-1}.\left(\frac{4R_e(R_e+R)}{(R_e+R)^4}\right)\\ &= \alpha.(\alpha-1).\left(\frac{2R}{R_e+R}\right)^{\alpha-2}.\frac{4R_e^2}{(R_e+R)^4}-\alpha.\left(\frac{2R}{R_e+R}\right)^{\alpha-1}.\left(\frac{4R_e^2+4R_e.R)}{(R_e+R)^4}\right)\\ &= \alpha.(\alpha-1).\left(\frac{2R}{R_e+R}\right)^{\alpha-2}.\frac{1}{(R_e+R)^4}.\left((\alpha-1).4R_e^2-\frac{2R}{R_e+R}.(4R_e^2+4RR_e)\right)\\ &= \alpha.(\alpha-1).\left(\frac{2R}{R_e+R}\right)^{\alpha-2}.\frac{1}{(R_e+R)^4}.\left((\alpha-1).4R_e^2-8RR_e\right)\end{aligned}$$

hence final second order derivative is-

$$\frac{\partial^2 Y}{\partial R^2} = \alpha.(\alpha-1).\left(\frac{2R}{R_e+R}\right)^{\alpha-2}.\frac{1}{(R_e+R)^4}.\left((\alpha-1).4R_e^2-8RR_e\right) \tag{72}$$

for concavity eq.3 must be greater than 0 so,

$$\alpha.(\alpha-1).\left(\frac{2R}{R_e+R}\right)^{\alpha-2}.\frac{1}{(R_e+R)^4}.\left((\alpha-1).4R_e^2-8RR_e\right)>0 \tag{73}$$

on solving above inequality we will get ,

$$\alpha-1>2\frac{R}{R_e} \tag{74}$$

similarly it can be proved for other 3 elasticity constants which gives us constrained conditions

$$\beta-1>2\frac{D}{D_e}, \tag{75}$$

$$\gamma-1>2\frac{T}{T_e}, \tag{76}$$

$$\delta-1>2\frac{V}{V_e} \tag{77}$$

Summing up equations (5),(6),(7) and (8),

$$\alpha+\beta+\gamma+\delta-4>2\left(\frac{R}{R_e}+\frac{D}{D_e}+\frac{T}{T_e}+\frac{V}{V_e}\right) \tag{78}$$

$$\Rightarrow \alpha+\beta+\gamma+\delta>2\left(\frac{R}{R_e}+\frac{D}{D_e}+\frac{T}{T_e}+\frac{V}{V_e}\right)+4 \tag{79}$$

Above Equation (10) shows that sum of the four elasticity constants cannot be less than or

equal to 1 (in fact cannot be less than 1). This is the case of IRS (increasing return to scale) in CDHPF function, which means that function is neither concave nor convex. The new metric holds for IRS condition only which doesn't ensure a global maxima implying lack of theoretical foundation for the ESI input structure.

## REFERENCES


[1] Bora, K., Saha, S., Agrawal, S., Safonova, M., Routh, S., Narasimhamurthy, A., 2016. CD-HPF: New Habitability Score Via Data Analytic Modeling. *Astronomy and Computing*, 17, 129-143

[2] Schulze-Makuch, D., Méndez, A., Fairén, A. G., et al., 2011. A Two-Tiered Approach to Assessing the Habitability of Exoplanets. *Astrobiology*, 11, 1041.

[3] Saha S., Sarkar J., Dwivedi A., Dwivedi N., Anand M. N., Roy R., 2016. A novel revenue optimization model to address the operation and maintenance cost of a data center. *Journal of Cloud Computing*, 5:1, 1-23.

[4] Saha, S., Basak, S., Agrawal, S., Safonova, M., Bora, K., Sarkar, P., Murthy, J. 2018. Theoretical Validation of Potential Habitability via Analytical and Boosted Tree Methods: An Optimistic Study on Recently Discovered Exoplanets. *Astronomy and Computing*, Arxiv, arXiv.org > astro-ph > arXiv:1712.01040

[5] Cobb, C. W. and Douglas, P. H., 1928. A Theory of Production. *American Economic Review*, 18 (Supplement), 139.

[6] Ginde, G., Saha, S., Mathur, A., Venkatagiri, S., Vadakkepat, S., Narasimhamurthy, A., B.S. Daya Sagar., 2016. ScientoBASE: A Framework and Model for Computing Scholastic Indicators of Non-Local Influence of Journals via Native Data Acquisition Algorithms. *J. Scientometrics*, 107:1, 1-51

[7] Wolf, E. T., 2017. Assessing the Habitability of the TRAPPIST-1 System Using a 3D Climate Model. *Astrophys. J.*, 839:L1.

[8] B. Goswami, J. Sarkar, S. Saha and S. Kar., CD-SFA: Stochastic Frontier Analysis Approach to Revenue Modeling in Cloud Data Centers, International Journal of Computer Networks and Distributed Systems, Inderscience (Forthcoming)

# 7 SUPERNOVA CLASSIFICATION

## 7.1 Introduction

This work describes the classification of supernova into various types. The focus is given on the classification of Type-Ia supernova. But the question is why we need to classify supernovae or why is it important? Astronomers use Type-Ia supernovae as âĂIJstandard candlesâĂİ to measure distances in the Universe. Classification of supernovae is mainly a matter of concern for the astronomers in the absence of spectra.

A supernova is a violent explosion of a star, whose brightness for an amazingly short period of time, matches that of the galaxy in which it occurs. This explosion can be due to the nuclear fusion in a degenerated star or by the collapse of the core of a massive star, both leads in the generation of massive amount of energy. The shock waves due to explosion can lead to the formation of new stars and also helps astronomers indicate the astronomical distances. Supernovae are classified according to the presence or absence of certain features in their orbital spectra. According to Rudolph Minkowski there are two main classes of supernova, the Type-I and the Type-II. Type-I is further subdivided into three classes i.e. the Type-Ia, the Type-Ib and the Type-Ic. Similarly, Type II supernova are further sub-classified as Type IIP and Type IIn. Astronomers face lot of problem in classifying them because a supernova changes itself over the time. At one instance a supernovae belonging to a particular type, may get transformed into the supernovae of other type. Hence, at different time of observation, it may belong to different type. Also, when this spectra is not available, it poses a great challenge to classify them. They have to rely only on photometric measurements for their classification which poses a big challenge in front of astronomers to do their studies.

Machine learning methods help researchers to analyze the data in real time. Here, we build a model from the input data. A learning algorithm is used to discover and learn knowledge from the data. These methods can be supervised (that rely on training set of objects for which target property is known) or unsupervised (require some kind of initial input data but unknown class).

In this chapter, classification of Type Ia supernova are taking in considerations from a supernova dataset defined in [Davis et al.2007],[Riess et al.2007] and [Wood-Vassey et al.2007] using several machine learning algorithms. To solve this problem, the dataset is classified in two classes which may aid astronomers in the classification of new supernovae with high

accuracy.

## 7.2 Categorization of Supernova

The basic classification of supernova is done depending upon the shape of their light curves and the nature of their spectra. But there are different ways of classifying the supernovae-
**a) Based on presence of hydrogen in spectra** If hydrogen is not present in the spectra then it belongs to the Type I supernova; otherwise, it is the Type II.
**b) Based on type of explosion** There are two types of explosions that may takes place in the star- thermonuclear and core-collapse . Core collapse, happens at the final phase in the evolution of a massive star , whereas thermonuclear explosions are found in white dwarfs.

The detailed classification of supernova is given below where both types are discussed in correspondence to each other. The classification is the basic classification depending on Type I and Type II .

## 7.3 Type I supernova

Supernova are classified as Type I if their light curves exhibit sharp maxima and then die away smoothly and gradually. The spectra of Type I supernovae are hydrogen poor. As discussed earlier they have three more types- Type-Ia, Type-Ib and Type-Ic. According to [Fraser] and [supernova tutorial], Type Ia supernova are created when we have binary star where one star is a white dwarf and the companion can be any other type of star, like a red giant, main sequence star, or even another white dwarf. The white dwarf pulls off matter from the companion star and the process continues till the mass exceeds the **Chandrasekhar limit** of 1.4 solar masses (According to [Philipp], the Chandrasekhar limit/mass is the maximum mass at which a self gravitating object with zero temperature can be supported by electron degeneracy method). This causes it to explode. Type-Ia is due to the thermonuclear explosion and has strong silicon absorption lines at 615 nm and this type is mainly used to measure the astronomical distances. This is the only supernova that appears in all type of galaxies. Type-Ib have strong helium absorption lines and no silicon lines, Type-Ic have no silicon and no helium absorption lines. Type Ib and Type Ic are core collapse supernova like Type II without hydrogen lines. The reason of Type-Ib and Type-Ic to fall in core collapse is that they produce little Ni [Phillips93] and are found within or near star formation regions. Core

collapse explosion mechanism happens in massive stars for which hydrogen is exhausted and sometimes even He (as in case of Type-Ic). Both the mechanisms are shown in Figure 20

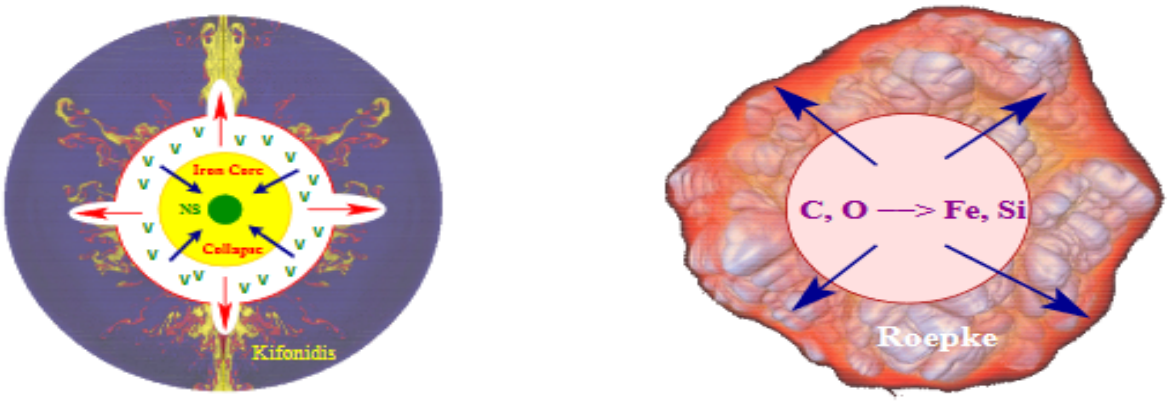



**Figure 20:** Core collapse supernova (*Left*) and Thermonuclear Mechanism(*Right*)

## 7.4 Type II supernova

Type-II is generally due to core collapse explosion mechanism. These supernovae are modeled as implosion-explosion events of a massive star. An evolved massive star is organized in the manner of an onion, with layers of different elements undergoing fusion. The outermost layer consists of hydrogen, followed by helium, carbon, oxygen, and so forth. According to [Fraser], a massive star, with 8-25 times the mass of the Sun, can fuse heavier elements at its core. When it runs out of hydrogen, it switches to helium, and then carbon, oxygen, etc, all the way up the periodic table of elements. When it reaches iron, however, the fusion reaction takes more energy than it produces. The outer layers of the star collapses inward in a fraction of a second, and then detonates as a Type II supernova. Finally the process left with a dense neutron star as a remnant. This show a characteristic plateau in their light curves a few months after initiation. They have less sharp peaks at maxima and peak at about 1 billion solar luminosity. They die away more sharply than the Type I. It has visible strong hydrogen and helium absorption lines. If the massive star have more than 25 times mass of the Sun, the force of the material falling inward collapses the core into a black hole. The main characteristics of Type II supernova is the presence of hydrogen lines in itâĂŹs spectra. These lines have P Cygni profiles and are usually very broad, which indicates rapid expansion velocities for the material in the supernova.
Type II supernova are sub-divided based on the shape of their light curves. Type II-Linear

(Type II-L) supernova has fairly rapid, linear decay after maximum light. Type II-plateau (Type II-P) remains bright for a certain period of time after maximum light i.e. they shows a long phase that lasts approximately 100d and here light curves are almost constant(plateau phase). TypeII-L is rarely found and doesnâĂŹt show the plateau phase, but decreases logarithmically after their light curve is peaked. As they drops on logarithmic scale, more or less linearly , hence L stands for âĂIJLinearâĂİ. In Type II-narrow (Type IIn) supernova, hydrogen lines had a vague or no P Cygni profile, and instead displayed a narrow component superimposed on a much broader base. Some type Ib/Ic and IIn supernova with explosion energies E >1052 erg are often called **hypernovae**. The classification of supernova is shown in Figure 21 with the flowchart as-

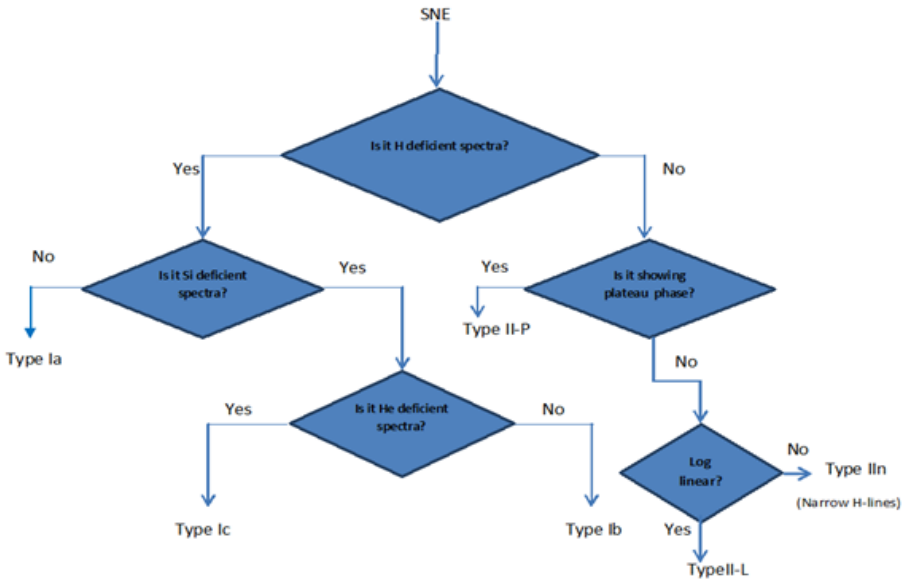



**Figure 21:** Classification of Supernova

## 7.5 Machine Learning Techniques

Machine learning is a discipline that constructs and study algorithms to build a model from input data. The type and the volume of the dataset will affect the learning and prediction performance. Machine learning algorithms are classified into supervised and unsupervised methods, also known as predictive and descriptive, respectively. Supervised methods are

also known as classification methods. For them class labels or category is known. Through the data set for which labels are known, machine is made to learn using a learning strategy, which uses parametric or non-parametric approach to get the data. In parametric model, there are fixed number of parameters and the probability density function is specified as p(x|$\theta$) which determines the probability of pattern x for the given parameter $\theta$ (generally a parameter vector). In nonparametric model, there are no fixed number of parameters, hence cannot be parameterized. Parametric models are basically probabilistic models like Bayesian model, Maximum Aposteriori Classifiers etc. and non- parametric where directly decision boundaries are determined like Decision Trees, KNN etc. These models ( parametric and nonparametric) mainly talks about the distribution of data in the data set, which helps to take the decision upon the use of appropriate classifiers.

If class labels are not known (unsupervised case), and data is taken from different distributions it is hard to assess. In these cases, some distance measure, like Euclidean distance, is considered between two data points, and if this distance is 0 or nearly 0, the two points are considered as similar. All the similar points are kept in the same group, which is called as cluster. Likewise the clusters are devised. While clustering main aim is to keep high intracluster similarity and low intercluster similarity. There are several ways in which clustering can be done. It can be density based, distance based, grid based etc. Shapes of the cluster also can be spherical, ellipsoidal or any other based on the type of clustering being performed. Most basic type of clustering is distance based, on the basis of which K-means algorithm is devised which is most popular algorithm. Other clustering algorithms to name a few are K-medoids, DB Scan, Denclue etc. Each has its own advantages and limitations. They have to be selected based on the dataset for which categorization has to be performed. Data analytic uses machine learning methods to make decision for a system.

According to [Nicholas et al.2010], supervised methods rely on a training set of objects for which the target property, for example a classification, is known with confidence. The method is trained on this set of objects, and the resulting mapping is applied to further objects for which the target property is not available. These additional objects constitute the testing set. Typically in astronomy, the target property is spectroscopic, and the input attributes are photometric, thus one can predict properties that would normally require a spectrum for the generally much larger sample of photometric objects.

On the other hand, unsupervised methods do not require a training set. These algorithms usually require some prior information of one or more of the adjustable parameters, and the solution obtained can depend on this input.

In between supervised and unsupervised algorithms there is one more type of model-

semi-supervised method is there that aims to capture the best from both of the above methods by retaining the ability to discover new classes within the data, and also incorporating information from a training set when available.

## 7.6 Supernovae Data source and classification

The selection of classification algorithm not only depends on the dataset, but also the application for which it is employed. There is, therefore, no simple method to select the best optimal algorithm. Our problem is to identify Type Ia supernova from the given dataset in [Davis et al.] which contains 292 different supernova information. Since the classification is binary classification, as one need to identify Type Ia supernova from the list of 292 supernovas, the best resulting algorithms are used for this purpose. The algorithms used for classification are NaÃŕve Bayes, LDA, SVM, KNN, Random Forest and Decision Tree.

The dataset used is retrieved from [Davis et al.]. These data are a combination of the ESSENCE, SNLS and nearby supernova data reported in Wood-Vasey et al. (2007) and the new Gold dataset from Riess et al.(2007). The final dataset used is combination of ESSENCE / SNLS / nearby dataset from **Table 4** of Wood-Vasey et al. (2007), using only the supernova that passed the light-curve-fit quality criteria. It has also considered the HST data from **Table 6** of Riess et al. (2007), using only the supernovae classified as gold. These were combined for Davis et al. (2007) and the data are provided in 4 columns: redshift, distance modulus, uncertainty in the distance modulus and quality as âĂIJGoldâĂİ or âĂIJSilverâĂİ. The supernova with quality labeled as âĂIJGoldâĂİ are Type Ia with high confidence and those with label âĂIJSilverâĂİ are Likely but uncertain SNe Ia. In the dataset, all the supernova with redshift value less than 0.023 and quality value Silver are discarded.

## 7.7 Results and Analysis

The experimental study was setup to evaluate performance of various machine learning algorithms to identify Type-Ia supernova from the above mentioned dataset. The data set mentioned above is tested on 6 major classification algorithms namely NaÃŕve Bayes, Decision tree, LDA, KNN, Random Forest and SVM respectively. A ten-fold cross validation procedure was carried out to make the best use of data, that is, the entire data was divided into ten bins in which one of the bins was considered as test-bin while the remaining 9 bins were taken as training data. We observe the following results and conclude that the outcome of the experiment is encouraging, considering the complex nature of the data. Table 40 shows the result of classification.

**Table 40:** Results of Type -Ia supernova classification

| Algorithm | Accuracy (%) |
|---|---|
| NaÃŕve Bayes | 98.86 |
| Decision Tree | 98.86 |
| LDA | 65.90 |
| KNN | 96.59 |
| Random Forest | 97.72 |
| SVM | 65.90 |
| | |

Performance analysis of the algorithms on the dataset is as follows.
1. NaÃŕve BayesâĂŹ and Decision Tree top the accuracy table with the accuracy of 98.86%.
2. Random Forest ranks 2 with accuracy of 97.72% and KNN occupies 3rd position with 96.59% accuracy.
3. The dramatic change was observed in the case of SVM, which occupied the last position with LDA with an accuracy of 65.9%. The geometric boundary constraints inhibit the performance of the two classifiers.

Overall, we can conclude NaÃŕve BayesâĂŹ, Decision Tree and Random Forest perform exceptionally well with the dataset, while KNN acts as an average case.

## 7.8 Conclusion

In this chapter, we have compared few classification techniques to identify Type Ia supernova. Here it is seen that Naive Bayes, Decision Tree and Random Forest algorithms gave best result among all. This work is relevant to astroinformatics, especially for classification of supernova, star-galaxy classification etc. The dataset used is a well-known which is the combination of ESSENCE, SNLS and nearby supernova data.

## 7.9 Future Research Directions

Supernova classification is an emerging problem that scientists, astronomers and astrophysicists are working on to solve using various statistical techniques. In the absence of spectra, how this problem can be solved. In this chapter, Type-Ia supernova are classified

using machine learning techniques based on redshift value and distance modulus. The same techniques can be applied to solve the overall supernova classification problem. It can help us to differentiate Type I supernova from Type II, Type Ib from Type Ic or so on. Machine learning techniques along with various statistical methods help us to solve such problems.

# 8 MACHINE LEARNING DONE RIGHT: A CASE STUDY IN QUASAR-STAR CLASSIFICATION

## 8.1 Introduction

Quasars are *quasi-stellar radio sources*, which were first discovered in 1960. They emit radio waves, visible light, ultraviolet rays, infrared rays, X-rays and gamma rays. They are very bright and the brightness causes the light of all the other stars becoming relatively faint in that galaxy which houses these quasars. The source of their brightness is generally the massive black hole present in the center of the host galaxy. Quasars are many light-years away from the Earth and the energy from quasars takes billions of years to reach the earth's atmosphere; they may carry signatures of the early stages of the universe. This information gathering exercise and subsequent physical analysis of quasars pose strong motivation for the study. It is difficult for astronomers to study quasars by relying on telescopic observations alone since quasars are not distinguishable from the stars due to their great distance from Earth. Evolving some kind of semi-automated or automated technique to classify quasars from stars is a pressing necessity.

Identification of large numbers of quasars/active galactic nuclei (AGN) over a broad range of redshift and luminosity has compelled astronomers to distinguish them from stars. Historically, quasar candidates have been identified by virtue of color, variability, and lack of proper motion but generally not all of these combined. The standard way of identifying large numbers of candidate quasars is to make *color cuts* using optical or infrared photometry. This is because the majority of quasars at $z < 2.5$ emit light that mostly falls in the frequencies corresponding to blue than the majority of stars in the optical range, and light whose frequencies are much lower than infrared. This establishes the inadequacy to distinguish stars from quasars based on color, variability, and proper motion. Machine learning techniques have turned out to be extremely effective in performing classification of various celestial objects.

*Machine Learning* (ML) [Basak et al.(2016)] is a sub-field of computer science which relies on statistical methods for predictive analysis. Machine learning algorithms broadly fall into two categories: *supervised* and *unsupervised methods*. In supervised methods, target values are assigned to every entity in the data set. These may be class labels for *classification*, and continuous values for *regression*. In unsupervised methods, there are no target values associated with entities and thus the algorithms must find similarities between different entities. *Clustering* is an unsupervised machine learning approach. ML algorithms may broadly make use of one *strong* classifier, or a combination of *weak* classifiers. A *strong*

classifier or a strong learner is a single model implementation which may effectively be able to predict the outcome of an input, based on training samples. A weak learner, on the contrary, is not a robust classifier itself and may be only slightly better than a random guess. Combinations of weak learners may be used to make strong predictions. Namely two broad approaches exist for this: bootstrap aggregation (*bagging*) and *boosting*. In bagging [Breiman(1996)], the attribute set of a training sample is a subset of all the attributes. Often, successive learners complement each other for making a prediction. In boosting, each learner makes a prediction, usually on the entire attribute set, which is very close to a random guess: based on accuracy of each weak learner, a weight for each class is assigned to the weak learner successively constructed; the contribution of each weak learner to the final prediction depends on these weights. Consequently, the model is built based on a scheme of checks-and-balances to get the best results over many learners. AdaBoost was introduced by Freund and Schapire [Freund & Schapire(1996)], which is based on the aforementioned principles of boosting. Over time, many variations of the original algorithm have been suggested which take into consideration biases present in the data set and uneven costs of misclassification such as AdaCost, AdaBoost. MH, Sty-P Boost, Asymmetric AdaBoost etc.

Machine learning algorithms have been used in various fields of astronomy. In this manuscript, the strength of such algorithms has been utilized to classify quasars or stars to complement the astronomers' task of distinguishing them. Some evidence of data classification methods for quasar-star classification such as Support Vector Machines [Gao et al.(2008), Elting et al.(2008)] and SVM-kNN [Peng et al.(2013)] is available in the existing literature. However, there is room for critical analysis and re-examination of the published work and significant amendments are not redundant! Machine learning has the potential to provide good predictions (in this case, determining whether the class a stellar object belongs to is that of a star or a quasar): but only if aptly and correctly applied; otherwise, it may lead to wrong classifications or predictions. For example, a high *accuracy* may not necessarily be an indication of a proper application of machine learning as these statistical indicators themselves may lead to controversial results when improperly used. The presentation of the results, inclusive of appropriate validation methods depending on the nature of data, may reveal the correctness of the methods used. Incorrect experimental methods and critical oversight of nuances in data may not be a faithful representation of the problem statement; this is elaborated in Sections 8.4 to 13.5.

The remainder of the paper is organized as follows: Section 8.2 presents the motivation behind re-investigating the problems and the novel contribution in the solution scheme. This is followed by a literature survey where the existing methods are highlighted Section 8.3.

Section 8.4 discusses the data source, acquisition method, and nuances present in data, used in existing work. Section 8.5 discusses a few machine learning methods that are used with emphasis on the effectiveness of a particular approach bolstered by the theoretical analysis of the methods employed by the authors of this paper. Section **??** of the paper discusses various metrics used for performance analysis of the given classification approaches. Section 13.5 presents and analyzes the results obtained using the approaches used in Section 8.5; this section elaborates on the comparison between the work surveyed and the work presented in this paper. In Section **??**, a discussion reconciles all the facts discovered while exploring the dataset and various methods, and our ideas on an appropriate workflow in any data analytic pursuit. We conclude in Section **??** by reiterating and fortifying the motivation for the work presented and document a workflow thumb rule (Figure **??**) for the benefit of the larger readership.

## 8.2 Motivation and Contribution

The contribution of this paper is two-fold: novelty and critical scrutiny. Once the realization about data imbalance dawned upon us, we proposed a method, Asymmetric Adaboost, tailor made to handle such imbalance. This has not been attempted before in star-quasar classification, the problem under consideration. Secondly, the application of this method makes it imperative to critically analyze other methods reported in the existing literature and this exercise helped unlock the nuances of this problem, otherwise unknown. This exercise sheds some light on the distinction between classical pattern recognition and machine learning. The former typically assumes that data are balanced across the classes and the algorithms and methods are written to handle balanced data. However, the latter is designed to tackle data cleaning and preparation issues within the algorithmic framework. Thus, beneath the hype, the rationality, and science behind choosing machine learning over classical pattern recognition emerges; machine learning is more convenient and powerful. The problem turns out to be a case study for investigating such a paradigm shift. This has been highlighted by the authors through critical mathematical and computational analysis and should serve as another significant contribution.

Different algorithms are not just explored on a random basis but are chosen carefully in cognizance of papers available in the public domain. Extremely high accuracy reported in the papers surveyed (refer to Section 8.3), not consistent with the data distribution raised reasonable doubt. Hence the authors decided to adopt careful scrutiny of the work accomplished in the literature, having set the goals on falsification; scientific validation of those

results required re-computation and investigative analysis of the ML methods used in the past. This has brought up several anomalies in the published work. The authors intend to highlight those in phases throughout the remainder of the text. AstroInformatics is an emerging field and is thus prone to erroneous methods and faulty conclusions. Correcting and re-evaluating those are important contributions that the community should not ignore! This is the cornerstone of the work presented apart from highlighting the correct theory behind ML methods in astronomy and science. The detailed technical contributions made by authors are summarized as follows:

- We have attempted to demonstrate the importance of linear separability tests. This is done to check whether the data points are linearly separable or not. Certain algorithms like SVM with linear kernel cannot be used if data is not linearly separable. The implications of a separability test, an explanation for which has been given in Section **??**, has been overlooked in the available literature.

- A remarkable property of this particular data set, the presence of data bias, has been identified. If the classification is performed without considering the bias in the data set, it may lead to biased results; for example, if two classes $C_1$ and $C_2$ are present in the dataset and one class is dominating in the dataset then directly applying any classification algorithm may return results which are biased by the dominating class. We argue that a dataset must be balanced i.e. the selected training set for classification must contain almost the same amount of data belonging to both classes. This is presented in Section 8.5.1 as the concept of artificial balancing of the dataset.

- We have also proposed an approach which mathematically handles the bias in the dataset. This approach can be used directly in the presence of inherent data bias. Known as Asymmetric AdaBoost, this has been discussed in Section **??**.

It is important to note that the authors have used the same datasets from previous work by other researchers. Also, the paper is not only about highlighting the efficacy of a method by exhibiting marginal improvements in accuracy. Astronomy is becoming increasingly data driven and it is imperative that such new paradigms be embraced by the leaders of the astronomical community. However, such endeavor should be carefully pursued because of the possible loopholes that can arise due to oversight or lack of adequate foundation in data science. Through this paper, apart from demonstrating effective methods for automatic classification of stars and quasars, the authors laid down some fundamental ideas that should be kept in mind and adhered to. The ideas/working rules are for anyone wishing to pursue

*astroInformatics* or data analytics in any area. A flowchart presenting the knowledge base is presented in the conclusion section (Figure **??**).

All the experiments were performed in Python3, using the machine learning toolkit *scikit-learn* [Pedregosa et al.(2011)].

## 8.3 Star-Quasar Classification: Existing Literature

Support Vector Machine (SVM) is one of the most powerful machine learning methods, discussed in detail in Section 8.5. It is used generally for binary classification. However, variants of this method can also be applied for multi-class classification. Since the classification is based on two classes, namely stars and quasars, SVM has been widely used in the existing literature to classify quasars and stars.

[Gao et al.(2008)] used SVM to separate quasars from stars listed in the Sloan Digital Sky Survey (SDSS) database (refer to Section 8.4 for details on data acquisition). Four colors: $u-g$, $g-r$, $r-i$, and $i-z$ are used for photometric classification of quasars from stars. SVM was used for the classification of quasars and stars. A non-linear radial basis function (RBF) kernel was used for SVM. The main reason for the usage of RBF kernels was to tune the parameters $\gamma$ and $c$ (trade-off) to increase the accuracy. The highest accuracy of classification obtained was equal to 97.55%. However, the manuscript fails to check for linear separability of the two classes. [Elting et al.(2008)] used SVM for the classification of stars, galaxies, and quasars. A data set comprising the $u-g$, $g-r$, $r-i$ and $i-z$ colors is used for the prediction on unbalanced data set. A non-linear RBF kernel was used for classification and an accuracy of 98.5% was obtained.

The aforementioned papers used non-linear RBF kernel which is commonly used when data distribution is Gaussian. It is imprudent to cite increase the accuracy as the reason for using any kernel. The choice of kernel depends on the data. Therefore, authors in the present manuscript have performed a linear separability test on the data set, discussed in Section 8.5, which clearly shows that the data is mostly linearly separable and hence, a linear kernel should be used. [Peng et al.(2013)] used an SVM-KNN method which is a combination of SVM and KNN. SVM-KNN strengthens the generalization ability of SVM and applies KNN to correct some forecasting errors of SVM and improve the overall forecast accuracy. SVM-KNN was applied for classification using a linear kernel. SVM-KNN (the ratio of the number of samples in the training set to the testing set as 9:1) was applied on the unbalanced SDSS data set which was dominated by the "star" class. This gave an overall accuracy of 98.85% as the data was unbalanced and the classes were biased. The total percentage of stars and quasars

which were classified using this method was 99.41% and 98.19% respectively.

SVM should not be used without performing a separability test and therefore, choice of linear or RBF kernel depends on the linear separability of data. If data is linearly separable, then SVM may be implemented using a linear kernel. The absence of linear separability and evidence of a normal trend in data may justify SVM implementation in conjunction with the RBF kernel. However, that evidence was not forthcoming in the works by [Gao et al.(2008)], [Elting et al.(2008)] or [Peng et al.(2013)]. In fact, one should select the kernel and then accordingly should apply SVM depending on the data distribution. There was no evidence of a separability analysis being performed by [Gao et al.(2008)], [Elting et al.(2008)] or [Peng et al.(2013)], thus forcing the conclusion that the kernel was chosen without proper examination. [Gao et al.(2008)] and [Elting et al.(2008)] used a non-linear RBF kernel is used along with SVM. Similarly, in [Peng et al.(2013)] used a linear kernel in SVM-KNN without a proper justification. Moreover, the class dominance was ignored in [Gao et al.(2008), Elting et al.(2008), Peng et al.(2013)]. Class dominance must be considered, otherwise, the accuracy of classification obtained will be biased by the dominant class and it will always be numerically very high. We have performed *artificial balancing* of data to counter the effects of class bias; the process of artificial balancing has been elaborated in 8.5.1.

The authors would like to emphasize that the manuscript is not a *black-box assembly* of several ML techniques but a careful study of those methods, eventually picking the right classifier based on the nature of data (such as Asymmetric Adaboost, as discussed in Section **??**). The algorithm's ability to handle class imbalance, and establishing the applicability of such an algorithm to solve similar kind of problems have been addressed in our work.

*Sensitivity* and *specificity* are measures of performance for binary classifiers. The accuracy obtained without calculating sensitivity and specificity is not always meaningful. Sensitivity and specificity are used to check the correctness of the obtained accuracy but were not reported in [Gao et al.(2008), Elting et al.(2008), Peng et al.(2013)]. This makes accuracy validation difficult.

The comparison of the results obtained from these [Gao et al.(2008), Elting et al.(2008), Peng et al.(2013)] are presented in Table 41.

## 8.4 Data Acquisition

The Sloan Digital Sky Survey (SDSS) [Adelman-McCarthy et al.(2008)] has created the most detailed three-dimensional maps of the Universe ever made, with deep multi-color images of

**Table 41:** Results of classification obtained by [Gao et al.(2008), Elting et al.(2008), Peng et al.(2013)]: the critical and challenging issues not addressed in the cited literature are tabulated as well.

| Methods | Accuracy (%) | Class Bias | Data Imbalance | Linear Separability Test Done |
|---|---|---|---|---|
| [Gao et al.(2008)] | 97.55 | YES | YES | NO |
| [Elting et al.(2008)] | 98.5 | YES | YES | NO |
| [Peng et al.(2013)] | 98.85 | YES | YES | NO |

one-third of the sky and spectra for more than three million astronomical objects. It is a major multi-filter imaging and spectroscopic redshift survey using a dedicated 2.5m wide-angle optical telescope at the Apache Point Observatory in New Mexico, USA. Data collection began in 2000 and the final imaging data release covers over 35% of the sky, with photometric observations of around 500 million objects and spectra for more than 3 million objects. The main galaxy sample has a median redshift of $z = 0.1$; there are redshifts for luminous red galaxies as far as $z = 0.7$, and for quasars as far as $z = 5$; and the imaging survey has been involved in the detection of quasars beyond a redshift $z = 6$. Stars have a redshift of $z = 0$.

SDSS makes the data releases available over the Internet. Data release 7 (DR7) [Abazajian et al.(2009)], released in 2009, includes all photometric observations taken with SDSS imaging camera, covering 14,555 square degrees of the sky. Data Release 9 (DR9) [Ahn et al.(2013)], released to the public on 31 July 2012, includes the first results from the Baryon Oscillations Spectroscopic Survey (BOSS) spectrograph, including over 800,000 new spectra. Over 500,000 of the new spectra are of objects in the Universe 7 billion years ago (roughly half the age of the universe). Data release 10 (DR10), released to the public on 31 July 2013. DR10 is the first release of the spectra from the SDSS-III's Apache Point Observatory Galactic Evolution Experiment (APOGEE), which uses infrared spectroscopy to study tens of thousands of stars in the Milky Way. The SkyServer provides a range of interfaces to an underlying Microsoft SQL Server. Both spectra and images are available in this way, and interfaces are made very easy to use. The data are available for non-commercial use only, without written permission. The SkyServer also provides a range of tutorials aimed at everyone from school children up to professional astronomers. The tenth major data release, DR10, released in July 2013, provides images, imaging catalogs, spectra, and redshifts via a variety of search interfaces. The datasets are available for download from the casjobs website (`http://skyserver.sdss.org/casjobs`).

The spectroscopic data is stored in the *SpecObj* table in the SkyServer. Casjobs is a flexible and advanced SQL-based interface to the Catalog Archive Server (CAS), for all data releases. It is used to download the SDSS DR6 [Elting et al.(2008)] data set which contains spectral information of 74463 quasars and 430827 stars. Spectral information like the colors $u - g$,

$g-r$, $r-i$, $i-z$ and redshift are obtained by running a SQL query. The output obtained from running the query is downloaded in the form of a comma-separated value (CSV) file.

## 8.5 Methods

### 8.5.1 Artificial Balancing of Data

Since the dataset is dominated by a single class (stars), it is essential for the training sets used for training the algorithms to be artificially balanced. In the data set, the number of entities in the stars' class is about six times greater than the number of entities in the quasars' class; it is a cause of concern as a data bias is imminent. In such a case, voting for the dominating class naturally increases as the number of entities belonging to this class is greater. The number of entities classified as stars is far greater than the number of entities classified as quasars. The extremely high accuracy reported by [Gao et al.(2008)], [Elting et al.(2008)], and, [Peng et al.(2013)] is because of the dominance of one class and not because the classes are correctly identified. In such cases, the sensitivity and specificity are also close to 1.

*Artificial balancing* of data needs to be performed such that the classes present in the dataset used for training a model don't present a bias to the learning algorithm. In quasar-star classification, the stars' class dominates the quasars' class. This causes an increase in the influence of the stars' class on the learning algorithm and results in a higher accuracy of classification. Algorithms like SVM cannot handle the imbalance in classes if the separating boundary between the two classes is thin, or slightly overlapping (which is often the case in many datasets) and end up classifying more number of samples as belonging to the dominant class, thereby increasing the accuracy of classification, numerically. It is found that the accuracy of classification decreases with the artificial balancing of the dataset as shown in Section 13.5. In artificial balancing, an equal number of samples from both the classes are taken for training the classifier. This eliminates the class bias and the data imbalance. The dataset used for analysis has a larger number of samples belonging to the stars' class as compared to the number of samples in the quasars' class. The samples that are classified as belonging to the stars' class are more when compared to the number of sampled classified as belonging to the quasars' class as the voting for the dominating class increases with imbalance and results in a higher accuracy of classification. Hence, the voting for the stars' class was found to be 99.41% which is higher than the voting of quasars, which is 98.19%, by [Peng et al.(2013)]. The accuracy claimed is doubtful as there data imbalance and class bias is prevalent.

# 9 An Introduction to Image Processing

## 9.1

# 10 A STUDY IN EMERGENCE OF ASTROINFORMATICS: A NOVEL METHOD IN BIG DATA MINING

## 10.1 INTRODUCTION

Scientometrics evaluates the impact of the results of scientific research by placing focus on the work's quantitative and measurable aspects. Statistical mathematical models are employed in this study and evaluation of journals and conference proceedings to asses their quality. The implosion of journals and conference proceedings in the science and technology domain coupled with the insistence of different rating agencies and academic institutions to use journal metrics for evaluation of scholarly contribution present a big data accumulation and analysis problem. This high volume of data requires an efficient metric system for fair rating of the journals. However, certain highly known and widely used metrics such as the Impact Factor and the H factor have been misused lately through practices like non-contextual self-citation, forced citation, copious-citation etc. [7] Thus, the way this volume of data is modeled needs improvement because it influences the evaluation and processing of this data to draw useful conclusions. One effective way to deal with this problem is to characterize a journal by a single metric or a reduced set of metrics that hold more significance. The volumes of data scraped from various sources are organized as a rectangular $m$x$n$ matrix where $m$ is the rows representing the number of articles in a journal and $n$ columns of various Scientometric parameters. An effective dimensionality and rank reduction technique such as the Singular Value Decomposition (SVD) applied on the original data matrix not only helps to obtain a single ranking metric(based on the different evaluation parameters enlisted as various columns) but also identifies pattern used for efficient analysis of the big data. Apache Mahout, Hadoop, Spark, R, Python, Ruby are some tools that can be used to implement SVD and other similar dimensionality reduction techniques. [5]

One notable characteristic of the Scientometric data matrix is its sparsity. The matrix is almost always rectangular and most metric fields (columns) do not apply to many of the articles(rows). For instance, a lot of journals may not have patent citations. Similarly, a number of other parameters might not apply to a journal as a whole. Usually, $n$ and $m$ differ from each other by a good integer difference. Thus, by virtue of this sparsity, the efficiency of the SVD algorithms can be enhanced when coupled with norms like $l_1$-norm, $l_2$-norm or the group norms. In general, both sparsity and structural sparsity regularization methods utilize the assumption that the output $\boldsymbol{Y}$ can be described by a reduced number of input variables in the input space $\boldsymbol{X}$ that best describe the output. In addition to this, structured

sparsity regularization methods can be extended to allow optimal selection over groups of input variables in $\boldsymbol{X}$.

## 10.2 The depths of Dimensionality Reduction

Dimensionality reduction has played a significant role in helping us ascertain results of the analysis for voluminous data set [2]. The propensity to employ such methods comes from the phenomenal growth of data and the velocity at which it is generated. Dimensional reduction such as Singular Value Decomposition and Principle component Analysis solves such big data problems by means of extracting more prominent features and obtaining a better representation of the data. This data tends to be much smaller to store and much easier to handle to perform further analysis. These dimensionality reduction methods are very often found in most of the tools which handle large data sets and perform rigorous data analysis. Such tools include Apache Mahout, Hadoop, Spark, R, Python etc. The ease of employing such methods is directly dependent on the performance of such tools to be able to compute and assess the results quickly and store it efficiently, all this while managing resources available at an optimal rate. The divergence in the methods used in these tools to compute such algorithms gives us scope to study and evaluate such case scenarios and help us choose the right kind of tools to perform these tasks.

### 10.2.1 *PCA*

**Principal Component Analysis**, a technique mostly used in statistics to transform a set of observations of possibly correlated variables into a set of linearly uncorrelated variables called as Principal Components. These Principal Components are the representation of the underlying structure in the data or the directions in which the variance is more and where the data is more concentrated.

The procedure lays emphasis on variation and identification of strong patterns in the dataset. PCA extracts a low dimensional set of features from a higher dimension dataset, simultaneously serving the objective of capturing as much useful information as possible. PCA is most commonly implemented in two ways:-

- **Eigenvalue Decomposition** of a data covariance(or correlation) matrix into canonical form of eigenvalues and eigenvectors. However, only square/diagonalizable matrices can be factorized this way and hence it also takes the name Matrix Diagonalization.

- **Singular Value Decomposition** of the initial higher dimension matrix. This approach

is relatively more suitable for the problem being discussed since it exists for all matrices: singular, non-singular, dense, sparse, square or rectangular.

### 10.2.2 *Singular Value Decomposition*

Singular Value Decomposition is the factorization of a real or complex matrix. Large scale of Scientometric data is mined using suitable web scraping techniques and is modeled as a matrix in which the rows represent the articles in a journal published over the years, and the columns represent various Scientometrics or indicators proposed by experts of evaluation agencies [3]. The original data matrix, say **A** of dimension *m*x*n* and rank *k* is factorized into three unique matrices **U**, **V** and $\mathbf{W^H}$.

- **U** - Matrix of Left Singular Vectors of dimension *m*x*r*
- **V** - Diagonal matrix of dimension *r*x*r* containing singular values in decreasing order along the diagonal
- $\mathbf{W^H}$ - Matrix of Right Singular Vectors of dimension *n*x*r*. The Hermitian, or the conjugate transpose of **W** is taken, changing its dimension to *r*x*n* and hence the original dimension of the matrix is maintained after the matrix multiplication. In this case of Scientometrics, since the data is represented as a real matrix, Hermitian transpose is simply the transpose of **W**.

*r* is a very small number numerically representing the approximate rank of the matrix or the number of "concepts" in the data matrix **A**. *Concepts* refer to latent dimensions or latent factors showing the association between the singular values and individual components [3]. The choice of *r* plays a vital role in deciding the accuracy and computation time of the decomposition. If *r* is equal to *k*, then the SVD is said to be a Full Rank Decomposition of **A**. Truncated SVD or Reduced Rank Approximation of **A** is obtained by setting all but the first *r* largest singular values equal to zero and using the first *r* columns of **U** and **W** [Manyika et al.()Manyika, Chui, Brown, Bughin, Dobbs, Roxburgh, and Byers].

Therefore, choosing a higher value of *r* closer to *k* would give a more accurate approximation whereas a lower value would save a lot of computation time and increase efficiency.

### 10.2.3 *Regularization Norms*

In the case of Big Data, parsimony is central to variable and feature selection, which makes the data model more intelligible and less expensive in terms of processing.

$l_p$-norm of a matrix or vector **x**, represented as $||\mathbf{x}_p||$ is defined as, $||\mathbf{x}_p|| = \sqrt[p]{\Sigma_i |x|_i{}^p}$ i.e the $p^{th}$ root of summation of all the elements raised to the power p. Hence, by definition, $l_1$ norm $= ||\mathbf{x}||_1 = \Sigma_i |x|_i$

Sparse approximation, inducing structural sparsity as well as regularization is achieved by a number of norms, the most common ones being $l_1$ norm and the mixed group $l_1$-$l_q$ norm. The relative structure and position of the variable in the input vector, and hence the inter-relationship between the variables is inconsequential as a variable is chosen individually in $l_1$ regularization. Prior knowledge aids in improving the efficacy of estimation through these techniques.

The $l_1$ norm concurs to only the cardinality constraint and is unaware to any other information available about the patterns of non-zero coefficients.[1]

### 10.2.4 *Sparsity via the $l_1$ norm*

Most variable or feature selection problems are presented as combinatorial optimization problems. Such problems focus on selecting the optimal solution through a discrete, finite set of feasible solutions. Additionally, $l_1$ norm turns these problems to convex problems after dropping certain constraints from the overall optimization problem. This is known as convex relaxation. Convex problems classify as the class of problems in which the constraints are convex functions and the objective function is convex if minimizing, or concave if maximizing.

$l_1$ regularization for sparsity through supervised learning involves predicting a vector **y** from a set of usually reduced values/observations consisting a vector in the original data matrix **x**. This mapping function is often known as the hypothesis **h : x→y**. To achieve this, we assume there exists a joint probability distribution P(x,y) over **x** and **y** which helps us model anomalies like noise in the predictions.

In addition to this, another function known as a loss function L(y',y) is required to measure the difference in the prediction y'=h(x) from the true result y. Consider the resulting vectors consisting of the predicted value and the true value to be **y'** and **y** respectively. A characteristic called *Risk*, R(h) associated with loss function, and hence in turn with the hypothesis-h(x) is defined as the expectation of the loss function.

$$R(h) = \mathbf{E}[L(y', y)] = \int L(y', y)\, dP(x, y)$$

Thus, the hypothesis chosen for mapping should be such that the risk, R(h) is minimum. This refers to as risk minimization. However, in usual cases, the joint probability distribution of the problem in hand, P(x,y) is not known. So, an approximation called *empirical risk* is computed by taking the average of the loss function of all the observations. Empirical Risk is given by :

$$R_{emp}(h) = \frac{1}{n}\sum_{i=1}^{n} L(\mathbf{y'}_i, \mathbf{y}_i)$$

The empirical risk minimization principle states that the hypothesis(h') selected must be such it that reduces the empirical risk $R_{emp}(h)$:

$$h' = \min_{h} R_{emp}(h)$$

While mapping observations x in $n$ dimensional vector $\mathbf{x}$ to outputs y in vector $\mathbf{y}$, we consider $p$ pairs of data points - $(\mathbf{x}_i, \mathbf{y}_i) \in \mathbb{R}^n \times \mathbf{y}$ where i = 1,2...p.

Thus the optimization problem for the data matrix in Scientometrics takes the form:

$$\min_{\mathbf{w}\in\,\mathbb{R}^n} \frac{1}{p}\sum_{i=1}^{p} L(\mathbf{y'}_i, \mathbf{w}^T\mathbf{x}_i) + \lambda\Omega(\mathbf{w})$$

L is a loss function which can either be square loss for least squares regression, $L(y', y) = \frac{1}{2}(y' - y)^2$, or a logistic loss function. Now, the problem thus takes the form:

$$\min_{\mathbf{w}\in\mathbb{R}^n} ||\mathbf{y'} - \mathbf{A}\mathbf{w}||^2$$

Since the variables in the vector space/groups can overlap, it is ideal to choose $\Omega(\mathbf{w})$ to be a group norm for better predictive performance and structure. The $m$ rows of data matrix $\mathbf{A}$ are treated as vectors or groups($g$) of these variables, forming a partition equal to the vector dimension, [1:$n$]. If $\boldsymbol{G}$ is the set of all these groups and $d_g$ is a scalar weight indexed by each group $g$, the norm is said be a $l_1$-$l$–$q$ norm where $q \in [2,\infty)$. [1]

$$\Omega(\mathbf{w}) = \sum_{g\in\boldsymbol{G}} d_g ||\mathbf{w}_g||_q$$

The choice of the indexed weight $d_g$ is critical because it is responsible for the discrepancies of sizes between the groups. It must also compensate for the possible penalization of parameters which can increase due to high-dimensional scaling. The factors that affect

the selection are the choice of q in the group norm and the consistency that is expected of the result. In addition to this, accuracy and efficiency can be enhanced by weighing each coefficient in a group rather than weighing the entire group as a whole. The initial sparse data matrix is first manipulated using the $l_1$-norm. [1]

## 10.3 Methodology

An estimate of a journal's scholastic indices is necessary to judge its effective impact. The nuances of scientometric factors such as Total Citation Count and Self-citation Count come into play when deciding the impact of a journal. However, these factors unless considered in ideal circumstances don't by themselves become a good indicator to represent the importance of a journal. Many anomalies arise when considering these indices directly which may misrepresent or falsify a journal's true influence. The necessity to use these indices in context with a ranking algorithm is imperative to better utilize these indices. The resulting transformation of $l_1$-norms gives rise to a row matrix which is of the length equal to the number of features of the pristine Scientometric data. This row matrix effectively represents the entire dataset at any given iteration. The application of the Singular Value Decomposition operation on this row matrix is key in determining the necessary norm values to remove through a recursive approach.

The $singval$ array contains the Normalized Singular Values of all the individual $l_1$-norm transformed columns. These values act as scores while addressing the impact of any given journal. In the context of Singular Values the one with the lowest $singval$ score is the most influential journal. Utilizing these scores we can formulate a list of Journals which give preference to subtle factors such as high or low Citation Counts and give an appropriate ranking. Identifying the influential journals from a column norm and contrasting it with the Singular values is the equivalent of recursively eliminating the a low impact journal by comparing it's Singular Value to its Frobenius norm. This allows the algorithm to repeatedly eliminate the journals and find the score simultaneously to give a more judicious ranking system. Our method is different from the SCOPUS journal rank (SJR) algorithm. The SJR indicator computation uses an iterative algorithm that distributes prestige values among the journals until a steady-state solution is reached. The method is similar to eigen factor score [9] where the score is influenced by the size of the journal so that the score doubles when the journal doubles in size. Our method, on the contrary, adopts a recursive approach and doesn't assume initial prestige values. Therefore, the eigen factor approach may not be suitable for evaluating the short-term influence of peer-reviewed journals. In contrast, our

**Algorithm 1** Recursive $l_1$-norm SVD

```
A ← Input Transposed Feature Matrix A
procedure LASSO
    row_matrix ← Coefficents of Lasso Regression
    return row_matrix
procedure SVD
    U,Σ,V ← Matrices of SVD
    return Σ
procedure NORMALIZE
    Norm_Data ← Normalized using l1-norms
    return Norm_Data
procedure RECURSIVE
    L1_row ← LASSO(A)
    singval [] ← SVD(L1_row)
    Row_Norm ← Normalize(L1_row)
    Col_Norm ← Normalize(All columns of A)
    Col_i ← Closest Col_Norm Value to Row_Norm
    Delete Col_i from A
    goto RECURSIVE
```

method works well under such restrictions.

## 10.4 The Big Data Landscape

The appeal of modern-day computing is its flexibility to handle volumes of data through an aspect of coordination and integration. Advancements in Big Data frameworks and technologies has allowed us to break the barriers of memory constraints for computing and implement a more scalable approach to employ methods and algorithms. [5] The aforementioned journal ranking scheme is one such algorithm which thrives under the improvements made to scalability in Big Data. With optimized additions such as Apache Spark to the distributed computing family, the enactment of $l_1$ Regularization and Singular Value Decomposition has reached an all new height. Implementing the SVD algorithm with the help of Spark can not only improve spatial efficiency but temporal as well. The $l_1$-norm SVD scheme utilizes the SVD and regularization implementation of *ARPACK* and *LAPACK* libraries along with a cluster setup to enhance the speed of execution by a magnitude of at least three times depending on the configuration. Collecting data is also a very important aspect of Big Data topography. The necessity of a cluster based system is rendered useless without the requisite data to substantiate it. Scientometric data usually deals with properties of the journals such as Total Citation, Self-Citation etc. This data could be collected using

Web Scraping methodologies but also can be found by most journal ranking organizations, available for open source use; SCOPUS and SCIMAGO. For the $l_1$-norm SVD scheme, we used SCOPUS as it had an eclectic set of features which were deemed appropriate to showcase the effectiveness of the algorithm. The inclusion of the two important factors such as CiteScore and SJR indicators gave a better enhancement over just considering one over the other. For more information about the data and code used to develop this algorithm (please refer to [Aedula(2018)], Github repository of the project).

### 10.4.1 *Case Study*: Astronomy and Computing

SCOPUS and SCIMAGO hold some of the best journal ranking systems to this day, using their CiteScore and SJR indicators respectively to rank journals. However, due to the manner in which both these indicators are considered, it is often the case that the ranking might not display the true potential of a specified scientific journal. To demonstrate this we considered the case of the Journal *Astronomy and Computing* within the context of SCOPUS Journals in the relevant domain of Astronomy and Astrophysics.

The primary focus of this case study is to determine where the Journal *Astronomy and Computing* stand with respect other journals which were established prior to it. The algorithm also tests the validity of the ranking and suggests an alternative rank which used a more holistic approach towards the features.

Using the publicly available SCOPUS dataset we implemented the aforementioned $l_1$-norm SVD scheme to rank all its corresponding journals and simultaneously determine the potency of the algorithm. SCOPUS contains approximately around 46k Journals listed in different domains. Discarding few redundancies, SCOPUS effectively covers a large range of metrics and provides adequate resources for verification. For this demonstration, we have considered SCOPUS's 7 different metrics to be used as features in our algorithm. These features include *Citation Count, Scholarly Output, SNIP, SJR, CiteScore, Percentile and Percent Cited.*

To cross verify the results of the algorithm they were compared to SJR based ranking of SCIMAGO to articulate the discrepancies. The $l_1$-norm SVD scheme worked brilliantly in rating the journals and approached the data in a more wholesome sense. The result was a ranking system which ranked *Astronomy and Computing* much higher than most older journals and also at the same time highlighting the niche prominence of the particular journal. Similarly, this method also highlighted the rise of other journals which were underrepresented due to the usage of the aforementioned SCOPUS and SCIMAGO indicators. This was method was largely successful in rectifying the rank of such journals. This $l_1$-norm SVD scheme

| Journal Name | L1 Scheme Rank | SJR based Rank | Year |
|---|---|---|---|
| Astronomy and Computing | 39 | 31 | 2013 |
| Astronomy and Astrophysics Review | 40 | 5 | 1999 |
| Radiophysics and Quantum Electronics | 41 | 51 | 1969 |
| Solar System Research | 42 | 48 | 1999 |
| Living Reviews in Solar Physics | 43 | 3 | 2005 |
| Astrophysical Bulletin | 44 | 45 | 2010 |
| Journal of Astrophysics and Astronomy | 45 | 55 | 1999 |
| Revista Mexicana de Astronomia y Astrofisica | 46 | 23 | 1999 |
| Acta Astronomica | 47 | 20 | 1999 |
| Journal of the Korean Astronomical Society | 48 | 32 | 2009 |
| Cosmic Research | 49 | 58 | 1968 |
| Geophysical and Astrophysical Fluid Dynamics | 50 | 46 | 1999 |
| New Astronomy Reviews | 51 | 12 | 1999 |
| Kinematics and Physics of Celestial Bodies | 52 | 65 | 2009 |
| Astronomy and Geophysics | 53 | 67 | 1996 |
| Chinese Astronomy and Astrophysics | 54 | 72 | 1981 |

**Table 42:** Case Study: Astronomy and Computing, SJR and L1-SVD ranks

can be extrapolated to other data entries as well. It can also be used to study the impact of individual articles. Utilizing similar features such as Total Citation, Self Citation, and NLIQ. The algorithm can be used to rank articles within a journal with great accuracy along with a holistic consideration.

### 10.4.2 *Contrasting Performances of $l_1$ and $l_2$ Norms*

Being recursive in nature the Norm-based algorithms are subjected to some lapse while parallelizing its execution. However, they can be improved by using the right kind of suitable norm to enhance its running time. The decision of using $l_1$-norm over the $l_2$-norm was made because of a pragmatic choice for the following recursive scheme. The facet of the $l_1$-norm to use a loss function over the $l_2$-norm's squared data approach proves to be significantly better in structuring the data for a high-density computation. This type of method allows the overall dataset to reduce to a row matrix the size of the smallest dimension of the original data. This gives the added benefit of having a very consistent execution time and scale accordingly with the increase in data size.

| Norm | Time per row |
|---|---|
| $l_1$ Norm | 0.172s |
| $l_2$ Norm | 0.188s |

**Table 43:** Performance time for a row matrix of size 46k.

The execution time mentioned in Table 2 of this article gives the time-based performance of the different norms. This will only get significant with the increase in the size of the rows. This dereliction in parallelization can be compensated by the expected speed increase in the execution of the $l_1$-norm and SVD routines in a cluster setup. Optimized settings like Apache Spark which uses the aforementioned *LAPACK* and *ARPACK* libraries are able to boost the speed even further. The biggest benefit of opting such Big Data settings is that by increasing the size of the cluster the overall speed of the algorithm also scales appropriately.

| Data Framework | Overall Time |
|---|---|
| Python | 2hrs + |
| R | 58 mins |
| L1 SVD | 15 mins |

**Table 44:** Performance time for SVD of size 100k X 100k.

Table 3 indicates the performance time for the SVD algorithms in different ecosystems. The usage of SVD function in the algorithm to determine the individual singular values of the reduced row matrices of the columns can also be enhanced by using the corresponding Eigen Value optimization which are usually provided within the Big Data environment. Algorithms such as Lanczo's algorithm can not only enhance the speed of the operation but also can be very easily parallelized.

Hence, this combination of $l_1$-norm and SVD can effectively make the best version of algorithm; being fast in execution at the same time delivering a holistic approach.

## 10.5 Knowledge Discovery from Big Data Computing: The Evolution of ASCOM

Even though Astronomy and Computing (ASCOM) has been in publication for five years only, its reputation has grown quickly as seen from the ranking system proposed here. This is

despite the fact that ASCOM is severely handicapped in size. ASCOM is ranked 39 according to our method, slightly lower than its 31 rank in SCOPUS. This is due to the fact that we haven't used " citations from more prestigious journals" as a feature. Nonetheless, it is ranked higher than many of its peers which have been in publication over 20 years. This is also due to the fact that ASCOM is "one of its kind" and uniquely positioned in the scientific space shepherded by top notch editors. Such qualitative feature, regrettably is not visible from the big data landscape.

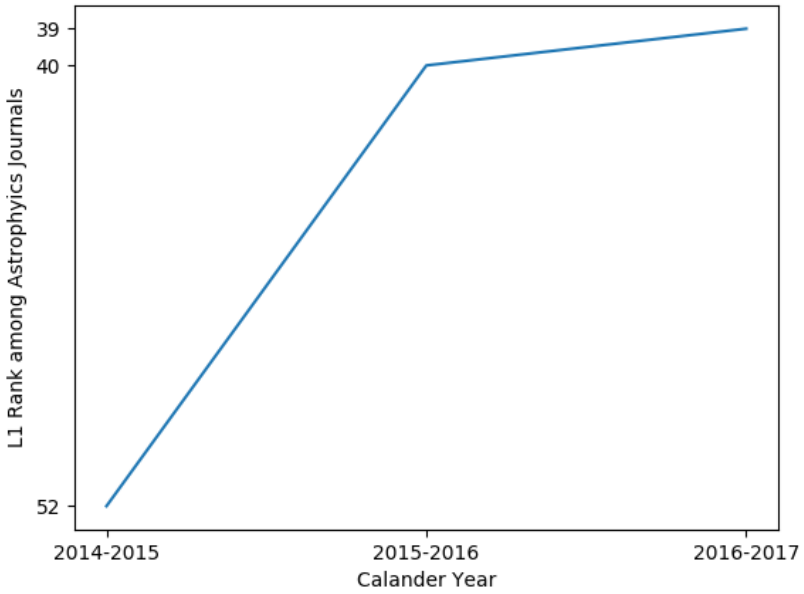



**Figure 22:** $l_1$ Rank Progression of ASCOM based on SCOPUS data computed by the proposed method. The steady ascendancy in the journal's rank is unmistakable. it will be interesting to investigate the behavior of the journal rank in the long run once enough data is gathered.

There is another interesting observation to take note of. By ignoring the "size does matter" paradigm, the ranks of some journals (many years in publication with proportionate volumes and issues) suffered. A few examples include Living Reviews in Solar Physics, ranked 43 according to our scheme while it is ranked 3 in SCOPUS and Astronomy and Astrophysics Review, ranked 40 in our scheme while it is ranked 5 according to SCOPUS. This is important as our goal was to investigate the standing of a journal relatively new and in a niche area. This indicates that years in publication may sometimes dominate over other quality indicators and may not capture the growth of journals in "short time windows". Our study also reveals that ASCOM is indeed a quality journal as far as early promise is concerned.

## 10.6 Conclusion

The Big Data abode adds a new dimension to the already existing domain of Machine Learning; where the computation aspect is as important as the algorithmic and operational facet. The $l_1$-norm SVD scheme does just that, it introduces a brand new way of ranking data by considering all the features to its entirety. The added benefit of optimizing the required norms and methodologies in terms of a Big Data domain suggests its vast flexibility in the area of Big Data Mining. This article covered its application in the Scientometric Domain. However it can be extended to any type of data, provided that the nuances are well understood. The aforementioned recursive methodology of the scheme allows us to carefully consider the important feature of the dataset and make prudent decisions based on the outcome of an iteration. This allows us to take a more wholesome approach which is very similar to the page rank algorithm which gives a specific importance to each one of the features under computation.

In the context of Scientometrics, this scheme is also applicable as a way to rank specific articles in a given journal with the result that their respective scholastic indices are available. We can conduct similar data experiments using indicators like *Total Citations, Self Citations* etc to categorize them of their various other features available for articles. We have also done some extensive studies based on the scholastic indices of the ACM journal whose case study lies outside the scope of this article and were able to successfully rank the corresponding journals and article. The scheme proved to be successful in evaluating the parameters with their nuances intact. More often than not, most Scientometric indicators do not apply to the journal being evaluated. As a consequence of this, the data matrix in which the rows represent the articles in the journal and the columns represent the different evaluation metrics is clearly sparse. Exploiting this sparsity, using certain structural sparsity inducing norms and applying recursive Singular Value Decomposition to eliminate metrics can make the process more efficient. Sparse approximation is ideal in such cases because although the data is represented as a matrix in a high-dimensional space, it can actually be obtained in some lower-dimensional subspace due to it being sparse.

With the ever-expanding necessity to process voluminous amounts of data, there needs to be a need to provide solutions which can adapt to the fluctuating technological climate. The $l_1$-norm SVD scheme tries to achieve similar potency, the usage of norm-based dimensionality reduction enhances the over-all efficiency on how we interpret data. The usage of techniques like sparsity norms suppresses outliers and only highlights the most meaningful data in store. The evolution of such methods will prove to be an absolute prerequisite in the future to compute copious amounts of data. Moving forward Dimensionality Reduction

based techniques will become the foundation of salient data identification and the $l_1$-norm SVD scheme is such a step along that direction.

## REFERENCES


[1] Francis Bach, Rodolphe Jenatton, Julien Mairal and Guillaume Obozinski, *Structured Sparsity through Convex Optimization, Statistical Science*. 27:450-468 (2011).

[2] Golub, G.H., Van Loan, C.F.: *Matrix Computations*. 3rd ed. Baltimore, MD: John Hopkins University (2012).

[3] Kalman D.: *A singularly valuable decomposition: The SVD of a matrix*. College Mathematics Journal 27:2âĂŞ23 (1996).

[4] Manyika, J., Chui, M., Brown, B., Bughin, J., Dobbs, R., Roxburgh, C. & Byers, A. H.: *Big Data: The Next Frontier for Innovation, Competition, and Productivity ()*. *McKinsey Global Institute* (2011)

[5] Ginde G,Aedula R, Saha S, Mathur A, Dey S R, Sampatrao G S, Sagar B.: *Big Data Acquisition, Preparation and Analysis using Apache Software Foundation Projects, Somani, A. (Ed.), Deka, G. (Ed.)., Big Data Analytics, New York: Chapman and Hall/CRC. (2017)*

[6] Bora, K., Saha, S., Agrawal, S., Safonova, M., Routh, S., Narasimhamurthy, A.: *CD-HPF: New Habitability Score Via Data Analytic Modeling, Astronomy and Computing*, 17, 129-143 (2016)

[7] Ginde, G., Saha, S., Mathur, A., Venkatagiri, S., Vadakkepat, S., Narasimhamurthy, A., B.S. Daya Sagar.: *ScientoBASE: A Framework and Model for Computing Scholastic Indicators of Non-Local Influence of Journals via Native Data Acquisition Algorithms, J. Scientometrics*, 107:1, 1-51 (2016)

[8] Aedula, R.: rahul-aedula95/L1_Norm, https://github.com/rahul-aedula95/L1_Norm.

[9] Ramin, S., Shirazi, A.S.: Comparison between Impact factor, SCImago journal rank indicator and Eigenfactor score of nuclear medicine journals. Nuclear Medicine Reviews. (2012).

# 11 Machine learning Based analysis of Gravitational Waves and classification of Exoplanets

## 11.1 Introduction

Gravitational waves (GW) are the ripples in space-time curvature [**?**]. In other words, GW are emitted when the mass quadrupole moment changes with time. Einstein's theory of general relativity first gave us a glimpse to the concept of Gravitational Waves, now after a span of 100 years with help of the extraordinary efforts of LIGO [**?**] and Virgo their existence has been confirmed. On February 11th 2016, LIGO announced its first discovery of Gravitational waves [**?**] corresponding to two black holes of 36 and 29 solar masses merging. So far at the time of writing this manuscript five such events of black hole mergers have been recorded along with a neutron star merger.

Gravitational waves are detected as strains in laser interferometers when it passes through it. These strains given by, $h(t) = \Delta L/L$ where $\Delta L$ is the change in length, is often converted into numerical relativity based waveforms to better interpret the source. Numerical relativity (NR) is a subsidiary of general relativity. It is often employed to study cosmological entities such as black holes and neutron stars. The principles of NR regarding GW for a binary pair however remain the same regardless of the type of entity which is being studied with some minor adjustments. This leads to a postulation that NR can be extrapolated to other entities, particularly weaker ones such as Exoplanets assuming that the necessary conditions of the source are met. In our case the binary in-spiral of black holes is extended to the Star-Planet Binary pair of an Exoplanet system [**?**], [**?**].

### 11.1.1 Premise and Solution

The primary problem with employing NR directly to weaker entities is that the complex nature of NR algorithms and its reliance on a converging point gives it an enormous overhead in terms of time complexity and therefore is ill-suited to obtain results quickly. Typically with the new advancements in reduced order models, the current simulation times go as long a couple of days. There is a significant necessity to find an optimized solution for generating NR waveforms, as NR waveforms give valuable insight about the source. Software Modules such as PyCBC make an attempt for finding approximated waveforms using semi-analytical models, which still have considerable overheads. Aside from the computational aspects of generating NR based waveforms for GW, there is also a need to use GW as a feature which can supplement the existing cohort of information for entities such as Exoplanets. This additional

information can provide valuable insight and address issues with detection of new exoplanets as GW unlike light waves have negligible absorption and dispersion, therefore retaining a large amount of information about the masses of the source objects.

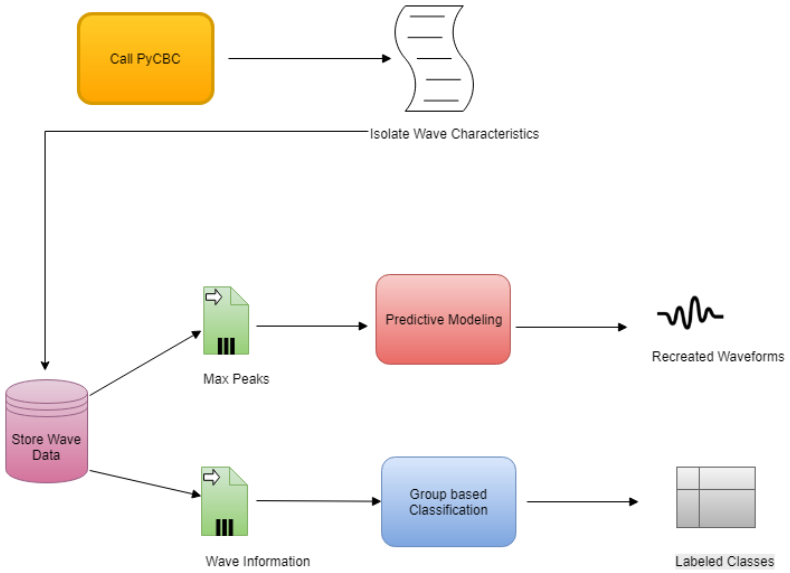



**Figure 23:** Road map

Utilizing a meticulously constructed dataset from semi-analytical model softwares such as PyCBC we can create a base to train our Machine Learning model. Given the correct assumptions this model will be able to get a good approximation for generating waveforms for weaker entities, while not accurate but is faster than the traditional methods. Methods based of Regression not only enhance the speed of generating waveforms by a great magnitude but also helps it in making it more consistent. Also, the added benefit of using the GW data for entities like Star-Planet system of Exoplanets gives a greater depth of understanding the intrinsic properties. Techniques such as Decision Trees and most importantly Random Forests show how well the Gravitational Waves data not only integrate with the already existing cohort of information, it also provides details about the GW frequency and spin.

### 11.1.2 Assumptions made to simplify computation

Extending Numerical Relativity using Machine Learning poses a few challenges when interpreting the results. Primarily, a direct one to one conversion is not possible, features such as ring down of the NR waveform are exclusive to that of a system of black holes. To address this issue we ensured that we used the appropriate adjustment and set the ring down to zero

after merger. The spin associated with the Star-planet pair is also assumed to be zero to ease calculations, since the spin of a planet or a star is of much lesser magnitude compared to that of a black hole. The Gravitational Wave frequency is taken as twice the orbital frequency of the planet around the star which is $2\omega$ where $\omega$ is the calculated orbital frequency [**?**]. The sampling rate of the simulated data is also set at a constant $del\_t = 1/4096$ for ease of generating data. Note that this can be customized to generate particular data depending on the required sampling rate. Lastly, since these mergers take an extraordinary amount of time this simulation only generates data in the last stages of coalescence, effectively finding the peak amplitude of coalescence on a time domain and use that peak amplitude in the classification. Therefore, the most important part of the waveform is the sinusoidal aspect pre-coalescence for information retrieval of the source.

## 11.2 Basic Properties and features of Gravitational waves

### 11.2.1 Physical Properties

In the context of physical nature, Gravitational waves cause the stretching and squeezing of matter in space, it also distorts time around the object causing a slowing and speeding up of time. GW also has polarization similar to light they are $(i)$ Plus and $(ii)$ Cross type polarizations respectively. This polarization is caused due to the precession of the binary in-spiral pair [**?**]. Gravitational Waves being ripples in Space-time propagate at the speed of light. The necessary conditions required for the propagation of GW is

$$\lambda << R$$

where $\lambda$ is the wavelength of the GW and $R$ is the Radius of Curvature (ROC) of the background space-time. Other properties such as absorption and dispersion are negligible in Gravitational Waves.

### 11.2.2 Wave Characteristics

Gravitational Waves can only be studied and discerned by their waveforms and not just the strain alone. It is not possible to map GW as a figure or a picture. Waveforms contains the details of the source. They can hold many attributes such as mass of the binary pair, GW frequency etc. Normally to decide these attributes it has to be compared with NR simulations which as mentioned before takes a long time even with the capabilities of existing super computers. To circumvent this problem, numerical relativity approximates are used. Numerical

relativity approximates are simulated waveforms which are done in a weak field paradigm. Under the assumption that the entities in question are not moving fast enough or to better phrase it moving slower than the speed of light. This application of Einstein's equations in a weak field paradigm is also called Post Newtonian Expansion.

$$h(f) = \frac{1}{r} M_{ch}^{5/6} f^{-7/6} exp(i\psi(f)) \tag{80}$$

$$\psi(f) = 2\pi f t_c - \phi_c - \frac{\pi}{4} + \frac{3}{128}(\pi M_{ch} f)^{-}5/3 \tag{81}$$

Here $t_c$ is time at coalescence, $\phi_c$ phase at coalescence and $M_{ch}$ is chirp mass which will be discussed in the upcoming sections. The $h$ is a first order approximation of the strain. Both these equations correspond to the frequency domain. A few adjustments can be made regarding the location of the source but for a basic scenario these formula can be generalized [**?**]. These formula give insight into how these source attributes can be obtained by the wave.

### 11.2.3 Existing computational approximation methods

As previously mentioned NR takes a lengthy amount of time to approach generalized solutions. To ease the arduous computational task approximation is used. This provides an optimized solution with the help of the aforementioned Post Newtonian Expansion methods. A common type of waveform is that of phenomenological waveform or phenom waveform [**?**]. These type of waveforms have shown a very successful rate in mimicking NR waveforms as closely as possible. It has some inaccuracies as it has come to be expected, because of the approximated nature of its generation but over all efficiency has been significantly high compared to most approximates. One such existing methods which uses phenom [**?**] based waveforms is PyCBC[**?**]. **PyCBC** - an open source python implemented stable module could be used to obtain the theoretical gravitational waveforms for specific input parameters such as the masses, lower frequency and so on. This powerful module gives an optimized solution to theoretical equations by means of Bayesian Belief Networks and computes different types of gravitational waveform such as SEOBNR, TAYLOR and many more. The one problem with PyCBC is that even though it is an optimized solution it can only give results for black holes and other heavy entities like neutron stars. GW cannot be studied effectively regarding other lesser mass entities like Exoplanets etc which are also in in-spiral with their corresponding star. Through the course of this paper we will be extrapolating the approach used for black holes on multiple Star-Exoplanet Systems. The aforementioned SEOBNR (Spin Effective One

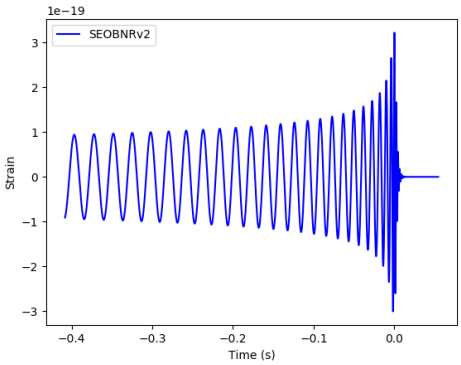



**Figure 24:** PyCBC generated SEOBNR waveform for black holes

Body Numerical Relativity) is a phenom based waveform which will be the primary focus as we progress with this article.

### 11.2.4 Proposed Computational Approach

The problem lies in creating a flexible and elegant solution by discerning both aspects, the waveform and the germane physics required to correlate the idea to other weaker entities. On one hand there has to be a lucid interpretation of the waveform, that is to say there should be clear understanding of how the trend of the waveform changes with respect to different parameters which influence it and on the other hand the proposed astrophysical model should not only validate but also make inferences by a supervised learning procedure. The way to go about achieving these goals is to approach the problem in two simultaneous subroutines.
These proposed mechanisms are :

- Regression Analysis
- Classification

Regression Analysis[**?**] deals with discerning the trend of the SEOBNR waveform. It tries to correlate the various parameters that are inclusive of the generation of these waveforms and tries to have a pellucid grasp on how it can be made computationally efficient and deals with the process of extrapolation outside the domain of its limiting factors. Classification aims to propose a new model to group these entities in question with the help of GW. Not only does such a model not exist so far but also it helps in validating and correcting the regression results in a peculiar way. The upcoming sections of this article delves deep into

these subroutines and aids in surfacing a new computational model which is created by mending these approaches.

## 11.3 Regression Analysis

The previous section speaks about the PyCBC module and why it is marked to be of utmost importance in the study of GW. It is a robust module to perform tasks at various levels studying GW, one such task is the generation of GW based on numerical relativistic equations[**?**]. The module also presents diverse waveforms to pick for study, ranging from the Taylor series representation to the SEOBNR[**?**] and many more. An important observation here is - the numerical relativistic equations or the theoretical equations involved behind the generation of these waveforms are intrinsic by nature. This intrinsic property results in a lot of computational overhead while using the module for some specific values supplied as part of functional requirements such as the masses of the celestial objects for the generation of GW. An other observation is that, beyond certain limit of the input parameters such as the masses of celestial bodies in-spiral, PyCBC fails to compute values and generate waveforms, though in reality a similar in-spiral system would naturally generate a GW. This paper aims to introduce few basic concepts of ML[**?**], Regression and analyze their contribution to bring down the computational overhead and extend the domain of the input parameters while trading off the accuracy of the waveform generated by a small amount. This trade off should be meager, shouldn't interfere producing a result too deviated from the theoretical result. The most interesting phase of in-spiral is that of the Coalescence. As discussed in the previous sections about the in-spiral, there is a certain period of time after which they gain rapid acceleration and spin vehemently about each other. This is followed by both of the celestial bodies colliding and thus merging into a single body which is a common phenomenon in binary Black holes. The start of this merger is marked by the coalescence. The peak amplitude of the GW happens at the coalescence and plays a vital role to study the properties of celestial bodies involved in generating this peak amplitude.

The above Fig. depicts a simple PyCBC generated waveform for the parameters $m_1 = 10$, $m_2 = 10$, $spin1z = 0.0$, $delta\_t = 1.0/4096$ and $f\_lower = 60$. The peak amplitude during coalescence ($h_0$) as seen from the waveform occurs at $Time = 0.00$

To understand how the peak amplitude varies with various masses of the celestial bodies in-spiral, a huge dataset was created with a mixture of masses, recording the peak amplitudes during coalescence from the PyCBC module. In fact, the peak amplitude was recorded against

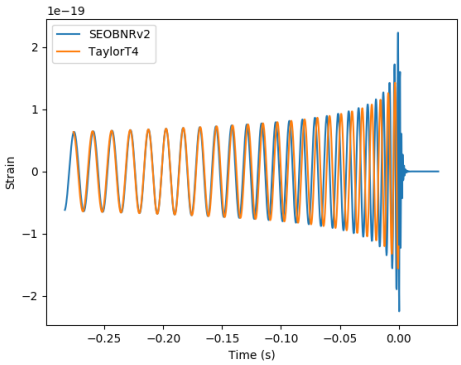



**Figure 25:** A sample PyCBC Waveform for blackholes of masses 10 and 10

the chirp mass which is given by,

$$M_{ch} = \frac{(m_1 m_2)^{3/5}}{(m_1 + m_2)^{1/5}}$$

where $m_1$, $m_2$ correspond to the masses of the celestial bodies with $m_1$ being the mass of the more massive body among the two. The generation of the dataset involves a lot of time as the generation of the waveform even for a single input requires a lot of time. Thus, parallelization was used with the help of Multiprocessing module in python to build this dataset using all cores of the CPU. The dataset created comprises peak amplitudes during coalescence (SEOBNR) of celestial bodies recorded for different values of masses $m_1$, $m_2$ and lower frequency $f$ as follows:

| parameter | Range |
|---|---|
| $m_1$ | 10 - 99 |
| $m_2$ | 10 - $m_1$ |
| $f$ | $\{35, 40, 45, ...60\}$ |
| spz | $\approx 0$ (very negligible) |
| del | $6.103515625E-05$ |

It has to be noted that the peak amplitude during coalescence for any input lower frequency got to be equal in magnitude, but for PyCBC generated waveforms the peak amplitudes are not strictly equal though approximately equal. Thus, the lower frequency is also considered while generating the data set. For the preliminary analysis on the relation between the peak amplitude during coalescence against the masses of the celestial bodies, a scatter plot is plotted with these parameters.

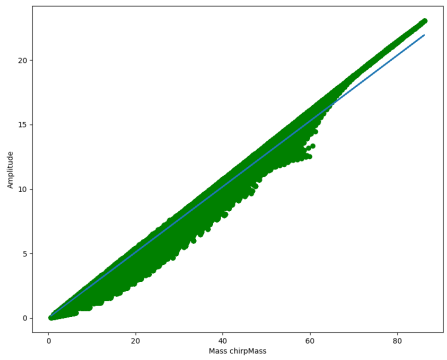



**Figure 26:** Regression on amplitude peaks

From the above Fig. it could be observed that the scatter plot approximates to a linear curve. The x-axis corresponds to $M_c$ while the y-axis corresponds to $h_0 \times 10^{19}$. The linear model of this data set could be imagined to be

$$h = \beta_0 + \beta_1 M_{ch} + \epsilon$$

In the above equation, $\beta_0$ and $\beta_1$ is some coefficient and $\epsilon$ the error. The blue line in the above equation is a linear fit which could be calculated using the Sum of squares method, minimizing the sum of the squared errors. Thus, for the linear fit $\hat{h} = \hat{\beta}_0 + \hat{\beta}_1 M_{ch}$, we obtained the parameter $\hat{\beta}1 = 0.25464428$. The term $\hat{\beta}_0$ or the intercept is zero, since it is logical that when $M_c = 0$, $h_0 = 0$. It is now possible that we make use of this model to obtain $h_0$ for other celestial bodies such as the Exoplanets, which could in turn be used for the classification of Exoplanets as discussed in the later sections of this paper.

## 11.4 Complete Waveform generation

In the previous sections we have obtained a model to predict $h_0$ for any given $M_{ch}$. In this section we discuss about an approach which could be used in the generation of complete waveform. The approach taken here does not generate the exact waveform but an approximation of the waveform. Generating the complete waveform would result in a lot of GW characteristics which have potential applications in many fields of astronomy. As we can observe from the PyCBC waveform, output for SEOBNR, it is evident that the amplitude versus time waveform is a chirp equation whereby the frequency increases with time.
To generate a waveform with unique characteristics, the waveform could be identified with the amplitude, the frequency and its phase. Now, to generate the GW waveform envelope

we need to have at least two characteristics, the amplitude and the frequency. The reason being that, the peak amplitude during coalescence always occurs at the time = 0. It could be observed that the amplitude peaks versus time of a gravitational wave could be split into two parts. By observation, it could be concluded that in the first part of the GW the time increases exponentially with the amplitudes peaks. i.e., the relation between amplitude peaks and time is with time $t$ and amplitude $h$ follows:

$$t = e_a.e^{h.e_b} + e_c$$

The coefficients $e_a$, $e_b$ and $e_c$ of this model cannot be obtained analytically and hence we resort to use approximation algorithms to obtain these values. For obtaining these values we make use of the $curve\_fit()$ method inside $scipy.optimize$. The initial guesses for these constants were set to $(-1,-1,-1)$ Now, $h_{max}$ or the envelope could be obtained from $t$ just using

$$h_{max} = \left(\frac{1}{e_b}\right).ln(\frac{t-e_c}{e_a})$$

The fit of the curve looks as depicted in Fig. 4

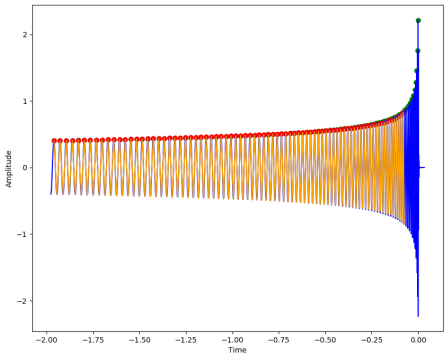



**Figure 27:** PyCBC generated SEOBNR waveform

For the fit in th, the points marked in the red color, forming an envelope above are the predicted peaks. The curve fits pretty well for the amplitude peaks, but as we could see, reversing the equation presents two issues viz. the first thing the domain error and the second thing that the predicted values increase rapidly after a certain interval of time. It could be defined that the time interval up to which neither of the issues occur as the pre-coalescence phase or the non-coalescence phase. The time interval after which either of the two issues occur could be defined as the coalescence phase, where by the current exponential

model fails to fit the model and hence we would have to go with another model which could be a model such as:

$$t = x_a.h^{x_b}$$

Reversing the above equation for amplitude gives,

$$h_{max} = \left(\frac{t}{x_a}\right)^{\frac{1}{x_b}}$$

It could be seen from the fit that the model fits perfectly for the values of time after the non-coalescence period. Fig. 5 shows the fit of the amplitude peaks for the time after the non-coalesce period or during the coalescence period.

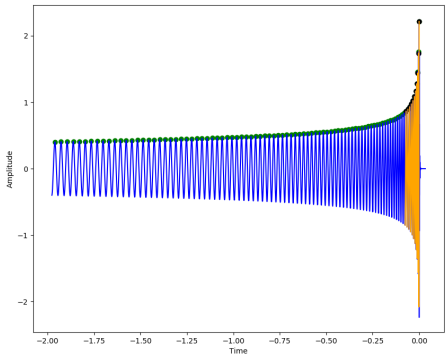



**Figure 28:** PyCBC generated SEOBNR waveform for a lower mass couple

The points in black color, forming an envelope towards coalescence are the predicted peaks during the coalescence phase.

An envelope around the graph peaks could be obtained by using the models for both the positive peaks as well as the negative peaks. Now in order to obtain this envelope for all the GW for given masses and lower frequency, we need to run a regression analysis on how these Model parameters viz. $e_a$, $e_b$, $e_c$, $x_a$ and $x_b$ vary with respect to masses and the frequency. The images - Fig. 6 to Fig. 9 show 3D scatter plot of the Model parameters with respect to chirp mass and frequency for the non-coalesce phase. The points green in color are the predicted points while the points in blue color are the observed points. From the below images we could infer that the distribution of these parameters against $M_{ch}$ and $f$ is exponential.

It could be visually confirmed that the relation of $e_a$ with chirp mass and the frequency is exponential, in fact it has a score of roughly about 98.5 percent. Similarly, $e_b$ and $e_c$ follow

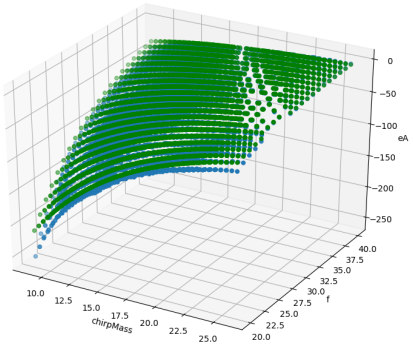



**Figure 29:** $e_a$ against $M_{ch}$ and $f$

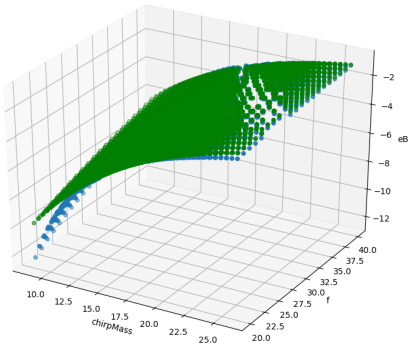



**Figure 30:** $e_b$ against $M_{ch}$ and $f$

an exponential distribution. The image of $x_a$ against $M_{ch}$ and $f$ shows a 3D scatter plot of the Model parameters with respect to the chirp mass and the frequency for the coalescence phase.

It could be visually confirmed that the relation of $x_b$ with chirp mass and the frequency also follows an exponential fit. But the relation of $x_b$ with respect to the chirp mass and frequency is slightly scattered. Thus, for this we could take a density estimate. The reason for this being that the scatter plot becomes uniform as the plot approaches toward the lower masses and a lower frequency but still the regression line passes through the mean density of the scatter.

After we are able to obtain an estimate of these model parameters, we could again obtain

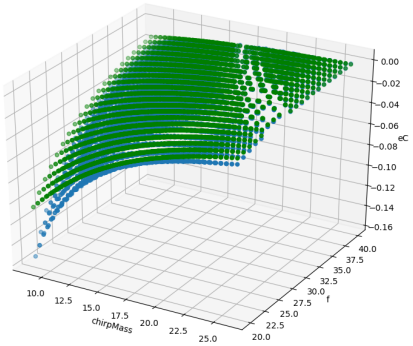



**Figure 31:** $e_c$ against $M_{ch}$ and $f$

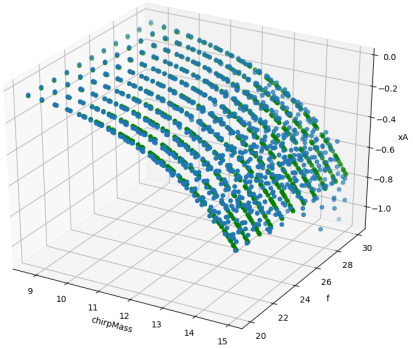



**Figure 32:** $x_a$ against $M_{ch}$ and $f$

an envelope of the amplitude for the time of a GW for a given chirp mass and a given frequency by just using a version of gradient descent on the equations up to the lower frequency.

The procedure for construction of the envelope goes as follows:

1. Compute $h_0$
2. Initial time,

$$t = x_a.h^{x_b}$$

$$\frac{dt}{dh} = x_a.x_b.h^{x_b-1}$$

3. Use $\frac{dt}{dh}$ for new time,

$$t_{new} = t_{old} - \frac{dt_{old}}{dh}$$

4. Compute $h_{new}$

$$h_{new} = \left(\frac{t_{new}}{x_a}\right)^{1/x_b}$$

5. Compute $f_c$ (current frequency)

$$f_c = |\frac{1}{t_{new} - t_{old}}|$$

if $f_c <= f_{lower}$: stop else if:

$$h' = \frac{1}{e_b} \ln\left(\frac{t - e_c}{e_a}\right) > h_{new}$$

then goto step 3 else: continue

6.

$$\frac{dt}{dh} = e_a.e_b.e^{e_b.h}$$

7.

$$t_{new} = t_{old} - \frac{dt_{old}}{dh_{old}}$$

8.

$$h_{new} = \frac{1}{e_b} \ln\left(\frac{t - e_c}{e_a}\right)$$

9. if

$$f_c = |\frac{1}{t_{new} - t_{old}}| <= f_{lower}$$

then stop else goto step 3.

Now that the envelope of the wave would be generated the next question would be on identifying the points at which the envelope of the wave has to be considered to roughly approximate the original wave. For this a simple scatter plot between the chronological order of positive peaks of amplitude versus time gives us a hyperbolic curve as shown in Fig. 10

Thus, going with a hyperbolic fit, the right lower part of the equation

$$\frac{t^2}{a^2} - \frac{i^2}{b^2} = 1$$

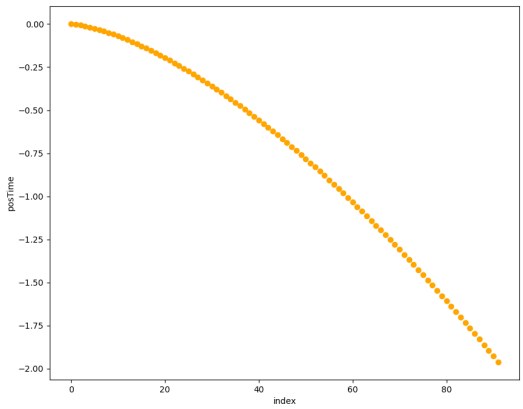



**Figure 33:** index vs time (positive peaks

where $t$ corresponds to the positive time and $i$ corresponds to the chronological order or index of the positive peaks. The next section deals with how the predicted amplitude during coalescence could be applied for exo-planets and help in their classification.

## 11.5 Gravitational Waves Based Classification

### 11.5.1 Need for Classification

Classification is the grouping of entities with similar attributes under a defined class label. The necessity for classification is that it serves two purposes, it validates the astrophysical model and corroborates all the assumptions made so far in extrapolating the model outside its usual domain, but also it helps in a subtle form of corrigible operations which corrects any types of inconsistency that may occur due to the regression analysis. Primarily it is used to bolster the claims made so far that the model can indeed be flexible enough to operate outside the domain. The center of focus in this section is dealing with the implications of extrapolating the already existing GW concepts to weaker entities, followed by some computational evidence that supports the usage of the GW concepts in this manner. The weaker entities under scrutiny in this scenario are Star-planet system of Exoplanets. The reason to choose this particular set of celestial objects is because they follow the in-spiral mechanisms which is similar to black holes. As mentioned earlier some necessary adjustments have been made such as disregarding ring down so as to keep it as relevant to star-planet system as possible. The revolution of an Exoplanet around their respective star which is considerably far away behave as a weak pair of binary coalescing black holes. The term "weak pair" here

describes the reduced magnitude of mass compared to that of a black hole. The assumption is that coalescence takes place a few million years later rather than quickly like that of two merging black holes. Newtonian mechanics dictate that the gravitational orbits are stable and once a celestial body such as a planet enters into orbit it remains in revolution forever. But the introduction of Einstein's general relativity showed otherwise, indeed there is a decay in the gravitational orbits over time and this decay of energy is emitted out in the form of Gravitational Waves. This Gravitational waves can be observed in planets the problem being that the wave itself is too weak to be detected by any conventional detectors like LIGO. The reason being that a planet system would emit a GW due to the motion of masses which are many times weaker than that of black holes, particularly LIGO is currently more tuned to extract GW information for only black holes. This means that to practically detect these waves from such small sources would probably take more calibration on LIGO's end but that doesn't say anything about interpreting the theoretical results. These theoretical results can assist in creating a more suitable model to correlate weak mass entities with GW. The waveforms are well discerned in the previous section Regression Analysis, the results generated for each unique wave can be used as aids to help create a more pellucid model. Results such as the maximum peak amplitude of the supposed coalescence, the GW frequency and the known mass of the planets all help in creating a better classification model.

### 11.5.2 Classification Overview

Gravitational waves have a lot of factors which influence them but none more prominent than mass. As mentioned in the previous sections the necessity of Chirp Mass in extracting important information about the source is essential. When GW is so prominently influenced by mass, it can also be used as an aid to help better sort the masses of the entities themselves. The mass class is a parameter defined by astronomers used to classify various Exoplanets by virtue of their mass. These masses have a defined set of constraining values which segregate them to their corresponding mass classes. The numeric mass class are the integer allocated values of the respective mass class done for easy graphic representation. The key idea here is to relate the different masses of these Exoplanets to the Gravitational waves they may produce while orbiting their corresponding star. A supervised machine learning model can be proposed on the grounds of training data with already confirmed masses along with their respective mass classes to that of the supposed Gravitational Wave information. The Regression Analysis section shows the various wave information that can be extracted by using various predictive modeling techniques. Although to reach this coalescence point where there is an immense burst of GW from the corresponding potential merger it would

| Mass Class | Numerical Mass Class(NMC) |
|---|---|
| Jovian | 1 |
| Terran | 2 |
| Superterran | 3 |
| Subrterran | 4 |
| Neptunian | 5 |
| Mercurian | 6 |

**Table 45:** Planet Mass Classes, *NMC is used for denoting the mass classes in the classification algorithm

take a considerable amount of time, at least a few billion years. However, this potential maximum peak amplitude (GWPeakAmp) can be used as a feature in classification of these Exoplanets. This is because the maximum peak still denotes a unique point in the NR waveform. It can be used to uniquely identify a binary pair based on the Chirp Mass, thereby making it a trainable feature in the classification model. Another feature which is used to train the classification algorithm is the Gravitational Wave frequency (GWFrequency). The frequency of the wave itself can tell us lot about the source, it can correlate to the orbital dynamics of the binary pair of the star-planet system of the Exoplanet. The features which determine the outcome of Machine learning based classification are as follows:

- GWPeakAmp
- SunMassSU
- PlanetMassSU
- GWFrequency

These features provide a rudimentary extension to the already existing mass class grouping mechanism, so by associating them we can not only increase the accuracy of the classification but also provide a new model, a catalog which encompasses Gravitational Waves along with the existing features in the Exoplanet Catalog.

### 11.5.3 Exoplanet Catalog Dataset

The application of Gravitational Waves makes it a requisite to use the most reliable catalog to extrapolate the idea. One such catalog is the Planetary Habitability Laboratory Exoplanet Catalog (PHL-EC) by the University of Puerto Rico. Although this dataset is known for its

study for habitability of planets the usage of the data set in this scenario is very different. The PHL-EC data set is being used only to derive the orbital mechanics related information such as orbital frequency and the corresponding masses of the planets. The mass classes are also a part of this robust dataset.

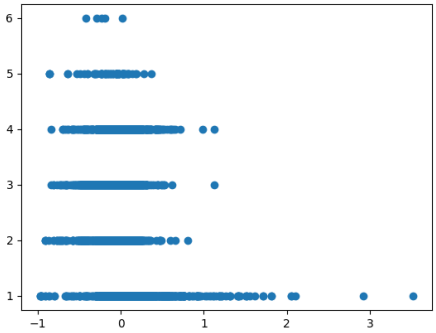



**Figure 34:** Dimensionally reduced data distribution

Although, there might exist some inconsistencies when masses are taken into account, such as compression of gravity and the scaling of mass with volume. All these factors might cause a skew in the data set for some of the mass classes. The dimensionally reduced distribution figure Jovian class for example contains the most varying range of possible planets, but there may be cases in the data set where some of the observations are misclassified. For this reason there is no hard limit set on the mass classes. This problem can be overcome by our classification strategy by incorporating the Gravitational Waves as a feature giving more clarity and distinction to the mass classes. Another recognizable problem with the dataset is that it is unbalanced. This may cause problems with the algorithm as it causes a bias in the ML model. The reason a bias like this might exist is because if there are more samples present in the training data which belong to a particular class compared to the other classes, this will cause the algorithm to identify the majority sample class with more efficiency compared to that of the minority sample class. This type of imbalance problems can be resolved in various over-sampling or under-sampling techniques. The next section will cover these methods in more detail.

### 11.5.4 Oversampling using SMOTE

The two methods which are employed to tackle class imbalance problems are $(i)$ Oversampling and $(ii)$ under-sampling techniques. Over-sampling techniques include procedures to artificially generate data to fill the gaps so as to minimize then imbalance, where as Under-

sampling includes using only part of the data to which all the classes are balanced, without using the entire dataset. For this specific GW based classification problem, the better choice was over-sampling. This was due to the high amounts of samples for the majority of the classes making it easier to generate data with more accuracy. Another reason for performing oversampling is that future data which has yet to be observed and recorded has not been accounted for. The generation of synthetic data will help test the model in more robust conditions. There are many well known oversampling techniques such as ADASYN etc, but for this problem, Synthetic Minority Over-Sampling Technique (SMOTE) was the superior method. Other methods such as Kernel Density Estimation (non parametric methods), ADASYN were also evaluated, but SMOTE provided the most agreeable values. This is due to the inherent nature of the dataset. There was a necessity that the synthetically generated values for the planet have to resemble actual observable values. With a wide range of values for most of the planet masses, it was imperative that the values have to be in the plausible range. The Synthetic data has to also maintain it's integrity with that of the original PHL-EC data. SMOTE does exactly this, it provides values in an aggregated range (close to the mean) and also manages to maintain the magnitudes of the parameters within domain constraints. Thus while performing classification the data under operation will be as close to the probable values. Thus solving the imbalance problem.

### 11.5.5 Classification Strategy using Random Forests

While the imbalance problem has been conquered by oversampling methods, classifying the data and the features as mentioned in the previous section is a challenge all to its own. Primarily, a clear distinction of mass class has to be established. As mentioned before there are some small amount of inconsistencies in the dataset. The classification algorithm must take this nature of data into consideration and should provide an improvement. The following features are an integral part in determining the exact outcome of the mass class: ($i$) GWPeakAmp ($ii$) SunMassSU ($iii$) PlanetMassSU ($iv$) GWFrequency. The primary feature which has the highest correlation and can contribute significantly to finding the mass class is without a doubt is PlanetMassSU or in this case PMass according to Figure 13. Since the entire classification revolves around the mass classification of exoplanets PMass becomes the contributing factor. However, the primary purpose of this whole model is to use Gravitational Wave data. Because of the aforementioned inconsistencies in section 5.3 we know that just classifying with the mass of the planet fails to identify the exact mass class. Hence utilizing features such as GWPeakAmp and GWFrequency we can make a more accurate prediction. The binary nature of the source requiring a binary in-spiral pair means that there might be

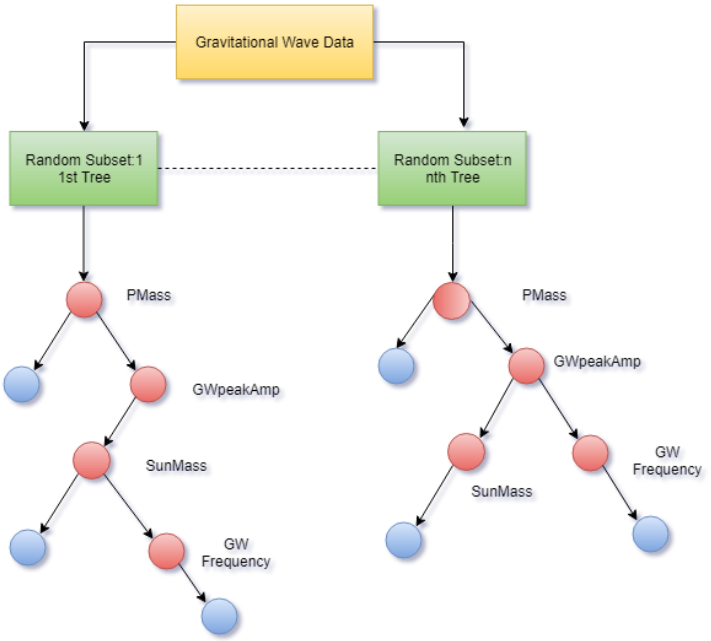



**Figure 35:** Random Forests on Gravitational Wave Data

some certain combinations of source masses which could produce the same GWPeakAmp. For this reason including features such as SunMass can help in classifying the star-exoplanet pair more accurately. The appropriate algorithm which satisfactorily tends to these nuances is **Random Forests**. Random Forests being based on decision trees, finds the boundaries of the classes. This is needed as previously stated, all the mass classes have to be uniquely identified and should provide the best accuracy for each class. Under ideal circumstances such an algorithm would be more than perfect to reliably complete the task. The reasons for choosing Random Forests over Decision Trees is because Random Forests uses a significant amount of voting based conclusions as compared to that of Decision Trees. It runs a bagging based routine by using a large number of de-correlated Decision Trees to classify a predicted class. This course of operations is highly suitable for the GW data and it's associated mass classification as it meticulously examines the feature space to make better judgments over which mass class to finalize as the expected result.

### 11.5.6 Classification Performance Metrics

The accuracy metrics decide how efficiently the model has performed over a set of constraining factors. In the case of classification, it could be simply defined as the accuracy involved in predicting the correct class for a given set of parameters. The results show various metrics

such as True Positive Rate (TPR) or Sensitivity and True Negative Rate (TNR) Specificity etc. These metrics give valuable insight into how each class is being treated and gives a more lucid interpretation of all the nuances of the data. The following results displayed consists metric scores of both classification without using SMOTE and classification after using SMOTE.

| PM | OD | OSD |
|---|---|---|
| Overall Accuracy | 89.4505% | 84.9932% |
| 95% CI | (0.8625 , 0.9211) | (0.8325, 0.8651) |
| Kappa | 0.8488 | 0.8184 |

**Table 46:** Overall Metrics:PM = Performance Metrics; Original Data = OD; Original + Synthetic Data = OSD

The scores mentioned in Table III show the overall performance of the Random Forest classifier. As the results indicate there is a drop in the overall accuracy in the results of the SMOTE generated data. This is because the classifier is trained without any imbalance and hence showing a decrease in accuracy. But this classifier trained without imbalance is more robust compared to that of the classifier which trained only on the original data. The higher accuracy score of the original data can only be attributed to the skew in the data. Because of imbalance the classifier tends to recognize class 1 (Jovian) more than any other class. Also, in the actual data set the number of planets in the class 6 or Mercurian category are very less which is visible in its peculiar class wise scores. This also factors into the classification algorithm's accuracy.

| | Specificity | | Sensitivity | | Accuracy | | F1 Score | |
|---|---|---|---|---|---|---|---|---|
| NMC | OD | OSD | OD | OSD | OD | OSD | OD | OSD |
| 1 | 0.99 | 0.98 | 1.00 | 0.99 | 0.99 | 0.99 | 0.99 | 0.98 |
| 2 | 0.89 | 0.88 | 0.99 | 0.97 | 0.97 | 0.95 | 0.92 | 0.83 |
| 3 | 0.70 | 0.73 | 0.96 | 0.92 | 0.92 | 0.89 | 0.74 | 0.71 |
| 4 | 0.78 | 0.54 | 0.92 | 0.93 | 0.90 | 0.87 | 0.69 | 0.56 |
| 5 | 0.88 | 0.91 | 0.99 | 0.99 | 0.99 | 0.97 | 0.91 | 0.93 |
| 6 | 0 | 0.90 | 1.00 | 0.99 | 0.99 | 0.99 | 0.000 | 0.98 |

**Table 47:** Class wise Performance Metrics: Original Data = OD; Original + Synthetic Data = OSD

The class-wise scores give a better insight on how the classifier is handling each class separately. Understanding the variations in Specificity and Sensitivity are key in discerning how to efficiently boost up the classification model. As shown in Table IV the class wise scores,

the over all performance of the synthetic model improved compared to it's counterpart. This goes to show that even though overall accuracy is higher, the robustness of model might not be prominent. The model built from using SMOTE and then applying Random forests might score less in the overall accuracy (not by much), the robustness of the model is far more superior. It has validated that it can handle the data it may have to process in the future and show promising results.

## 11.6 CONCLUSIONS

Gravitational waves and its significance has just started emerging to the forefront. This manuscript has taken steps in a creative direction of applying these physical phenomenon to other weaker entities. The proposed computational model serves as a rudimentary approach to a far more perplexing design. While nowhere do we claim in this article that we are reinventing Gravitational Wave physics but what is being provided is an automated efficient solution through the lens of Statistics. The strongest aspect of this model is the use of Machine Learning tools to not only ease the understanding of this complex phenomenon but also making it efficient to operate with it. The perspective of Data Science and Machine Learning is that of elegance. Physicists already understand a vast amount about Gravitational Waves. Data Science not only enhances that plethora of knowledge but also gives a unique outlook. An eloquent solution to generalize one of the most arcane paradigms of the universe. The progress made in this article is a stepping stone for more elaborate models yet to be made. The understanding of Gravitational waves is absolutely vital in advancement of sciences. It's correlation with some of the most enigmatic entities like black holes make it all the more reason to delve deep to discern them. In search of these answers the proposed approaches suggested in this paper is the understanding of these waveforms and extrapolating the information learned from these trends to far outside the domain. Our understanding of the waveform and how it behaves over variation of various parameters such as mass and frequency has been enhanced. The application of this knowledge outside the usual domain is a step on the creative side of the paradigm. It led us to develop an efficient way of producing waveforms, and also a new method for classifying Exoplanets based on the GW released by the star-planet binary system, maturing a computational model comprising the two which enhance each other. The application of this model lies in optimization algorithms to generate waveforms and also catalog extraction which incorporates GW with Exoplanets.

As mentioned before, both our exoplanet data and our base PyCBC data were conditioned with the required orbital frequency to ensure that our case study is appropriate for numerical

relativity. As far as Hubble time is concerned, our use of PyCBC is only to generate hypothetical waveforms which are weak to detect and would otherwise take a couple of days to simulate in under a few hours. Our methods are purely based on the semi analytical models and is a novel attempt to integrate Machine Learning into GWs. We never claimed anywhere these are the most accurate waveforms that can be produced. However, it is the most efficient semi analytical model that is out there by our estimates.

## 11.7 Future Scope

This proposed computational model is the first of many approaches that will unfold in coming years, hopefully. To ensure simplicity, lot of assumptions were made to ease the computational aspects. Section 1.2 discusses this in detail. Most of the these assumptions were made using baseline conditions. With better understanding, a more refined model can be created by considering a set of more elaborate factors. The proposed classification model may use more discernible features and make the classification more robust. Fine tuning the features in this data set and adding more depth to the assumed features can not only add a higher degree of accuracy to the model but also might assist in knowledge discovery. Better calibration of the LIGO sensors can help in physically validating the proposed waveforms.

# 12 Time Reversed Delay Differential Equation Based Modeling Of Journal Influence In An Emerging Area

## 12.1 Introduction

It is well-known that ranking of journals, whether in science, technology, engineering or in social sciences, such as Economics, is a contentious issue. For many subjects, there is no correct ranking, but a universe of rankings, each a result of subjective criteria included by its creators. In this regard, the following studies are instructive: ([Engemann and Wall(2009)], [Jangid et al.(2014)Jangid, Saha, Gupta, and Rao]). With the creators' choices and rules laid out explicitly, the users of such ranking still need to use own judgments and institutional requirements to choose ranks appropriately. The subjective element in journal rankings not only complicates matters about what is correct, if any, but also about outcomes that depend crucially on adoption and analysis of rankings. For science and related subjects, SCOPUS and SCIMAGO hold some of the best journal ranking systems to this day, using their Cite Score and SJR indicators respectively, to rank journals.([Kianifar et al.(2014)Kianifar, Sadeghi, and Zarifmahmoudi]) However, owing to the manner in which both these indicators are considered, it is often the case that the received ranking might not always display the true quality and outreach of a specific scientific journal. Obviously, this could be true for a large number of subjects across length and breadth of contemporary research therefore recourse to a scientifically more acceptable method should always be of interest and often beneficial for a large set of users. To demonstrate this, we therefore considered the case of the Journal entitled, *Astronomy and Computing*, within the context of SCOPUS Journals in the relevant domain of AstroInformatics ([Bora et al.2016]) , in particular and Astronomy and Astrophysics, in general.

The primary focus of this case study is to determine the standing of Journal *Astronomy and Computing* with respect to other journals which were established prior to it. More importantly, the reasons for such standing need to be investigated which is a more complex and qualitative study. The algorithm also tests the validity of the ranking and suggests an alternative rank that used a more holistic approach towards the features. While this paper focuses on a specific journal, it is easy to see that the purpose of this construct is broad-based and deep-seated at the same time, such that the applications of the algorithms can be adopted by numerous other subjects grappling with the same problem.

**Table 48:** Case Study: Astronomy and Computing, SJR ([González-Pereira et al.(2009)González-Pereira, Bote, and de M
and L1-SVD ranks ([Aedula et al.(2018)Aedula, Yashasvi Madhukumar, Snehanshu Saha, Mathur, Kakoli Bora, and Su

| Journal Name | L1 Scheme Rank | SJR based Rank | Year |
|---|---|---|---|
| Astronomy and Computing | 39 | 31 | 2013 |
| Astronomy and Astrophysics Review | 40 | 5 | 1999 |
| Radiophysics and Quantum Electronics | 41 | 51 | 1969 |
| Solar System Research | 42 | 48 | 1999 |
| Living Reviews in Solar Physics | 43 | 3 | 2005 |
| Astrophysical Bulletin | 44 | 45 | 2010 |
| Journal of Astrophysics and Astronomy | 45 | 55 | 1999 |
| Revista Mexicana de Astronomia y Astrofisica | 46 | 23 | 1999 |
| Acta Astronomica | 47 | 20 | 1999 |
| Journal of the Korean Astronomical Society | 48 | 32 | 2009 |
| Cosmic Research | 49 | 58 | 1968 |
| Geophysical and Astrophysical Fluid Dynamics | 50 | 46 | 1999 |
| New Astronomy Reviews | 51 | 12 | 1999 |
| Kinematics and Physics of Celestial Bodies | 52 | 65 | 2009 |
| Astronomy and Geophysics | 53 | 67 | 1996 |
| Chinese Astronomy and Astrophysics | 54 | 72 | 1981 |

## 12.2 Motivation: The ranking scheme

We implemented $l_1$-norm SVD scheme using the publicly available SCOPUS dataset to rank all Astronomy journals, and simultaneously determine the potency of the algorithm. The outcome of the ranking scheme posed interesting and compelling questions which led us to model the growing influence of the particular journal. We discuss the detailed method in Appendix A, for the simple reason that the focus of the manuscript is not on the ranking methods, rather on the model formulation and interpretation explaining such rank. SCOPUS contains approximately 46,000 Journals listed in different domains. Discarding few redundancies, SCOPUS effectively covers a large range of metrics and provides adequate resources for verification. For this demonstration, we have considered 7 different metrics from SCOPUS to be used as features in our algorithm. These features include *Citation Count, Scholarly Output, SNIP, SJR, Cite Score, Percentile and Percent Cited.*

The results of the algorithm are cross-verified with SJR based ranking of SCIMAGO for suitable articulation of the discrepancies. It seems that the $l_1$-norm SVD scheme works quite successfully ([Aedula et al.(2018)Aedula, Yashasvi Madhukumar, Snehanshu Saha, Mathur, Kakoli Bora, and Su
in rating the journals and approaches the data in a more comprehensive way. The result is a ranking system which ranks *Astronomy and Computing* much higher than most of the

older journals and at the same time highlights the niche prominence of the particular journal. Similarly, this method also highlights the rise of other journals which were underrepresented due to the usage of the SCOPUS and SCIMAGO indicators only. This method, therefore, has been largely successful in rectifying the rank of such journals. Importantly, the $l_1$-norm SVD scheme can be extrapolated to other data as well. It can be used to study the impact of individual articles, for example. Utilizing similar features such as Total Citation, Self Citation, and NLIQ ([Ginde2016]), the algorithm can be used to rank articles within a journal with great accuracy along with a holistic coverage. To re-apprise the scope of this research, it is important to remember that the common practice ([Engemann and Wall(2009)])has been to control for the size of the journal (measures like pages, number of articles, even characters), age of article, age of citation, reference intensity, exclusion of self-citations, etc.

### 12.2.1 Knowledge Discovery and the Evolution of ASCOM: Key Motivation for the model

Even though Astronomy and Computing (ASCOM) has been in publication for five years only, its reputation has grown quickly as observed from the ranking system proposed here. (please refer Fig. 1) This is despite the fact that ASCOM is severely handicapped in size. There is no journal focused on the interface of astronomy and computing in the same way as ASCOM. It can be observed from Table 1 that, ASCOM, in comparison with the other journals listed, is significantly younger! Unless the number of volumes and issues published are significant, a journal is unlikely to create the equivalent impact of an established journal. This is a notable handicap for any new journal, ASCOM being no exception. We define this as "size handicap".

Despite the "size handicap" explained above, ASCOM is ranked 39 according to our method, slightly lower than its 31 rank in SCOPUS. This is due to the fact that we have not used "citations from more prestigious journals" as a feature (this data are not readily available). Nonetheless, it is ranked higher than many of its peers which have been in publication for over 20 years. This is also due to the fact that ASCOM is "one of its kind" and uniquely positioned in the scientific space steered by appropriate editorial support. However SUBJECTIVE the statement may sound, it seems that interdisciplinary, diversity in background of the Editors and authors and novelty in theme have been instrumental in placing journals uniquely ([NAKAWATASE(2017)]. [Rodríguez(2016)] [Jacobs and Rebecca.(2012)] [Erfanmanesh(2017)]). Such qualitative feature, regrettably is not visible from the big data landscape alone. This is another significant driving factor behind framing and interpreting a novel model that explains trends arising from investigating the big data landscape. There is another interesting observation to take note of. By ignoring the "size does matter" paradigm, the ranks of some journals (many years in publication with several volumes and issues)

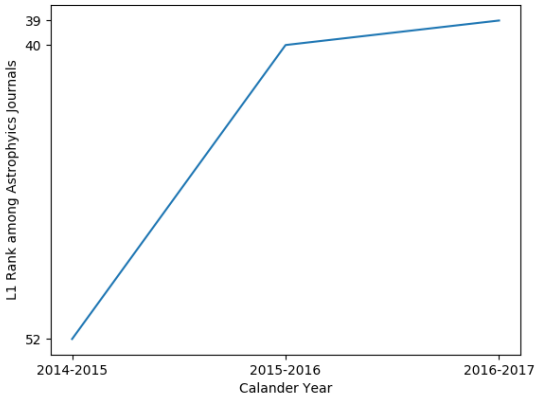



**Figure 36:** $l_1$ Rank Progression of ASCOM based on SCOPUS data computed by the proposed method. Since, the steady ascendancy in the journal's rank is unmistakable, it will be interesting to investigate the behavior of the journal rank in the long run once enough data is gathered. Please see Appendix A for details.

suffered according to our method. A few examples include Living Reviews in Solar Physics, ranked 43 according to our scheme while it is ranked 3 in SCOPUS; and Astronomy and Astrophysics Review, ranked 40 in our scheme while it is ranked 5 according to SCOPUS. This reversal of positions should be considered as important findings, because existing methods do not offer appropriate weights to journals that are new, despite catering to a niche and important area of research. In other words, the results indicate that years in publication may sometimes dominate over other indicators of quality and may not capture the growth of journals in "short time windows". Our study also reveals that ASCOM is indeed a quality journal as far as early promise is concerned.

Scientometrics deals with analyzing and quantifying works in science, technology, and innovation. It is a study that focuses on quality rather than quantity. The journals are evaluated against several metrics such as the impact of the journals, scientific citation, SJR, SNIP indicators as well as the indicators used in policy and management contexts. The practice of using journal metrics for evaluation involves handling a large volume of data to derive useful patterns and conclusions ([**?**]). These metrics play an important role in the measurement and evaluation of research performance. Due to the fact that most metrics are easily susceptible to manipulation and misuse, it becomes essential to judge and evaluate a journal by using a single metric or a reduced set of significant metrics. We proposed $l_1$-norm Singular Value Decomposition($l_1$-SVD) ([Aedula et al.(2018)Aedula, Yashasvi Madhukumar, Snehanshu Saha, Mathur, Kak
to efficiently solve this problem. The code of the proposed method is available at [Aedula()].

### 12.2.2 The Big Data Landscape

Advancements in Big Data frameworks ([Ginde et al.(2017)Ginde, Aedula, Saha, Mathur, Dey, Sampatrao, an
and technologies allow us to break the barriers of memory constraints for computing and implement a more scalable approach to employ methods and algorithms. The aforementioned journal ranking scheme is one such algorithm which thrives under the improvements made to scalability in Big Data. Implementing the SVD algorithm with the help of Spark can not only improve spatial efficiency but temporal as well. The $l_1$-norm SVD scheme utilizes the SVD and regularization implementation of *ARPACK* and *LAPACK* libraries along with a cluster setup to enhance the speed of execution by a magnitude of at least three times (depending on the configuration). We need to remind ourselves that The necessity of a cluster based system is rendered useless without the requisite data to substantiate it. Scientometric data usually deals with properties of the journals such as Total Citation, Self-Citation etc. This data could be collected using Web Scraping methodologies but also can be found by most journal ranking organizations, available for open source use; SCOPUS and SCIMAGO. For the $l_1$-norm SVD scheme, we used SCOPUS as it had an eclectic set of features which were deemed appropriate to showcase the effectiveness of the algorithm. The inclusion of the two important factors such as Cite Score and SJR indicators gave a better enhancement over just considering one over the other. For more information about the data and code used to develop this algorithm (please refer to [Aedula(2018)], Github repository of the project). The landscape and the big data framework are able to capture the rapid growth of ASCOM but are insufficient to explain it! This brings us to the next topic of deliberation.

## 12.3 Beyond the ranking framework: Seeking motivation for the model

In order to be precise, the ranking scheme raises some important questions which can be reasonably challenging. Standard scientometric features used to study influence/reputation of journals are not adequate for explaining the ascendancy of ASCOM in influence. The importance of investigating intrinsic dynamics is rarely stressed upon in scientometric literature ([Fei et al.(2015)Fei, Chong, and Bell]). Usually, the analysis is static, based on citations and other factors. The authors intend to bring out the missing dynamics via the DDE based model. The following set of questions are addressed in this study. What are the non-quantitative factors (could be qualitative and difficult to quantify) explaining the rapid growth of this journal? What is the direction of causation and how do we frame it? Does the big data landscape help? Can we formulate a model that reasonably accounts for such surge in influence? Are there features/factors, not statistically significant but play crucial roles as implicit control

variables toward the phenomena? The proposed model (Section 4 onward) addresses the following questions:

- What are the principal factors behind the quick rise in impact of the journal, Astronomy & Computing (ASCOM)?
- The argument of quality articles from good authors contributing is a vicious cycle since scholars of high intellect have a wide range of impressive journals in Astronomy & Astrophysics to choose from. Why are they attracted to contributing to ASCOM?
- Are there extraneous factors which accelerate the growth of ASCOM? Is it the Editorial board or the reputation of the publication house (Elsevier) or both that are responsible for attracting quality scholars?
- Is history i.e. pedigree a key factor behind such growth? If so, what defines the pedigree here?
- Finally, if the ranking framework is unable to answer the above questions, does there exist a modeling approach to explain all of the above? In the absence of it, can we propose a model that addresses the interesting questions raised by the ranking exercise?

We shall seek answers to these questions in the remainder of the paper. The rest of the manuscript is organized as follows. We begin by presenting the motivation for Delay Differential Equation (DDE) based modeling by outlining key strengths of such modeling concept. Next, we consider time reversed DDE to model the growth by including historical effects, a fundamental contribution in section 4. Section 5 contains solutions, analytically and computationally investigated and interpreted in light of the big data landscape. Section 6 considers further modifications in the model by adding Editorial reputation and Publisher Goodwill value. We discuss the implications of these additional factors and the fundamental assumptions in Discussion & Conclusion Sections, 7 and 8.

## 12.4 Scope of our study and Motivation for modeling via DDE

The manuscript strives to achieve two fundamental objectives:

- We establish and quantify current journal influence as a function of its past influence. If the past influence is positive (good inheritance), the present journal influence benefits immensely from it (Please see sections 4, 5 and Figures 2, 3 and 4).

- The manuscript proposes a doctrine of " self-serving incentivization" by exploiting implicit control variables (publisher goodwill value and editorial reputation-the celebrity effect). The so-called " incentivized model" is proposed to propagate a positive "start-up boost" to the journal influence. Thereby, these control variables and the modifications form the second and more advanced, complex layer in modeling journal influence (Please see sections 6, 7 and Figures 5-10) and help quantify the theory of " celebrity effect".

The factors mentioned above and the resulting model explained in the subsequent sections also account for the remarkable growth in influence and ASCOM discussed in sections 1 and 2. We achieve this by the DDE based model presented below.

DDE is a well known concept for over two centuries, which has found application in various problems in the fields of dynamical modeling of biomedical systems, biochemical reactions as well as in the newer models of interpersonal/romantic relationships!! DDEs also find useful applications like dynamic population growth, economic growth and spread of diseases like HIV, cancer, etc. Delay Differential Equations belong to the class of Partial Differential Equations. These are used by the scientific community for modeling dynamic systems for many of the obvious advantages. These equations describe the rate of change of a function, at time 't' as a function of earlier times. A DDE in its general form can be given by:

$$p'(t) = f(p(t), p(t-\tau)); p(0) = p_0 \tag{82}$$

considering a constant delay of $\tau$. Some of the advantages of DDEs are:

- DDEs take care of the "hereditary effects" during modeling a system. This implies if the influence of a journal is positive in the past and/or intrinsic factors have been responsible for surge in reputation, such features are naturally modeled in DDEs.

- In system modeling, it is desirable that the model is closer to the real process (in our case, influence diffusion and percolation) and it has been observed that DDEs offer a better model than others.

- DDEs are seen to provide better control over the system since historical data is directly modeled in to the system (using time-reversed structure). This is particularly desirable.

- In case of a DDE, the initial point $p_0$ defined over the interval [-$\tau$, 0] , is a function and not just a point. The solution $p(t)$ is also a function in the same interval. Hence, the solution becomes infinite dimensional, unlike an ODE. Moreover, in a dynamical

system, DDE takes care of rate of growth, which is a robust form of looking at the real world problem than just reading from hereditary events and inferring from them.

## 12.5 TIME REVERSED DDE: Our Contribution

Let $p'(t)$ denote rate of change of influence over time whereas $p(t)$ stands for influence at time $t$. Moreover, the history function is represented by $p(-t)$ implying influence at time $t=-t$. ($t=1, p'(1)=ap(1)+bp(-1)$ or $p'(2)=ap(2)+bp(-2)$ and so on). The Time Reversed equation can now be written as

$$p'(t)=ap(t)+bp(-t) \tag{83}$$

which implies the rate of change of influence is represented as a combination of present and past influence. To begin with, let us consider a simple growth model given as

$$\begin{aligned} ap'(t) &= b+cp(t) \\ p(0) &= c \end{aligned}$$

where $p(0)$ is not the initial condition but is the value at the instant of time under the interval of consideration. Please note, $a,b,c$ are constants and are estimated from data in the model fitting process explained in section 5. We represent this linear growth in the form of time reversed structures as follows:

$$\begin{aligned} \implies p'(t) &= \frac{b}{a}+\frac{c}{a}p(t) \\ &= \frac{b}{a}+\frac{c}{2a}p(t)+\frac{c}{2a}p(t) \\ &= \frac{b}{a}+\frac{c}{2a}p(t)+\frac{c}{2a}p(-t) \end{aligned}$$

We assume symmetric influence function from two possibilities, symmetric and non-symmetric influence. (If $p(t)=p(-t)$, the influence function is symmetric. The symmetry is a known concept in calculus. The implication is astounding for the simple reason that symmetric influence function carries over the good effects from the past and help a journal grow quickly if the effects are utilized in a forward manner. This is exactly what is hypothesized in section

2.3). Now, differentiating the above equation again w.r.t $t$,

$$\begin{aligned} p''(t) &= \frac{c}{2a}p'(t) - \frac{c}{2a}p'(-t) \\ &= \frac{c}{2a}\Big(\frac{b}{a} + \frac{c}{2a}p(t) + \frac{c}{2a}p(-t)\Big) - \frac{c}{2a}\Big(\frac{b}{a} + \frac{c}{2a}p(-t) + \frac{c}{2a}p(t)\Big) \\ &= 0 \end{aligned}$$

Therefore, $p(t) = Ct + C_1$ where $C, C_1$ are constants. This implies $p(t)$ may exhibit linear growth under the assumption that there is a certain repeatability in the journal influence.

### 12.5.1 The model under non-symmetric influence:

Let us not consider the symmetric influence function since it is too strong an assumption to begin with (fluctuations are absent, unidirectional slope, elements of uncertainty almost absent). Let us consider the same model given as

$$ap'(t) = b + cp(t); p(0) = c \tag{84}$$

without the assumption of symmetric influence ($p(t) = p(-t)$). Here also, $p(0)$ is not the initial condition but is the value at the instant of time under the interval of consideration. Reorganizing equation 3,

$$p'(t) + (-\frac{c}{a})p(t) = \frac{b}{a} \tag{85}$$

Assuming $(-\frac{c}{a})$ and $(\frac{b}{a})$ are continuous functions (constants in our case), we fix $(-\frac{c}{a}) = r(t)$ and $(\frac{b}{a}) = s(t)$. Putting this in the equation, we obtain

$$p'(t) + r(t)p(t) = s(t) \tag{86}$$

Let $\mu(t)$ be an integrating factor giving by the following equation ([Saha(2011)]). Multiply both sides of equation (5) with $\mu(t)$ and integrating, we arrive at the following form:

$$\mu(t) = Ke^{\int r(t)dt}$$

where K is a constant. Eventually the expression for journal influence is written as

$$p(t) = \frac{\int e^{\int r(t)dt}s(t)dt + \frac{c}{K}}{e^{\int r(t)dt}} \tag{87}$$

Please see Appendix E for computational details. Under the assumption of non-symmetric influence (more realistic), the influence experiences exponential growth or decay depending on the coefficients but not a combination of both in a single expression. We shall see a different picture in the next section when we encounter non-linear growth in influence for a slightly more complicated, time reversed model.

**Remark:** Please note the above model does not contain "history" functions. Hence the solution does not display a convex combination of exponential functions, which can be easily interpreted in light of historical data. This is in contrast to the simple case (we assume a symmetric influence) where we can safely conclude that if either the historic influence or the current influence of the journal is high then the journal is most likely going to experience further rise in influence in the near future.

### 12.5.2 Modeling Non-linear growth using symmetric influence effects

Let us consider eq.(1) with the condition $p(0) = c$ by mapping these to the following DDE:

$$y'(t) = a_1(t)y(t) + a_2(t)y(t-d), t >= 0$$
$$y(t) = p(t), t \in [-d, 0]$$

Consider $d = 2t; a = a_2(t), b = a_1(t); y(t) \equiv p(t) \forall\, t \in [-d, d]$. Our proposed model is a special case of DDE and it will be shown later that eq.(1) has at least one solution (see Appendix B), which may not be necessarily unique.

**Sketch of the Solution Methodology:** Let us consider the time reversed model eq.(2):

$$p'(t) = ap(-t) + bp(t)$$
$$p(0) = k$$
$$p'(0) = (a+b)k$$

A clever manipulation yields (see Appendix F for details)

$$p'(-t) = -ap(t) - bp(-t) \tag{88}$$

Eventually we obtain,

$$p''(t) = (a^2 - b^2)p(t)$$

where $r = \sqrt{a^2 - b^2}$ [Please see Appendices F and D, E for further clarification and missing steps] Again, by symmetry,

$$p''(-t) = r^2 p(-t) \tag{89}$$

Solution is of the form,

$$p(t) = Ae^{rt} + Be^{-rt} \tag{90}$$

It is evident that $p(t)$ is an exponential function. Using initial conditions, solving for A & B in terms of a & b we get,

$$p(t) = \frac{c}{2r}\Big(r + a + b\Big)e^{rt} + \frac{c}{2r}\Big(r - a - b\Big)e^{-rt} \tag{91}$$

Appendix D contains the detailed derivation leading up to the solution obtained above, in (10). Depending on the coefficient values, either positive or negative exponents will dominate. The two possible solutions depend on the value of $r$.

- When $r > 0 (i.e., b > a)$, we can expect an exponential real solution

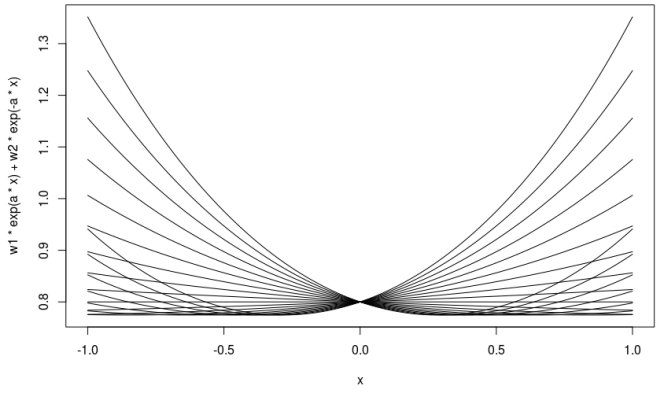



**Figure 37:** Plot of eq. (9) where $p(t)$ is represented by the Y-axis and t is represented by the X-axis. The present influence, $p(t)$ is controlled by past influence, $p(-t)$

- When $r < 0 (i.e., b < a)$, there will be oscillatory solutions, due to r being imaginary. Again, these solutions are deemed infeasible due to lack of fixed periodicity.

$t = 0$ is considered to be in the middle of a short time frame, at which, we are measuring the influence. Hence, this is not considered as initial value problem and hence we are not guaranteed of a unique solution. $p(-t)$ is the mirror image of $p(t)$ and it will result in a sharp spike in influence provided its value is high. This is typically observed in a short time

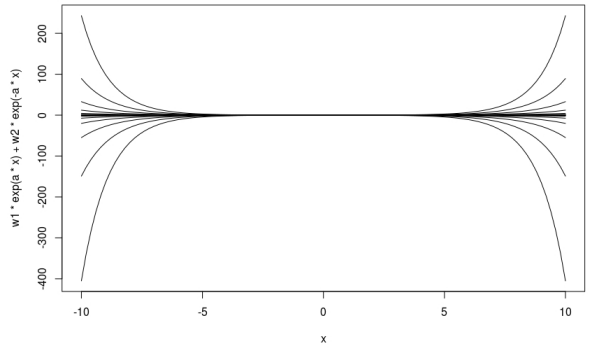



**Figure 38:** Plot of eq. (9) where $p(t)$ is represented by the Y-axis and $t$ by the X-axis.Imaginary solutions are obtained when $a$ and $b$ are varied such that $w1 < 0$. This is infeasible as the model explains real solutions for obvious reasons.

window and averages out in the longer time span. We see that depending on the values of the parameters $a$ and $b$, either the historical or the current data dominates. The curve shows that, in the first few years, the influence is largely dominated by the past reputation of the editors represented by the historical part of the DDE. After a certain point (we have assumed this point to be at the center of time series data), other parameters such as the current journal citations and the current reputation of the editors begin to reflect on the influence.

## 12.6 Model Fitting:

Let us recall Eq.(1):

$$\begin{aligned} p'(t) &= ap(t) + bp(-t) \\ p(0) &= c \\ p'(0) &= (a+b)c \end{aligned}$$

We also know, by approximation that,

$$\begin{aligned} p'(t) &\approx \frac{p(t+h) - p(t)}{h} \\ &\approx \frac{p(t) - p(t-h)}{h} \end{aligned}$$

where $h$ is the step size. Let us consider the spread at discrete time intervals corresponding to one to five years (obtained from the data set) indicated as $p'(1), p'(2), p'(3), p'(4)$ and $p'(5)$

respectively. Here, we can write $p'(1) = ap(-1) + bp(1)$. Also,

$$\begin{aligned} \implies \frac{p(1)-p(0)}{1} &= ap(-1) + bp(1) \\ &= a[Ae^{-r} + Be^{r}] + b[Ae^{r} + Be^{-r}] \\ &= (aA + bB)e^{-r} + (aB + bA)e^{r} \end{aligned}$$

The value on LHS is obtained from the data set. Similarly, we can compute $p'(\frac{3}{4}), p'(\frac{1}{2}), p'(\frac{1}{4})$, etc. obtained from the data set, where the fractions represent the quarters in a year. We are now required to estimate the coefficients $a, b, A\&B$ [6]. This is an overestimation problem with number of equations exceeding number of unknowns. We can solve this by method of Least Squares and use the solution to predict future influence and rate of journal influence spread.

### 12.6.1 Least Square Method to fit the data:

From eq. (1), we obtain

$$p'(t) = ap(t) + bp(-t)$$

Let $p'(t) = z, p(t) = x, p(-t) = y$. Therefore, eq. (1) becomes

$$z = ax + by$$

Let

$$S = \sum (z - (ax + by))^2$$

Differentiating w.r.t a,

$$\begin{aligned} S &= 2\sum (z - (ax + by))(-x) = 0 \\ S &= \sum (-zx + ax^2 + bxy) = 0 \end{aligned}$$

$$\sum zx = a\sum x^2 + b\sum xy \tag{92}$$

[6] the coefficients a, b are constants and arise from the differential equation based model proposed in (2) and subsequent derivatives of it. A & B are constants, obtained by integrating the differential equation to yield the desired solution, $p(t)$

Differentiating w.r.t b,

$$S = 2\sum (z - (ax + by))(-y) = 0$$
$$S = \sum (-zy + axy + by^2) = 0$$

$$\sum zy = a\sum xy + b\sum y^2 \tag{93}$$

On solving eq. (11) and eq. (12), we obtain the values of $a$ and $b$.
**ESTIMATING 'A' and 'B':** We have found that

$$p(t) = (aA + bB)e^{-rt} + (aB + bA)e^{rt}$$

Let,

$$p(t) = y$$
$$aA + bB = w1$$
$$aB + bA = w2$$
$$e^{rt} = x$$

Taking log

$$\log(x) = rt$$
$$\log(x^{-1}) = -rt$$
$$e^{-rt} = \frac{1}{x}$$

Therefore,

$$y = w1 * x + \frac{w2}{x}$$
$$xy = w1 * x^2 + w2$$

Let,

$$Y = xy$$
$$X = x^2$$

Now,

$$Y = w1 * X + w2$$

$$\sum Y = w1 \sum X + w2 * n \tag{94}$$

$$\sum X \sum Y = w1 \sum X^2 + w2 \sum X \tag{95}$$

On solving eq. (13) and eq. (14) we can obtain values of w1 and w2. Hence,we can also find the values of A and B. We present the algorithm below.

**Algorithm 2** Model Fit using Least Square Method

*p(t)* ← *Input journal influence data*
*EQU* ← *Model_EQ(p(t))*
*NEW_EQU* ← *LSM(EQU)*
**procedure** MODEL_EQ(P(T))
    *p'(t)* ← *ap(t) + bp(-t)*
    *Discretize the derivative using present and past data*
    *p'(t)* ← *p(t+h)-p(t)/h*
    *(p(1)-p(0))*/1 ← *ap(-1) + bp(1)*
    $= a[Ae^{-r} + Be^{r}] + b[Ae^{r} + Be^{-r}]$
    *Return* $(aA + bB)e^{-r} + (aB + bA)e^{r}$
**procedure** LSM(EQU)
    *Derive the values of 'a' and 'b' of EQU using Equations*
    $\sum zx = a\sum x^2 + b\sum xy$ *and*
    $\sum zy = a\sum xy + b\sum y^2$
    *Derive the values of 'A' and 'B' of EQU using Equations*
    $\sum Y = w1 \sum X + w2 * n$
    $\sum X \sum Y = w1 \sum X^2 + w2 \sum X$
    $return$ *EQU with derived values*

**Goodness Of Fit:** Percentage Accuracy in estimating the coefficients $a, b$ turned out to be 97.1. Moreover, 70% Confidence interval of the coefficients $a, b$ are in the range (-0.4005416,-0.15684858) and (-0.3127150, -0.01080603) respectively implying the coefficient values are significant. This is testimony of the goodness of fit of our model validated against real data. The Confidence percentage would improve with additional data whenever available.

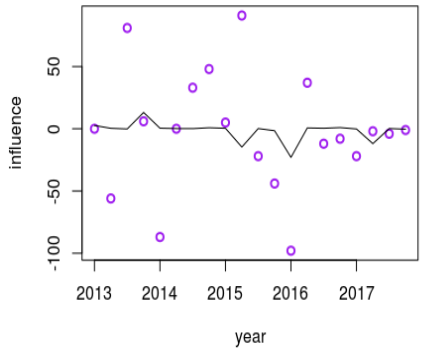



**Figure 39:** $p'(t)$ v/s $t$ ((Non-linear least square curve): The rate of change in influence over the span of 5 years shows small fluctuations but maintains an overall steady value. This means that the influence in 5 years will not suffer drastically.

## 12.7 Model Modification to accommodate implicit control variables

Additionally, we consider implicit control variables which play important roles in the growth of any journal. These variables pose challenges to the modeling set up and without these, the scope is limited to empirical verification at a minor scale only. Next step in modeling data is to carry out modifications to this structure in order to accommodate implicit control parameters such as publisher goodwill value and âĂIJstart-up initiativeâĂİ by editors (editorial reputation). We define this initiative as the reputation of editors who steered the journal and offered a strong attraction for quality submissions from scholars across the globe. It is realistic to hypothesize that reputed scholars acting as editors add value and credibility to an emerging journal. This value however is extremely hard to quantify and therefore modeling such phenomenon is novel and imperative to understand the journalâĂŹs growth pattern. We propose to present the model and the analytical solution, repeat the exercise of sections 3 and 4 and discuss the implication of the proposed modification.

The Time Reversed equation with the additive influence term (Publisher goodwill value)

can now be re-written as

$$p'(t) = ap(t) + bp(-t) + \eta + \theta \tag{96}$$

where $\eta$ is an additive term implying goodwill of the publishing house, Elsevier, in our case! $\theta$, OTOH represents Editors' reputation.

### 12.7.1 Additional Considerations

- Let us assume $\eta$ to be either linear or exponential. Such considerations are justified since any reputed publisher, in order to remain competitive, would strive to enhance goodwill. Thus, $\eta$ can't possibly be a constant.
- We pose the next question pertinent to quantification of goodwill. It is modeled as a function of the percentage of accepted papers over time, a trend that accommodates a fixed number of accepted articles and the selection criteria of additional papers becomes increasingly stringent. It is modeled as

  $$\eta(.) = e^{-art} + \alpha(a-b) \tag{97}$$

  where $art$ is the percentage of articles accepted after the initial threshold of $\alpha$ articles. $\alpha(a-b)$ is the initial threshold, conveniently set to ensure that the influence doesn't hover to the negative.
- Thus, $\eta(.)$ is a control variable in the formulation and explanation of publisher goodwill. This implies, increasingly the percentage of accepted articles will diminish. Such stringent measures in peer-review bolster publisher goodwill.
- The formulation being in place, we now integrate $\eta(.)$ with the modified model.
- Editorial reputation may be any of the three: a constant function, linear or exponential growth. The first one is more likely since the Editors of ASCOM are well established in their fields. Therefore, it is less likely that their phase of influence is still growing at quadratic rate or higher. In fact, we have observed that the influence pattern (citations) is steady. Nonetheless, we have considered all three possibilities and discuss the implications after integrating Editorial reputation, $\theta$ (which is a function of time) in to the model.

### 12.7.2 Temporal evolution of publisher goodwill value

Figures 5(a) and 5(b) throw some useful insights. We hypothesize that the linear graph (fig. 5(b)) is a subset of the non-linear one (fig. 5(a)). Fig 5(b), which is a time-series plot of publisher goodwill value is linear upon fitting the ASCOM data. Fig. 5(a) is an extended time window plot of the same journal which is accomplished by simulating the data available from 5 years, extended to 10 years. The 5-year trend, if we take the time-slice off from fig. 5(a), produces fig. 5(b). This is done to establish the hypothesis that, available data to understand and predict longer time average behavior is insufficient.

This synthetic experiment implies that, if commendable work in the past continues (good inheritance in terms of positive influence of the implicit control variable i.e. the publisher goodwill, it shall continue to grow in non-linear fashion). The observation is in agreement with the publisher in question, Elsevier, who pursues aggressive and stringent quality practices toward the larger goal of monopoly in the business of publishing. At this point, we may note that, the nonlinear time dependent trend shall influence the overall journal growth in influence to a greater proportion in comparison with the model we assumed in eq. (16) (which is time-independent). We draw such inferences from the goodwill value as a time series plot by re-solving the equation with fitted goodwill value model from time-series data. We show that in the ensuing discussion accompanied by the figures below (Fig. 6, 7, 8). Let us now consider eq. (16). On adding the publishers goodwill as a function of time we obtain the equation,

$p''(t) - (b^2 - a^2)p(t) = (a+b)\theta(t) + k * e^{k_1 t}$ ——————————————————————————(*)

On solving the above equation on similar lines outlined in Appendix C, we obtain expressions of journal influence as solutions for the three different cases of $\theta(t)$ being constant, linear and exponential and $\eta(t)$ being the time dependent function instead of a function of accepted articles as discussed earlier[7].

1. CASE 1 (Fig. 6): Let us assume that $\theta(t) = \theta = constant$
   $\implies p(t) = c_1 e^{t\sqrt{b^2-a^2}} + c_2 e^{-t\sqrt{b^2-a^2}} + \frac{\theta}{(a-b)} + \frac{ke^{(k_1)t}}{(k_1)^2-(b^2-a^2)}$

2. CASE 2 (Fig. 7): Let us assume that $\theta(t)$ is linear: $\theta(t)$ = At+B
   $\implies p(t) = c_1 e^{t\sqrt{b^2-a^2}} + c_2 e^{-t\sqrt{b^2-a^2}} + \frac{At+B}{a-b} + \frac{ke^{(k_1)t}}{(k_1)^2-(b^2-a^2)}$

3. CASE 3 (Fig. 8): Let us assume that $\theta(t)$ is exponential:
   $\theta(t) = e^{At} \longrightarrow p(t) = c_1 e^{t\sqrt{b^2-a^2}} + c_2 e^{-t\sqrt{b^2-a^2}} + \frac{(a+b)e^{At}}{A^2-(b^2-a^2)} + \frac{ke^{(k_1)t}}{(k_1)^2-(b^2-a^2)}$

[7] $e^{rt}$ and $exp(rt)$ have been used interchangeably in the manuscript.

These plots (shown in Figure 5, 6, 7 and 8) demonstrate clearly that if publisher goodwill value is modeled as a time dependent evolution, the influence of the journal grows at a faster pace in the longer run. Therefore, it complements our observation that, publisher goodwill value has a small role to play in the growth of journal influence in short time span but evolves gradually as time progresses.

### 12.7.3 Temporal evolution of Editorial reputation

We observe the celebrity effect here ([Fei et al.(2015)Fei, Chong, and Bell]). Editors are well established scholars and by the time they assumed editorial responsibility, they are in the "cool off state" implying the surge in reputation they experienced when they were rising stars had stabilized. Therefore, steep gradient shall no longer be expected. This is what we observe in Fig 9 where the editorial influence between 2004 and 2014 is plotted. Please note ASCOM was founded in 2013. The influence trend of all the editors during that time (2010-14) is approximately constant.

Next section will deliberate on the contributions of these variables, in particular and model modification, in general on the rate of change in influence observed in ASCOM. The role of control variables are evident in the visualization we present below.

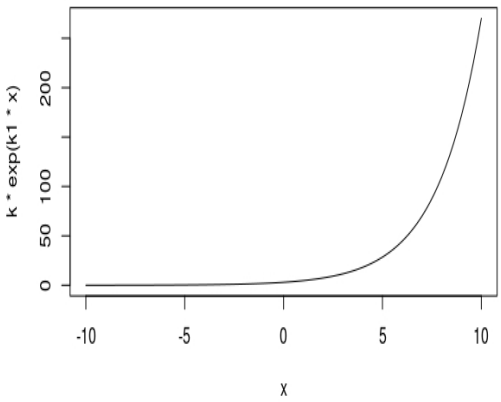
## 12.8 DISCUSSION

We develop a model to study its effect on astronomy and computer science domains and analyze parameters that have contributed in building the reputation of ASCOM. In this specific case study of journal influence, the spread is clearly dependent on present as well as history dependent functions. This strengthens the motivation of using DDE model for the study. The model explains the growth pattern of the journal well by capturing the intrinsic attributes and historical data. The time reversed model works as a mirror and helps carry over the good deeds of the past (quality of articles in niche areas and open problems solved by interdisciplinary efforts reflected in citation history). Our model exploits the âĂIJhereditary effectsâĂİ through time-reversed structure built in. Additionally, the phenomenon of observing a journal in an emerging and interdisciplinary area, modeled as a function of spatial variables renders the system infinite degrees of freedom. Thus, the proposed model is robust and provides better control over the system.

However, the data is limited since the journal is in publication for just over five years. Therefore the influence of historical data does not translate to overwhelming quantitative evidence in the way we liked it to. Nonetheless, if we extrapolate the interval by extending the

time window of consideration (since the historical data is assumed to influence the present one), we observe profound effects (Please see the discussion on temporal evolution of publisher goodwill value where the observed linear growth in goodwill is really a 5-year snapshot subset of the longer window; (please see figs. 5(a) and (b) and the discussion in section 6.2). Additionally, we considered implicit control variables such as Editorial reputation and Publisher goodwill which play important roles in the growth of any journal. These variables pose challenges to the modeling set up and without those, the scope is limited to empirical verification at minor scale.

## References



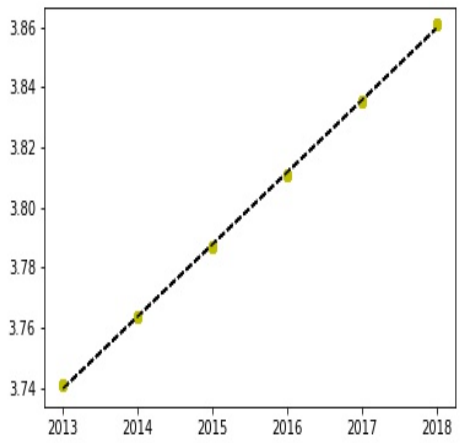




**Figure 40:** Plot of publishers goodwill VS time. We observe that the publishers good will shows a linear rise in the span of the 5 years between 2013 and 2017. Extrapolated to 10 years, the linear trend becomes non-linear and eventually impact the overall influence of the journal by a margin.

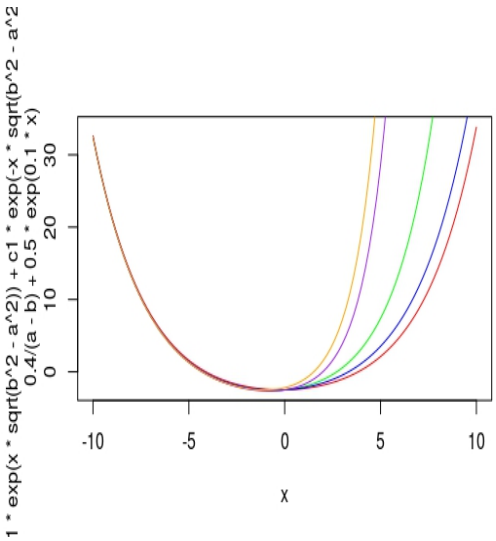



**Figure 41:** Plot of $p(t)$ v/s $t$ when $p(t)$, the influence is the solution to the equation (*). We observe that the slope of the graph becomes steeper as $k_1$ increases.

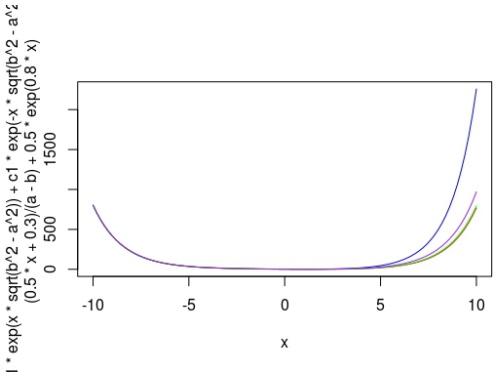



**Figure 42:** Plot of $p(t)$ v/s. $t$ when $p(t)$, the influence is the solution to the equation (*). We observe that the slope of the graph becomes steeper as $k_1$ increases.

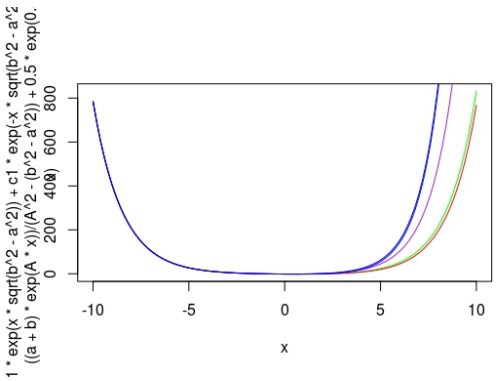



**Figure 43:** Plot of $p(t)$ v/s. $t$ when $p(t)$, the influence is the solution to the equation (*). We observe that the slope of the graph becomes steeper as $k_1$ increases.

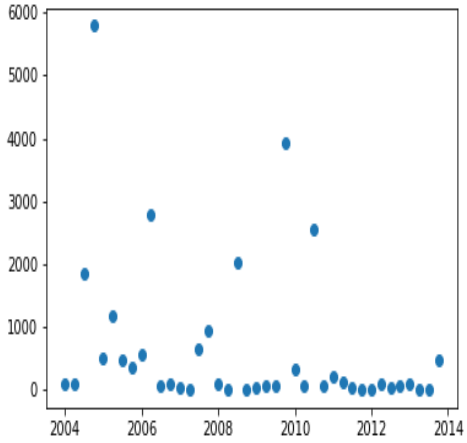



**Figure 44:** The above plot represents editors influence against time. We see that the editorial influence is almost constant with time. This is possible because the editors are already well established. Hence, the influence is steady with little fluctuations.

## REFERENCES


[Aedula()] Rahul Aedula. URL https://github.com/rahul-aedula95/L1_Norm.

[Aedula(2018)] Rahul Aedula. Github repository, 2018.

[Aedula et al.(2018)Aedula, Yashasvi Madhukumar, Snehanshu Saha, Mathur, Kakoli Bora, and Surbhi Agra Rahul Aedula, Yashasvi Madhukumar, Snehanshu Saha, Archana Mathur, Kakoli Bora, and Surbhi Agrawal. L1 norm svd based ranking scheme: A novel method in big data mining. pages –, 2018. doi: 10.13140/rg.2.2.23721.29280.

[Bora et al.(2016)Bora, Saha, Agrawal, Safonova, Routh, and Narasimhamurthy] K. Bora, S. Saha, S. Agrawal, M. Safonova, S. Routh, and A. Narasimhamurthy. CD-HPF: New habitability score via data analytic modeling. *Astronomy and Computing*, 17:129–143, October 2016. doi: 10.1016/j.ascom.2016.08.001.

[Engemann and Wall(2009)] Kristie M. Engemann and Howard J. Wall. A journal ranking for the ambitious economist. *Review*, (May):127–140, 2009. URL https://ideas.repec.org/a/fip/fedlrv/y2009imayp127-140nv.91no.3.html.

[Erfanmanesh(2017)] Amin Erfanmanesh. Composition of journalsâĂŹ editorial board members as an indicator of the interdisciplinarity: The case of iranian journals in social sciences and humanities. *Library and Information Science*, 19(1):81–106, June 2017.

[Fei et al.(2015)Fei, Chong, and Bell] Qiang Fei, H Gin Chong, and Reginald Bell. The diminishing influence of celebrity authors in a diversified world of accounting journals. 15: 37–57., 03 2015.

[Ginde et al.(2017)Ginde, Aedula, Saha, Mathur, Dey, Sampatrao, and Sagar] G Ginde, R Aedula, S Saha, A Mathur, S R Dey, S Sampatrao, and B S Sagar. Big data acquisition, preparation and analysis using apache software foundation projects. In Arun K. Somani Ganesh Chandra Deka, editor, *Big data analytics tools and technology for effective planning*, chapter 9. CRC Press, 2017.

[Ginde et al.(2016)Ginde, Saha, Mathur, Venkatagiri, Vadakkepat, Narasimhamurthy, and Daya Sagar] Gouri Ginde, Snehanshu Saha, Archana Mathur, Sukrit Venkatagiri, Sujith Vadakkepat, Anand Narasimhamurthy, and B. S. Daya Sagar. Scientobase: a framework and model for computing scholastic indicators of non-local influence of journals via native data acquisition algorithms. *Scientometrics*, 108(3):1479–1529, Sep 2016. ISSN 1588-2861. doi: 10.1007/s11192-016-2006-2. URL https://doi.org/10.1007/s11192-016-2006-2.

[González-Pereira et al.(2009)González-Pereira, Bote, and de Moya Anegón] Borja González-Pereira, Vicente P. Guerrero Bote, and Félix de Moya Anegón. The SJR indicator: A new indicator of journals' scientific prestige. *CoRR*, abs/0912.4141, 2009. URL http://arxiv.org/abs/0912.4141.

[Jacobs and Rebecca.(2012)] A. Jacobs and Henderson Rebecca. Interdisciplinarity in recently founded academic journals. *Scholarly Common Repository*, August 2012.

[Jangid et al.(2014)Jangid, Saha, Gupta, and Rao] Neelam Jangid, Snehanshu Saha, Siddhant Gupta, and J. Mukunda Rao. Ranking of journals in science and technology domain: A novel and computationally lightweight approach. *IERI Procedia*, 10:57 – 62, 2014. ISSN 2212-6678. doi: https://doi.org/10.1016/j.ieri.2014.09.091. URL http://www.sciencedirect.com/science/article/pii/S2212667814001397. International Conference on Future Information Engineering (FIE 2014).

[Kianifar et al.(2014)Kianifar, Sadeghi, and Zarifmahmoudi] Hamidreza Kianifar, Ramin Sadeghi, and Leili Zarifmahmoudi. Comparison between impact factor, eigenfactor metrics, and SCimago journal rank indicator of pediatric neurology journals. *Acta Informatica Medica*, 22(2):103, 2014. doi: 10.5455/aim.2014.22.103-106. URL https://doi.org/10.5455/aim.2014.22.103-106.

[Manyika et al.()Manyika, Chui, Brown, Bughin, Dobbs, Roxburgh, and Byers] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A. H. Byers. Recursive object model (ROM)-modelling of linguistic information in engineering design. URL https://www.mckinsey.com/business-functions/digital-mckinsey/our-insights/big-data-the-next-frontier-for-innovation.

[NAKAWATASE(2017)] Hidekazu NAKAWATASE. The present state analysis of emerging interdisciplinary research areas: An analysis based on a classified table of grant-in-aid for scientific research. *Joho Chishiki Gakkaishi*, 27(1):1–5, 2017. doi: 10.2964/jsik_2017_007. URL https://doi.org/10.2964/jsik_2017_007.

[Rodríguez(2016)] Jorge Mañana Rodríguez. Disciplinarity and interdisciplinarity in citation and reference dimensions: knowledge importation and exportation taxonomy of journals. *Scientometrics*, 110(2):617–642, nov 2016. doi: 10.1007/s11192-016-2190-0. URL https://doi.org/10.1007/s11192-016-2190-0.

[Saha(2011)] Snehanshu Saha. *Ordinary Differential Equations: A Structured Approach*. Cognella, 2011. ISBN 160927704X. URL https://www.amazon.com/

Ordinary-Differential-Equations-Structured-Approach/dp/160927704X?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=160927704X.

# 13 A Novel Exoplanetary Habitability Score via Particle Swarm Optimization of CES Production Functions

## 13.1 Introduction

The search for extra-terrestrial life [Shostak2017, Schwieterman2017] and potentially habitable extrasolar planets [Johnson2018, Presto2017] has been an international venture since Frank Drake's attempt with Project Ozma in the mid-20th century [Shuch2011]. Cochran, Hatzes, and Hancock[Cochran1991] confirmed the first exoplanet in 1991. This marked the start of a trend that has lasted 25 years and yielded over 3,700 confirmed exoplanets. There have been attempts to assess the habitability of these planets and to assign a score based on their similarity to Earth. Two such habitability scores are the Cobb-Douglas Habitability (CDH) score [Bora et al.2016, Saha2017] and the Constant Elasticity Earth Similarity Approach (CEESA) score. Estimating these scores involves maximizing a production function while observing a set of constraints on the input variables.

Under most paradigms, maximizing a continuous function requires calculating a gradient. This may not always be feasible for non-polynomial functions in high-dimensional search spaces. Further, subjecting the input variables to constraints, as needed by CDH and CEESA, are not always straightforward to represent within the model. This paper details an approach to Constrained Optimization (CO) using the swarm intelligence metaheuristic. Particle Swarm Optimization (PSO) is a method for optimizing a continuous function that does away with the need for calculating the gradient. It employs a large number of randomly initialized particles that traverse the search space, eventually converging at a global best solution encountered by at least one particle [Eberhart1995, Shi1998].

Particle Swarm Optimization is a distributed method that requires simple mathematical operators and short segments of code, making it a lucrative solution where computational resources are at a premium. Its implementation is highly parallelizable. It scales with the dimensionality of the search space. The standard PSO algorithm does not deal with constraints but, through variations in initializing and updating particles, constraints are straightforward to represent and adhere to, as seen in Section 13.3.2. Poli[Poli2007, Poli2008] carried out extensive surveys on the applications of PSO, reporting uses in Communication Networks, Machine Learning, Design, Combinatorial Optimization and Modeling, among others.

This paper demonstrates the applicability of Particle Swarm Optimization in estimating habitability scores, CDHS and CEESA of an exoplanet by maximizing their respective production functions (discussed in Sections 13.2.1 and 13.2.2). CDHS considers the planet's Radius,

Mass, Escape Velocity and Surface Temperature, while CEESA includes a fifth parameter, the Orbital Eccentricity of the planet. The Exoplanet Catalog hosted by the Planetary Habitability Laboratory, UPR Arecibo records these parameters for each exoplanet in Earth Units [**?**]. Section 13.5 reports the performance of PSO and discusses the distribution of the habitability scores of the exoplanets.

## 13.2 Habitability Scores

### 13.2.1 Cobb-Douglas Habitability Score

Estimating the Cobb-Douglas Habitability (CDH) score [**?**] requires estimating an interior score ($CDHS_i$) and a surface score ($CDHS_s$) by maximizing the following production functions,

$$Y_i = CDHS_i = R^{\alpha}.D^{\beta}, \tag{98a}$$

$$Y_s = CDHS_s = {V_e}^{\gamma}.{T_s}^{\delta}, \tag{98b}$$

where, $R$, $D$, $V_e$ and $T_s$ are density, radius, escape velocity and surface temperature respectively. $\alpha$, $\beta$, $\gamma$ and $\delta$ are the elasticity coefficients subject to $0 < \alpha, \beta, \gamma, \delta < 1$. Equations 98a and 98b are convex under either Constant Returns to Scale (CRS), when $\alpha + \beta = 1$ and $\gamma + \delta = 1$, or Decreasing Returns to Scale (DRS), when $\alpha + \beta < 1$ and $\gamma + \delta < 1$. The final CDH score is the convex combination of the interior and surface scores as given by,

$$Y = w_i.Y_i + w_s.Y_s. \tag{99}$$

### 13.2.2 Constant Elasticity Earth Similarity Approach Score

The Constant Elasticity Earth Similarity Approach (CEESA) uses the following production function to estimate the habitability score of an exoplanet,

$$Y = (r.R^{\rho} + d.D^{\rho} + t.{T_s}^{\rho} + v.{V_e}^{\rho} + e.E^{\rho})^{\frac{\eta}{\rho}}, \tag{100}$$

where, $E$ is the fifth parameter denoting Orbital Eccentricity. The value of $\rho$ lies within $0 < \rho \leq 1$. The coefficients $r$, $d$, $t$, $v$ and $e$ lie in $(0,1)$ and sum to 1, $r + d + t + v + e = 1$. The value of $\eta$ is constrained by the scale of production used, $0 < \eta < 1$ under DRS and $\eta = 1$ under CRS.

## 13.3 Particle Swarm Optimization

Particle Swarm Optimization (PSO) [**?**] is a biologically inspired metaheuristic for finding the global minima of a function. Traditionally designed for unconstrained inputs, it works by iteratively converging a population of randomly initialized solutions, called particles, toward a globally optimal solution. Each particle in the population keeps track of its current position and the best solution it has encountered, called *pbest*. Each particle also has an associated velocity used to traverse the search space. The swarm keeps track of the overall best solution, called *gbest*. Each iteration of the swarm updates the velocity of the particle towards its *pbest* and the *gbest* values.

### 13.3.1 PSO for Unconstrained Optimization

Let $f(x)$ be the function to be minimized, where $x$ is a $d$-dimensional vector. $f(x)$ is also called the fitness function. Algorithm 45 outlines the approach to minimizing $f(x)$ using PSO. A set of particles are randomly initialized with a position and a velocity, where $l$ and $u$ are the lower and upper boundaries of the search space. The position of the particle corresponds to its associated solution. The algorithm initializes each particle's *pbest* to its initial position. The *pbest* position that corresponds to the minimum fitness is selected to be the *gbest* position of the swarm. Shi and Eberhart[**?**] discussed the use of inertial weights to regulate velocity to balance the global and local search. Upper and lower bounds limit velocity within $\pm v_{max}$.

On each iteration, the algorithm updates the velocity and position of each particle. For each particle, it picks two random numbers $u_g, u_p$ from a uniform distribution, $U(0,1)$ and updates the particle velocity. Here, $\omega$ is the inertial weight and $\lambda_g, \lambda_p$ are the global and particle learning rates. If the new position of the particle corresponds to a better fit than its *pbest*, the algorithm updates *pbest* to the new position. Once the algorithm has updated all particles, it updates *gbest* to the new overall best position. A suitable termination criteria for the swarm, under convex optimization, is to terminate when *gbest* has not changed by the end of an iteration.

### 13.3.2 PSO with Leaders for Constrained Optimization

Although PSO has eliminated the need to estimate the gradient of a function, as seen in Section 13.3.1, it still is not suitable for constrained optimization. The standard PSO algorithm does not ensure that the initial solutions are feasible, and neither does it guarantee that the individual solutions will converge to a feasible global solution. Solving the initialization

**Input:** $f(x)$, the function to minimize.
**Output:** global minimum of $f(x)$.

**for** each particle $i \leftarrow 1, n$ **do**
  $p_i \sim U(l,u)^d$
  $v_i \sim U(-|u-l|, |u-l|)^d$
  $pbest_i \leftarrow p_i$
$gbest \leftarrow \underset{pbest_i, i=1\ldots n}{\arg\min} f(pbest_i)$
**repeat**
  $oldbest \leftarrow gbest$
  **for** each particle $i \leftarrow 1 \ldots n$ **do**
    $u_p, u_g \sim U(0,1)$
    $v_i \leftarrow \omega.v_i + \lambda_g u_g(gbest - p_i) + \lambda_p u_p(pbest_i - p_i)$
    $v_i \leftarrow \text{sgn}(v_i).\max\{|v_{max}|, |v_i|\}$
    $p_i \leftarrow p_i + v_i$
    **if** $f(p_i) < f(pbest_i)$ **then**
      $pbest_i \leftarrow p_i$
  $gbest \leftarrow \underset{pbest_i, i=1\ldots n}{\arg\min} f(pbest_i)$
**until** $|oldbest - gbest| < threshold$
**return** $f(gbest)$

**Figure 45:** Algorithm for PSO.

problem is straightforward, resample each random solution from the uniform distribution until every initial solution is feasible. To solve the convergence problem each particle uses another particle's *pbest* value, called *lbest*, instead of its own to update its velocity. Algorithm 46 describes this process.

On each iteration, for each particle, the algorithm first picks two random numbers $u_g, u_p$. It then selects a *pbest* value from all particles in the swarm that is closest to the position of the particle being updated as its *lbest*. The *lbest* value substitutes $pbest_i$ in the velocity update equation. While updating *pbest* for the particle, the algorithm checks if the current fit is better than *pbest*, and performs the update if the current position satisfies all constraints. The algorithm updates *gbest* as before.

**Input:** $f(x)$, the function to minimize.
**Output:** global minimum of $f(x)$.

**for** each particle $i \leftarrow 1, n$ **do**
  **repeat**
    $p_i \sim U(l,u)^d$
  **until** $p_i$ satisfies all constraints
  $v_i \sim U(-|u-l|, |u-l|)^d$
  $pbest_i \leftarrow p_i$
$gbest \leftarrow \underset{pbest_i,\, i=1\ldots n}{\arg\min} f(pbest_i)$
**repeat**
  $oldbest \leftarrow gbest$
  **for** each particle $i \leftarrow 1 \ldots n$ **do**
    $u_p, u_g \sim U(0,1)$
    $lbest \leftarrow \underset{pbest_j,\, j=1\ldots n}{\arg\min} \|pbest_j - p_i\|^2$
    $v_i \leftarrow \omega.v_i + \lambda_g u_g(gbest - p_i) + \lambda_p u_p(lbest - p_i)$
    $p_i \leftarrow p_i + v_i$
    **if** $f(p_i) < f(pbest_i)$ **and** $p_i$ satisfies all constraints **then**
      $pbest_i \leftarrow p_i$
  $gbest \leftarrow \underset{pbest_i,\, i=1\ldots n}{\arg\min} f(pbest_i)$
**until** $|oldbest - gbest| < threshold$
**return** $f(gbest)$

**Figure 46:** Algorithm for CO by PSO.

## 13.4 Representing the Problem

A Constrained Optimization problem can represented as,

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & g_k(x) \le 0, \quad k = 1 \ldots q, \\ & h_l(x) = 0, \quad l = 1 \ldots r. \end{aligned}$$

Ray and Liew[**?**] describe a way to represent non-strict inequality constraints when optimizing using a particle swarm. Strict inequalities and equality constraints need to be converted to non-strict inequalities before being represented in the problem. Introducing an error threshold $\epsilon$ converts strict inequalities of the form $g_k{}'(x) < 0$ to non-strict inequalities of the form $g_k(x) = g_k{}'(x) + \epsilon \le 0$. A tolerance $\tau$ is used to transform equality constraints to a

pair of inequalities,

$$\begin{aligned} g_{(q+l)}(x) &= \quad h_l(x) - \tau \leq 0, \quad l = 1 \ldots r\,, \\ g_{(q+r+l)}(x) &= -h_l(x) - \tau \leq 0, \quad l = 1 \ldots r\,. \end{aligned}$$

Thus, $r$ equality constraints become $2r$ inequality constraints, raising the total number of constraints to $s = q + 2r$. For each solution $p_i$, $c_i$ denotes the constraint vector where, $c_{ik} = \max\{g_k(p_i), 0\},\ k = 1 \ldots s$. When $c_{ik} = 0,\ \forall k = 1 \ldots s$, the solution $p_i$ lies within the feasible region. When $c_{ik} > 0$, the solution $p_i$ violates the $k^{\text{th}}$ constraint.

### 13.4.1 Representing CDH Score Estimation

Under the aforementioned guidelines, the representation of CDH score estimation under CRS is,

$$\begin{aligned} \underset{\alpha,\beta,\gamma,\delta}{\text{minimize}} \quad & Y_i = -R^{\alpha}.D^{\beta}\,,\ Y_s = -{V_e}^{\gamma}.{T_s}^{\delta}\,, \\ \text{subject to} \quad & -\phi + \epsilon \leq 0,\ \forall \phi \in \{\alpha, \beta, \gamma, \delta\}\,, && \text{(100a)} \\ & \phi - 1 + \epsilon \leq 0,\ \forall \phi \in \{\alpha, \beta, \gamma, \delta\}\,, && \text{(100b)} \\ & \alpha + \beta - 1 - \tau \leq 0\,, && \text{(100c)} \\ & 1 - \alpha - \beta - \tau \leq 0\,, && \text{(100d)} \\ & \gamma + \delta - 1 - \tau \leq 0\,, && \text{(100e)} \\ & 1 - \gamma - \delta - \tau \leq 0\,. && \text{(100f)} \end{aligned}$$

Under DRS the constraints 100c to 100f are replaced with,

$$\begin{aligned} & \alpha + \beta + \epsilon - 1 \leq 0\,, && \text{(101a)} \\ & \gamma + \delta + \epsilon - 1 \leq 0\,. && \text{(101b)} \end{aligned}$$

### 13.4.2 Representing CEESA

The representation of CEESA score estimation (described in Section 13.2.2) under DRS is,

**Table 49:** Parameters from the PHL-EC used for the experiment.

| Parameter | Description | Unit |
|---|---|---|
| P. Radius | Estimated radius | Earth Units (EU) |
| P. Density | Density | Earth Units (EU) |
| P. Esc Vel | Escape velocity | Earth Units (EU) |
| P. Ts Mean | Mean Surface temperature | Kelvin (K) |
| P. Eccentricity | Orbital eccentricity | |

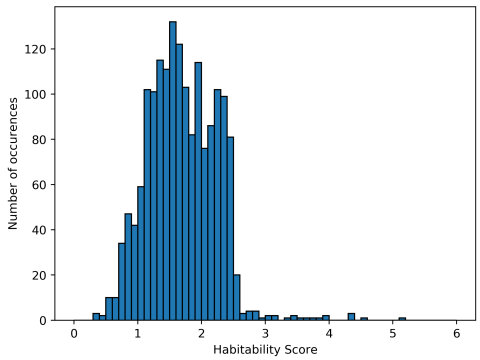
$$\begin{aligned}
\underset{r,d,t,v,e,\rho,\eta}{\text{minimize}} \quad & Y = -(r.R^{\rho} + d.D^{\rho} + t.T_s{}^{\rho} + v.V_e{}^{\rho} + e.E^{\rho})^{\frac{\eta}{\rho}} \\
\text{subject to} \quad & \rho - 1 \leq 0\,, && \text{(101c)} \\
& \rho - 1 + \epsilon \leq 0\,, && \text{(101d)} \\
& -\phi + \epsilon \leq 0,\ \forall \phi \in \{r, d, t, v, e, \eta\}\,, && \text{(101e)} \\
& \phi - 1 + \epsilon \leq 0,\ \forall \phi \in \{r, d, t, v, e, \eta\}\,, && \text{(101f)} \\
& (r + d + t + v + e) - 1 - \tau \leq 0\,, && \text{(101g)} \\
& 1 - (r - d - t - v - e) - \tau \leq 0\,. && \text{(101h)}
\end{aligned}$$

Under CRS there is no need for the parameter $\eta$ (since $\eta = 1$). Thus, the objective function for the problem reduces to,

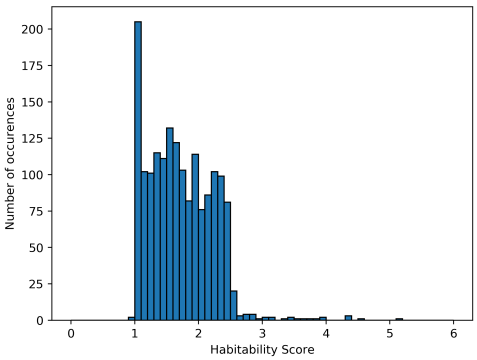
$$\underset{r,d,t,v,e,\rho,\eta}{\text{minimize}} \quad Y = -(r.R^{\rho} + d.D^{\rho} + t.T_s{}^{\rho} + v.V_e{}^{\rho} + e.E^{\rho})^{\frac{1}{\rho}}$$

## 13.5 Experiment and Results

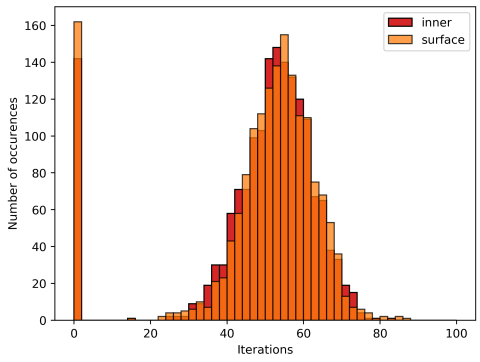
The data set used for estimating the Habitability Scores of exoplanets was the Confirmed Exoplanets Catalog maintained by the Planetary Habitability Laboratory (PHL) [**?**]. The catalog records observed and modeled parameters for exoplanets confirmed by the Extrasolar Planets Encyclopedia. Table 49 describes the parameters from the PHL Exoplanets' Catalog (PHL-EC) used for the experiment. Since surface temperature and eccentricity are not recorded in Earth Units, we normalized these values by dividing them with Earth's surface temperature (288 K) and eccentricity (0.017). PHL-EC assumes an Eccentricity of 0 when unavailable. The PHL-EC records empty values for planets whose surface temperature is not known. We chose to drop these records from the experiment.



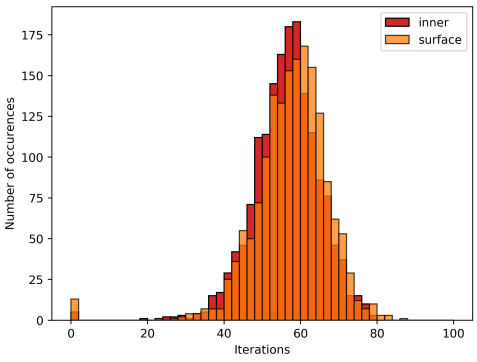

**(a)** CRS Score Distribution




**(b)** DRS Score Distribution

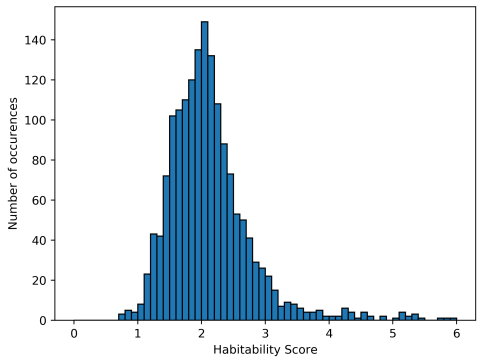



**(c)** CRS Iterations Distribution

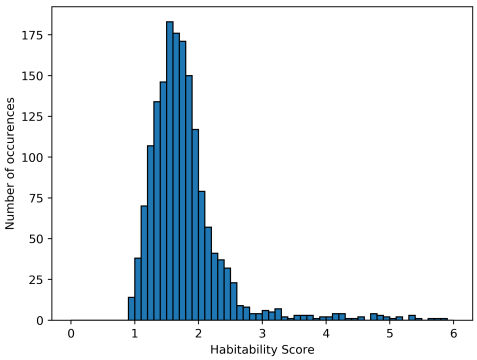



**(d)** DRS Iterations Distribution

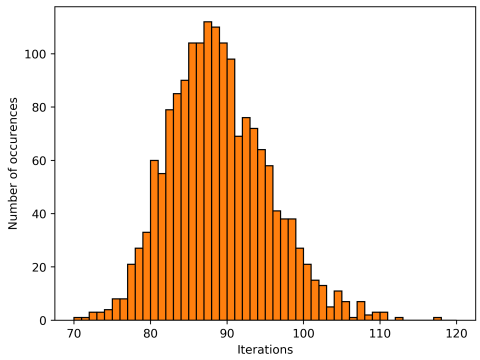
**Figure 47:** Plots for the Cobb-Douglas Habitability Score.



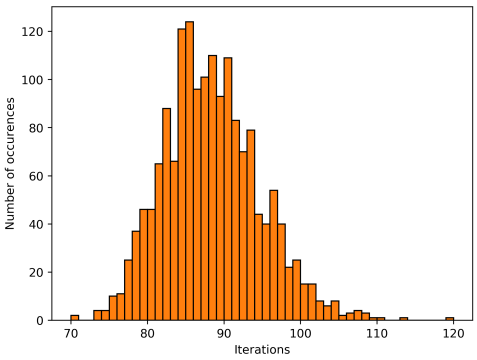

**(a)** CRS Score Distribution




**(b)** DRS Score Distribution




**(c)** CRS Iterations Distribution




**(d)** DRS Iterations Distribution

**Figure 48:** Plots for the Constant Elasticity Earth Similarity Approach.

The implementation resulted in a Python library now available on the Python Packaging Index under the name PSOPy [**?**]. It can be installed through pip as `pip install psopy` (also available in the github folder AstrIRG). Our implementation used $n = 25$ particles to traverse the search space, with learning rates $\lambda_g = 0.8$ and $\lambda_p = 0.2$. It used an inertial weight of $\omega = 0.6$ and upper and lower bounds $\pm 1.0$. We used an error threshold of $\epsilon = 1 \times 10^{-6}$ to convert strict inequalities to non-strict inequalities, and a tolerance of $\tau = 1 \times 10^{-7}$ to transform an equality constraint to a pair of inequalities. Further implementation details are discussed in the Appendix.

The plots in Figures 47a and 47b describe the distribution of the CDH scores across exoplanets tested from the PHL-EC. Figures 47c and 47d show the distribution of iterations required to converge to a global maxima. The spike at 0 is caused by particles converging to a *gbest* that does not shift from the original position (for a more detailed explanation see Appendix **??**). The plots in Figures 48b and 48b describe the distribution of the CEESA score across the exoplanets, while Figures 48d and 48c show the distribution of iterations to convergence. These graphs aggregate the results of optimizing the Habitability Production Functions (Equations 101, 101, 102 and 102) for each exoplanet in the PHL-EC by the method described in Algorithm 46.

Table 50 records the CDH scores for a sample of exoplanets under CRS at $w_i = 0.99$ and $w_s = 0.01$. $\alpha$, $\beta$, $\gamma$ and $\delta$ record the parameters of Equation 101. $Y_i$ and $Y_s$ record the maxima for the objective functions. $i_i$ and $i_s$ specify the number of iterations taken to converge to a stable *gbest* value. Under the Class column there are four categories for the planets — Psychroplanets (psy), Mesoplanets (mes), Non-Habitable planets (non) and Hypopsychroplanets (hyp). Table 51 records the CDH scores for a sample under DRS, with $\alpha$, $\beta$, $\gamma$ and $\delta$ recording the parameters of Equation 101.

Tables 52 and 53 record the estimated CEESA scores under CRS and DRS respectively. $r$, $d$, $t$, $v$, $e$, $\rho$ and $eta$ record the parameters of Equation 102 in Table 53 and the parameters of Equation 102 in Table 52. However, since under the CRS constraint, $\eta = 1$, there is no need for the parameter $\eta$ in Table 52. $i$ specifies the number of iterations taken to converge to the maxima.

These tables indicate that although CEESA has 7 parameters and 16 constraints under DRS, PSO takes a little over twice the number of iterations to converge as in each step of the CDH score estimation, which has 2 parameters and 5 constraints. This is a promising result as it indicates that the iterations required for converging increases sub-linearly with the number of parameters in the model. As for real time taken to converge, PSO took 666.85 s ($\approx$ 11 min 7 s) to estimate the CDH score under CRS for 1683 exoplanets, at an average of

**Table 50:** Estimated Cobb-Douglas Habitability scores under CRS.

| Name | Class | $\alpha$ | $\beta$ | $Y_i$ | $i_i$ | $\gamma$ | $\delta$ | $Y_s$ | $i_s$ | *CDHS* |
|---|---|---|---|---|---|---|---|---|---|---|
| GJ 176 b | non | 0.460 | 0.540 | 1.90 | 50 | 0.107 | 0.893 | 2.11 | 61 | 1.90 |
| GJ 667 C b | non | 0.423 | 0.577 | 1.71 | 58 | 0.692 | 0.308 | 1.81 | 54 | 1.71 |
| GJ 667 C e | psy | 0.129 | 0.871 | 1.40 | 50 | 0.258 | 0.742 | 1.39 | 55 | 1.40 |
| GJ 667 C f | psy | 0.534 | 0.466 | 1.40 | 48 | 0.865 | 0.135 | 1.39 | 47 | 1.40 |
| GJ 3634 b | non | 0.409 | 0.591 | 1.89 | 58 | 0.724 | 0.276 | 2.09 | 48 | 1.89 |
| HD 20794 c | non | 0.260 | 0.740 | 1.35 | 50 | 0.096 | 0.904 | 1.34 | 58 | 1.35 |
| HD 40307 e | non | 0.168 | 0.832 | 1.50 | 49 | 0.636 | 0.364 | 1.53 | 63 | 1.50 |
| HD 40307 f | non | 0.702 | 0.298 | 1.52 | 68 | 0.303 | 0.697 | 1.55 | 45 | 1.52 |
| HD 40307 g | psy | 0.964 | 0.036 | 1.82 | 51 | 0.083 | 0.917 | 1.98 | 55 | 1.82 |
| Kepler-186 f | hyp | 0.338 | 0.662 | 1.17 | 50 | 0.979 | 0.021 | 1.12 | 40 | 1.17 |
| Proxima Cen b | psy | 0.515 | 0.484 | 1.12 | 37 | 0.755 | 0.245 | 1.07 | 0 | 1.12 |
| TRAPPIST-1 b | non | 0.319 | 0.681 | 1.09 | 0 | 0.801 | 0.199 | 0.89 | 0 | 1.09 |
| TRAPPIST-1 c | non | 0.465 | 0.535 | 1.06 | 0 | 0.935 | 0.065 | 1.14 | 26 | 1.06 |
| TRAPPIST-1 d | mes | 0.635 | 0.365 | 0.77 | 34 | 0.475 | 0.525 | 0.73 | 47 | 0.77 |
| TRAPPIST-1 e | psy | 0.145 | 0.855 | 0.92 | 0 | 0.897 | 0.103 | 0.83 | 55 | 0.92 |
| TRAPPIST-1 g | hyp | 0.226 | 0.774 | 1.13 | 43 | 0.876 | 0.124 | 1.09 | 0 | 1.13 |

198.11 ms for each planet for each individual score (interior and surface) of the CDH score. For CDH estimation under DRS, it took 638.69 s (≈ 10 min 39 s) at an average of 189.75 ms for each part of the CDH score. The CEESA calculations, requiring a single estimate, took a little over half the CDH estimation execution time to run. Under DRS it took a total of 370.86 s (≈ 6 min 11 s) at 220.36 ms per planet, while under CRS it took 356.92 s (≈ 5 min 57 s) at 212.07 ms per planet.

## 13.6 Conclusion

Particle Swarm Optimization mainly draws its advantages from being easy to implement and highly parallelizable. The algorithms described in Section 13.3 use simple operators and straightforward logic. What is especially noticeable is the lack of the need for a gradient, allowing PSO to work in high dimensional search spaces with a large number of constraints, precisely what is needed in a potential Habitability score estimate. Further, particles of the swarm, in most implementations operate independently during each iteration, their updates can occur simultaneously and even asynchronously, yielding much faster execution times than those outlined in Section 13.5. However, since strict inequalities and equality constraints are not exactly represented, the resulting solution may not be as accurate as direct methods.

**Table 51:** Estimated Cobb-Douglas Habitability scores under DRS.

| Name | Class | $\alpha$ | $\beta$ | $Y_i$ | $i_i$ | $\gamma$ | $\delta$ | $Y_s$ | $i_s$ | *CDHS* |
|---|---|---|---|---|---|---|---|---|---|---|
| GJ 176 b | non | 0.395 | 0.604 | 1.90 | 59 | 0.372 | 0.627 | 2.11 | 56 | 1.90 |
| GJ 667 C b | non | 0.781 | 0.218 | 1.71 | 58 | 0.902 | 0.097 | 1.81 | 57 | 1.71 |
| GJ 667 C e | psy | 0.179 | 0.820 | 1.40 | 49 | 0.234 | 0.765 | 1.39 | 60 | 1.40 |
| GJ 667 C f | psy | 0.704 | 0.295 | 1.40 | 64 | 0.398 | 0.601 | 1.39 | 61 | 1.40 |
| GJ 3634 b | non | 0.602 | 0.397 | 1.89 | 59 | 0.429 | 0.570 | 2.09 | 77 | 1.89 |
| HD 20794 c | non | 0.014 | 0.985 | 1.35 | 50 | 0.116 | 0.883 | 1.34 | 45 | 1.35 |
| HD 40307 e | non | 0.752 | 0.247 | 1.50 | 60 | 0.677 | 0.322 | 1.53 | 50 | 1.50 |
| HD 40307 f | non | 0.887 | 0.112 | 1.52 | 51 | 0.261 | 0.738 | 1.55 | 60 | 1.52 |
| HD 40307 g | psy | 0.300 | 0.699 | 1.82 | 62 | 0.785 | 0.214 | 1.98 | 56 | 1.82 |
| Kepler-186 f | hyp | 0.073 | 0.926 | 1.17 | 46 | 0.740 | 0.259 | 1.12 | 51 | 1.17 |
| Proxima Cen b | psy | 0.045 | 0.954 | 1.12 | 57 | 0.216 | 0.783 | 1.07 | 53 | 1.12 |
| TRAPPIST-1 b | non | 0.102 | 0.897 | 1.09 | 41 | 0.000 | 0.000 | 1.00 | 65 | 1.09 |
| TRAPPIST-1 c | non | 0.471 | 0.528 | 1.06 | 44 | 0.227 | 0.772 | 1.14 | 57 | 1.06 |
| TRAPPIST-1 d | mes | 0.000 | 0.000 | 1.00 | 67 | 0.000 | 0.000 | 1.00 | 59 | 1.00 |
| TRAPPIST-1 e | psy | 0.000 | 0.000 | 1.00 | 55 | 0.000 | 0.000 | 1.00 | 57 | 1.00 |
| TRAPPIST-1 g | hyp | 0.888 | 0.111 | 1.13 | 47 | 0.949 | 0.050 | 1.09 | 46 | 1.13 |

**Table 52:** Estimated Constant Elasticity Earth Similarity Approach scores under CRS.

| Name | Class | $r$ | $d$ | $t$ | $v$ | $e$ | $\rho$ | $\eta$ | *CDHS* | $i$ |
|---|---|---|---|---|---|---|---|---|---|---|
| GJ 176 b | non | 0.194 | 0.020 | 0.315 | 0.465 | 0.006 | 0.398 | 1.000 | 1.88 | 86 |
| GJ 667 C b | non | 0.162 | 0.289 | 0.090 | 0.087 | 0.372 | 0.836 | 1.000 | 3.54 | 107 |
| GJ 667 C e | psy | 0.373 | 0.032 | 0.134 | 0.304 | 0.157 | 0.217 | 1.000 | 1.25 | 71 |
| GJ 667 C f | psy | 0.394 | 0.006 | 0.043 | 0.360 | 0.196 | 0.490 | 1.000 | 1.44 | 81 |
| GJ 3634 b | non | 0.351 | 0.122 | 0.006 | 0.069 | 0.453 | 0.439 | 1.000 | 2.89 | 96 |
| HD 20794 c | non | 0.101 | 0.077 | 0.691 | 0.071 | 0.059 | 0.756 | 1.000 | 1.58 | 94 |
| HD 40307 e | non | 0.069 | 0.091 | 0.097 | 0.173 | 0.569 | 0.768 | 1.000 | 5.29 | 94 |
| HD 40307 f | non | 0.285 | 0.161 | 0.053 | 0.443 | 0.058 | 0.342 | 1.000 | 1.42 | 73 |
| HD 40307 g | psy | 0.156 | 0.010 | 0.081 | 0.302 | 0.451 | 0.612 | 1.000 | 7.15 | 94 |
| Kepler-186 f | hyp | 0.036 | 0.017 | 0.082 | 0.383 | 0.483 | 0.929 | 1.000 | 1.68 | 85 |
| Proxima Cen b | psy | 0.352 | 0.383 | 0.103 | 0.059 | 0.103 | 0.936 | 1.000 | 0.89 | 83 |
| TRAPPIST-1 b | non | 0.148 | 0.147 | 0.344 | 0.269 | 0.093 | 0.767 | 1.000 | 0.94 | 81 |
| TRAPPIST-1 c | non | 0.038 | 0.060 | 0.575 | 0.321 | 0.005 | 0.602 | 1.000 | 1.17 | 86 |
| TRAPPIST-1 d | mes | 0.023 | 0.065 | 0.475 | 0.391 | 0.045 | 0.830 | 1.000 | 0.84 | 79 |
| TRAPPIST-1 e | psy | 0.176 | 0.464 | 0.253 | 0.103 | 0.004 | 0.920 | 1.000 | 0.86 | 81 |
| TRAPPIST-1 g | hyp | 0.060 | 0.086 | 0.310 | 0.540 | 0.004 | 0.848 | 1.000 | 0.97 | 86 |

**Table 53:** Estimated Constant Elasticity Earth Similarity Approach scores under DRS.

| Name | Class | $r$ | $d$ | $t$ | $v$ | $e$ | $\rho$ | $\eta$ | $CDHS$ | $i$ |
|---|---|---|---|---|---|---|---|---|---|---|
| GJ 176 b | non | 0.304 | 0.001 | 0.375 | 0.271 | 0.050 | 0.467 | 0.808 | 1.52 | 85 |
| GJ 667 C b | non | 0.297 | 0.010 | 0.318 | 0.052 | 0.322 | 0.682 | 0.730 | 2.36 | 90 |
| GJ 667 C e | psy | 0.230 | 0.286 | 0.137 | 0.199 | 0.148 | 0.551 | 0.906 | 1.14 | 85 |
| GJ 667 C f | psy | 0.397 | 0.035 | 0.152 | 0.402 | 0.014 | 0.793 | 0.999 | 1.31 | 100 |
| GJ 3634 b | non | 0.178 | 0.175 | 0.005 | 0.194 | 0.447 | 0.894 | 0.657 | 2.07 | 94 |
| HD 20794 c | non | 0.073 | 0.142 | 0.452 | 0.190 | 0.144 | 0.953 | 0.635 | 1.20 | 78 |
| HD 40307 e | non | 0.156 | 0.307 | 0.185 | 0.033 | 0.319 | 0.428 | 0.939 | 2.69 | 88 |
| HD 40307 f | non | 0.272 | 0.231 | 0.064 | 0.305 | 0.127 | 0.676 | 0.802 | 1.28 | 77 |
| HD 40307 g | psy | 0.113 | 0.219 | 0.066 | 0.454 | 0.148 | 0.711 | 0.991 | 3.26 | 92 |
| Kepler-186 f | hyp | 0.039 | 0.159 | 0.116 | 0.329 | 0.357 | 0.253 | 0.919 | 1.35 | 70 |
| Proxima Cen b | psy | 0.272 | 0.173 | 0.284 | 0.193 | 0.079 | 0.615 | 0.114 | 0.99 | 75 |
| TRAPPIST-1 b | non | 0.488 | 0.151 | 0.039 | 0.193 | 0.129 | 0.151 | 0.014 | 0.99 | 87 |
| TRAPPIST-1 c | non | 0.172 | 0.236 | 0.275 | 0.242 | 0.075 | 0.969 | 0.962 | 1.06 | 80 |
| TRAPPIST-1 d | mes | 0.106 | 0.308 | 0.075 | 0.218 | 0.293 | 0.844 | 0.017 | 0.99 | 93 |
| TRAPPIST-1 e | psy | 0.189 | 0.266 | 0.192 | 0.094 | 0.260 | 0.371 | 0.006 | 0.99 | 84 |
| TRAPPIST-1 g | hyp | 0.326 | 0.186 | 0.143 | 0.278 | 0.067 | 0.315 | 0.021 | 1.00 | 76 |

Despite this, using PSO to calculate the habitability scores is beneficial when the number of input parameters are large, which further increases the number of constraints, resulting in a model too infeasible for traditional optimization methods.

Determining habitability from exoplanet requires that determining parameters are collectively considered [**?**] before coming up with a conclusion as no single factor alone contributes to it. Our proposed model would serve as an indicator while looking for new habitable worlds. Eccentricity may have some effect on habitabilty and the models for computation should address that. CDHS doesn't, at least for the Trappist system (otherwise considered a set of potentially habitable exoplanets) since the eccentricities for all memebers of the Trappist system are 0 identically. CDHS, being a product model then will render the habitability score to be 0, not in agreement with observations and overall opinion in the community. This is the reason CESSA is considered in the process of habitability score computation (additive nature of the model). One might wonder why metaheuristic optimization was applied on two different optimization problems. We hope, our clarification would suffice.

However, the functional forms considered to compute the habitability score pose challenges. As we intend to add more parameters (such as eccentricity) to the basic model [**?**][**?**], the functional form tends to suffer from curvature violation [**?**][**?**]. Even though global op-

tima is guaranteed, premature convergence and local oscillations are hard to mitigate. An attempt to address such issues, with moderate success, could be found in [**?**]. The greatest contribution of the manuscript is to propose an evolutionary algorithm to track dynamic functions of the type that allow for the oscillation that were instead mitigated with SGA in [**?**]. Consequently, a Pyhton library is integrated with the open source tool suite, an add on for coding enthusiasts to test our method.

## REFERENCES


[Shostak2017] Shostak S., Do Aliens Exist? Your Tax Dollars May Hold the Truth, 2017, Newsweek url:http://www.newsweek.com/do-aliens-exist-your-tax-dollars-may-hold-answer-753151

[Schwieterman2017] Schwieterman E. W. et al., 2017, Exoplanet Biosignatures: A Review of Remotely Detectable Signs of Life, arXiv preprint arXiv:1705.05791

[Johnson2018] Johnson M., Kepler and K2 Mission Overview, 2018, NASA, url:https://www.nasa.gov/mission_pages/kepler/overview

[Presto2017] LoPresto M. C., Ochoa H., 2017, Searching for potentially habitable extra solar planets: a directed-study using real data from the NASA Kepler-Mission, Physics Education, Vol. 52, Number 6, p. 065016

[Shuch2011] Shuch H. P., 2011, Searching for Extraterrestrial Intelligence: SETI past, present, and future, Springer Science and Business Media, Chapter 2, p. 13-18

[Cochran1991] Cochran W. D., Hatzes A. P., Hancock T. J., 1991, Constraints on the companion object to HD 114762, The Astrophysical Journal, Vol. 380, p. L35-L38

[Bora2016] Bora K. et al., 2016, CD-HPF: New habitability score via data analytic modeling, Astronomy and Computing, Vol. 17, p. 129-143

[Saha2017] Saha S. et al., 2017, Theoretical Validation of Potential Habitability via Analytical and Boosted Tree Methods: An Optimistic Study on Recently Discovered Exoplanets, arXiv preprint arXiv:1712.01040

[Eberhart1995] Eberhart R., Kennedy J., 1995, A new optimizer using particle swarm theory, Micro Machine and Human Science, 1995. MHS'95., Proceedings of the Sixth International Symposium on, IEEE, p. 39-43

[Shi1998] Shi Y., Eberhart R., A modified particle swarm optimizer, Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on, IEEE, p. 69-73

[Poli2007] Poli R., 2007, An analysis of publications on particle swarm optimization applications, Essex, UK: Department of Computer Science, University of Essex

[Poli2008] Poli R., 2008, Analysis of the publications on the applications of particle swarm optimisation, Journal of Artificial Evolution and Applications

[Méndez2017] Méndez A., PHL's Exoplanets Catalog, 2017, Planetary Habitability Laboratory, University of Puerto Rico at Arecibo., url:http://phl.upr.edu/projects/habitable-exoplanets-catalog/data/database"

[Ray2001] Ray T., Liew K. M., 2001, A swarm with an effective information sharing mechanism for unconstrained and constrained single objective optimisation problems, Evolutionary Computation, 2001. Proceedings of the 2001 Congress on, IEEE, Vol. 1, p. 75-80

[Sobin2016] Sobin CC et al., 2016, CISER: An Amoebiasis inspired Model for Epidemic Message Propagation in DTN, arXiv preprint arXiv:1608.07670

[Sarkar2016] Sarkar J. et al., 2016, CDSFA Stochastic Frontier Analysis Approach to Revenue Modeling in Large Cloud Data Centers, arXiv preprint arXiv:1610.00624

[Saha2018] Saha S. et al., 2018, Model Visualization in understanding rapid growth of a journal in an emerging area, arXiv preprint arXiv:1803.04644

[Saha2018] Saha S. et al., 2018, MACHINE LEARNING IN ASTRONOMY: A WORKMANâĂŹS MANUAL, ResearchGate, p. 1-255

[Saha2016] Saha S. et al., 2016, A novel revenue optimization model to address the operation and maintenance cost of a data center, Journal of Cloud Computing, SpringerOpen, Vol. 5, Number 1, p. 1

[Ginde2016] Ginde G. et al., 2016, ScientoBASE: a framework and model for computing scholastic indicators of non-local influence of journals via native data acquisition algorithms, Scientometrics, Springer, Vol. 108, Number 3, p. 1479-1529

[Agrawal2018] Agrawal S. et al., 2018, A Comparative Analysis of the Cobb-Douglas Habitability Score (CDHS) with the Earth Similarity Index (ESI), arXiv preprint arXiv:1804.11176

[Theophilus2018] Theophilus A. J., Saha S., Basak S., 2018, PSOPy: A Python implementation of Particle Swarm Optimization, url:"https://pypi.org/project/psopy/"

# 14 Economics, Chaos theory and Machine Learning

# 15 ADAPTIVE LEARNING RATES: A NEW PARADIGM IN DEEP LEARNING

# 16 PYTHON CODES

## 16.1 Fuzzy Neural Nets implementation

The code shown in this section is an implementation of Back Propagation algorithm written in python. No in-built libraries are used and the code is written from scratch. The reason behind implementing the algorithm from scratch is that one can clearly look at every step from core and can visualize the movement of data between layers and the propagation of error back to the network. Stochastic gradient Descent is used to reduce the error margins. The code runs for 500 epochs and the learning rate is kept at 0.001. Vowel data set is used as input to the network that comprises of 871 patterns, 3 input features and 6 output classes . The input layer and the output layer is fuzzified by using PI membership function. Code 1 uses the class-independent fuzzification and Code 2 fuzzifies input layer on the basis of class values. 5-fold cross validation is used to achieve good accuracy and classes are balanced in each fold.

Code 1 (Class Independent fuzzification)

```
from random import seed
from random import randrange
from random import random
from csv import reader
from math import exp
from sklearn.metrics import confusion\_matrix
import numpy as np
import math

# Load a CSV file
def load_csv(filename):
        dataset = list()
        with open(filename, 'r') as file:
                csv_reader = reader(file)
                for row in csv_reader:
                        if not row:
                                continue
                        dataset.append(row)
        return dataset

def minmax(dataset):
        minmax = list()
```

```
        stats = [[min(column), max(column)] for column in zip(*dataset)]
        return stats

# Rescale dataset columns to the range 0-1
def normalize(dataset, minmax):
        for row in dataset:
                for i in range(len(row)-1):
                        row[i] = (row[i] - minmax[i][0]) / (minmax[i][1] - minmax[i
                            ][0])

# Convert string column to float
def column_to_float(dataset, column):
        for row in dataset:
                try:
                        row[column] = float(row[column].strip())
                except ValueError:
                        print("Error with row",column,":",row[column])
                        pass

# Convert string column to integer
def column_to_int(dataset, column):
        for row in dataset:
                row[column] = int(row[column])

# Find the min and max values for each column


# Split a dataset into k folds
def cross_validation_split(dataset, n_folds):
        dataset_split = list()
        dataset_copy = list(dataset)
        fold_size = int(len(dataset) / n_folds)
        for i in range(n_folds):
                fold = list()
                while len(fold) < fold_size:
                        index = randrange(len(dataset_copy))
                        fold.append(dataset_copy.pop(index))
                dataset_split.append(fold)
        return dataset_split

# Calculate accuracy percentage
def accuracy_met(actual, predicted):
```

```
        correct = 0
        for i in range(len(actual)):
                if actual[i] == predicted[i]:
                        correct += 1
        return correct / float(len(actual)) * 100.0

# Evaluate an algorithm using a cross validation split
def run_algorithm(dataset, algorithm, n_folds, *args):
        #print(dataset)
        folds = cross_validation_split(dataset, n_folds)
        #for fold in folds:
                #print("Fold {} \n \n".format(fold))
        scores = list()
        for fold in folds:
                #print("Test Fold {} \n \n".format(fold))
                train_set = list(folds)
                train_set.remove(fold)
                train_set = sum(train_set, [])
                test_set = list()
                for row in fold:
                        row_copy = list(row)
                        test_set.append(row_copy)
                        row_copy[-1] = None
                predicted = algorithm(train_set, test_set, *args)
                actual = [row[-1] for row in fold]
                #print(predicted)
                #print(actual)
                accuracy = accuracy_met(actual, predicted)
                cm = confusion_matrix(actual, predicted)
                print('\n'.join(['  '.join(['{:4}'.format(item) for item in row]) for
                    row in cm]))
                #confusionmatrix = np.matrix(cm)
                FP = cm.sum(axis=0) - np.diag(cm)
                FN = cm.sum(axis=1) - np.diag(cm)
                TP = np.diag(cm)
                TN = cm.sum() - (FP + FN + TP)
                print('False␣Positives\n␣{}'.format(FP))
                print('False␣Negetives\n␣{}'.format(FN))
                print('True␣Positives\n␣{}'.format(TP))
                print('True␣Negetives\n␣{}'.format(TN))
                TPR = TP/(TP+FN)
                print('Sensitivity␣\n␣{}'.format(TPR))
```

```
                TNR = TN/(TN+FP)
                print('Specificity \n {}'.format(TNR))
                Precision = TP/(TP+FP)
                print('Precision \n {}'.format(Precision))
                Recall = TP/(TP+FN)
                print('Recall \n {}'.format(Recall))
                Acc = (TP+TN)/(TP+TN+FP+FN)
                print('ÃĄccuracy \n{}'.format(Acc))
                Fscore = 2*(Precision*Recall)/(Precision+Recall)
                print('FScore \n{}'.format(Fscore))
                scores.append(accuracy)


# Calculate neuron activation for an input
def activate(weights, inputs):
        activation = weights[-1]
        for i in range(len(weights)-1):
                activation += weights[i] * inputs[i]
        return activation

# Transfer neuron activation
def function(activation):
        return 1.0 / (1.0 + exp(-activation))

# Forward propagate input to a network output
def forward_propagate(network, row):
        inputs = row
        #print("input row{}\n".format(inputs))
        for layer in network:
                new_inputs = []
                for neuron in layer:
                        activation = activate(neuron['weights'], inputs)
                        neuron['output'] = function(activation)
                        new_inputs.append(neuron['output'])
                inputs = new_inputs
        #print("output row{}\n".format(inputs))
        return inputs

# Calculate the derivative of an neuron output
def function_derivative(output):
        return output * (1.0 - output)
```

```
# Backpropagate error and store in neurons
def backprop_error(network, expected):
        for i in reversed(range(len(network))):
                layer = network[i]
                errors = list()
                if i != len(network)-1:
                        for j in range(len(layer)):
                                error = 0.0
                                for neuron in network[i + 1]:
                                        error += (neuron['weights'][j] * neuron['
                                            delta'])
                                errors.append(error)
                else:
                        for j in range(len(layer)):
                                neuron = layer[j]
                                errors.append(expected[j] - neuron['output'])
                for j in range(len(layer)):
                        neuron = layer[j]
                        neuron['delta'] = errors[j] * function_derivative(neuron['
                            output'])

# Update network weights with error
def change_weights(network, row, l_rate):
        for i in range(len(network)):
                inputs = row[:-1]
                if i != 0:
                        inputs = [neuron['output'] for neuron in network[i - 1]]
                for neuron in network[i]:
                        for j in range(len(inputs)):
                                neuron['weights'][j] += l_rate * neuron['delta'] *
                                    inputs[j]
                        neuron['weights'][-1] += l_rate * neuron['delta']

#To fuzzify the output layer
def fuzzyout(row,mean,stdev,n_outputs):
    z=list()
    mu=list()
    muINT=list()
    rowclass=row[-1]-1
    #print(rowclass)
    for k in range(n_outputs):
        sumz=0
```

```
        for j in range(9):
            interm=pow((row[j]-mean[k][j])/stdev[k][j],2)
            sumz=sumz+interm
            #print("row{}".format(row[j]))
            #print("mean{}".format(mean[rowclass][j]))
            #print("sum{}".format(sumz))
        weightedZ=math.sqrt(sumz)
        memMU=1/(1+(weightedZ/5))
        if 0 <= memMU <= 0.5:
            memMUINT=2*pow(memMU,2)
        else:
            temp=1-memMU
            memMUINT=1-(2*pow(temp,2))
        mu.append(memMU)
        z.append(weightedZ)
        muINT.append(memMUINT)
    return muINT

# Train a network for a fixed number of epochs
def neural_network_train(network, train, l_rate, n_epoch, n_outputs):
        #print(dataset)
        for epoch in range(n_epoch):
                #print(train)
                for row in train:
                        outputs = forward_propagate(network, row)
                        #print(outputs)
                        expected = fuzzyout(row,mean,stdev,n_outputs)
                        #print("input row{}\n".format(row))
                        #expected[row[-1]-1] = 1
                        #print("expected row{}\n".format(expected))
                        backprop_error(network, expected)
                        change_weights(network, row, l_rate)

# Initialize a network
def init_net(n_inputs, n_hidden, n_outputs):
        network = list()
        hidden_layer = [{'weights':[random() for i in range(n_inputs + 1)]} for i in
            range(n_hidden)]
        network.append(hidden_layer)
        output_layer = [{'weights':[random() for i in range(n_hidden + 1)]} for i in
            range(n_outputs)]
        network.append(output_layer)
```

```
        return network

# Make a prediction with a network
def predict(network, row):
        outputs = forward_propagate(network, row)
        #print(outputs)
        indexOut=outputs.index(max(outputs))+1
        #print(indexOut)
        return indexOut

# Backpropagation Algorithm With Stochastic Gradient Descent
def back_propagation(train, test, l_rate, n_epoch, n_hidden):
        n_inputs = len(train[0]) - 1

        n_outputs = len(set([row[-1] for row in train]))

        network = init_net(n_inputs, n_hidden, n_outputs)
        #print("initialize network {}\n".format(network))
        neural_network_train(network, train, l_rate, n_epoch, n_outputs)
        #print("network {}\n".format(network))
        predictions = list()
        for row in test:
                prediction = predict(network, row)
                predictions.append(prediction)
        return(predictions)

# Test Backprop on Seeds dataset
seed(1)
# load and prepare data
filename = 'data.csv'
dataset = load_csv(filename)
for i in range(len(dataset[0])-1):
        column_to_float(dataset, i)
# convert class column to integers
column_to_int(dataset, len(dataset[0])-1)
#normalize input variables
#minmax = minmax(dataset)
#normalize(dataset, minmax)
# evaluate algorithm
n_folds = 5             #k=5
l_rate = 0.2            #learning rate
n_epoch = 500           #epochs
```

```
n_hidden = 7              #since the number of inputs are 3*3, and number of classes
    are 6

#this part of the computation is to fuzzify inputs, PI membership function is taken
    for fuzzyfication. each feature vector is fuzzyfied into low, meadium and high
    degree of membership
center=[[250,575,900],[700,1625,2550],[1800,2500,3200]]
#print(center)
rad=[[650,325,650],[1850,925,1850],[1400,700,1400]]
fuzzy=list()

for i in range(870):
    data=dataset[i]
    #print(data)
    l=0
    value=list()
    for j in range(3):
        d=data[j]
        for k in range(3):
            r=rad[j][k]/2
            eucleanD=math.pow((d-center[j][k]),2)
            eDist=math.sqrt(eucleanD)
            if eDist <= r :
                val = 1- 2*math.pow(eDist/(r*2),2)
                value.insert(l,val)
            else:
                y=eDist/rad[j][k]
                x=(1-(eDist/rad[j][k]))
                val = 2*math.pow(x,2)
                value.insert(l,val)
            l=l+1
    value.insert(l,data[-1])
    fuzzy.insert(i,value)
#fuzzy is the modified dataset after fuzzification
#print(fuzzy)

#This part of the code computes Mean and standard deviation of every feature of all
    classes to fuzzyfies the output layer of the network
classes=[row[-1] for row in fuzzy]
Unique=np.unique(classes)
dataset_split=list()
fold_size=int(len(Unique))
```

```
for i in range(fold_size):
        fold=list()
        for row in fuzzy:
                if row[-1] == Unique[i]:
                        fold.append(row)
        dataset_split.append(fold)
i=0
mean=list()
stdev=list()
j=0
for fold in dataset_split:
        x=list()
        y=list()
        z=list()
        x1=list()
        y1=list()
        z1=list()
        x2=list()
        y2=list()
        z2=list()
        for row in fold:
            if row[-1] == Unique[j]:
                x.append(row[0])
                y.append(row[1])
                z.append(row[2])
                x1.append(row[3])
                y1.append(row[4])
                z1.append(row[5])
                x2.append(row[6])
                y2.append(row[7])
                z2.append(row[8])
        m1=sum(x)/float(len(x))
        m2=sum(y)/float(len(y))
        m3=sum(z)/float(len(z))
        m4=sum(x1)/float(len(x1))
        m5=sum(y1)/float(len(y1))
        m6=sum(z1)/float(len(z1))
        m7=sum(x2)/float(len(x2))
        m8=sum(y2)/float(len(y2))
        m9=sum(z2)/float(len(z2))
        mean.append([m1,m2,m3,m4,m5,m6,m7,m8,m9])
        st1=sum([pow(val-m1,2) for val in x])/float(len(x)-1)
```

```
        st2=sum([pow(val-m2,2) for val in y])/float(len(y)-1)
        st3=sum([pow(val-m3,2) for val in z])/float(len(z)-1)
        st4=sum([pow(val-m4,2) for val in x1])/float(len(x1)-1)
        st5=sum([pow(val-m5,2) for val in y1])/float(len(y1)-1)
        st6=sum([pow(val-m6,2) for val in z1])/float(len(z1)-1)
        st7=sum([pow(val-m7,2) for val in x2])/float(len(x2)-1)
        st8=sum([pow(val-m8,2) for val in y2])/float(len(y2)-1)
        st9=sum([pow(val-m9,2) for val in z2])/float(len(z2)-1)
        std1=math.sqrt(st1)
        std2=math.sqrt(st2)
        std3=math.sqrt(st3)
        std4=math.sqrt(st4)
        std5=math.sqrt(st5)
        std6=math.sqrt(st6)
        std7=math.sqrt(st7)
        std8=math.sqrt(st8)
        std9=math.sqrt(st9)
        stdev.append([std1,std2,std3,std4,std5,std6,std7,std8,std9])
        j=j+1
#print(mean)
#print(stdev)
run_algorithm(fuzzy, back_propagation, n_folds, l_rate, n_epoch, n_hidden)
```

## 16.2 II (Class dependent fuzzification-after fuzzification the number of input features is 18, the network has single hidden layer with 10 neurons)

```
# Backpropogation algorithm implemented on the Vowel Dataset. The dataset is attached
    with the file. Number of input features are three and output classes are six and
    sample size is 871.
# The fuzzification is class dependent. The input features are fuzzified using PI
    membership function and fed into the neural network. The output layer is also
    fuzzified.
# 5-fold cross validation is used to achieve better accuracy. There are unbalanced
    classes in training set.
# Confusion matrix, precision, recall, accuracy and Fscore is computed using sklearn
    package


from random import seed
from random import randrange
from random import random
from csv import reader
from math import exp
from sklearn.metrics import confusion_matrix
import numpy as np
import math

def Fuzzy_input(N):
        fuzzy=list()
        coeff=pow(2,N-1)
        for i in range(871):
            data=dataset[i]
            #print(data)
            l=0
            member=list()
            value=list()
            for j in range(3):
                d=data[j]
                for k in range(6):
                        if d<= a[k][j]:
                                member.append(0)
                        elif a[k][j]<d<=p[k][j]:
                                member.append(coeff* pow((d-a[k][j])/(r[k][j]-a[k][j
```

```
                                ]),N))
                        elif p[k][j] < d <= r[k][j]:
                                member.append(1-(coeff* pow((r[k][j]-d)/(r[k][j]-a[k
                                    ][j]),N)))
                        elif r[k][j] < d <= q[k][j]:
                                member.append(1-(coeff* pow((d-r[k][j])/(b[k][j]-r[k
                                    ][j]),N)))
                        elif q[k][j] < d <= b[k][j]:
                                member.append(coeff* pow((b[k][j]-d)/(b[k][j]-r[k][j
                                    ]),N))
                        else:
                                member.append(0)
                        l=l+1
                member.append(data[-1])
                #print("member {}".format(member))
                fuzzy.append(member)
        #print(fuzzy)
        return fuzzy


# Load a CSV file
def load_csv(filename):
        dataset = list()
        with open(filename, 'r') as file:
                csv_reader = reader(file)
                for row in csv_reader:
                        if not row:
                                continue
                        dataset.append(row)
        return dataset

def minmax(dataset):
        minmax = list()
        stats = [[min(column), max(column)] for column in zip(*dataset)]
        return stats

# Rescale dataset columns to the range 0-1
def normalize(dataset, minmax):
        for row in dataset:
                for i in range(len(row)-1):
                        row[i] = (row[i] - minmax[i][0]) / (minmax[i][1] - minmax[i
                            ][0])
```

```
# Convert string column to float
def column_to_float(dataset, column):
        for row in dataset:
                try:
                        row[column] = float(row[column].strip())
                except ValueError:
                        print("Error with row",column,":",row[column])
                        pass

# Convert string column to integer
def column_to_int(dataset, column):
        for row in dataset:
                row[column] = int(row[column])

# Find the min and max values for each column




# Split a dataset into k folds
def cross_validation_split(dataset, n_folds):
        size_class=list()
        size_class.append(int(72/n_folds))
        size_class.append(int(89/n_folds))
        size_class.append(int(172/n_folds))
        size_class.append(int(151/n_folds))
        size_class.append(int(207/n_folds))
        size_class.append(int(180/n_folds))
        class_info = [1,2,3,4,5,6]
        #print(size_class)
        dataset_split = list()
        dataset_copy = list(dataset)


        fold_size = int(len(dataset) / n_folds)
        for k in range(n_folds):
                #print("k = {}".format(k))
                i=0
                fold=list()
                for j in range(len(class_info)):
                #for j in range():
```

```
                        count=0
                        while (count < size_class[j]):
                                data=dataset_copy[i]
                                #print("{}    ".format(dataset_copy[i]))
                                if data[-1] == class_info[j]:
                                        #print("{}    {}   {}  {}   ".format(data[-1],
                                            count,size_class[j],class_info[j]))
                                        item=dataset_copy.pop(i)
                                        fold.append(item)
                                        count=count+1
                                else:
                                        i=i+1

                dataset_split.append(fold)
                #print("dataset {} \n\n\n".format(dataset_split))
        return dataset_split

# Calculate accuracy percentage
def accuracy_met(actual, predicted):
        correct = 0
        for i in range(len(actual)):
                if actual[i] == predicted[i]:
                        correct += 1
        return correct / float(len(actual)) * 100.0

# Evaluate an algorithm using a cross validation split
def run_algorithm(dataset, algorithm, n_folds, *args):
        #print(dataset)
        folds = cross_validation_split(dataset, n_folds)
        #for fold in folds:
                #print("Fold {} \n \n".format(fold))
        scores = list()
        for fold in folds:
                #print("Test Fold {} \n \n".format(fold))
                train_set = list(folds)
                train_set.remove(fold)
                train_set = sum(train_set, [])
                test_set = list()
                for row in fold:
                        row_copy = list(row)
                        test_set.append(row_copy)
                        row_copy[-1] = None
```

```
                predicted = algorithm(train_set, test_set, *args)
                actual = [row[-1] for row in fold]
                #print(predicted)
                #print(actual)
                accuracy = accuracy_met(actual, predicted)
                cm = confusion_matrix(actual, predicted)
                print('\n'.join([''.join(['{:4}'.format(item) for item in row]) for
                    row in cm]))
                #confusionmatrix = np.matrix(cm)
                FP = cm.sum(axis=0) - np.diag(cm)
                FN = cm.sum(axis=1) - np.diag(cm)
                TP = np.diag(cm)
                TN = cm.sum() - (FP + FN + TP)
                print('False␣Positives\n␣{}'.format(FP))
                print('False␣Negetives\n␣{}'.format(FN))
                print('True␣Positives\n␣{}'.format(TP))
                print('True␣Negetives\n␣{}'.format(TN))
                TPR = TP/(TP+FN)
                print('Sensitivity␣\n␣{}'.format(TPR))
                TNR = TN/(TN+FP)
                print('Specificity␣\n␣{}'.format(TNR))
                Precision = TP/(TP+FP)
                print('Precision␣\n␣{}'.format(Precision))
                Recall = TP/(TP+FN)
                print('Recall␣\n␣{}'.format(Recall))
                Acc = (TP+TN)/(TP+TN+FP+FN)
                print('ÃĄccuracy␣\n{}'.format(Acc))
                Fscore = 2*(Precision*Recall)/(Precision+Recall)
                print('FScore␣\n{}'.format(Fscore))
                scores.append(accuracy)


# Calculate neuron activation for an input
def activate(weights, inputs):
        activation = weights[-1]
        for i in range(len(weights)-1):
                activation += weights[i] * inputs[i]
        return activation

# Transfer neuron activation
def function(activation):
        return 1.0 / (1.0 + exp(-activation))
```

```
# Forward propagate input to a network output
def forward_propagate(network, row):
        inputs = row
        #print("input row{}\n".format(inputs))
        for layer in network:
                new_inputs = []
                for neuron in layer:
                        activation = activate(neuron['weights'], inputs)
                        neuron['output'] = function(activation)
                        new_inputs.append(neuron['output'])
                inputs = new_inputs
        #print("output row{}\n".format(inputs))
        return inputs

# Calculate the derivative of an neuron output
def function_derivative(output):
        return output * (1.0 - output)

# Backpropagate error and store in neurons
def backprop_error(network, expected):
        for i in reversed(range(len(network))):
                layer = network[i]
                errors = list()
                if i != len(network)-1:
                        for j in range(len(layer)):
                                error = 0.0
                                for neuron in network[i + 1]:
                                        error += (neuron['weights'][j] * neuron['
                                            delta'])
                                errors.append(error)
                else:
                        for j in range(len(layer)):
                                neuron = layer[j]
                                errors.append(expected[j] - neuron['output'])
                for j in range(len(layer)):
                        neuron = layer[j]
                        neuron['delta'] = errors[j] * function_derivative(neuron['
                            output'])

# Update network weights with error
def change_weights(network, row, l_rate):
```

```
        for i in range(len(network)):
                inputs = row[:-1]
                if i != 0:
                        inputs = [neuron['output'] for neuron in network[i - 1]]
                for neuron in network[i]:
                        for j in range(len(inputs)):
                                neuron['weights'][j] += l_rate * neuron['delta'] *
                                    inputs[j]
                        neuron['weights'][-1] += l_rate * neuron['delta']

#To fuzzify the output layer
def fuzzyout(row,mean,stdev,n_outputs):
    z=list()
    mu=list()
    muINT=list()
    rowclass=row[-1]-1
    #print(rowclass)
    for k in range(n_outputs):
        sumz=0
        for j in range(9):
            interm=pow((row[j]-mean[k][j])/stdev[k][j],2)
            sumz=sumz+interm
            #print("row{}".format(row[j]))
            #print("mean{}".format(mean[rowclass][j]))
            #print("sum{}".format(sumz))
        weightedZ=math.sqrt(sumz)
        memMU=1/(1+(weightedZ/5))
        if 0 <= memMU <= 0.5:
            memMUINT=2*pow(memMU,2)
        else:
            temp=1-memMU
            memMUINT=1-(2*pow(temp,2))
        mu.append(memMU)
        z.append(weightedZ)
        muINT.append(memMUINT)
    return muINT

# Train a network for a fixed number of epochs
def neural_network_train(network, train, l_rate, n_epoch, n_outputs):
        #print(dataset)
        for epoch in range(n_epoch):
                #print(train)
```

```
                for row in train:
                        outputs = forward_propagate(network, row)
                        #print(outputs)
                        expected = fuzzyout(row,mean,stdev,n_outputs)
                        #print("input row{}\n".format(row))
                        #expected[row[-1]-1] = 1
                        #print("expected row{}\n".format(expected))
                        backprop_error(network, expected)
                        change_weights(network, row, l_rate)

# Initialize a network
def init_net(n_inputs, n_hidden, n_outputs):
        network = list()
        hidden_layer = [{'weights':[random() for i in range(n_inputs + 1)]} for i in
            range(n_hidden)]
        network.append(hidden_layer)
        output_layer = [{'weights':[random() for i in range(n_hidden + 1)]} for i in
            range(n_outputs)]
        network.append(output_layer)
        return network

# Make a prediction with a network
def predict(network, row):
        outputs = forward_propagate(network, row)
        #print(outputs)
        indexOut=outputs.index(max(outputs))+1
        #print(indexOut)
        return indexOut

# Backpropagation Algorithm With Stochastic Gradient Descent
def back_propagation(train, test, l_rate, n_epoch, n_hidden):
        n_inputs = len(train[0]) - 1

        n_outputs = len(set([row[-1] for row in train]))

        network = init_net(n_inputs, n_hidden, n_outputs)
        #print("initialize network {}\n".format(network))
        neural_network_train(network, train, l_rate, n_epoch, n_outputs)
        #print("network {}\n".format(network))
        predictions = list()
        for row in test:
                prediction = predict(network, row)
```

```
                predictions.append(prediction)
        return(predictions)

# Test Backprop on Seeds dataset
seed(1)
# load and prepare data
filename = 'data.csv'
dataset = load_csv(filename)
for i in range(len(dataset[0])-1):
        column_to_float(dataset, i)
# convert class column to integers
column_to_int(dataset, len(dataset[0])-1)


#To divide the dataset class-wise; each fold has individual class
classes=[row[-1] for row in dataset]
Unique=np.unique(classes)
dataset_split=list()
fold_size=int(len(Unique))
for i in range(fold_size):
    fold=list()
    for row in dataset:
        if row[-1] == Unique[i]:
            fold.append(row)
    dataset_split.append(fold)


#this part of the computation is to fuzzify inputs, PI membership function is taken
    for fuzzyfication. each feature vector is fuzzyfied into six class based fuzzy set

a=list()
p=list()
r=list()
q=list()
b=list()
j=0
for fold in dataset_split:
        x=list()
        y=list()
        z=list()
        x=[row[0] for row in fold]
        y=[row[1] for row in fold]
```

```
        z=[row[2] for row in fold]
        m1=sum(x)/float(len(x))
        m2=sum(y)/float(len(y))
        m3=sum(z)/float(len(z))
        r.append([m1,m2,m3])
        p1=m1-((max(x)-min(x))/2)
        p2=m2-((max(y)-min(y))/2)
        p3=m3-((max(z)-min(z))/2)
        p.append([p1,p2,p3])
        q1=m1+((max(x)-min(x))/2)
        q2=m2+((max(y)-min(y))/2)
        q3=m3+((max(z)-min(z))/2)
        q.append([q1,q2,q3])
        a1=m1-(q1-p1)
        a2=m2-(q2-p2)
        a3=m3-(q3-p3)
        b1=m1+(q1-p1)
        b2=m2+(q2-p2)
        b3=m3+(q3-p3)
        a.append([a1,a2,a3])
        b.append([b1,b2,b3])
N=1
fuzzy=list()
fuzzy=Fuzzy_input(N)
#fuzzy is the modified dataset after fuzzification
#print(fuzzy)

#This part of the code computes Mean and standard deviation of every feature of all
    classes to fuzzyfies the output layer of the network
classes=[row[-1] for row in fuzzy]
Unique=np.unique(classes)
dataset_split=list()
fold_size=int(len(Unique))
for i in range(fold_size):
        fold=list()
        for row in fuzzy:
                if row[-1] == Unique[i]:
                        fold.append(row)
        dataset_split.append(fold)
i=0
mean=list()
stdev=list()
```

```
j=0
for fold in dataset_split:
        x=list()
        y=list()
        z=list()
        x1=list()
        y1=list()
        z1=list()
        x2=list()
        y2=list()
        z2=list()
        for row in fold:
            if row[-1] == Unique[j]:
                x.append(row[0])
                y.append(row[1])
                z.append(row[2])
                x1.append(row[3])
                y1.append(row[4])
                z1.append(row[5])
                x2.append(row[6])
                y2.append(row[7])
                z2.append(row[8])
        m1=sum(x)/float(len(x))
        m2=sum(y)/float(len(y))
        m3=sum(z)/float(len(z))
        m4=sum(x1)/float(len(x1))
        m5=sum(y1)/float(len(y1))
        m6=sum(z1)/float(len(z1))
        m7=sum(x2)/float(len(x2))
        m8=sum(y2)/float(len(y2))
        m9=sum(z2)/float(len(z2))
        mean.append([m1,m2,m3,m4,m5,m6,m7,m8,m9])
        st1=sum([pow(val-m1,2) for val in x])/float(len(x)-1)
        st2=sum([pow(val-m2,2) for val in y])/float(len(y)-1)
        st3=sum([pow(val-m3,2) for val in z])/float(len(z)-1)
        st4=sum([pow(val-m4,2) for val in x1])/float(len(x1)-1)
        st5=sum([pow(val-m5,2) for val in y1])/float(len(y1)-1)
        st6=sum([pow(val-m6,2) for val in z1])/float(len(z1)-1)
        st7=sum([pow(val-m7,2) for val in x2])/float(len(x2)-1)
        st8=sum([pow(val-m8,2) for val in y2])/float(len(y2)-1)
        st9=sum([pow(val-m9,2) for val in z2])/float(len(z2)-1)
        std1=math.sqrt(st1)
```

```
        std2=math.sqrt(st2)
        std3=math.sqrt(st3)
        std4=math.sqrt(st4)
        std5=math.sqrt(st5)
        std6=math.sqrt(st6)
        std7=math.sqrt(st7)
        std8=math.sqrt(st8)
        std9=math.sqrt(st9)
        stdev.append([std1,std2,std3,std4,std5,std6,std7,std8,std9])
        j=j+1
#print(mean)
#print(stdev)
n_folds = 5             #k=5
l_rate = 0.2            #learning rate
n_epoch = 500           #epochs
n_hidden = 10           #since the number of inputs are 18, and number of classes are 6
run_algorithm(fuzzy, back_propagation, n_folds, l_rate, n_epoch, n_hidden)
```

**********

**Github Repository:** https://github.com/mathurarchana77/neuralnetwork

**Data Set**: Vowel Dataset - 871 rows, 3 input features and 6 output classes. Classes are unbalanced in original dataset and the code balances the data to keep the distribution of every class in every fold uniform.

## 16.3 Code III (Quick Reduct Algorithm)

```
#QuickReduct is an algorithm for feature selection that computes degree of
    dependencies of a set of features on another set

#Rough set theory deals with concepts that are vague and creates an approximate
    description of patterns for data processing. It basically defines the crisp set
    with rough representation. The quick reduct starts with the complete initial set
    and removes one feature at a time by ensuring identical predictive capability of
    the decision feature as that of the original feature set.

from random import seed
from random import randrange
from random import random
from csv import reader
from math import exp
from sklearn.metrics import confusion_matrix
```

```
import numpy as np
import math
from itertools import combinations

def load_csv(filename):
        dataset = list()
        with open(filename, 'r') as file:
                csv_reader = reader(file)
                for row in csv_reader:
                        if not row:
                                continue
                        dataset.append(row)
        return dataset

# Convert string column to float
def str_column_to_float(dataset, column):
        for row in dataset:
                try:
                        row[column] = float(row[column].strip())
                except ValueError:
                        print("Error with row",column,":",row[column])
                        pass

# Convert string column to integer
def str_column_to_int(dataset, column):
        for row in dataset:
                row[column] = int(row[column])


def dependency(dataset,num):
        total=0
        dependency=0
        for j in range(3):
                fold=list()
                for i in range(len(dataset)):
                        data=dataset[i]
                        #print(i)
                        if data[-1]==j:
                                fold.append(dataset[i])
                #print("Fold {}".format(fold))
                count=len(fold)
                #print("count {}".format(count))
```

```
                for k in range(len(fold)):
                        list1=fold[k]
                        for l in range(len(dataset)):
                                #print("len{}".format(len(fold)))
                                list2=dataset[l]
                                if list1[:num]==list2[:num] and list1[-1]!=list2[-1]:
                                        count = count-1
                                        #print("Count inside {}".format(count))
                                        break
                total=total+count
                #print("total {}",format(total))
        dependency=total/len(dataset)
        #print("{}".format(dependency))
        return dependency

def generate_new_dataset(row,l):
        #print(row)
        X = np.empty((8, 0))  # there are 8 samples in the dataset
        #print(X)
        for i in range(len(row)):
                col=row[i]
                x=[row_new[col] for row_new in dataset]
                x=np.array([x])
                #print(np.transpose(x))
                #print(x)
                x=x.T
                X=np.append(X, x, axis=1)
        x=[row[-1] for row in dataset]
        X=np.append(X, [[x[0]],[x[1]],[x[2]],[x[3]],[x[4]],[x[5]],[x[6]],[x[7]]],
            axis=1)
        X=np.array(X).tolist()
        print("X {}".format(X))
        return X


filename = 'TestData.csv'
dataset = load_csv(filename)
for i in range(len(dataset[0])-1):
        str_column_to_float(dataset, i)
# convert class column to integers
str_column_to_int(dataset, len(dataset[0])-1)
#print(dataset)
```

```
#this is fuzzify inpput based on class belongin granulation
dp=dependency(dataset,4)
#print("dp{}".format(dp))
n=4    # the number of features are 4
initial_val=[0,1,2,3]
comb=combinations([0,1,2,3],3)
while n>1:
        for row in  comb:
                #print(row)
                data_X=generate_new_dataset(row,len(row))
                #print(data_X)
                dp_data_X=dependency(data_X,len(row))
                #print("new dp{}".format(dp_data_X))
                if dp > dp_data_X:
                        continue
                else:
                        n=n-1
                        break

        #print(row)
        comb=combinations(row,n)
print("final reduct {}".format(row))
```

**********

Github repository : https://github.com/mathurarchana77/QuickReduct

**********

## 16.4 Code IV (Multi Layer Perceptron)

```
# Multilayer Perceptron on the Vowel Dataset
from random import seed
from random import randrange
from random import random
from csv import reader
from math import exp
from sklearn.metrics import confusion_matrix
from sklearn.metrics import cohen_kappa_score
import numpy as np
import csv

# Load a CSV file
```

```
def loadCsv(filename):
        trainSet = []

        lines = csv.reader(open(filename, 'r'))
        dataset = list(lines)
        for i in range(len(dataset)):
                for j in range(4):
                        #print("DATA {}".format(dataset[i]))
                        dataset[i][j] = float(dataset[i][j])
                trainSet.append(dataset[i])
        return trainSet

def minmax(dataset):
        minmax = list()
        stats = [[min(column), max(column)] for column in zip(*dataset)]
        return stats

# Rescale dataset columns to the range 0-1
def normalize(dataset, minmax):
        for row in dataset:
                for i in range(len(row)-1):
                        row[i] = (row[i] - minmax[i][0]) / (minmax[i][1] - minmax[i
                            ][0])

# Convert string column to float
def column_to_float(dataset, column):
        for row in dataset:
                try:
                        row[column] = float(row[column])
                except ValueError:
                        print("Error with row",column,":",row[column])
                        pass

# Convert string column to integer
def column_to_int(dataset, column):
        class_values = [row[column] for row in dataset]
        unique = set(class_values)
        lookup = dict()
        for i, value in enumerate(unique):
                lookup[value] = i
        for row in dataset:
                row[column] = lookup[row[column]]
```

```
        return lookup

# Find the min and max values for each column


# Split a dataset into k folds
def cross_validation_split(dataset, n_folds):
        dataset_split = list()
        dataset_copy = list(dataset)
        fold_size = int(len(dataset) / n_folds)
        for i in range(n_folds):
                fold = list()
                while len(fold) < fold_size:
                        index = randrange(len(dataset_copy))
                        fold.append(dataset_copy.pop(index))
                dataset_split.append(fold)
        return dataset_split

# Calculate accuracy percentage
def accuracy_met(actual, predicted):
        correct = 0
        for i in range(len(actual)):
                if actual[i] == predicted[i]:
                        correct += 1
        return correct / float(len(actual)) * 100.0

# Evaluate an algorithm using a cross validation split
def run_algorithm(dataset, algorithm, n_folds, *args):

        folds = cross_validation_split(dataset, n_folds)
        #for fold in folds:
                #print("Fold {} \n \n".format(fold))
        scores = list()
        for fold in folds:
                #print("Test Fold {} \n \n".format(fold))
                train_set = list(folds)
                train_set.remove(fold)
                train_set = sum(train_set, [])
                test_set = list()
                for row in fold:
                        row_copy = list(row)
                        test_set.append(row_copy)
```

```
                row_copy[-1] = None
            predicted = algorithm(train_set, test_set, *args)
            actual = [row[-1] for row in fold]
            accuracy = accuracy_met(actual, predicted)
            cm = confusion_matrix(actual, predicted)
            print('\n'.join(['' .join(['{:4}'.format(item) for item in row]) for
                row in cm]))
            #confusionmatrix = np.matrix(cm)
            FP = cm.sum(axis=0) - np.diag(cm)
            FN = cm.sum(axis=1) - np.diag(cm)
            TP = np.diag(cm)
            TN = cm.sum() - (FP + FN + TP)
            print('False␣Positives\n␣{}'.format(FP))
            print('False␣Negetives\n␣{}'.format(FN))
            print('True␣Positives\n␣{}'.format(TP))
            print('True␣Negetives\n␣{}'.format(TN))
            TPR = TP/(TP+FN)
            print('Sensitivity␣\n␣{}'.format(TPR))
            TNR = TN/(TN+FP)
            print('Specificity␣\n␣{}'.format(TNR))
            Precision = TP/(TP+FP)
            print('Precision␣\n␣{}'.format(Precision))
            Recall = TP/(TP+FN)
            print('Recall␣\n␣{}'.format(Recall))
            Acc = (TP+TN)/(TP+TN+FP+FN)
            print('ÃĄccuracy␣\n{}'.format(Acc))
            Fscore = 2*(Precision*Recall)/(Precision+Recall)
            print('FScore␣\n{}'.format(Fscore))
            k=cohen_kappa_score(actual, predicted)
            print('ÃĞohen␣Kappa␣\n{}'.format(k))
            scores.append(accuracy)
        return scores

# Calculate neuron activation for an input
def activate(weights, inputs):
        activation = weights[-1]
        for i in range(len(weights)-1):
                activation += weights[i] * inputs[i]
        return activation

# Transfer neuron activation
def transfer(activation):
```

```
        return 1.0 / (1.0 + exp(-activation))

# Forward propagate input to a network output
def forward_propagate(network, row):
        inputs = row
        for layer in network:
                new_inputs = []
                for neuron in layer:
                        activation = activate(neuron['weights'], inputs)
                        neuron['output'] = transfer(activation)
                        new_inputs.append(neuron['output'])
                inputs = new_inputs
        return inputs

# Calculate the derivative of an neuron output
def transfer_derivative(output):
        return output * (1.0 - output)

# Backpropagate error and store in neurons
def backward_propagate_error(network, expected):
        for i in reversed(range(len(network))):
                layer = network[i]
                errors = list()
                if i != len(network)-1:
                        for j in range(len(layer)):
                                error = 0.0
                                for neuron in network[i + 1]:
                                        error += (neuron['weights'][j] * neuron['
                                            delta'])
                                errors.append(error)
                else:
                        for j in range(len(layer)):
                                neuron = layer[j]
                                errors.append(expected[j] - neuron['output'])
                for j in range(len(layer)):
                        neuron = layer[j]
                        neuron['delta'] = errors[j] * transfer_derivative(neuron['
                            output'])

# Update network weights with error
def update_weights(network, row, l_rate):
        for i in range(len(network)):
```

```
                inputs = row[:-1]
                if i != 0:
                        inputs = [neuron['output'] for neuron in network[i - 1]]
                for neuron in network[i]:
                        for j in range(len(inputs)):
                                temp = l_rate * neuron['delta'] * inputs[j] + mu *
                                    neuron['prev'][j]

                                neuron['weights'][j] += temp
                                #print("neuron weight{} \n".format(neuron['weights'][
                                    j]))
                                neuron['prev'][j] = temp
                        temp = l_rate * neuron['delta'] + mu * neuron['prev'][-1]
                        neuron['weights'][-1] += temp
                        neuron['prev'][-1] = temp


# Train a network for a fixed number of epochs
def train_network(network, train, l_rate, n_epoch, n_outputs):
        for epoch in range(n_epoch):
                for row in train:
                        outputs = forward_propagate(network, row)
                        #print(network)
                        expected = [0 for i in range(n_outputs)]
                        expected[row[-1]] = 1
                        #print("expected row{}\n".format(expected))
                        backward_propagate_error(network, expected)
                        update_weights(network, row, l_rate)

# Initialize a network
def initialize_network(n_inputs, n_hidden, n_outputs):
        network = list()
        hidden_layer = [{'weights':[random() for i in range(n_inputs + 1)], 'prev':[0
            for i in range(n_inputs+1)]} for i in range(n_hidden)]
        network.append(hidden_layer)
        hidden_layer = [{'weights':[random() for i in range(n_inputs + 1)], 'prev':[0
            for i in range(n_inputs+1)]} for i in range(n_hidden)]
        network.append(hidden_layer)
        output_layer = [{'weights':[random() for i in range(n_hidden + 1)],'prev':[0
           for i in range(n_hidden+1)]} for i in range(n_outputs)]
        network.append(output_layer)
        #print(network)
```

```
        return network

# Make a prediction with a network
def predict(network, row):
        outputs = forward_propagate(network, row)
        return outputs.index(max(outputs))

# Backpropagation Algorithm With Stochastic Gradient Descent
def back_propagation(train, test, l_rate, n_epoch, n_hidden):
        n_inputs = len(train[0]) - 1
        n_outputs = len(set([row[-1] for row in train]))
        network = initialize_network(n_inputs, n_hidden, n_outputs)
        train_network(network, train, l_rate, n_epoch, n_outputs)
        #print("network {}\n".format(network))
        predictions = list()
        for row in test:
                prediction = predict(network, row)
                predictions.append(prediction)
        return(predictions)

# Test Backprop on Seeds dataset
seed(1)
# load and prepare data
filename = 'data.csv'
dataset = loadCsv(filename)
for i in range(len(dataset[0])-1):
        column_to_float(dataset, i)
# convert class column to integers
column_to_int(dataset, len(dataset[0])-1)
# normalize input variables
minmax = minmax(dataset)
normalize(dataset, minmax)
# evaluate algorithm
n_folds = 5
l_rate = 0.1
mu=0.001
n_epoch = 1500
n_hidden = 4
scores = run_algorithm(dataset, back_propagation, n_folds, l_rate, n_epoch, n_hidden)

#print('Scores: %s' % scores)
#print('Mean Accuracy: %.3f%%' % (sum(scores)/float(len(scores))))
```

## 16.5 Code V (k Nearest Neighbors)

```
import math
import operator
import numpy as np
from sklearn.metrics import confusion_matrix
#from pandas_ml import ConfusionMatrix
def loaddata(filename, file1):
        trainSet = []
        testSet = []
        lines = csv.reader(open(filename, 'r'))
        dataset = list(lines)
        for i in range(len(dataset)):
                for j in range(4):
                        dataset[i][j] = float(dataset[i][j])
                trainSet.append(dataset[i])
        lines = csv.reader(open(file1, 'r'))
        dataset = list(lines)
        for i in range(len(dataset)):
                for j in range(4):
                        dataset[i][j] = float(dataset[i][j])
                testSet.append(dataset[i])
        return trainSet, testSet

def eud(i1, i2, length):
        dist = 0
        for x in range(length):
                dist += pow((i1[x] - i2[x]), 2)
        return math.sqrt(dist)

def nb(train, test, k):
        d = []
        length = len(test)-1
        #print(trainingSet)
        for x in range(len(train)):
                #print('Test {}\n'.format(testInstance))
                dist = eud(test, train[x], length)
                d.append((train[x], dist))
                #print(distances)
        d.sort(key=operator.itemgetter(1))
        #print(distances)
        nei = []
```

```
        for x in range(k):
                nei.append(d[x][0])
        #print(neighbors)
        return nei

def res(neighbors):
        classVotes = {}
        for x in range(len(neighbors)):
                response = neighbors[x][-1]
                if response in classVotes:
                        classVotes[response] += 1
                else:
                        classVotes[response] = 1
        #print(classVotes)
        sortedVotes = sorted(classVotes.items(), key=operator.itemgetter(1), reverse=
            True)
        return sortedVotes[0][0]

def main():
        filename = 'DataBalancedcsv.csv'
        file1 = 'Databalancedtest.csv'
        #sRatio = 0.80
        train, test = loaddata(filename, file1)
        predictions=[]
        k = 7
        trueValue = []
        for x in range(len(test)):
                nei = nb(train, test[x], k)
                result = res(nei)
                predictions.append(result)
                trueValue.append(test[x][-1])
        #print('> predicted=' + repr(predictions) + ', actual=' + repr(trueValue))
        cm = confusion_matrix(trueValue, predictions)
        #for i in range(6):
                #for j in range(6):
                        #print('{:4}'.format(cm[i][j])),
                #print
        print('k={}'.format(k))
        print('\n\n Confusion Matrix \n')
        print('\n'.join([''.join(['{:4}'.format(item) for item in row]) for row in cm
            ]))
        #confusionmatrix = np.matrix(cm)
```

```
        FP = cm.sum(axis=0) - np.diag(cm)
        FN = cm.sum(axis=1) - np.diag(cm)
        TP = np.diag(cm)
        TN = cm.sum() - (FP + FN + TP)
        print('False Positives\n {}'.format(FP))
        print('False Negetives\n {}'.format(FN))
        print('True Positives\n {}'.format(TP))
        print('True Negetives\n {}'.format(TN))
        TPR = TP/(TP+FN)
        print('Sensitivity \n {}'.format(TPR))
        TNR = TN/(TN+FP)
        print('Specificity \n {}'.format(TNR))
        Precision = TP/(TP+FP)
        print('Precision \n {}'.format(Precision))
        Recall = TP/(TP+FN)
        print('Recall\n {}'.format(Recall))
        Acc = (TP+TN)/(TP+TN+FP+FN)
        print('Accuracy \n{}'.format(Acc))
        Fscore = 2*(Precision*Recall)/(Precision+Recall)
        print('FScore \n{}'.format(Fscore))
main()
```

## 16.6 VI (Bayesian Classification0

```
import csv
import random
import math
import numpy as np
from sklearn.metrics import confusion_matrix
#from pandas_ml import ConfusionMatrix
def loadCsv(filename):
        lines = csv.reader(open(filename, 'r'))
        dataset = list(lines)
        for i in range(len(dataset)):
                dataset[i] = [float(x) for x in dataset[i]]
        return dataset

def mean(numbers):
        return sum(numbers)/float(len(numbers))

def std(numbers):
        avg = mean(numbers)
        variance = sum([pow(x-avg,2) for x in numbers])/float(len(numbers)-1)
        return math.sqrt(variance)

def splitData(dataset, sRatio):
        trainSize = int(len(dataset) * sRatio)
        trainSet = []
        copy = list(dataset)
        while len(trainSet) < trainSize:
                index = random.randrange(len(copy))
                trainSet.append(copy.pop(index))
        return [trainSet, copy]

def process(dataset):
        foreveryclass=[]
        for attribute in zip(*dataset):
                x = mean(attribute)
                y = std(attribute)
                foreveryclass.append([x,y])
        del foreveryclass[-1]
        return foreveryclass

def ClassData(dataset):
```

```
        classdivision = {}
        for i in range(len(dataset)):
                vector = dataset[i]
                if (vector[-1] not in classdivision):
                        classdivision[vector[-1]] = []
                classdivision[vector[-1]].append(vector)
        return classdivision

def summary(dataset):
        divided = ClassData(dataset)
        #print(separated)
        PValues = {}  # a dictionary to store mean stdev of all attributes classwise
        for classValue, instances in divided.items(): #returns a list of key, value
            pairs for tuples
                PValues[classValue] = process(instances)

        return PValues

def Prob(x, mean, stdev):
        exponent = math.exp(-(math.pow(x-mean,2)/(2*math.pow(stdev,2))))
        return (1 / (math.sqrt(2*math.pi) * stdev)) * exponent

def ClassProb(ProcessValues, inputVector):
        probabilities = {}
        for classValue, classSummaries in ProcessValues.items():
                probabilities[classValue] = 1
                for i in range(len(classSummaries)):
                        mean, stdev = classSummaries[i]
                        x = inputVector[i]
                        probabilities[classValue] *= Prob(x, mean, stdev)
        #print(probabilities)
        return probabilities

def predict(ProcessValues, inputVector):
        probabilities = ClassProb(ProcessValues, inputVector)
        bestLabel, bestProb = None, -1
        for classValue, probability in probabilities.items():
                if bestLabel is None or probability > bestProb:
                        bestProb = probability
                        bestLabel = classValue
        return bestLabel
```

```
def getPredictions(ProcessValues, testSet):
        predictions = []
        y_true = []
        for i in range(len(testSet)):
                result = predict(ProcessValues, testSet[i])
                predictions.append(result)
        #print(predictions)
        for i in range(len(testSet)):
                vector=testSet[i]
                y_true.append(vector[-1])
        #print(y_true)
        return [y_true, predictions]

def getAccuracy(testSet, predictions):
        correct = 0
        for i in range(len(testSet)):
                if testSet[i][-1] == predictions[i]:
                        correct += 1
        return (correct/float(len(testSet))) * 100.0

def main():
        #filename = 'DataBalancedcsv.csv'
        file = 'data.csv'
        sRatio = 0.80
        dataset = loadCsv(file)
        training, test = splitData(dataset, sRatio)
        #print('Split {} rows into train={} and test={} rows'.format(len(dataset),
            len(trainingSet), len(testSet)))
        # prepare model
        PV = summary(training)
        # test model
        y_true, predictions = getPredictions(PV, test)
        cm = confusion_matrix(y_true, predictions)

        print('\n\n Confusion Matrix \n')
        print('\n'.join([''.join(['{:4}'.format(item) for item in row]) for row in cm
            ]))
        #confusionmatrix = np.matrix(cm)
        FP = cm.sum(axis=0) - np.diag(cm)
        FN = cm.sum(axis=1) - np.diag(cm)
        TP = np.diag(cm)
        TN = cm.sum() - (FP + FN + TP)
```

```
        print('False Positives\n {}'.format(FP))
        print('False Negetives\n {}'.format(FN))
        print('True Positives\n {}'.format(TP))
        print('True Negetives\n {}'.format(TN))
        TPR = TP/(TP+FN)
        print('Sensitivity \n {}'.format(TPR))
        TNR = TN/(TN+FP)
        print('Specificity \n {}'.format(TNR))
        Precision = TP/(TP+FP)
        print('Precision \n {}'.format(Precision))
        Recall = TP/(TP+FN)
        print('Recall \n {}'.format(Recall))
        Acc = (TP+TN)/(TP+TN+FP+FN)
        print('ÃĄccuracy \n{}'.format(Acc))
        Fscore = 2*(Precision*Recall)/(Precision+Recall)
        print('FScore \n{}'.format(Fscore))

main()
```

**********

```
 Confusion Matrix

   9   3   0   0   3   0
   0  19   0   0   0   0
   0   0  23   0   7   0
   0   0   0  23   0   1
   5   0   5   0  30   2
   0   2   0   8   1  34
False Positives
 [ 5  5  5  8 11  3]
False Negetives
 [ 6  0  7  1 12 11]
True Positives
 [ 9 19 23 23 30 34]
True Negetives
 [155 151 140 143 122 127]
Sensitivity
 [ 0.6         1.          0.76666667  0.95833333  0.71428571  0.75555556]
Specificity
 [ 0.96875     0.96794872  0.96551724  0.94701987  0.91729323  0.97692308]
Precision
 [ 0.64285714  0.79166667  0.82142857  0.74193548  0.73170732  0.91891892]
Recall
 [ 0.6         1.          0.76666667  0.95833333  0.71428571  0.75555556]
Áccuracy
[ 0.93714286  0.97142857  0.93142857  0.94857143  0.86857143  0.92      ]
FScore
[ 0.62068966  0.88372093  0.79310345  0.83636364  0.72289157  0.82926829]
```

**Figure 49:** Output of Bayesian classification showing a confusion matrix and computation of precison, recall, Specificity, Accuracy and Fscore.

## 16.7 VII (K Means Clustering)

```
from PIL import Image, ImageStat
import numpy
from scipy.spatial import distance

def get_Minimum(pixel, centroids):
        minDist = 9999
        minIndex = 0

        for i in range(0, len(centroids)):
                d = numpy.sqrt(int((centroids[i][0] - pixel[0]))**2 + int((centroids[
                    i][1] - pixel[1]))**2 + int((centroids[i][2] - pixel[2]))**2)
                if d < minDist:
                        minDist = d
                        minIndex = i

        return minIndex


def converged(centroids, old_centroids):
        if len(old_centroids) == 0:
                return False


        if len(centroids) <= 5:
                a = 1
        elif len(centroids) <= 10:
                a = 2
        else:
                a = 4

        for i in range(0, len(centroids)):
                cent = centroids[i]
                old_cent = old_centroids[i]

                if ((int(old_cent[0]) - a) <= cent[0] <= (int(old_cent[0]) + a)) and
                    ((int(old_cent[1]) - a) <= cent[1] <= (int(old_cent[1]) + a)) and
                    ((int(old_cent[2]) - a) <= cent[2] <= (int(old_cent[2]) + a)):
                        continue
                else:
                        return False
```

```
        return True

def Pixel_ass(centroids):
        clusters = {}

        for x in range(0, img_width):
                for y in range(0, img_height):
                        p = px[x, y]
                        minIndex = get_Minimum(px[x, y], centroids)

                        try:
                                clusters[minIndex].append(p)
                        except KeyError:
                                clusters[minIndex] = [p]
        #print(clusters)
        return clusters




def ad_Cent(centroids, clusters):
        new_centroids = []
        keys = sorted(clusters.keys())
        #print(keys)

        for k in keys:
                n = numpy.mean(clusters[k], axis=0) #axis=0 indicates means to be
                    computed across each column
                new = (int(n[0]), int(n[1]), int(n[2]))
                print(str(k) + ": " + str(new))
                new_centroids.append(new)

        return new_centroids

def startKmeans(someK):
        centroids = []
        old_centroids = []
        rgb_range = ImageStat.Stat(im).extrema
        #print(rgb_range)
        i = 1
```

```
        #Initializes someK number of centroids for the clustering
        for k in range(0, someK):

                cent = px[numpy.random.randint(0, img_width), numpy.random.randint(0,
                    img_height)]
                centroids.append(cent)
                #print(centroids)
        print("Centroids Initialized.")
        while not converged(centroids, old_centroids) and i <= 20:
                print("Iteration #" + str(i))
                i += 1

                old_centroids = centroids
                                                                                #
                   Make the current centroids into the old centroids
                clusters = Pixel_ass(centroids)
                                                        #Assign each pixel in
                    the image to their respective centroids
                centroids = ad_Cent(old_centroids, clusters)    #Adjust the centroids
                    to the center of their assigned pixels

        print("Converged")
        print(centroids)
        #print("{} : {}".format(1,clusters[0]))
        return centroids, clusters

def drawWindow(result):
        img = Image.new('RGB', (img_width, img_height), "white")
        p = img.load()

        for x in range(img.size[0]):
                for y in range(img.size[1]):
                        RGB_value = result[get_Minimum(px[x, y], result)]
                        p[x, y] = RGB_value

        img.show()

def compute_beta_index(img_width, img_height, px, centroids, clusters, nc):
        # print x
        beta_R = 0.0
        val = []
        numerator=0
```

```
        denom=0
        sum1=0
        sq_sum_num=0
        for k in clusters.keys():
                sq_sum=0
                n = numpy.mean(clusters[k], axis=0)
                new = (int(n[0]), int(n[1]), int(n[2]))
                #print(str(k) + ": " + str(new))
                for l in range(0, len(clusters[k])):
                        val=clusters[k][l]
                        diff=(val[0]-n[0])**2
                        sq_sum=sq_sum+diff
                denom=denom+sq_sum

        #print(denom)
        size=img_width*img_height
        for x in range(img_width):
                for y in range(img_height):
                        sum1=sum1+px[x,y][0]
        num_mean=sum1/size
        #print(num_mean)
        for x in range(img_width):
                for y in range(img_height):
                        diff=(px[x,y][0]-num_mean)**2
                        sq_sum_num=sq_sum_num+diff
        print(sq_sum_num)


        beta_index = sq_sum_num/denom
        return beta_index


k_input = int(input("Enter␣K␣value:␣"))

img = "flower.jpg"
im = Image.open(img)
img_width, img_height = im.size
px = im.load()
clusters = {}
centroids, clusters = startKmeans(k_input)
drawWindow(centroids)
#print("The cluster{}".format(clusters[0]))
```

```
index_beta_val = compute_beta_index(img_width, img_height, px, centroids, clusters,
    k_input)
print("The value of Davies Bouldin index for a K-Means cluster of size {} {}".format
    (str(k_input),str(index_beta_val)))
```

```
Enter K value: 9 Centroids Initialized.
Iteration #1
0: (119, 142, 31)
1: (108, 131, 47)
2: (94, 83, 54)
3: (235, 193, 56)
4: (77, 114, 19)
5: (45, 86, 3)
6: (90, 112, 38)
7: (121, 135, 69)
8: (52, 80, 10)
Iteration #2
0: (120, 142, 31)
1: (106, 129, 50)
2: (94, 86, 53)
3: (235, 193, 56)
4: (75, 111, 18)
5: (43, 79, 3)
6: (90, 114, 38)
7: (124, 137, 71)
8: (52, 78, 11)
Iteration #3
0: (119, 141, 31)
1: (107, 129, 52)
2: (94, 88, 53)
3: (235, 193, 56)
4: (73, 108, 17)
5: (37, 68, 3)
6: (90, 117, 35)
7: (125, 138, 72)
8: (56, 83, 12)
Converged
[(117, 141, 34), (109, 128, 55), (94, 92, 51), (235, 193, 56), (71, 106, 17), (36, 60, 5), (89, 121, 31), (127,
140, 73), (55, 86, 10)]
180827851.15958127
The value of Davies Bouldin index for a K-Means cluster of size 9 32.0069079716
```

**Figure 50:** Output of K means clustering with k=9.

## 16.8 VIII (Fuzzy MLP with initial encoding)

This is an implementation of 3 layered MLP using rough-set-theoretic concepts for initial weight encoding of neurons of input, hidden and output layer. The code runs on Vowel Dataset. The features are fuzzy-coded using PI membership function, and n dimensional pattern is represented in 3n dimensional vector. The fuzzy MLP uses rough set concepts and a methodology using rough sets is formulated for encoding initial knowledge of the information system. The initial connection weights of fuzzy MLP are computed using the concepts of Dependency factor and rule generation. D reducts are computed using method described in rough sets, discernibility matrix is derived, discernibility function is obtained and corresponding rules are generated. These rules help in deciding the connection of neurons in the hidden layer and dependency factor calculates the initial weights of the connection and also computes the number of neurons in the hidden layer.

```
from random import seed
from random import randrange
import random
from csv import reader
from math import exp
from sklearn.metrics import confusion_matrix
import numpy as np
import math
import itertools
from collections import Counter

def load_csv(filename):
        dataset = list()
        with open(filename, 'r') as file:
                csv_reader = reader(file)
                for row in csv_reader:
                        if not row:
                                continue
                        dataset.append(row)
        return dataset

# Convert string column to float
def str_column_to_float(dataset, column):
        for row in dataset:
                try:
                        row[column] = float(row[column].strip())
                except ValueError:
```

```
                        print("Error with row",column,":",row[column])
                        pass

# Convert string column to integer
def str_column_to_int(dataset, column):
        for row in dataset:
                row[column] = int(row[column])

def splitData(dataset, sRatio):
        trainSize = int(len(dataset) * sRatio)
        trainSet = []
        copy = list(dataset)
        seed(8)
        while len(trainSet) < trainSize:
                index = random.randrange(len(copy))

                trainSet.append(copy.pop(index))
        return [trainSet, copy]

def generate_new_dataset(new_sam,row):
        #print(row)
        X = np.empty((len(new_sam), 0))
        #print(X)
        for i in range(len(row)):
                col=row[i]
                x=[row_new[col-1] for row_new in new_sam]

                x=np.array([x])
                #print(np.transpose(x))
                #print(x)
                x=x.T
                X=np.append(X, x, axis=1)
        x=[row[-1] for row in new_sam]
        x=np.array([x])
        x=x.T
        X=np.append(X, x, axis=1)
        X=np.array(X).tolist()
        #print("X {}".format(X))
        return X

def fuzzify(trainingSet, testSet):
        center=list()
```

```
rad=list()
for i in range(3):
        feature1=list()
        feature1=[row[i] for row in trainingSet]
        #print("feature {}".format(feature1))
        maxi=max(feature1)
        mini=min(feature1)
        radius_medium=0.5*(maxi-mini)
        #print("rad {}".format(radius_medium))
        center_medium= mini + radius_medium
        radius_low= 2*(center_medium-mini)
        center_low=center_medium-(0.5*radius_low)
        radius_high = 2*(maxi - center_medium)
        center_high = center_medium + (0.5 * radius_high)
        center.append([center_low,center_medium,center_high])        rad.
            append([radius_low,radius_medium,radius_high])

fuzzy=list()
for i in range(len(trainingSet)):
    data=trainingSet[i]
    #print(data)
    l=0
    value=list()
    for j in range(3):
        d=data[j]
        for k in range(3):
            r=rad[j][k]/2
            eucleanD=math.pow((d-center[j][k]),2)
            eDist=math.sqrt(eucleanD)
            if eDist <= r :
                val = 1- 2*math.pow(eDist/(r*2),2)
                value.insert(l,val)
            else:
                y=eDist/rad[j][k]
                x=(1-(eDist/rad[j][k]))
                val = 2*math.pow(x,2)
                value.insert(l,val)
            l=l+1
    value.insert(l,data[-1])
    fuzzy.insert(i,value)
#print(fuzzy)
fuzzy_test=list()
```

```
        for i in range(len(testSet)):
            data=testSet[i]
            #print(data)
            l=0
            value=list()
            for j in range(3):
                d=data[j]
                for k in range(3):
                    r=rad[j][k]/2
                    eucleanD=math.pow((d-center[j][k]),2)
                    eDist=math.sqrt(eucleanD)
                    if eDist <= r :
                        val = 1- 2*math.pow(eDist/(r*2),2)
                        value.insert(l,val)
                    else:
                        y=eDist/rad[j][k]
                        x=(1-(eDist/rad[j][k]))
                        val = 2*math.pow(x,2)
                        value.insert(l,val)
                    l=l+1
            value.insert(l,data[-1])
            fuzzy_test.insert(i,value)

        fuzzy_threshold=list()
        for i in range(len(trainingSet)):
                row=list()
                temp=0
                data= fuzzy[i]
                for j in range(len(data)-1):
                        d=data[j]
                        if d < 0.8:
                                temp=0
                        else:
                                temp=1
                        row.append(temp)
                row.append(data[-1])
                fuzzy_threshold.append(row)
        #print(fuzzy_threshold, len(fuzzy_threshold))
        return [fuzzy_threshold, fuzzy, fuzzy_test]

def process(trainingSet,fuzzy_threshold):
        folds=list()
```

```
        for j in range(6):
                fold=list()
                for i in range(len(trainingSet)):
                        data=fuzzy_threshold[i]
                        if data[-1]==j+1:
                                fold.append(fuzzy_threshold[i])
                folds.append(fold)
        #print(folds)
        final_feature=list()
        for fold in folds:
                fold.sort()
                #print("    {} \n\n".format(fold))
                output=set(tuple(i) for i in fold)
                tup=(list(o) for o in output)
                largest_count=0
                for ele in tup:
                        #print(ele)
                        count=0
                        for i in range(len(fold)):
                                ele_fold=fold[i]
                                compare = lambda a,b: len(a)==len(b) and len(a)==sum
                                    ([1 for i,j in zip(a,b) if i==j])
                                if compare(ele, ele_fold):
                                        count=count+1
                        if count > largest_count:
                                largest_count=count
                                rep_feature=ele
                final_feature.append(rep_feature)
        #for i in range(6):
                #print(final_feature[i])
        return final_feature

def dependency(new_data,num):
        count=len(new_data)
        dependency=0
        for row in new_data:
                del row[-1]

        no_dupes = [x for n, x in enumerate(new_data) if x in new_data[:n]]
        #print(no_dupes)

        for i in range(len(no_dupes)):
```

```
                while no_dupes[i] in new_data:
                        new_data.remove(no_dupes[i])

        #print(new_data)
        dependency=len(new_data)/count
        return dependency


def compute_minterm(sample_set):
        total_minterm=list()
        for i in range(6):
                j=i+1
                temp1 = sample_set[i]
                #print(len(temp1))
                while (j<6):
                        temp2 = sample_set[j]
                        #print("1 2 {} {}".format(temp1, temp2))
                        minterm=list()
                        for k in range(len(temp1)-1):
                                if temp1[k] != temp2[k]:
                                        minterm.insert(k,k+1)
                        #print(" minterm{}".format(minterm))
                        total_minterm.insert(i,minterm)
                        j=j+1

        #print(total_minterm)
        s=list()
        #this part of the code removes duplicate entries
        for i in total_minterm:
                if i not in s:
                        s.append(i)

        ##this part of the code removes null set
        s1 = [x for x in s if x]

        #this part of the code captures all minterms that should be removed because
            of the absorption law
        #The minterms are collected in 'rem'
        s1.sort(key=len)
        #print("\n\n {}".format(s1))
        rem=list()
        for i in range(len(s1)):
```

```
                one=s1[i]
                #print("one{}".format(one))
                j=i+1
                while j<len(s1):
                        two=s1[j]
                        #print("j={} two{}".format(j,two))
                        if all(x in two for x in one) and two not in rem:
                                rem.append(two)
                                #print(rem)
                        j=j+1

        #The minterms are now removed
        for item in rem:
                s1.remove(item)
        return s1

def minterm_for_rules(sample_set,x):
        mint=list()
        inp_neuron_count=0
        dict_weight={}
        #hidden=[]
        dict_wei_output={}

        #output_layer=[]
        count_n=0
        for i in range(6):
                total_minterm=list()
                j=0
                temp1 = sample_set[i]
                print(len(temp1))
                while (j<6):
                        temp2 = sample_set[j]
                        #print(" {} {}".format(temp1, temp2))
                        minterm=list()
                        for k in range(len(temp1)-1):
                                if temp1[k] != temp2[k]:
                                        minterm.insert(k,x[k])
                        #print(" minterm{}".format(minterm))
                        total_minterm.insert(i,minterm)
                        j=j+1
                s=list()
                #this part of the code removes duplicate entries
```

```
for p in total_minterm:
        if p not in s:
                s.append(p)
s1 = [x for x in s if x]
#this part of the code captures all minterms that should be removed
    because of the absorption law
#The minterms are collected in 'rem'
s1.sort(key=len)
#print("\n\n {}".format(s1))
rem=list()
for w in range(len(s1)):
        one=s1[w]
        #print("one{}".format(one))
        j=w+1
        while j<len(s1):
                two=s1[j]
                #print("j={} two{}".format(j,two))
                if all(x in two for x in one) and two not in rem:
                        rem.append(two)
                        #print(rem)
                j=j+1
#The minterms are now removed
for item in rem:
        s1.remove(item)
#print(s1)

inp_neuron_count=inp_neuron_count+len(s1)
if i==0:
        dep_one=0
        class_one.append(s1)
        for j in itertools.product(*s1):
                #print(i)
                new_s=generate_new_dataset(final_sample,j)
                dep=dependency(new_s,len(j))
                dep_one=dep_one+dep
        #print("dep_one {}".format(dep_one))
        wei=dep_one/len(s1)
        print(final_sample[i])
        wt_output=[0 for i in range(12)]
        for p in range(len(s1)):
                wt=[0 for i in range(9)]
```

```
                for j in range(len(s1[p])):
                        pos=s1[p][j]
                        if final_sample[i][pos-1] ==1:
                                wt[pos-1]=wei/len(s1[p])
                        else:
                                wt[pos-1]=-wei/len(s1[p])
                        dict_weight['weights']=wt

                wt_output[count_n]=wei
                count_n=count_n+1
                hidden_layer.append(dict_weight)
                dict_weight={}
                #dict_wei_output={}
            dict_wei_output['weights']=wt_output
            output_layer.append(dict_wei_output)
            dict_wei_output={}
            #print(hidden_layer)
            #print("OUTPUT {}".format(output_layer))

    elif i==1:
            dep_two=0
            class_two.append(s1)
            for j in itertools.product(*s1):
                    new_s=generate_new_dataset(final_sample,j)
                    dep=dependency(new_s,len(j))
                    dep_two=dep_two+dep
            #print("dep_two {}".format(dep_two))
            wei=dep_two/len(s1)
            wt_output=[0 for i in range(12)]
            for p in range(len(s1)):
                    wt=[0 for i in range(9)]

                    for j in range(len(s1[p])):
                            pos=s1[p][j]
                            if final_sample[i][pos-1] ==1:
                                    wt[pos-1]=wei/len(s1[p])
                            else:
                                    wt[pos-1]=-wei/len(s1[p])
                            dict_weight['weights']=wt

                    wt_output[count_n]=wei
                    count_n=count_n+1
```

```
                hidden_layer.append(dict_weight)
                dict_weight={}

        dict_wei_output['weights']=wt_output
        output_layer.append(dict_wei_output)
        dict_wei_output={}
        #print(hidden_layer)
        #print("OUTPUT {}".format(output_layer))

elif i==2:
        dep_three=0
        class_three.append(s1)
        for j in itertools.product(*s1):
                new_s=generate_new_dataset(final_sample,j)
                dep=dependency(new_s,len(j))
                dep_three=dep_three+dep
        #print("dep_three {}".format(dep_three))
        wei=dep_three/len(s1)
        wt_output=[0 for i in range(12)]
        for p in range(len(s1)):
                wt=[0 for i in range(9)]
                wt_output=[0 for i in range(12)]
                for j in range(len(s1[p])):
                        pos=s1[p][j]
                        if final_sample[i][pos-1] ==1:
                                wt[pos-1]=wei/len(s1[p])
                        else:
                                wt[pos-1]=-wei/len(s1[p])
                        dict_weight['weights']=wt

                wt_output[count_n]=wei
                count_n=count_n+1
                hidden_layer.append(dict_weight)
                dict_weight={}

        dict_wei_output['weights']=wt_output
        output_layer.append(dict_wei_output)
        dict_wei_output={}
        #print(hidden_layer)
        #print("OUTPUT {}".format(output_layer))

elif i==3:
```

```
            dep_four=0
            class_four.append(s1)
            for j in itertools.product(*s1):
                    new_s=generate_new_dataset(final_sample,j)
                    dep=dependency(new_s,len(j))
                    dep_four=dep_four+dep
            #print("dep_four {}".format(dep_four))
            wei=dep_four/len(s1)
            wt_output=[0 for i in range(12)]
            for p in range(len(s1)):
                    wt=[0 for i in range(9)]

                    for j in range(len(s1[p])):
                            pos=s1[p][j]
                            if final_sample[i][pos-1] ==1:
                                    wt[pos-1]=wei/len(s1[p])
                            else:
                                    wt[pos-1]=-wei/len(s1[p])
                            dict_weight['weights']=wt

                    wt_output[count_n]=wei
                    count_n=count_n+1
                    hidden_layer.append(dict_weight)
                    dict_weight={}

            dict_wei_output['weights']=wt_output
            output_layer.append(dict_wei_output)
            dict_wei_output={}
            #print(hidden_layer)
            #print("OUTPUT {}".format(output_layer))

    elif i==4:
            dep_five=0
            class_five.append(s1)
            for j in itertools.product(*s1):
                    new_s=generate_new_dataset(final_sample,j)
                    dep=dependency(new_s,len(j))
                    dep_five=dep_five+dep
            #print("dep_five {}".format(dep_five))
            wei=dep_five/len(s1)
            wt_output=[0 for i in range(12)]
            for p in range(len(s1)):
```

```
                    wt=[0 for i in range(9)]

                    for j in range(len(s1[p])):
                            pos=s1[p][j]
                            if final_sample[i][pos-1] ==1:
                                    wt[pos-1]=wei/len(s1[p])
                            else:
                                    wt[pos-1]=-wei/len(s1[p])
                            dict_weight['weights']=wt
                    wt_output[count_n]=wei
                    count_n=count_n+1
                    hidden_layer.append(dict_weight)
                    dict_weight={}
            dict_wei_output['weights']=wt_output
            output_layer.append(dict_wei_output)
            dict_wei_output={}
            #print(hidden_layer)
            #print("OUTPUT {}".format(output_layer))

    elif i==5:
            dep_six=0
            class_six.append(s1)
            for j in itertools.product(*s1):
                    new_s=generate_new_dataset(final_sample,j)
                    dep=dependency(new_s,len(j))
                    dep_six=dep_six+dep
            #print("dep_six {}".format(dep_six))
            wei=dep_six/len(s1)
            wt_output=[0 for i in range(12)]
            for p in range(len(s1)):
                    wt=[0 for i in range(9)]

                    for j in range(len(s1[p])):
                            pos=s1[p][j]
                            if final_sample[i][pos-1] ==1:
                                    wt[pos-1]=wei/len(s1[p])
                            else:
                                    wt[pos-1]=-wei/len(s1[p])
                            dict_weight['weights']=wt

                    wt_output[count_n]=wei
                    count_n=count_n+1
```

```
                                hidden_layer.append(dict_weight)
                                dict_weight={}
                        dict_wei_output['weights']=wt_output
                        output_layer.append(dict_wei_output)
                        dict_wei_output={}
                        #print(hidden_layer)
                        #print("OUTPUT {}".format(output_layer))

        #print(class_one)
        #print(class_two)
        #print(class_three)
        #print(class_four)
        #print(class_five)
        #print(class_six)
        #print(inp_neuron_count)
        return inp_neuron_count



def accuracy_met(actual, predicted):
        correct = 0
        for i in range(len(actual)):
                if actual[i] == predicted[i]:
                        correct += 1
        return correct / float(len(actual)) * 100.0

# Evaluate an algorithm using a cross validation split
def run_algorithm(fuzzy, testSet, algorithm, *args):
        #print(dataset)
        predicted = algorithm(fuzzy, testSet, *args)
        actual = [row[-1] for row in testSet]
        #print(predicted)
        #print(actual)
        accuracy = accuracy_met(actual, predicted)
        cm = confusion_matrix(actual, predicted)
        print('\n'.join(['  '.join(['{:4}'.format(item) for item in row]) for row in cm
            ]))
        #confusionmatrix = np.matrix(cm)
        FP = cm.sum(axis=0) - np.diag(cm)
        FN = cm.sum(axis=1) - np.diag(cm)
        TP = np.diag(cm)
        TN = cm.sum() - (FP + FN + TP)
```

```
        print('False Positives\n {}'.format(FP))
        print('False Negetives\n {}'.format(FN))
        print('True Positives\n {}'.format(TP))
        print('True Negetives\n {}'.format(TN))
        TPR = TP/(TP+FN)
        print('Sensitivity \n {}'.format(TPR))
        TNR = TN/(TN+FP)
        print('Specificity \n {}'.format(TNR))
        Precision = TP/(TP+FP)
        print('Precision \n {}'.format(Precision))
        Recall = TP/(TP+FN)
        print('Recall \n {}'.format(Recall))
        Acc = (TP+TN)/(TP+TN+FP+FN)
        print('ÃĄccuracy \n{}'.format(Acc))
        Fscore = 2*(Precision*Recall)/(Precision+Recall)
        print('FScore \n{}'.format(Fscore))
```

```
# Calculate neuron activation for an input
def activate(weights, inputs):
        #print("weight neuorn {} {}".format(weights,inputs))
        activation = weights[-1]
        for i in range(len(weights)-1):
                activation += weights[i] * inputs[i]
        return activation

# Transfer neuron activation
def function(activation):
        return 1.0 / (1.0 + exp(-activation))

# Forward propagate input to a network output
def forward_propagate(network, row):
        inputs = row
        #print("input row{}\n".format(inputs))
        for layer in network:
                new_inputs = []
                for neuron in layer:
                        activation = activate(neuron['weights'], inputs)
                        neuron['output'] = function(activation)
```

```
                new_inputs.append(neuron['output'])
            inputs = new_inputs
        #print("output row{}\n".format(inputs))
        return inputs

# Calculate the derivative of an neuron output
def function_derivative(output):
        return output * (1.0 - output)

# Backpropagate error and store in neurons
def backprop_error(network, expected):
        for i in reversed(range(len(network))):
            layer = network[i]
            errors = list()
            if i != len(network)-1:
                for j in range(len(layer)):
                    error = 0.0
                    for neuron in network[i + 1]:
                        error += (neuron['weights'][j] * neuron['
                            delta'])
                    errors.append(error)
            else:
                for j in range(len(layer)):
                    neuron = layer[j]
                    #print("neuron {} {}".format(j,neuron))
                    errors.append(expected[j] - neuron['output'])
            for j in range(len(layer)):
                neuron = layer[j]
                neuron['delta'] = errors[j] * function_derivative(neuron['
                    output'])

# Update network weights with error
def change_weights(network, row, l_rate):
        for i in range(len(network)):
            inputs = row[:-1]
            if i != 0:
                inputs = [neuron['output'] for neuron in network[i - 1]]
            for neuron in network[i]:
                for j in range(len(inputs)):
                    neuron['weights'][j] += l_rate * neuron['delta'] *
                        inputs[j]
                neuron['weights'][-1] += l_rate * neuron['delta']
```

```
#To fuzzify the output layer
def fuzzyout(row,n_outputs):
        z=list()
        mu=list()
        muINT=list()
        rowclass=row[-1]-1
        #print(rowclass)
        for k in range(n_outputs):
                sumz=0
                for j in range(9):
                    interm=pow((row[j]-mean[k][j])/stdev[k][j],2)
                    sumz=sumz+interm
                    #print("row{}".format(row[j]))
                    #print("mean{}".format(mean[rowclass][j]))
                    #print("sum{}".format(sumz))
                weightedZ=math.sqrt(sumz)
                memMU=1/(1+(weightedZ/5))
                if 0 <= memMU <= 0.5:
                    memMUINT=2*pow(memMU,2)
                else:
                    temp=1-memMU
                    memMUINT=1-(2*pow(temp,2))
                mu.append(memMU)
                z.append(weightedZ)
                muINT.append(memMUINT)
        return muINT

# Train a network for a fixed number of epochs
def neural_network_train(network, train, l_rate, n_epoch, n_outputs):
        #print(dataset)
        for epoch in range(n_epoch):
                #print(train)
                for row in train:
                        #print("epochs {} {}".format(epoch,row))
                        outputs = forward_propagate(network, row)
                        #print(outputs)
                        expected = fuzzyout(row, n_outputs)
                        #print("input row{}\n".format(row))
                        #expected = [0 for i in range(n_outputs)]
                        #expected[row[-1]-1] = 1
                        #print("expected row{}\n".format(expected))
```

```
                        backprop_error(network, expected)
                        change_weights(network, row, l_rate)

# Initialize a network
def init_net(n_inputs, n_hidden, n_outputs):
        network = list()
        #hidden_layer = [{'weights':[random() for i in range(n_inputs + 1)]} for i in
             range(n_hidden)]
        print(hidden_layer)
        network.append(hidden_layer)
        #output_layer = [{'weights':[random() for i in range(n_hidden + 1)]} for i in
             range(n_outputs)]
        network.append(output_layer)
        return network

# Make a prediction with a network
def predict(network, row):
        outputs = forward_propagate(network, row)
        #print(outputs)
        indexOut=outputs.index(max(outputs))+1
        #print(indexOut)
        return indexOut

# Backpropagation Algorithm With Stochastic Gradient Descent
def back_propagation(train, test, l_rate, n_epoch, n_hidden):
        n_inputs = len(train[0]) - 1

        n_outputs = len(set([row[-1] for row in train]))

        network = init_net(n_inputs, n_hidden, n_outputs)
        #print("initialize network {}\n".format(network))
        neural_network_train(network, train, l_rate, n_epoch, n_outputs)
        #print("network {}\n".format(network))
        predictions = list()
        for row in test:
                prediction = predict(network, row)
                predictions.append(prediction)
        return(predictions)


l_rate = 0.2
n_epoch = 100
```

```
filename = 'data.csv'
sRatio = 0.60
dataset = load_csv(filename)

for i in range(len(dataset[0])-1):
        str_column_to_float(dataset, i)
#convert class column to integers
str_column_to_int(dataset, len(dataset[0])-1)
trainingSet, testSet = splitData(dataset, sRatio)
fuzzy_threshold, fuzzy, fuzzy_test=fuzzify(trainingSet, testSet)
final_sample=process(trainingSet,fuzzy_threshold)
# for the computation of output class fuzzification

classes=[row[-1] for row in fuzzy]
Unique=np.unique(classes)
dataset_split=list()
fold_size=int(len(Unique))
for i in range(fold_size):
        fold=list()
        for row in fuzzy:
                if row[-1] == Unique[i]:
                        fold.append(row)
        dataset_split.append(fold)
i=0
mean=list()
stdev=list()
j=0
for fold in dataset_split:
        x=list()
        y=list()
        z=list()
        x1=list()
        y1=list()
        z1=list()
        x2=list()
        y2=list()
        z2=list()

        for row in fold:
                if row[-1] == Unique[j]:
                        x.append(row[0])
                        y.append(row[1])
```

```
                    z.append(row[2])
                    x1.append(row[3])
                    y1.append(row[4])
                    z1.append(row[5])
                    x2.append(row[6])
                    y2.append(row[7])
                    z2.append(row[8])
        m1=sum(x)/float(len(x))
        m2=sum(y)/float(len(y))
        m3=sum(z)/float(len(z))
        m4=sum(x1)/float(len(x1))
        m5=sum(y1)/float(len(y1))
        m6=sum(z1)/float(len(z1))
        m7=sum(x2)/float(len(x2))
        m8=sum(y2)/float(len(y2))
        m9=sum(z2)/float(len(z2))
        mean.append([m1,m2,m3,m4,m5,m6,m7,m8,m9])
        st1=sum([pow(val-m1,2) for val in x])/float(len(x)-1)
        st2=sum([pow(val-m2,2) for val in y])/float(len(y)-1)
        st3=sum([pow(val-m3,2) for val in z])/float(len(z)-1)
        st4=sum([pow(val-m4,2) for val in x1])/float(len(x1)-1)
        st5=sum([pow(val-m5,2) for val in y1])/float(len(y1)-1)
        st6=sum([pow(val-m6,2) for val in z1])/float(len(z1)-1)
        st7=sum([pow(val-m7,2) for val in x2])/float(len(x2)-1)
        st8=sum([pow(val-m8,2) for val in y2])/float(len(y2)-1)
        st9=sum([pow(val-m9,2) for val in z2])/float(len(z2)-1)
        std1=math.sqrt(st1)
        std2=math.sqrt(st2)
        std3=math.sqrt(st3)
        std4=math.sqrt(st4)
        std5=math.sqrt(st5)
        std6=math.sqrt(st6)
        std7=math.sqrt(st7)
        std8=math.sqrt(st8)
        std9=math.sqrt(st9)
        stdev.append([std1,std2,std3,std4,std5,std6,std7,std8,std9])
        j=j+1



#print(trainingSet,len(trainingSet))
#this is fuzzify input based on class belongin granulation
```

```
s1=compute_minterm(final_sample)
#print(s1)

#The cartesian product of the minterms(POS) is computed here, the product gathers all
    the
#terms in SOP form. The products thus obtained are reducts.
xyz=list()
for i in itertools.product(*s1):
        #print(i)
        xyz.append(set(i))
new_xyz=list()
for i in xyz:
        if i not in new_xyz:
                new_xyz.append(i)

x={1,2,4,7}
new_set=generate_new_dataset(final_sample,list(x))
#print("\n\n{}".format(new_set))
class_one=list()
class_two=list()
class_three=list()
class_four=list()
class_five=list()
class_six=list()
hidden_layer=[]
output_layer=[]
n_hidden=minterm_for_rules(new_set,list(x))

run_algorithm(fuzzy, fuzzy_test, back_propagation, l_rate, n_epoch, n_hidden)
```

## 16.9 Code IX (Saha Bora Activation Function)

```
from random import seed
from random import uniform
from random import randrange
from random import random
from csv import reader
from math import exp
from sklearn.metrics import confusion_matrix
from sklearn.metrics import cohen_kappa_score
import numpy as np
import csv
from decimal import Decimal


# Load a CSV file
def loadCsv(filename):
        trainSet = []

        lines = csv.reader(open(filename, 'r'))
        dataset = list(lines)
        for i in range(len(dataset)):
                for j in range(4):
                        #print("DATA {}".format(dataset[i]))
                        dataset[i][j] = float(dataset[i][j])
                trainSet.append(dataset[i])
        return trainSet

# Convert string column to float
def str_column_to_float(dataset, column):
        for row in dataset:
                try:
                        row[column] = float(row[column])
                except ValueError:
                        print("Error with row",column,":",row[column])
                        pass

# Convert string column to integer
def str_column_to_int(dataset, column):
        class_values = [row[column] for row in dataset]
        unique = set(class_values)
```

```
        lookup = dict()
        for i, value in enumerate(unique):
                lookup[value] = i+1
        for row in dataset:
                row[column] = lookup[row[column]]
        print(lookup)
        return lookup

# Find the min and max values for each column
def dataset_minmax(dataset):
        minmax = list()
        stats = [[min(column), max(column)] for column in zip(*dataset)]
        return stats

# Rescale dataset columns to the range 0-1
def normalize_dataset(dataset, minmax):
        for row in dataset:
                for i in range(len(row)-1):
                        row[i] = (row[i] - minmax[i][0]) / (minmax[i][1] - minmax[i
                            ][0])

def splitData(dataset, sRatio):
        trainSet = []
        copy = list(dataset)
        trainSize = int(len(dataset) * sRatio)
        seed(8)
        while len(trainSet) < trainSize:
                index = randrange(len(copy))
                trainSet.append(copy.pop(index))
        return [trainSet, copy]

# Calculate accuracy percentage
def accuracy_metric(actual, predicted):
        correct = 0
        for i in range(len(actual)):
                if actual[i] == predicted[i]:
                        correct += 1
        return correct / float(len(actual)) * 100.0

# Evaluate an algorithm using a cross validation split
def evaluate_algorithm(train_set, test_set, algorithm, *args):
```

```
                #print(test_set)
                predicted = algorithm(train_set, test_set, *args)
                print(predicted)
                actual = [row[-1] for row in test_set]
                print(actual)
                accuracy = accuracy_metric(actual, predicted)
                cm = confusion_matrix(actual, predicted)
                print('\n'.join(['' .join(['{:4}'.format(item) for item in row]) for
                    row in cm]))
                #confusionmatrix = np.matrix(cm)
                FP = cm.sum(axis=0) - np.diag(cm)
                FN = cm.sum(axis=1) - np.diag(cm)
                TP = np.diag(cm)
                TN = cm.sum() - (FP + FN + TP)
                print('False␣Positives\n␣{}'.format(FP))
                print('False␣Negetives\n␣{}'.format(FN))
                print('True␣Positives\n␣{}'.format(TP))
                print('True␣Negetives\n␣{}'.format(TN))
                TPR = TP/(TP+FN)
                print('Sensitivity␣\n␣{}'.format(TPR))
                TNR = TN/(TN+FP)
                print('Specificity␣\n␣{}'.format(TNR))
                Precision = TP/(TP+FP)
                print('Precision␣\n␣{}'.format(Precision))
                Recall = TP/(TP+FN)
                print('Recall␣\n␣{}'.format(Recall))
                Acc = (TP+TN)/(TP+TN+FP+FN)
                print('ÃĄccuracy␣\n{}'.format(Acc))
                Fscore = 2*(Precision*Recall)/(Precision+Recall)
                print('FScore␣\n{}'.format(Fscore))
                k=cohen_kappa_score(actual, predicted)
                print('ÃĞohen␣Kappa␣\n{}'.format(k))



# Calculate neuron activation for an input
def activate(weights, inputs):
        activation = weights[-1]
        for i in range(len(weights)-1):
                activation += weights[i] * inputs[i]
        return activation
```

```
# Transfer neuron activation
def transfer(act):
        val = 0.5
        y = 1.0 / (1.0 + (0.91 * (pow(act,val)*pow(1-act,1-val))))
        y = abs(y)
        return y

# Forward propagate input to a network output
def forward_propagate(network, row):
        inputs = row
        for layer in network:
                new_inputs = []
                for neuron in layer:
                        activation = activate(neuron['weights'], inputs)
                        #print("weight {}".format(neuron['weights']))
                        #print("x {}".format(activation))
                        neuron['input'] = activation
                        neuron['output'] = transfer(activation)
                        #print(" y {}".format(neuron['output']))
                        new_inputs.append(neuron['output'])
                        #new_inputs.append(neuron['input'])
                inputs = new_inputs
        return inputs

# Calculate the derivative of an neuron output
def transfer_derivative(output, inp):
        derv = output * (1.0 - output)/ (inp * ( 1.0 - inp ))
        derv = derv * (inp - 0.5)
        return derv

# Backpropagate error and store in neurons
def backward_propagate_error(network, expected):
        for i in reversed(range(len(network))):
                layer = network[i]
                errors = list()
                if i != len(network)-1:
                        for j in range(len(layer)):
                                error = 0.0
                                for neuron in network[i + 1]:
                                        error += (neuron['weights'][j] * neuron['
                                            delta'])
                                errors.append(error)
```

```
                else:
                        for j in range(len(layer)):
                                neuron = layer[j]
                                errors.append(expected[j] - neuron['output'])
                for j in range(len(layer)):
                        neuron = layer[j]
                        neuron['delta'] = errors[j] * transfer_derivative(neuron['
                            output'], neuron['input'])

# Update network weights with error
def update_weights(network, row, l_rate):
        for i in range(len(network)):
                inputs = row[:-1]

                if i != 0:
                        inputs = [neuron['output'] for neuron in network[i - 1]]
                for neuron in network[i]:
                        for j in range(len(inputs)):
                                temp = l_rate * neuron['delta'] * inputs[j] + mu *
                                    neuron['prev'][j]
                                neuron['weights'][j] += temp
                                #print("neuron weight{} \n".format(neuron['weights'][
                                    j]))
                                neuron['prev'][j] = temp
                        temp = l_rate * neuron['delta'] + mu * neuron['prev'][-1]
                        neuron['weights'][-1] += temp
                        neuron['prev'][-1] = temp
                #print("neuron {}".format(neuron['weights']))


# Train a network for a fixed number of epochs
def train_network(network, train, l_rate, n_epoch, n_outputs):
        for epoch in range(n_epoch):
                for row in train:
                        outputs = forward_propagate(network, row)
                        #print(network)
                        expected = [0 for i in range(n_outputs)]
                        expected[row[-1]-1] = 1
                        #print("expected row{}\n".format(expected))
                        backward_propagate_error(network, expected)
                        update_weights(network, row, l_rate)
```

```
# Initialize a network
def initialize_network(n_inputs, n_hidden, n_outputs):
        network = list()
        hidden_layer = [{'weights':[round(uniform(0.00,0.1),4) for i in range(
            n_inputs + 1)], 'prev':[0 for i in range(n_inputs+1)]} for i in range(
            n_hidden)]
        network.append(hidden_layer)
        #hidden_layer = [{'weights':[random() for i in range(n_hidden + 1)], 'prev
            ':[0 for i in range(n_hidden+1)]} for i in range(n_hidden)]
        #network.append(hidden_layer)
        output_layer = [{'weights':[round(uniform(0.0,0.1),4) for i in range(n_hidden
             + 1)],'prev':[0 for i in range(n_hidden+1)]} for i in range(n_outputs)]
        network.append(output_layer)
        print(network)
        return network

# Make a prediction with a network
def predict(network, row):
        outputs = forward_propagate(network, row)
        #print(outputs)
        value = outputs.index(max(outputs)) + 1
        return value

# Backpropagation Algorithm With Stochastic Gradient Descent
def back_propagation(train, test, l_rate, n_epoch, n_hidden):
        n_inputs = len(train[0]) - 1
        n_outputs = len(set([row[-1] for row in train]))
        #print("output {}".format(n_outputs))
        network = initialize_network(n_inputs, n_hidden, n_outputs)
        train_network(network, train, l_rate, n_epoch, n_outputs)
        #print("network {}\n".format(network))
        predictions = list()
        for row in test:
                prediction = predict(network, row)
                predictions.append(prediction)
        return(predictions)

# Test Backprop on Seeds dataset
seed(8)
# load and prepare data
filename = 'data.csv'
dataset = loadCsv(filename)
```

```
for i in range(len(dataset[0])-1):
        str_column_to_float(dataset, i)
# convert class column to integers
str_column_to_int(dataset, len(dataset[0])-1)
# normalize input variables

minmax = dataset_minmax(dataset)
normalize_dataset(dataset, minmax)
# evaluate algorithm
sRatio = 0.80
trainingSet, testSet = splitData(dataset, sRatio)
l_rate = 0.1
mu=0.001
n_epoch = 1000
n_hidden = 4
evaluate_algorithm(trainingSet,testSet, back_propagation, l_rate, n_epoch, n_hidden)
```

## 16.10 Code X (Stacked Autoencoder)

```
from random import seed
from random import randrange
from random import random
from csv import reader
from math import exp
import math
from sklearn.metrics import confusion_matrix
from sklearn.metrics import cohen_kappa_score
import numpy as np
import csv
import copy

def loadNew(file):
        trainSet = []
        lines = csv.reader(open(file, 'r'))
        dataset = list(lines)
        #print("training set {}".format(dataset))
        for i in range(len(dataset[0])-1):
                for row in dataset:
                        try:
                                row[i] = float(row[i])
                        except ValueError:
                                print("Error with row",column,":",row[i])
                                pass
        trainSet = dataset
        return trainSet


# Load a CSV file
def loadCsv(filename, file1):
        trainSet = []
        testSet = []
        lines = csv.reader(open(filename, 'r'))
        dataset = list(lines)
        #print("training set {}".format(dataset))
        for i in range(len(dataset[0])-1):
                for row in dataset:
                        try:
                                row[i] = float(row[i])
```

```
                        except ValueError:
                                print("Error with row",column,":",row[i])
                                pass
        trainSet = dataset
        lines = csv.reader(open(file1, 'r'))
        dataset = list(lines)
        for i in range(len(dataset[0])-1):
                for row in dataset:
                        try:
                                row[i] = float(row[i])
                        except ValueError:
                                print("Error with row",column,":",row[i])
                                pass
        testSet = dataset
        #print("training set {}".format(trainSet))
        return trainSet, testSet

# Convert string column to float
def str_column_to_float(dataset, column):
        for row in dataset:
                try:
                        row[column] = float(row[column])
                except ValueError:
                        print("Error with row",column,":",row[column])
                        pass

# Convert string column to integer
def str_column_to_int(dataset, column):
        class_values = [row[column] for row in dataset]
        unique = set(class_values)
        lookup = dict()
        for i, value in enumerate(unique):
                lookup[value] = i
        for row in dataset:
                row[column] = lookup[row[column]]
        return lookup

# Find the min and max values for each column
def dataset_minmax(dataset):
        minmax = list()
        stats = [[min(column), max(column)] for column in zip(*dataset)]
        return stats
```

```
# Rescale dataset columns to the range 0-1
def normalize_dataset(dataset, minmax):
        for row in dataset:
                for i in range(len(row)-1):
                        row[i] = (row[i] - minmax[i][0]) / (minmax[i][1] - minmax[i
                            ][0])



# Calculate accuracy percentage
def accuracy_metric(actual, predicted):
        correct = 0
        for i in range(len(actual)):
                if actual[i] == predicted[i]:
                        correct += 1
        return correct / float(len(actual)) * 100.0

# Evaluate an algorithm using a cross validation split
def evaluate_algorithm(train_set, test_set, algorithm, *args):
                test_set_for_class = copy.deepcopy(test_set)
                test_set_again = copy.deepcopy(test_set)
                for row in test_set:
                        del row[-1]
                #print(test_set_for_class)
                network_one, new_train, new_test = algorithm(train_set, test_set,
                    test_set_for_class, *args)


                test_set_for_class = copy.deepcopy(new_test)
                for row in new_test:
                        del row[-1]

                network_two, second_train, second_test = backpropagation_two(
                    new_train,new_test, test_set_for_class, *args)



                network_three, predicted=back_prop_for_classification(second_train,
                    second_test, *args)
                new_network = list()
                for layer in network_one:
```

```
                new_network.append(layer)
        for layer in network_two:
                new_network.append(layer)
        for layer in network_three:
                new_network.append(layer)

##          for layer in new_network:
##                  print("\n\n{}\n\n".format(layer))
        n_inputs = len(test_set_again[0]) - 1
        n_outputs = len(set([row[-1] for row in test_set_again]))

        #print("For Classification {}".format(network))
        predictions = list()
        for row in test_set_again:
                prediction = predict(new_network, row)
                predictions.append(prediction)
        #predicted = back_prop_for_classification(train_set,
            test_set_for_class, network, *args)
        actual = [int(row[-1]) for row in test_set_again]
        #print(" {} \n\n {} \n\n".format(actual,predicted))
        accuracy = accuracy_metric(actual, predictions)
        cm = confusion_matrix(actual, predictions)
        print('\n'.join(['  '.join(['{:4}'.format(item) for item in row]) for
            row in cm]))
        #confusionmatrix = np.matrix(cm)
        FP = cm.sum(axis=0) - np.diag(cm)
        FN = cm.sum(axis=1) - np.diag(cm)
        TP = np.diag(cm)
        TN = cm.sum() - (FP + FN + TP)
        print('False Positives\n {}'.format(FP))
        print('False Negetives\n {}'.format(FN))
        print('True Positives\n {}'.format(TP))
        print('True Negetives\n {}'.format(TN))
        TPR = TP/(TP+FN)
        print('Sensitivity \n {}'.format(TPR))
        TNR = TN/(TN+FP)
        print('Specificity \n {}'.format(TNR))
        Precision = TP/(TP+FP)
        print('Precision \n {}'.format(Precision))
        Recall = TP/(TP+FN)
        print('Recall \n {}'.format(Recall))
        Acc = (TP+TN)/(TP+TN+FP+FN)
```

```
                print('ÃĄccuracy \n{}'.format(Acc))
                Fscore = 2*(Precision*Recall)/(Precision+Recall)
                print('FScore \n{}'.format(Fscore))

                re_train_network(new_network, train_set, l_rate, n_epoch, n_outputs)
                #print("For Classification {}".format(network))
                predictions = list()
                for row in test_set_again:
                        prediction = predict(new_network, row)
                        predictions.append(prediction)
                actual = [int(row[-1]) for row in test_set_again]
                #print(" {} \n\n {} \n\n".format(actual,predicted))
                accuracy = accuracy_metric(actual, predictions)
                cm = confusion_matrix(actual, predictions)
                print('\n'.join([''.join(['{:4}'.format(item) for item in row]) for
                    row in cm]))
                #confusionmatrix = np.matrix(cm)
                FP = cm.sum(axis=0) - np.diag(cm)
                FN = cm.sum(axis=1) - np.diag(cm)
                TP = np.diag(cm)
                TN = cm.sum() - (FP + FN + TP)
                print('False Positives\n {}'.format(FP))
                print('False Negetives\n {}'.format(FN))
                print('True Positives\n {}'.format(TP))
                print('True Negetives\n {}'.format(TN))
                TPR = TP/(TP+FN)
                print('Sensitivity \n {}'.format(TPR))
                TNR = TN/(TN+FP)
                print('Specificity \n {}'.format(TNR))
                Precision = TP/(TP+FP)
                print('Precision \n {}'.format(Precision))
                Recall = TP/(TP+FN)
                print('Recall \n {}'.format(Recall))
                Acc = (TP+TN)/(TP+TN+FP+FN)
                print('ÃĄccuracy \n{}'.format(Acc))
                Fscore = 2*(Precision*Recall)/(Precision+Recall)
                print('FScore \n{}'.format(Fscore))


        #return scores

# Calculate neuron activation for an input
```

```
def activate(weights, inputs):
        activation = weights[-1]
        for i in range(len(weights)-1):
                activation += weights[i] * inputs[i]
        return activation

# Transfer neuron activation
def transfer(activation):
        return 1.0 / (1.0 + exp(-activation))

# Forward propagate input to a network output
def forward_propagate(network, row):
        inputs = row
        for layer in network:
                new_inputs = []
                for neuron in layer:
                        activation = activate(neuron['weights'], inputs)
                        neuron['output'] = transfer(activation)
                        new_inputs.append(neuron['output'])
                inputs = new_inputs
        return inputs

# Calculate the derivative of an neuron output
def transfer_derivative(output):
        return output * (1.0 - output)

# Backpropagate error and store in neurons
def backward_propagate_error(network, expected):
        for i in reversed(range(len(network))):
                layer = network[i]
                errors = list()
                if i != len(network)-1:
                        for j in range(len(layer)):
                                error = 0.0
                                for neuron in network[i + 1]:
                                        error += (neuron['weights'][j] * neuron['
                                            delta'])
                                errors.append(error)
                else:
                        for j in range(len(layer)):
                                neuron = layer[j]
                                errors.append(expected[j] - neuron['output'])
```

```
                for j in range(len(layer)):
                        neuron = layer[j]
                        neuron['delta'] = errors[j] * transfer_derivative(neuron['
                            output'])

# Update network weights with error
def update_weights(network, row, l_rate):
        for i in range(len(network)):
                inputs = row[:-1]

                if i != 0:
                        inputs = [neuron['output'] for neuron in network[i - 1]]
                for neuron in network[i]:
                        for j in range(len(inputs)):
                                temp = l_rate * neuron['delta'] * inputs[j] + mu *
                                    neuron['prev'][j]
                                neuron['weights'][j] += temp
                                #print("neuron weight{} \n".format(neuron['weights'][
                                    j]))
                                neuron['prev'][j] = temp
                        temp = l_rate * neuron['delta'] + mu * neuron['prev'][-1]
                        neuron['weights'][-1] += temp
                        neuron['prev'][-1] = temp


# Train a network for a fixed number of epochs
def train_network(network, train, l_rate, n_epoch, n_outputs):
        for epoch in range(n_epoch):
                for row in train:
                        outputs = forward_propagate(network, row)
                        expected=row
                        backward_propagate_error(network, expected)
                        update_weights(network, row, l_rate)

def re_train_network(network_two, train_set, l_rate, n_epoch, n_outputs):
        for epoch in range(n_epoch):
                for row in train_set:
                        outputs = forward_propagate(network_two, row)
                        #print(outputs)
                        expected = [0 for i in range(n_outputs)]
                        #print(row)
                        expected[int(row[-1])-1] = 1
```

```
                        #print("expected row{}\n".format(expected))
                        backward_propagate_error(network_two, expected)
                        update_weights(network_two, row, l_rate)

def prepare_dataset(network, test_with_class):
        new_data_train = list()
        for row in trainingSet:
                outputs = forward_propagate(network, row)
                outputs.append(int(row[-1]))
                #print(" {} \n".format(row))
                new_data_train.append(outputs)


        new_data_test = list()
        for row in test_with_class:
                outputs = forward_propagate(network, row)
                outputs.append(int(row[-1]))
                #print(" {} \n".format(row))
                new_data_test.append(outputs)


        return new_data_train, new_data_test

def prepare_dataset_two(network, train, test, test_with_class):

        second_data_train = list()
        for row in train:
                outputs = forward_propagate(network, row)
                outputs.append(row[-1])
                #print(" {} \n".format(row))
                second_data_train.append(outputs)

##        csvfile = "newdata_train_two.csv"
##        with open(csvfile, "w") as output:
##                writer = csv.writer(output, lineterminator='\n')
##                writer.writerows(second_data_train)

        second_data_test = list()
        for row in test_with_class:
                outputs = forward_propagate(network, row)
                outputs.append(row[-1])
                #print(" {} \n".format(row))
```

```
                second_data_test.append(outputs)

##          csvfile = "newdata_test_two.csv"
##          with open(csvfile, "w") as output:
##                  writer = csv.writer(output, lineterminator='\n')
##                  writer.writerows(second_data_test)
##          return second_data_train, second_data_test


# Initialize a network
def initialize_network(n_inputs, n_hidden, n_outputs):
        network = list()
        hidden_layer = [{'weights':[np.random.uniform(0.0, 0.2) for i in range(
            n_inputs + 1)], 'prev':[0 for i in range(n_inputs+1)]} for i in range(
            n_hidden)]
        network.append(hidden_layer)
##          hidden_layer = [{'weights':[random() for i in range(n_hidden + 1)], 'prev
    ':[0 for i in range(n_hidden+1)]} for i in range(n_hidden)]
##          network.append(hidden_layer)
        output_layer = [{'weights':[np.random.uniform(0, 0.2) for i in range(n_hidden
            + 1)],'prev':[0 for i in range(n_hidden+1)]} for i in range(n_outputs)]
        network.append(output_layer)
        #print("FIRST {} \n\n".format(network))
        return network


def reinitialize_to_classify(n_inputs, n_outputs):
        network_two = list()
        output_layer = [{'weights':[np.random.uniform(0, 0.2) for i in range(n_inputs
            + 1)],'prev':[0 for i in range(n_inputs + 1)]} for i in range(n_outputs)]
        network_two.append(output_layer)
        #print(network)
        return network_two


# Make a prediction with a network
def predict(network, row):
        outputs = forward_propagate(network, row)
        return outputs.index(max(outputs)) + 1
```

```
# Backpropagation Algorithm With Stochastic Gradient Descent
def back_propagation(train, test, test_with_class, l_rate, n_epoch, n_hidden):
        n_inputs = len(train[0]) - 1
        n_outputs = n_inputs
        network = initialize_network(n_inputs, n_hidden, n_outputs)
        train_network(network, train, l_rate, n_epoch, n_outputs)
        #print("network {}\n".format(network))
        avg_msq=0

        for row in test:
                output = forward_propagate(network, row)
                #print(" row {} ---- output {}".format(row,output))
                r = np.array(row)
                o = np.array(output)
                err = r - o
                mssq = np.sum(err**2)/len(err)
                #print("MSE = {}".format(mssq))
                avg_msq=avg_msq+ mssq
        avg_msq=avg_msq/len(test)
        print("MSE First  {}    ".format(avg_msq))
        #print("FIRST {} \n\n".format(network))
        #network = transform_auto(network,train, test, l_rate, n_epoch, n_hidden,
            n_outputs)
        network = network[:-1]
        new_train, new_test = prepare_dataset(network,test_with_class)
        #print(network)
        return network, new_train, new_test

def backpropagation_two(train ,test, test_with_class, *args):

        n_inputs = len(train[0]) - 1
        n_outputs = n_inputs
        n_hidden_two = 9
        network_two = initialize_network(n_inputs, n_hidden_two, n_outputs)
        train_network(network_two, train, l_rate, n_epoch, n_outputs)
        #print("network {}\n".format(network))
        avg_msq=0


        for row in test:
                output = forward_propagate(network_two, row)
                #print(" row {} ---- output {}".format(row,output))
```

```
                r = np.array(row)
                o = np.array(output)
                err = r - o
                mssq = np.sum(err**2)/len(err)
                #print("MSE = {}".format(mssq))
                avg_msq=avg_msq+ mssq
        avg_msq=avg_msq/len(test)
        print("␣MSE␣Second␣{}␣".format(avg_msq))
        network_two = network_two[:-1]
        second_train, second_test = prepare_dataset_two(network_two,train, test,
            test_with_class)
        #print(network)
        return network_two, second_train, second_test



def back_prop_for_classification(train_set,test_set_for_class, *args):
        n_inputs = len(train_set[0]) - 1
        n_outputs = len(set([row[-1] for row in train_set]))
        #print(n_outputs)
        network_three = reinitialize_to_classify(n_inputs, n_outputs)
        re_train_network(network_three, train_set, l_rate, n_epoch, n_outputs)
        #print("For Classification {}".format(network))
        predictions = list()
        for row in test_set_for_class:
                prediction = predict(network_three, row)
                predictions.append(prediction)
        #print(predictions)
        return network_three, predictions

# Test Backprop on Seeds dataset
seed(1)
# load and prepare data
filename = 'sat_train_new.csv'
file1 = 'sat_test_new.csv'
        #sRatio = 0.80
trainingSet, testSet = loadCsv(filename, file1)


# normalize input variables
minmax = dataset_minmax(trainingSet)
normalize_dataset(trainingSet, minmax)
```

```
# normalize input variables
minmax = dataset_minmax(testSet)
normalize_dataset(testSet, minmax)

# evaluate algorithm

l_rate = 0.3
mu=0.1
n_epoch = 300
n_hidden = 19
scores = evaluate_algorithm(trainingSet,testSet, back_propagation, l_rate, n_epoch,
    n_hidden)
```

Initial Image

For K=9

For K=20

**Figure 51:** Sample outputs to illustrate the effect of running k means clustering with K=9 and k=20 on initial image

```
 18   7   0   0   9   2
  4  27   0   0   0   1
  0   0  69   0   9   0
  1   0   0  49   0   4
  5   0   8   0  61   2
  3   0   0   5   2  63
False Positives
 [13  7  8  5 20  9]
False Negetives
 [18  5  9  5 15 10]
True Positives
 [18 27 69 49 61 63]
True Negetives
 [300 310 263 290 253 267]
Sensitivity
 [ 0.5         0.84375     0.88461538  0.90740741  0.80263158  0.8630137 ]
Specificity
 [ 0.95846645  0.97791798  0.9704797   0.98305085  0.92673993  0.9673913 ]
Precision
 [ 0.58064516  0.79411765  0.8961039   0.90740741  0.75308642  0.875     ]
Recall
 [ 0.5         0.84375     0.88461538  0.90740741  0.80263158  0.8630137 ]
Áccuracy
[ 0.91117479  0.96561605  0.9512894   0.9713467   0.89971347  0.94555874]
FScore
[ 0.53731343  0.81818182  0.89032258  0.90740741  0.77707006  0.86896552]
```

**Figure 52:** Output showing confusion matrix and other parameters

```
Confusion Matrix
3  4  0  4  6  1
0  7  0  1  0  7
0  0  2  0  39  0
0  0  0  24  0  0
2  1  2  2  33  0
0  1  0  27  0  9

False Positives [ 2  6  2 34 45  8]

False Negetives [15  8 39  0  7 28]

True Positives [3  7  2 24 33  9]

True Negetives [155 154 132 117  90 130]

Sensitivity  [ 0.16666667  0.46666667  0.04878049  1.        0.825       0.24324324]

Specificity [ 0.98726115  0.9625      0.98507463  0.77483444  0.66666667  0.94202899]

Precision [ 0.6         0.53846154  0.5         0.4137931   0.42307692  0.52941176]

Recall [ 0.16666667  0.46666667  0.04878049  1.        0.825       0.24324324]

Áccuracy [ 0.90285714  0.92        0.76571429  0.80571429  0.70285714  0.79428571]

FScore [0.26086957  0.5         0.08888889  0.58536585  0.55932203  0.33333333]

Cohen Kappa 0.32194927102057114
```

**Figure 53:** Output showing confusion matrix and other parameters

## REFERENCES


[Asimov1989] Asimov I., The Relativity of Wrong, 1989, p121-198, ISBN-13:9781558171695

[Anglada-Escudé et al.2016] Anglada-Escudé G. et al., 2016, A terrestrial planet candidate in a temperate orbit around Proxima Centauri, Nature, Vol. 536, Number 7617, p.437-440, doi:10.1038/nature19106

[Ball & Brunner2010] Ball N. M., Brunner R. J., 2010, Data Mining and Machine Learning in Astronomy, International Journal of Modern Physics D, Vol. 19, Number 7, p. 1049-1106, doi:10.1142/s0218271810017160

[Batalha 2014] Batalha, N. M., 2014, Exploring exoplanet populations with NASA's Kepler Mission, Proceedings of the National Academy of Sciences, Vol. 111, Number 35, p. 12647-12654, doi:10.1073/pnas.1304196111

[Bora et al.2016] Bora K., Saha S., Agrawal S., Safonova M., Routh S., Narasimhamurthy A. M., CD-HPF: New Habitability Score Via Data Analytic Modeling, 2016, Journal of Astronomy and Computing, Vol. 17, p. 129-143, doi:10.1016/j.ascom.2016.08.001

[Abraham2014] Botros A., Artificial Intelligence on the Final Frontier: Using Machine Learning to Find New Earths, 2014, Planet Hunter: http://www.abrahambotros.com/src/docs/AbrahamBotros

[Breiman2001] Breiman L., 2001, Random Forests, Machine Learning, Vol. 45, Number 1, p. 5-32, doi:10.1023/a:1010933404324

[Bylander & Hanzlik1999] Bylander T., Hanzlik D., 1999, Estimating generalization error using out-of-bag estimates. In Proceedings of the National Conference on Artificial Intelligence. AAAI, pp. 321-327, Proceedings of the 1999 16th National Conference on Artificial Intelligence (AAAI-99), 11th Innovative Applications of Artificial Intelligence Conference (IAAI-99), Orlando, FL, USA, 18-22 July.

[Cai, Duo & Cai2010] Cai Y. L., Duo J., Cai D., 2010, A KNN Research Paper Classification Method Based on Shared Nearest Neighbor, Proceedings of the 8th NTCIR Workshop Meeting on Evaluation of Information Access Technologies: Information Retrieval, Question Answering and Cross-Lingual Information Access, NTCIR-8, National Center of Sciences, Tokyo, Japan, June 15-18, p. 2010, 336-340

[Chawla et al.2002] Chawla N. V., Bowyer K. W., Hall L. O., 2002, Kegelmeyer W. P., SMOTE: Synthetic Minority Over-sampling Technique, Journal of Artificial Intelligence Research, Vol. 16, p. 321-357, doi:10.1613/jair.953

[Chen & Guestrin2016] Chen T., Guestrin C., 2016, XGBoost: A Scalable Tree Boosting System, Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16, doi:10.1145/2939672.2939785

[Danielski2014] Danielski C., 2014, Optimal Extraction Of Planetary Signal Out Of Instrumental and Astronphysical Noise, PhD Thesis, University of London(UCL)

[Dayal et al.2015] Dayal P., Cockell C., Rice K., Mazumdar A., 2015, The Quest for Cradles of Life: Using the Fundamental Metallicity Relation to Hunt for the Most Habitable Type of Galaxy, apjl, Vol. 810, Number 1, p. L2, doi:10.1088/2041-8205/810/1/L2

[Debray & Wu2013] Debray A., Wu R., 2013, Astronomical Implications of Machine Learning, http://cs229.stanford.edu/proj2013

[Eckart & Young1936] Eckart C., Young G., 1936, The approximation of one matrix by another of lower rank, Psychometrika Vol. 1, Number 3, p. 211-218, doi:10.1007/BF02288367

[Fischer et al.2014] Fischer D. A., Howard A. W., Laughlin G. P., Macintosh B., Mahadevan S., Sahlmann J., Yee J. C., 2014, Exoplanet Detection Techniques, Protostars and Planet VI, University of Arizona Press, Tucson, p.715-737, doi:10.2458/azu_uapress_9780816531240-ch031

[Gonzalez, Brownlee & Ward2001] Gonzalez G., Brownlee D., Ward P., 2001, The Galactic Habitable Zone: Galactic Chemical Evolution, Icarus, Vol. 152, Number 1, p. 185-200, doi:10.1006/icar.2001.6617

[Green et al.2015] Green G. M. et al., 2015, A Three-dimensional Map of Milky Way Dust, apjl, Vol. 810, Number 1, p. 25, doi:10.1088/0004-637x/810/1/25

[Heller & Armstrong2014] Heller R., Armstrong J., 2014, Superhabitable Worlds, Astrobiology, Volume 14, Issue 1, p. 50-66, doi:10.1089/ast.2013.1088

[Hestenes1958] Hestenes M. R., 1958, Inversion of Matrices by Biorthogonalization and Related Results, Journal of the Society for Industrial and Applied Mathematics, Vol. 6, Number 1, p. 51-90, doi:10.1137/0106005

[Hassanat et al.2014] Hassanat A. B., Abbadi M. A., Altarawaneh G. A., Alhasnat A. A., 2014, Solving the Problem of the K Parameter in the KNN Classifier Using an Ensemble Learning Approach, preprint(arxiv:1409.0919v1)

[Kaltenegger et al.2011] Kaltenegger L, Udry S., Pepe F, A Habitable Planet around HD 85512?, preprint(arXiv:1108.3561)

[Hsu, Chang & Lin2016] Hsu C. W., Chang C. C., Lin C. J., 2016, A Practical Guide to Support Vector Classification, url:http://www.csie.ntu.edu.tw/ cjlin/papers/guide/guide.pdf

[Huang1959] Huang, S. S., 1959, The Problem of Life in the Universe And the Mode of Star Formation, Publications of the Astronomical Society of the Pacific, Vol. 71, Number 422, p. 421, url:http://stacks.iop.org/1538-3873/71/i=422/a=421

[Irwin et al.2014] Irwin L. N., Méndez A., Fairén A. G., Schulze-Makuch D., 2014, Assessing the Possibility of Biological Complexity on Other Worlds, with an Estimate of the Occurrence of Complex Life in the Milky Way Galaxy, Challenges, Vol. 5, Number 1, p. 159-174, doi:10.3390/challe5010159

[Irwin & Schulze-Makuch2011] Irwin L. N., Schulze-Makuch D., 2011, Cosmic Biology: How Life Could Evolve on Other World, Springer-Praxis, New York, ISBN-13:978-1441916464

[Kasting1993] Kasting, J. F., 1993, Earth's Early Atmosphere, Science, Vol. 259, Number 5097, p. 920-926, doi:10.1126/science.11536547

[Ridden-Harper et al.2016] Ridden-Harper A. R. et al., 2016, Search for an exosphere in sodium and calcium in the transmission spectrum of exoplanet 55 Cancri e, A&A, Vol. 593, p. A129, doi:10.1051/0004-6361/201628448

[McCauliff et al.2014] McCauliff S. D. et.al., 2015, Automatic Classification of Kepler Planetary Transit Candidates, apjl, Vol. 806, Number 1, p. 6, doi:10.1088/0004-637X/806/1/6

[Méndez2011] Méndez A., 2011, A Thermal Planetary Habitability Classification for Exoplanets, University of Puerto Rico at Arecibo, url:http://phl.upr.edu/library/notes/athermalplanetaryhabitabilityclassi ficationforexoplanets

[Méndez2015] Méndez A. et al., 2015, PHL's Exoplanet Catalog of the Planetary Habitability Laboratory at University of Puerto Rico at Arecibo,

http://phl.upr.edu/projects/habitable-exoplanets-catalog/data/database, as accessed in June 2015

[Méndez2016] Méndez A. et al., 2016, PHL's Exoplanet Catalog of the Planetary Habitability Laboratory at University of Puerto Rico at Arecibo, http://phl.upr.edu/projects/habitable-exoplanets-catalog/data/database, as accessed on 20th May 2016

[Parzen1962] Parzen E., 1962, On Estimation of a Probability Density Function and Mode. Annals of Mathematical Statistics, 33, no. 3, 1065-1076. doi:10.1214/aoms/1177704472

[Pedregosa et al.2011] Pedregosa F. et al., 2011, Scikit-learn: Machine Learning in Python, Journal of Machine Learning Research, Vol. 12, p. 2825-2830

[Peng, Zhang & Zhao2013] Peng N., Zhang Y., Zhao Y., 2013, A SVM-kNN method for quasar-star classification, Science China Physics, Mechanics and Astronomy, Volume 56, Number 6, p. 1227-1234, doi:10.1007/s11433-013-5083-8

[Powell1977] Powell M. J. D., 1977, Restart procedures for the conjugate gradient method, Mathematical Programming, Vol. 12, Number 1, p. 241-254, doi:10.1007/BF01593790

[Quinlan1986] Quinlan J. R., Induction Of Decision Trees, Machine Learning, Vol. 1, Number 1, p. 81-106, doi:10.1007/BF00116251

[Rish2001] Rish I., An Empirical Study Of Naive Bayes Classifier, In IJCAI 2001 Workshop on Empirical Methods in Artificial Intelligence, New York: IBM, Vol. 3, Number 22, p. 41-46

[Roesnblatt1956] Rosenblatt M., 1956, Remarks on some nonparametric estimates of a density function, Annals of Mathematical Statistics, 27: 832-837.

[Saha et al.2015] Saha S., Agrawal S., Manikandan R., Bora K., Routh S., Narasimhamurthy A., 2015, ASTROMLSKIT: A New Statistical Machine Learning Toolkit: A Platform for Data Analytics in Astronomy, preprint(arXiv:1504.07865)

[Saha et al.2016] Saha S., Sarkar J., Dwivedi A., Dwivedi N., Narasimhamurthy A. M., Roy R., 2016, A novel revenue optimization model to address the operation and maintenance cost of a data center, Journal of Cloud Computing, Vol. 5, Number 1, doi: 10.1186/s13677-015-0050-8

[Sale2015] Sale S. E., 2015, Three-dimensional extinction mapping and selection effects, MNRAS, Vol. 452, No. 3, p. 2960-2972, doi:10.1093/mnras/stv1459

[Schulze-Makuch et al.2011] Schulze-Makuch D. et al., 2011, A Two-Tiered Approach to Assessing the Habitability of Exoplanets, Astrobiology, Vol. 11, Number 10, p. 1041-1052, doi:10.1089/ast.2010.0592

[Soutter2012] Soutter J. L., 2012, Finding Exoplanets Around Eclipsing Binaries: A Feasibility Study using Mt Kent and Moore Observations, thesis, University of Southern Queensland, url:https://eprints.usq.edu.au/23220/

[Strigari et al.(2012)] Strigari L. E., Barnabè M., Marshall P. J., Blandford R. D., 2012, Nomads of the Galaxy, mnras, Vol. 423, Number 2, p. 1856-1865, doi:10.1111/j.1365-2966.2012.21009.x

[Theophilus, Reddy & Basak2016] Theophilus A. J., Reddy M., Basak S., ExoPlanet, Astrophysics Source Code Library (submitted), url:http://ascl.net/code/v/1475

[Swift et al.2013] Swift J. J., Johnson J. A., Morton T. D., Crepp J. R., Montet B. T., Fabrycky D. C., Muirhead P. S., 2013, Characterizing the Cool KOIs IV: Kepler-32 as a prototype for the formation of compact planetary systems throughout the Galaxy, apj, Vol. 764, Number 1, p. 105, doi:10.1088/0004-637X/764/1/105

[Waldmann & Tinetti2012] Waldmann I. P., Tinetti G., 2012, Exoplanetary spectroscopy using unsupervised machine learning, European Planetary Science Congress, EPSC2012-195

[Welling2005] Welling M., 2005, Fischer Linear Discriminant Analysis, Department Of Computer Science, University Of Toronto, url:https://www.ics.uci.edu/ welling/teaching/273ASpring09/Fisher-LDA.pdf

[Öberg et al.2015] Öberg, K. I., Guzmán, V. V., Furuya, K., et al., 2015. The comet-like composition of a protoplanetary disk as revealed by complex cyanides. nat, 520, 198

[Cassan et al.2012] Cassan, A., Kubas, D., Beaulieu, J.-P., et al., 2012. One or more bound planets per Milky Way star from microlensing observations. nat, 481, 167

[Wittenmyer et al.2014] Wittenmyer, R. A., Tuomi, M., Butler, R. P., et al., 2014. GJ 832c: A Super-Earth in the Habitable Zone. apj, 791, 114

[Nemirovski & Todd2008] Nemirovski, Arkadi S., and Todd, M. J., 2008. Interior-point methods for optimization. *Acta Numerica*, 17, 191. doi:10.1017/S0962492906370018.

[Hájková & Hurnik2007] Hájková, D. and Hurnik, J., 2007. Cobb-Douglas: The Case of a Converging Economy, *Czech Journal of Economics and Finance (Finance a uver)*, 57, 465

[Wu2001] Wu, D.-M., 1975. Estimation of the Cobb-Douglas Production Function. *Econometrica*, 43, 739. http://doi.org/10.2307/1913082

[Hossain et al.2012] Hossain, M., Majumder,A. & Basak, T., 2012. An Application of Non-Linear Cobb-Douglas Production Function to Selected Manufacturing Industries in Bangladesh. *Open Journal of Statistics*, 2, 460, doi: 10.4236/ojs.2012.24058

[Hassani2012] Hassani, A., 2012. Applications of Cobb-Douglas Production Function in Construction Time-Cost Analysis (M.Sc. thesis). University of Nebraska, Lincoln.

[Anglada-EscudÃl'2016] Anglada-Escudé G. et al., 2016. A terrestrial planet candidate in a temperate orbit around Proxima Centauri. *Nature*, 536, 437-440.

[Witze2016] Witze, A. 2016. Earth-sized planet around nearby star is astronomy dream come true. *Nature*, 536, 381-382.

[Starshot] Breakthrough Starshot. "A Russian billionaire has a crazy plan to reach a nearby planet that might harbor life". http://www.businessinsider.in. Retrieved 12 2016.

[Trappist-1] http://exoplanet.eu/catalog/trappist-1_c/, as accessed on Feb 25, 2017.

[cobb-douglas] Cobb, C. W. and Douglas, P. H., 1928. A Theory of Production. *American Economic Review*, 18 (Supplement), 139.

[rogers2014] Rogers, L. A., 2014. Most 1.6 Earth-radius Planets are Not Rocky. *Ap. J.*, 801:1, 41.

[Agrawal1993] Agrawal, R., Imielinski, T. and Swami A., Mining Association Rules between Sets of Items in Large Databases. In Proc. 1993 ACM SIGMOD International Conference on Management of Data, Washington DC (USA), pp. 207-216.

[Agrawal 1994] Agrawal, R., Srikant, R.1994. Fast Algorithms for Mining Association Rules in Large Databases. In Proc. 20th International Conference on Very Large Data Bases (VLDB '94), Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo (Eds.). (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA), 487-499.

[Ginde2016] Ginde, G., Saha, S., Mathur, A., Venkatagiri, S., Vadakkepat, S., Narasimhamurthy, A., B.S. Daya Sagar., 2016. ScientoBASE: A Framework and Model for Computing Scholastic Indicators of Non-Local Influence of Journals via Native Data Acquisition Algorithms. *J. Scientometrics*, 107:1, 1-51

[Nicholas et al.2010] Nicholas M. Ball & Robert J. Brunner 2010, Overview of Data Mining and Machine Learning methods. Retrieved on 25-04-16, from http://ned.ipac.caltech.edu/level5/March11/Ball/Ball2.html

[Davis et al.2007] T. M. Davis, E. Mortsell, J. Sollerman, A. C. Becker, S. Blondin, P. Challis, A. Clocchiatti, A. V. Filippenko, R. J. Foley, P. M. Garnavich, S. Jha, K. Krisciunas, R. P. Kirshner, B. Leibundgut, W. Li, T. Matheson, G. Miknaitis, G. Pignata, A. Rest, A. G. Riess, B. P. Schmidt, R. C. Smith, J. Spyromilio, C. W. Stubbs, N. B. Suntzeff, J. L. Tonry and W. M. Wood-Vasey 2007, Scrutinizing Exotic Cosmological Models Using ESSENCE Supernova Data Combined with Other Cosmological Probes. Astrophys.J, 666:716-725, DOI: 10.1086/519988

[Riess et al.2007] Riess et al. 2007, New Hubble Space Telescope Discoveries of Type Ia Supernovae at z > 1: Narrowing Constraints on the Early Behavior of Dark Energy. Astrophys.J, 659:98-121, DOI: 10.1086/510378

[Wood-Vassey et al.2007] Wood-Vassey et al. 2007, âĂIJObservational Constraints on the Nature of the Dark Energy: First Cosmological Results from the ESSENCE Supernova SurveyâĂİ, Astrophys.J, 666:694-715, DOI: 10.1086/518642

[Davis et al.] Type Ia supernova data used by Davis, MÃűrtsell, Sollerman, et al. 2007, Retrieved from http://dark.dark-cosmology.dk/ tamarad/SN/

[Fraser] Fraser Cain 2016, What are the Different Kinds of Supernovae?, retrieved on 20-04-2016, from http://www.universetoday.com/127865/what-are-the-different-kinds-of-supernovae/

[supernova tutorial] Type I and Type II Supernovae, retrieved on 20-04-2016, from http://hyperphysics.phy-astr.gsu.edu/hbase/astro/snovcn.html#c3

[Philipp] Philipp Podsiadlowski, Supernovae and Gamma Ray Bursts, Department of Astrophysics, University of Oxford, Oxford, OX1 3RH, UK retrieved from http://www-

astroṗhysicsȯxȧcůk/ podsi/sn_podsiṗdf

[Phillips93] Phillips, M.M. The absolute magnitudes of Type IA supernovae. Astrophys. J, 413 , L105 1993

[Abazajian et al.(2009)] Abazajian KN, Adelman-McCarthy JK, Agüeros MA, et al. 2009, apjs, 182, 543-558

[Adelman-McCarthy et al.(2008)] Adelman-McCarthy JK, Agüeros MA., Allam SS, et al. 2008, apjs, 175, 297-313

[Ahn et al.(2013)] Ahn CP, Alexandroff R, Prieto CA, Anderson SF, Anderton T and Andrews BH. 2013, The ninth data release of the Sloan Digital Sky Survey: first spectroscopic data from the SDSS-III Baryon Oscillation Spectroscopic Survey, 203, 1-14.

[Basak et al.(2016)] Basak S, Saha S et al., 2016, Star FGalaxy Separation using Adaboost and Asymmetric Adaboost, DOI: 10.13140/RG.2.2.20538.59842, 10/2016.

[Breiman(1996)] Breiman L, 1996, Bagging predictors. In Machine Learning, pages 123-140.

[Elting et al.(2008)] Elting C, Bailer-Jones CAL and Smith KW, 2008, Photometric Classification of Stars, Galaxies and Quasars in the Sloan Digital Sky Survey DR6 Using Support Vector Machines, AIP Conference Proceedings, Volume 1082, Issue 1.

[Freund & Schapire(1996)] Freund Y and Schapire RE, 1996, Experiments with a New Boosting algorithm. In Proceedings of the Thirteenth International Conference on Machine Learning, Page 148 - 156.

[Gao et al.(2008)] Gao D, Zhang Y and Zhao Y, 2008, Support vector machines and kd-tree for separating quasars from large survey data bases. Monthly Notices of the Royal Astronomical Society, 386: 1417-1425.

[Pedregosa et al.(2011)] Pedregosa F et al., 2011, Scikit-learn: Machine Learning in Python, Journal of Machine Learning Research, 12, 2825-2830.

[Peng et al.(2013)] Peng N, Zhang Y & Zhao Y, 2013, Sci. China Phys. Mech. Astron., 56: 1227.

[Vázquez & Alba-Castro(2015)] Vázquez IL and Alba-Castro JL, 2015, Revisiting AdaBoost for Cost-Sensitive Classification. Part I: Theoretical Perspective. CoRR, abs/1507.04125.

[Vázquez & Alba-Castro(2012)] Vázquez IL and Alba-Castro JL, 2012, Shedding light on the asymmetric learning capability of adaboost. Pattern Recogn. Lett., 33(3):247-255.