Eur. Phys. J. Spec. Top. (2021) 230:2221–2251
https://doi.org/10.1140/epjs/s11734-021-00203-z

The European Physical Journal
Special Topics

Regular Article

Habitability classification of exoplanets: a machine learning insight

Suryoday Basak[1], Archana Mathur[2,a], Abhijit Jeremiel Theophilus[3], Gouri Deshpande[4], and Jayant Murthy[5]

[1] Department of Computer Science and Engineering, Pennsylvania State University, State College, PA 16801, USA
[2] Department of Information Science and Engineering, Nitte Meenakshi Institute of Technology, Bengaluru, India
[3] Microsoft India Private limited, Bengaluru, India
[4] Department of Computer Science, 602 ICT Building, University of Calgary, 2500 University Drive NW, Calgary AB, T2N 1N4, Canada
[5] Indian Institute of Astrophysics, Sarjapur Main Road, 2nd Block, Koramangala, Bengaluru, Karnataka 560034, India

Received 18 July 2020 / Accepted 23 June 2021 / Published online 21 July 2021
© The Author(s), under exclusive licence to EDP Sciences, Springer-Verlag GmbH Germany, part of Springer Nature 2021

Abstract We explore the efficacy of machine learning (ML) in characterizing exoplanets into different classes. The source of the data used in this work is University of Puerto Rico's Planetary Habitability Laboratory's Exoplanets Catalog (PHL-EC). We perform a detailed analysis of the structure of the data and propose methods that can be used to effectively categorize new exoplanet samples. Our contributions are twofold. We elaborate on the results obtained by using ML algorithms by stating the accuracy of each method used and propose a paradigm to automate the task of exoplanet classification for relevant outcomes. In particular, we focus on the results obtained by novel neural network architectures for the classification task, as they have performed very well despite complexities that are inherent to this problem. The exploration led to the development of new methods fundamental and relevant to the context of the problem and beyond. The data exploration and experimentation also result in the development of a general data methodology and a set of best practices which can be used for exploratory data analysis experiments.

1 Introduction

There has been a consensus among the community of astronomers regarding the possibility of the Earth as a very rare case of a habitable planet. From a machine learning perspective, the Earth is an instance of an anomaly among many other objects (which do not harbor life) within and outside the solar system. An anomaly is a deviation from the common trend, a peculiar phenomena that could be detected by computing dissimilarities from the general features of objects which are not anomalous. The Earth is considered an anomaly partly because so far, missions exploring specific planets like Mars and Venus have found no traces of life. However, over the past two decades, discoveries of exoplanets have multiplied manyfold. This helps us conjecture that planets around stars are a rule. It is also observed that the actual number of planets far exceeds the number of stars in our galaxy [1]. It is, therefore, pertinent, to investigate and analyze the conditions and possibilities that can lead to the emergenceof life [2].

[a] e-mail: mathurarchana77@gmail.com (corresponding author)

The necessity is to develop a quick and efficient (Scientifically correct) method to classify exoplanets based on their physical properties into relevant habitability classes. We note, the inference/prediction method is reliant on the veracity of training labels.

The discovery of exoplanets [3–5] is a complex process because of a number of reasons. The small size of exoplanets in comparison to other types of stellar objects makes such planets difficult to discover. On the other hand, objects such as stars, galaxies, quasars, etc., can be discovered with greater ease. Given the rapid technological improvements in inference and data processing pipelines to handle accumulation of a large amount of data, it is important to explore advanced methods of data analysis. These methods are capable of dealing with noise in data and make data ready for classification tasks into appropriate categories based on the physical characteristics. We call these characteristic features of data, in machine learning parlance. Existing work on characterizing exoplanets are based on assigning habitability scores to each planet which allows for a quantitative comparison with Earth. These metrics can provide us an idea of how similar or dissimilar an exoplanet is to Earth, with dissimilar exoplanets typically having a larger score than the similar planets. Habitability score is indicative of the potential of habitability, based on a restricted list of features. The Biological Complexity Index (BCI) [6], the Earth Similarity Index (ESI) and the Planetary Habitability Index (PHI) [7] are distance-based metrics which measure the similarity of a planet to that of Earth; the Cobb–Douglas Habitability Score (CDHS) [8–12], makes use of production economics to quantify the potential of habitability of an exoplanet [13–15]. In [16], a comparison between the ESI and CDHS has been put forth, highlighting the key differences in the methodology between the two. In particular, neural network architectures are becoming more and more popular in classification tasks, for its ability to predict using a repetitive epoch-based learning. Recently, NASA announced the confirmation of two exoplanets using artificial intelligence [17]. In [18] used an advanced tree-based classifier, Gradient Boosted Decision Trees (GBDT) [19,20] to classify Proxima b and planets in the TRAPPIST-1 system [21,22]. The accuracies were nearly perfect, giving us the basis of exploring other machine classifiers and approaches for the task.

The features we use for demonstrating the efficacy of the classifiers are small in number and are related to the real-world observations of exoplanets. Observing exoplanets is no easy task—to draw a complete picture of any exoplanet, observations from multiple missions are usually combined. For instance, the method of radial velocity can provide us the minimum mass of the planet and the distance of the planet from its parent star, transit photometry can provide us with information on the radius of a planet, and the method of gravitational lensing can provide us the information regarding the mass. In contrast to this, stars are generally easier to observe and profile through various methods of spectrometry and photometry in different ranges of wavelengths of the electromagnetic spectrum. Hence, by the time a planet has been discovered around a star, we would possibly have fairly rich information about the parent star, and this includes information such as the luminosity, radius, temperature, etc. For this, we use the data from PHL's exoplanet catalog, and at the present time, given that the number of samples that are potentially habitable (in the psychroplanet and mesoplanet classes, respectively; elaborated in Sect. 2), we use all the samples available, but use only a very small set of planetary parameters. The goal is to develop a pipeline based on various approaches which can detect interesting exoplanetary samples in databases, quickly after their discovery, thus potentially accelerating the thermal characterization of exoplanets before all the attributes are fully discovered and cataloged.

In this paper, we explore the efficacy of different ML approaches to classify exoplanets into thermal habitability classes [23] based on physical characteristics of the respective exoplanets and their parent stars. For this task, we compare the results of two families of classifiers: tree-based classifiers and neural networks. We emphasize on the challenges with the dataset which include the dominance in the dataset of non-habitable planetary samples and a high degree of inseparability, and we address these challenges using modern methods of sample balancing. The motivation for comparing the results of neural networks with those of tree classifiers follows from the results reported in [18]; modern neural networks are flexible and can be appropriately tuned to work on different kinds of data [24,25] . To upsample and balance the data from different classes in the dataset, we have used generative adversarial networks (GAN).

Our intention of comparing the results of various methods and providing explanations for the same is to demonstrate that the method of classification [26] which works the best is very closely related to what we can understand from the data: how the samples of various classes are distributed and how various features are related. To the best of our knowledge, such an exploration has not been done prior to the current work. A quick scan of feature importance reveals that surface temperature is the most important attribute/feature among all attributes in the PHL-EC data. Since surface temperature for many planets are estimated, and not measured and the fact that the class labeling is done using surface temperature, the reliability of training labels is questionable. Therefore, the efficacy of the machine learning algorithms is also not beyond reasonable doubt as most training algorithms are as good as data! Moreover, since surface temperature is the most dominant feature, the classification including it is almost equivalent to classification based on surface temperature alone. This implies, any machine classification method, in the context of this problem, is based on one feature primarily. This will limit the usefulness of surface temperature-based classification. This motivated us to modify existing methods and propose new ones which can work well on reduced feature sets and investigate the accuracy of prediction in such cases. A reduced feature set is equivalent to characterizing exoplanets based on attributes excluding surface temperature and all features related to surface temperature. This has not been attempted before.

The remainder of the paper is organized as follows. In Sect. 2, we briefly present the structure of the PHL-EC data and the complexities associated with a machine learning exploration of this data set. In Sect. 3, we talk about our contribution towards solving the complex classification task of exoplanets collected from PHL-EC. Sections 4 and 5 investigate the existing machine learning algorithms, such as Naive Bayes, Linear Discriminant Analysis, Support Vector Machine, kNN, Decision Tree, Random Forest and XGBoost. Not just methods, these sections cover the working principles of ML algorithms along with the results of executions for the methods. Section 6 begins with introduction to neural networks, and further elaborates on the working principle behind novel architectures proposed by authors. Next section presents the experimental results of these architectures for solving the aforementioned challenges associated with PHL-EC data. In parallel, the section also introduces SBAF, a novel activation function used in neural networks for PHL-EC data classification. We conclude our manuscript with discussion and conclusion.

2 Data and problem statement

Combined with stellar data from the Hipparcos catalog [23], the PHL-EC dataset [27] consists of a total of 68 features (of which 13 are categorical and 55 are continuous valued) and more than 3800 confirmed exoplanets (downloaded from the official website in 2018,[1]); the catalog includes important features like atmospheric type, mass, radius, estimated surface temperature, escape velocity, earth's similarity index, flux, orbital velocity, etc. The description of these features is presented in Table 12 of appendix and the downloaded catalog is available at github repository.[2] The PHL-EC catalog consists of observed as well as estimated attributes. Hence, it presents interesting challenges from the analysis point of view. There are six classes in the dataset, of which we can use three in our analysis as they are sufficiently large in size. These are namely non-habitable, mesoplanet, and psychroplanet classes. These three classes or types of planets can be described on the basis of their thermal properties as follows:

1. **Mesoplanets**: These are also referred to as M-planets. These planets have mean global surface temperature between 0 °C and 50 °C, a necessary condition for complex terrestrial life. These are generally referred as Earth-like planets.
2. **Psychroplanets**: These planets have mean global surface temperature between −50 °C and 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 remaining three classes in the data are those of thermoplanet, hypopsychroplanet and hyperthermoplanet. However, at the time of writing this paper, the number of samples in each of these classes is too small (each class has less than 3 samples) to reliably take them into consideration for an ML-based exploration.

While running the classification methods, we consider multiple features of the parent sun of the exoplanets that include mass, radius, effective temperature, luminosity, and the limits of the habitable zone. In addition to this, we consider the following planetary attributes in separate experimental settings:

1. **Minimum Mass** and **Distance from the Parent Star**, corresponding to observations of radial velocity
2. **Mass**, corresponding to observations using gravitational microlensing
3. **Radius**, corresponding to observations using transit photometry.

As a first step, data from PHL-EC are pre-processed. An important challenge in the dataset is that a total of about 1% of the data is missing (with a majority being of the feature P. Max Mass) and in order to overcome this, we used a simple method of removing instances with missing data after extracting the appropriate features for each experiment, as most of the missing data is from the non-habitable class. Following this, the ML approaches were used on these pre-processed datasets. The online data source for the current work is available at [28].

3 Contributions

Our contribution are largely of three kinds:

1. Application of Existing Methods in Machine Learning: We use existing families of classifiers and data processing methods which are popular in the literature. The purpose of this is twofold: first, to explore how the well-known and well-tested methods can be used to automate the classification of exoplanets and second, to set a performance baseline for new methods that will be subsequently developed for the same tasks.
2. Novel Methods of Characterization and Classification: Based on the nuances in the dataset, and results from existing methods, we have developed new methods to classify and characterize samples of exoplanets. We propose the elastic Gini splitting criteria. To establish efficacy and relevance of the proposed methods, results are compared with, where ever appropriate, the existing methods in literature.
3. Exploration of novel neural network architectures [29]—RWNN, GAN and Fusion net: We have introduced architectures based on conventional neural networks, and have compared their results using the estimated attributes from PHL-EC data set. In parallel, we also inspect the usage SBAF activation function on the same dataset and present a detailed analysis of its comparison with sigmoid.

The transition from one item to the other is effortless as demonstrated later in the manuscript. The effort was initially meant at exploring different classification algorithms in the study of automated discrimination of habitability. The exercise evolved into writing novel methods as the authors felt strong reasons to improve upon the existing ones, in the context of the problem. In all the results that we have reported, we have related the performance of the different classifiers to the nature and structure of the data. The algorithms are not treated as 'black-boxes' and are thoroughly examined for the purpose of determining the appropriateness of application to the PHL-EC dataset. The working principles of

[1] The exoplanet count may vary at the time of reading this paper, as the catalog is constantly updated.
[2] https://github.com/mathurarchana77/Habitability-PHL-EC-Dataset.

each classifier we have tried are not the same and we have provided justification for the results obtained.

4 Application of existing methods in machine learning

We briefly describe methods popular in the literature that are being used in the manuscript. We begin with the understanding of the existing machine learning classifiers and later investigate on methods that could be used for enhancing the accuracy of classification classifiers. Two methods of synthetic oversampling for creating a balance in an unbalanced data are also discussed in this section.

4.1 Existing methods used from machine learning algorithms

The following are the ML classification algorithms that are explored in the current work.

a. Probabilistic Classifiers: Gaussian Naive Bayes evaluates the classification labels based on class-conditional probabilities with class apriori probabilities [30]. GNB works on the assumptions that the features are independent of each other and that they all come from a Gaussian distribution.
b. Instance-Based Classifiers: The k-nearest neighbor classifier is an instance-based classifier where the distance between the neighbors in the input space is used as a measure for categorization [31]. Here, k is the number of nearest neighbors which are considered for automated classification: the class with the largest number of instances within the nearest k neighbors is predicted as the class of a test sample. We considered k to be 3 while the weights are assigned uniform values.
c. Hard-Boundary Classifiers: The working principle of support vector machines is to construct hard boundaries which are n-dimensional hyperplanes, between the samples of different classes [32], [33]. Here, n is the dimensionality of the feature space of the data as fed to the machine. We tried SVM without a kernel, and with a radial basis kernel [34].
d. The parameters setup for linear discriminant analysis classifier was implemented by the decomposition strategy similar to SVM [35]. No shrinkage metric was specified and no class prior probabilities were assigned.
e. Tree-Based Classifiers: Decision trees build tree based data structures by using a criterion, namely information gain (or simply, gain) [36]. The discrimination which leads to the best value of gain is used to develop a rule; should the rule not result in a perfect discrimination of the data (which is often a consequence of accepting the best rule), then the criterion is recursively applied to the partitions in the data created by the previous rule. A random forest is an ensemble of many decision trees [37] and gradient boosted decision trees are further sophisticated as they do not consider the data in their raw form but use a functional approximation prior to discrimination [19]. The splitting criteria that we used were Gini and elastic Gini (discussed in Section 4.2.3) We address the complexity and attribute correlations in the dataset in a two-fold manner: by considering all the features in the dataset in one set of automated classification experiments, and by considering only the basic observables of mass and radius in another set of experiments. The accuracies of either set of experiments are presented and discussed. The existing methods were implemented using the scikit-learn module in python [38], except GBDTs, which was implemented using XGBoost [20].

4.2 Novel methods of characterization and classification

4.2.1 Synthetic oversampling using Parzen window estimation

Artificial oversampling is an approach to handle the effects of bias due to a class with a large sample size in data. Here, we try to artificially add samples to the classes with lesser number of samples, namely the mesoplanet and the psychroplanet classes. Essentially, the paradigm of choice is to analyze the class-wise distribution of the data and to generate reasonable samples which reflect the general properties of their respective classes.

In this approach, we estimate the density of the data by approximating a distribution function empirically—we do not assume that the numeric values in the data samples are drawn from a standard probability distribution. The method we use for this is known as *window estimation*. This approach was developed by Rosenblatt and Parzen [39,40]. In a broader sense, window estimation is a method of kernel density estimation (KDE). In our work, we use this to augment the samples of the different classes in the data and we choose the features that we require in different experimental scenarios. Detailed exploration of this method is available in Appendix A.2.

4.2.2 Synthetic oversampling by assuming a distribution in the data

Assuming a distribution in data is a common approach in different simulations in physics. For our experiments, we assume that the Surface Temperature follows a Poisson distribution [41]. We randomly sample the remaining features as entire feature vectors from the original dataset; having estimated the value of surface temperature based on the distribution, we concatenate it with random feature vectors of the remaining parameters of samples from the same class. We use [42] to ensure the purity in the class belongingness of the synthetic samples generated by altering or rectifying the class of the synthetic samples which represent a different class. The steps in the algorithm are as follows:

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

step 2 : The distribution of either class is estimated as a Poisson distribution:

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

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 dataset.

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 dataset.

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 SVM and KNN are used here, along with density estimation, to rectify the class belongingness (class labels) of artificially generated random samples and not as classifiers. If an artificially 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) [43], the oversampling does not depend on individual samples in the data.

4.2.3 Elastic splitting criteria for decision trees

Decision Tree (DT) classifier is a popular classification algorithm. We used the standard Decision tree classifiers (Table 1 in Sect. 5) in habitability classification of exoplanets. DTs partition the feature space of data based on a partitioning heuristic or a splitting criterion. We introduce a new splitting criterion, which we call the elastic Gini Impurity. The elastic Gini differs from the two popular criteria, namely the Gini impurity criterion and Shannon's entropy—while the Gini impurity and Shannon's entropy are concave functions, the elastic Gini may be convex, concave, or non-convex and non-concave depending on the values of shape-controlling parameters called elasticities. We discuss the necessary conditions to determine the convexity of elastic Gini and suggest conditions for its usage. An interesting observation that can be made from the data is that the non-habitable class of exoplanets has a more expansive distribution in the feature space while the classes of mesoplanets and psychroplanets occupy a small range. This bias is more difficult to handle than a class-proportion bias as a small variance in some classes could easily be confused with the more distributed class, or be less representative to a classifier. Traditionally, splitting criteria for decision trees do not account for this and provide an equitable representation to all classes. However, taking into consideration structure and spread of the data, we try to assign elastic exponential parameters to all the probabilities in the Gini impurity index to develop an asymmetric representation for a split [44]. Our intention is to adjust the bias towards the mesoplanet and psychroplanet classes so that the final results obtained are more efficacious. The elastic splitting criteria is given by:

$$\begin{gathered} I(p_1, p_2, \ldots, p_n) = k\left(1 - \sum_{i=1}^{n} p_i^{\alpha_i}\right) \\ G = I_N - I_L - I_R, \end{gathered} \tag{1}$$

where I is the impurity of a node [36], n is the number of classes in the dataset, p_i is the probability of finding an instance of the i_{th} class in a node, α_i is the elasticity associated with the i_{th} class in the data. In order to ascertain the best split in a node in a decision tree, the gain G is used and $I_{\text{N}}, I_{\text{L}}$ and I_{R} represent the impurity of the parent node and potential left and right child nodes after a split, respectively. In our implementation, we have taken $k = 5$ so that this criteria is backward compatible with other existing libraries. The new addition to this splitting criteria are the elasticities α_i which are constants that skew the response of the splitting criteria based on our preference and are supplied to the algorithm as parameters. The skewed response is visualized in Fig. 1. What is clear is that for any $\alpha_i > 1$, the functional form of I is concave, which is a condition for a function to be used as a splitting criteria [45]. The first and second partial derivatives of $I(p_1; p_2; \ldots; p_n)$ are:

$$\begin{gathered} \frac{\partial I}{\partial p_i} = -\alpha_i p_i^{\alpha_i - 1} \\ \frac{\partial I}{\partial p_j \partial p_i} = \begin{cases} -\alpha_i p_i^{\alpha_i - 1}, & if \;\; i = j \\ 0, & \quad\;\; i \neq j \end{cases} . \end{gathered}$$

Table 1 Confusion matrices of the results of classification using the entire set of features as well as only mass and radius, both for undersampling and oversampling using Parzen window estimation

Algorithm	True	Method of handling bias due to class imbalance											
		Artificial undersampling						KDE using Parzen window estimation					
		Pred											
		All features			Mass and radius only			All features			Mass and radius only		
		N	P	M	N	P	M	N	P	M	N	P	M
	N	99.51	0.25	0.25	96.34	1.13	2.53	99.63	0.17	0.2	96.96	1.86	1.18
GNB	P	0.85	98.87	0.28	6.96	8.86	84.18	0.0	100.0	0.0	0.0	38.5	61.5
	M	0.0	6.54	93.46	0.0	7.13	92.87	0.0	0.08	99.92	0.0	5.92	94.08
	N	97.32	1.18	1.5	94.91	0.4	4.69	98.87	0.2	0.93	93.64	3.37	2.99
LDA	P	0.31	67.18	32.51	17.8	0.0	82.2	0.0	89.06	10.94	0.0	41.51	58.49
	M	0.0	3.93	96.07	10.94	0.0	89.06	0.0	5.62	94.38	0.0	21.96	78.04
	N	98.12	0.52	1.36	96.39	0.94	2.67	99.65	0.12	0.23	95.97	2.36	1.68
SVM	P	3.01	86.07	10.93	6.77	20.31	72.92	0.01	99.78	0.21	0.0	70.06	29.94
	M	3.02	2.01	94.97	0.16	13.46	86.38	0.01	0.08	99.92	0.0	29.16	70.84
	N	100.0	0.0	0.0	96.87	0.89	2.24	100.0	0.0	0.0	96.98	1.11	1.91
RBF-SVM	P	100.0	0.0	0.0	20.3	19.1	60.6	100.0	0.0	0.0	0.0	79.05	20.95
	M	100.0	0.0	0.0	19.11	14.18	66.72	100.0	0.0	0.0	0.02	17.96	82.03
	N	97.77	0.45	1.78	97.02	1.55	1.42	99.16	0.2	0.64	96.82	1.95	1.23
KNN	P	0.28	21.41	78.31	12.16	36.17	51.67	0.0	100.0	0.0	0.26	91.62	8.12
	M	0.16	14.81	85.02	11.73	25.74	62.52	0.0	0.0	100.0	0.43	12.99	86.58
	N	99.44	0.4	0.16	97.08	1.05	1.88	99.96	0.0	0.04	97.82	1.13	1.04
DT	P	3.98	92.33	3.69	9.31	68.77	21.92	0.0	100.0	0.0	0.57	94.96	4.47
	M	0.66	1.82	97.51	11.74	15.77	72.48	0.0	0.0	100.0	0.5	4.05	95.45
	N	99.75	0.17	0.08	97.01	1.22	1.77	99.94	0.02	0.04	98.07	0.89	1.05
RF	P	2.77	96.62	0.62	12.65	63.24	24.12	0.0	100.0	0.0	0.09	96.19	3.73
	M	0.16	2.72	97.12	10.82	21.13	68.05	0.0	0.05	99.95	0.22	4.67	95.11
	N	99.69	0.31	0.0	97.46	1.66	0.88	99.11	0.22	0.67	96.64	1.68	1.68
GBDT	P	1.16	96.51	2.33	4.0	72.0	24.0	0.0	99.96	0.04	0.1	88.43	11.47
	M	3.7	1.23	95.06	3.65	25.55	70.8	0.0	0.0	100.0	0.0	6.77	93.23

The larger the values of the diagonal elements in the confusion matrix of a classifier, the better is the performance of the respective classifier. Terminologies used: *GNB* Gaussian Naive Bayes, *LDA* linear discriminant analysis, *SVM* support vector machine, *RBF-SVM* radial basis function SVM, *KNN* K-nearest neighbor, *DT* decision tree, *RF* random forest, *GBDT* gradient boosting decision tree, *N* non-habitable, *P* psychroplanets, *M* mesoplanets

The Hessian matrix of $I(p_1; p_2; \ldots ; p_n)$ consequently is:

$$H(I(p_1, p_2, \ldots , p_n)) = \begin{bmatrix} -\alpha_1 p_1^{\alpha_1 - 1} & 0 & \cdots & 0 \\ 0 & -\alpha_2 p_2^{\alpha_2 - 1} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & -\alpha_n p_n^{\alpha_n - 1} \end{bmatrix}. \tag{2}$$

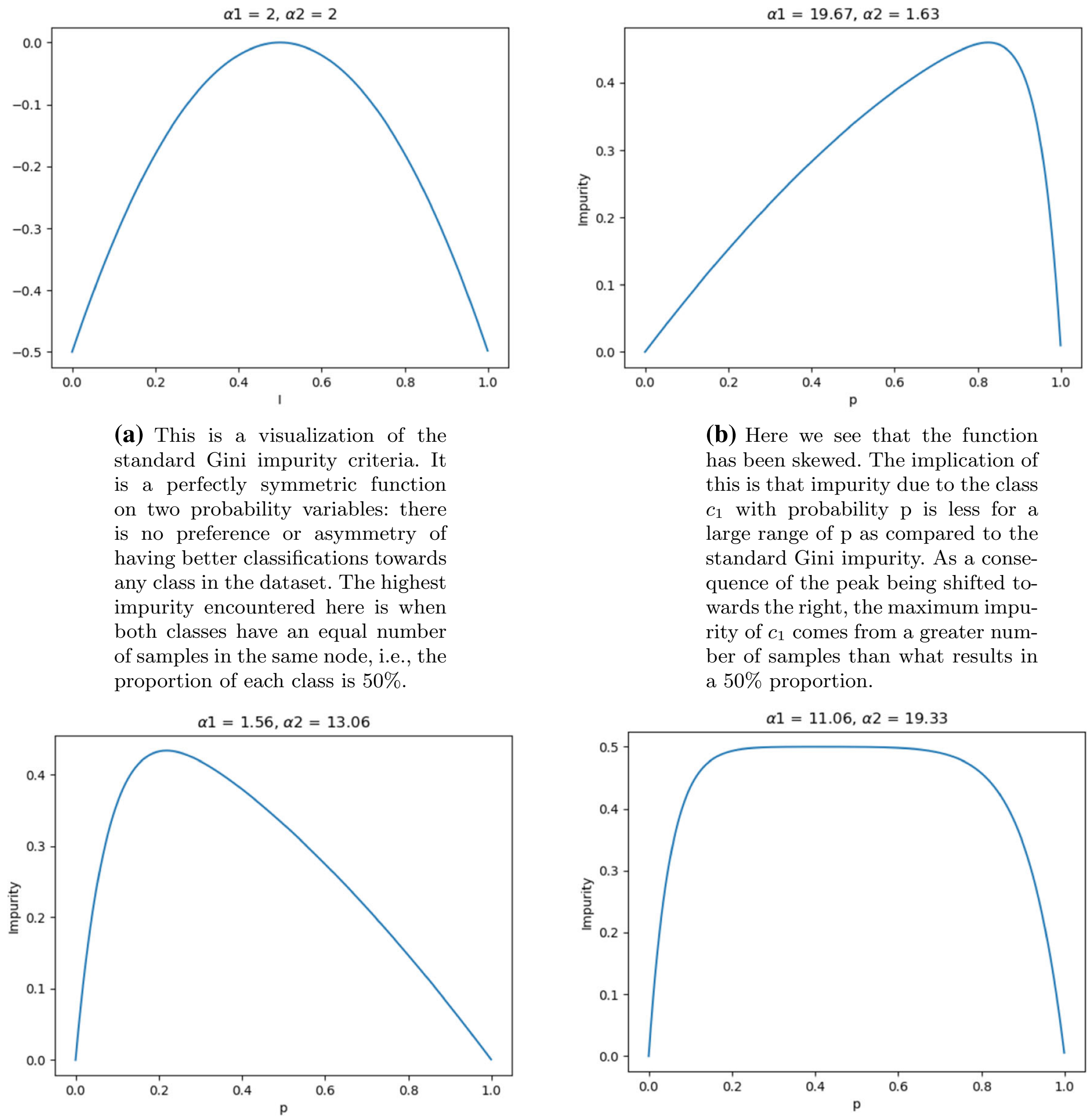
We briefly justify the concavity from the Hessian matrix by dealing with a three-class scenario as this is relevant to our work. For a function to be concave, its Hessian matrix must be symmetric and negative definite. This implies that the odd numbered primary minors must be negative and the even numbered primary minors must be positive . Let D_i represent the i_{th} primary minor. In the Hessian matrix of the elastic Gini, we have:

$$\begin{aligned} D_1 &= -\alpha_1(\alpha_1 - 1)p_1^{\alpha_1 - 2} \\ D_2 &= \alpha_1(\alpha_1 - 1)p_1^{\alpha_1 - 2} X \alpha_2(\alpha_2 - 1)p_2^{\alpha_2 - 2} \\ D_3 &= -\alpha_1(\alpha_1 - 1)p_1^{\alpha_1 - 2} X \alpha_2(\alpha_2 - 1)p_2^{\alpha_2 - 2} X \alpha_3(\alpha_3 - 1)p_3^{\alpha_3 - 2}. \end{aligned} \tag{3}$$

D1 and D3 can be negative and D2 can be positive $\alpha_i > 1 \forall i$. Thus, following this condition, the function is concave, and we ensure in our computer program that we supply elasticities that are strictly greater than 1. We implemented this method using the anytree module in the Python programming language running on a Linux-based OS on a computer with a 2.2GHz, dual-core, Core-i3 processor. Further explanations and elaboration of the properties of elastic gini used in the current work (and their continuation in the future) can be found in the Appendix B.

5 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 [12]. Machine Learning-based approaches can carry out this analysis effectively [46]. ML based approaches are broadly categorized into two main types: supervised and unsuper-



(a) This is a visualization of the standard Gini impurity criteria. It is a perfectly symmetric function on two probability variables: there is no preference or asymmetry of having better classifications towards any class in the dataset. The highest impurity encountered here is when both classes have an equal number of samples in the same node, i.e., the proportion of each class is 50%.

(b) Here we see that the function has been skewed. The implication of this is that impurity due to the class c_1 with probability p is less for a large range of p as compared to the standard Gini impurity. As a consequence of the peak being shifted towards the right, the maximum impurity of c_1 comes from a greater number of samples than what results in a 50% proportion.

(c) The impurity that results from the class with probability p - 1 is lesser. The curvature is similar to what we observe in the above graphs.

(d) Behavior due to the exponents (α_s) when both classes contribute proportionately, and neither gets a clear preference: confusion may arise and such cases should be avoided with appropriate parameter tuning.

Fig. 1 All the graphs plotted here are for a 2-class classification problem. If the probability of occurrence of a sample of one class in a node is p, then consequently, the probability of occurrence of a sample of the other class in the same node is $1 - \mathrm{p}$. We assume non-habitable planets as a class with probability p and meso/psychroplanets are merged as a class probability $1 - p$

vised 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 [47]. 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 similarity and distances of feature vectors, whereas non-metric classifiers are 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.

5.1 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 have 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 [30]. The method assumes independent distribution for attributes and thus estimates class conditional probability as in Eq. 4.

$$P(X \mid Y_i) = P(X_1 \mid Y_i) \times P(X_2 \mid Y_i) \times \cdots \times P(X_n \mid Y_i) \quad (4)$$

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 $X = \{x_1, x_2, \ldots, x_d\}$ and $C = \{c_1, c_2, \ldots, c_d\}$ be the set of *feature vectors* corresponding to each entity in the data set (where the number of attributes used are 49 , and the set of corresponding *class labels* (which is 3 here, *meso, psychro and non habitable planets*. Reiterating, the attributes in the PHL-EC data set are mass of the planet, estimated surface temperature, pressure, etc., of each cataloged planet. The subscript i refers to the ith class in the data set.

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 Eq. (5).

$$p(x \mid c_j) = \prod_{k=1}^{d} p(x_k \mid c_j). \quad (5)$$

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 given in Eq. (6).

$$p(c_j \mid x) = p(c_j) \prod_{k=1}^{d} p(x_k \mid c_j). \quad (6)$$

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 is a planet, the class labels meso, psychro or non habitable(whichever class has highest probability score for a particular planet) will be assigned to that class or category.

5.2 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 co-variance is same for all classes. The authors implemented an enhanced version of LDA (often called *Regularized Discriminant* Analysis). This involves *Eigen Decomposition* of the *sample* co-variance 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 [48]. The algorithm is expressed as:

step 1: Compute mean vectors, μ_i for $i = 1, 2, \ldots, c$ 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' (Sect. 5.1). Hence, μ_i is the mean vector of class i, where an element at position j in μ_i is the average of all the values of the jth attribute for the class i.

step 2: Compute scatter matrices between classes and within class as shown in Eqs. (7), (8) and (9).

$$S_B = \sum_{i=1}^{c} M_i(\mu_i - m)(\mu_i - m)^T, \quad (7)$$

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 μ_i is the sample mean.

$$S_w = \sum_{i=1}^{c} S_i, \quad (8)$$

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

$$S_i = \sum_{x \in D_i}^{n} (x_i - \mu_i)(x_i - \mu_i)^t. \tag{9}$$

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 dimension of X is $n \times d$, and ith row is the ith sample. Every row in the matrix X corresponds to an entity in the data set.

2. *Support Vector Machine* (SVM): SVM classifiers are effective for binary class discrimination [49]. The basic formulation is designed for the linear classification problem; the algorithm yields an optimal hyper-plane, i.e. one that maintains largest minimum distance from the training data, 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 *hyper-plane* may be found. The algorithm for the classification using SVM 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.

$$S = S \cup V. \tag{10}$$

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

Next, we survey another metric classifier, k-Nearest Neighbor(k-NN).

3. *k-Nearest Neighbor*: k-Nearest Neighbors is an instance-based classifier that compares new incoming instance with the data stored in memory [50]. k-Nearest Neighbors algorithm (or k-NN as popularly known) is a non-parametric method used for classification and regression. 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 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 $i \in S$.

Remark Surveying various machine learning algorithms was a key motivation even though some methods and algorithms could have been easily done with. 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 earlier. 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. The next subsection, non-metric classifiers which include Decision Tree, Random Forest and Boosted Trees, would bolster the logic behind discouraging "Black box" approaches to Data Analytics in the context of this problem or otherwise.

5.3 Non-metric classifiers

1. *Decision Tree*: A Decision Tree constructs a tree structure that can be used for classification or regression [36]. Each of the nodes in the tree splits the training set based on a feature (attribute); the first node is called the *root node*, which is based on a 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 called the *root node*. Classification is of interest only in cases with multiple classes and hence a root node is not 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. 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 index or entropy must be used to determine the best split attribute between any two subsequent levels in the decision tree.

step 4: *Pruning* must 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.

Remark It is important to observe here and in the later part of the manuscript that Decision Trees and Tree-based algorithms yield significantly better results for balanced and 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) [37]. The task of classifying new object from the data set is accomplished using randomly constructed tree. Classification implies 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 as belonging to class c_i.

Remark If a Random Forest consists of ten decision trees, out of which six trees classified a feature vector x as being habitable, and the remaining four trees classified x as being non-habitable, then we may conclude that the Random Forest classified x as a habitable planet.

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 learners. Given a training set $X = \{x_1, x_2, \ldots, x_n\}$ with responses $Y = \{y_1, y_2, \ldots, y_n\}$, bagging selects a random sample of the training set with iterative replacement and fits trees to these samples subsequently:

step 1: For $b = 1, \ldots, B$, sample with replacement, n training examples from X, Y; call these X_b, Y_b.

step 2: Next, decision or regression tree is trained: f_b on X_b, Y_b.

3: After training, predictions for unseen samples x' can be made by averaging the predictions from all the individual regression trees on x', or by considering the majority votes in the case of decision trees.

We terminate the brief primer on non-metric classifiers by including a recent tree-boosted machine learning algorithm, *XGB*.

3. *XGBoost*: XGBoost 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*. Similar to Random Forests, an objective function is minimized and each leaf has an associated score which determines the class membership of any test entity. The steps in XGBoosted trees are as follows:

step 1: For $b = 1, \ldots, B$, sample with replacement, n training examples X_b, Y_b from X, Y.

step 2: A decision or regression tree is trained: f_b on X_b, Y_b. An objective function is optimized.

step 3: Steps 1 and 2 are repeated by considering more trees. 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 the same way as in Random Forests.

5.4 Random forest: mathematical representation of binomial distribution and an example

A random forest uses 2/3rd of the observation for building individual decision trees and every tree generates an outcome or a vote as class prediction for a specific sample. Votes are counted for every class and one that acquires largest number of votes becomes the predicted class for that sample. The concept utilizes the wisdom of crowds, to make decisions for a particular sample. If the attribute used in building trees is binary, the vote could be a *yes* or *no*. The random forest score is the count of the affirmative votes received for the samples and probability that predicted outcome is correct is obtained through the percentage of affirmative votes.

In decision tree, inclusion of an attribute (x) is in the form of *yes*/*no*. This implies that binomial distribution can be used to find the probability of the presence of an attribute in decision trees. For example, if there are 20 random forests, probability that x will classify correctly is 0.7 and if k defines number of times attribute x is present in decision trees, then Probability Mass Function can be defined as -

$$Pr(X = k) = \binom{n}{k} p^k (1-p)^{n-k}. \tag{11}$$

Here, $n = 20$ and $p = 0.7$. Investigating the different values of k ranging between $0, 1, 2, \ldots, 20$, $k = 0$ shows the attribute was never used in decision making, and conversely, $k = 20$ implies the attribute was utilized in all the decision trees .

The Cumulative Distribution Function for $n = 20$, $p = 0.7$ and $m = 10$, is given by

$$Pr(X \le m+1) = \sum_{k=0}^{n} \binom{n}{k} p^k (1-p)^{n-k}, \tag{12}$$

$$Pr(X \le m+1) = \sum_{k=0}^{10} \binom{10}{k} (0.6)^k (0.4)^{n-k} = 1.0. \tag{13}$$

One can make out, as k becomes large, the probability of correct classification becomes large too, since Cumulative Distribution reaches close to 1.

5.5 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 Decision Trees of of a Random Forest. 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 Eq. 14.

$$X(n, p) = P[X(n, p_i) \ge k] = 1 - B(k; n, p_i). \tag{14}$$

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 overfitting 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 (15).

$$\begin{aligned} \text{var} &= \prod_{j=1}^{N} \min\{1 - B(k_{ij}; n_j, p_j\} \\ \mathcal{I} &= \begin{cases} 0, & \text{if var} < \text{confidence threshold} \\ \text{var}, & \text{otherwise}, \end{cases} \end{aligned} \tag{15}$$

subject to the conditions, $c \ge 1$, $p = [0, 1]$, confidence threshold $= [0, 1)$, $k_{ij} \le n_j$, $i = \{1, 2, \ldots, c\}$, $j = \{1, 2, \ldots, N\}$. 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 (Fig. 2), it is observed that the classification error decreases as the number of trees increases. In the current data set, balanced data sets with 39 entities equally distributed among three classes were used. However, since an appropriate confidence threshold depends on the number of entities in each class of the data set and the number of trees in the forest, the authors believe that the accuracy will increase with an increase in the number of entities in each class by virtue of this criteria.

5.6 Margins and convergence in random forests

The *margin* in a Random Forest measures the extent to which the average number of votes at $\mathbf{X}, Y$ for the right class exceeds the average vote for any other class. A larger margin thus implies a greater accuracy in classification [37]. The generalization error in Random

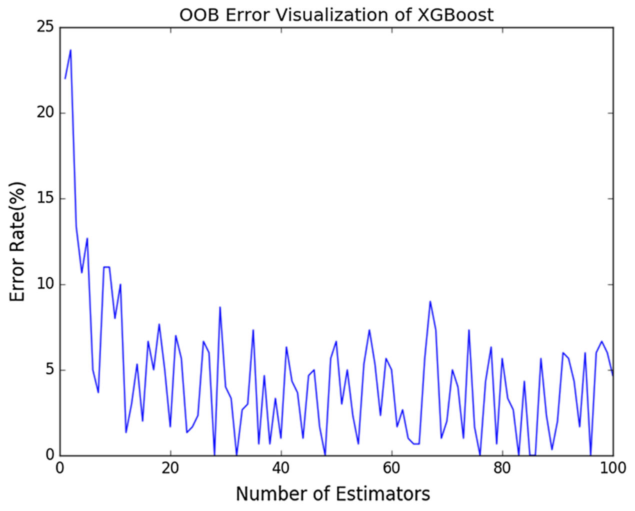


Fig. 2 OOB error rate as the number of trees increases (confidence split)

Forests converges almost surely, as the number of trees increases. A convergence in the generalization error is important. It shows that, as the number of tree classifiers increases, we tend towards a perfectly accurate classification.

Upper bound of error and Chebyshev inequality:

Measuring error bound for any classification method or approximation technique (these are used equivalently in machine learning) is important to ascertain robustness of classifiers. We ask the following question:
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 Eq. 16.

$$\text{Error} \leq \frac{\text{var}(\text{margin}_{RF}(x, y))}{s^2}. \tag{16}$$

5.7 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 [20] 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. 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 Sect. 5.8).

5.8 Working principles of XGBoost

XGBoosted Trees [20] may be understood by considering four central concepts.

5.8.1 Additive learning

For additive learning, functions f_i must be learned which contain the tree structure and leaf scores [20]. 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:

$$\begin{aligned}
\hat{y}_i^{(0)} &= 0\\
\hat{y}_i^{(1)} &= f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i)\\
\hat{y}_i^{(2)} &= f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i)\\
&\dots\\
\hat{y}_i^{(t)} &= \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i).
\end{aligned} \tag{17}$$

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:

$$\begin{aligned}
\text{obj}^{(t)} &= \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^{t} \Omega(f_i)\\
&= \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + \text{constant}.
\end{aligned} \tag{18}$$

Mean Squared Error 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.

$$\begin{aligned}\text{obj}^{(t)} &= \sum_{i=1}^{n}[l(y_i, \hat{y}_i^{(t-1)}) \\ &+ g_i f_t(x_i) + \frac{1}{2}h_i f_t^2(x_i)] + \Omega(f_t) + \text{constant},\end{aligned} \tag{19}$$

where g_i and h_i are defined as:

$$\begin{aligned}g_i &= \partial_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)}) \\ h_i &= \partial^2_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)}).\end{aligned} \tag{20}$$

By removing the lower order terms from Eq. 19, the objective function becomes:

$$\sum_{i=1}^{n}[g_i f_t(x_i) + \frac{1}{2}h_i f_t^2(x_i)] + \Omega(f_t), \tag{21}$$

which is the optimization equation of XGBoost.

5.8.2 Model complexity and regularized learning objective

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

$$f_t(x) = w_{q(x)}, w \in R^T, q : R^d \rightarrow \{1, 2, \ldots, T\}. \tag{22}$$

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:

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2. \tag{23}$$

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

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k). \tag{24}$$

5.8.3 Structure score

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

$$\begin{aligned}Obj^{(t)} &\approx \sum_{i=1}^{n}[g_i w_{q(x_i)} + \frac{1}{2}h_i w_{q(x_i)}^2] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \\ &= \sum_{j=1}^{T}[(\sum_{i\in I_j} g_i)w_j + \frac{1}{2}(\sum_{i\in I_j} h_i + \lambda)w_j^2] + \gamma T,\end{aligned} \tag{25}$$

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 Eq. 25, 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:

$$\text{obj}^{(t)} = \sum_{j=1}^{T}[G_j w_j + \frac{1}{2}(H_j + \lambda)w_j^2] + \gamma T. \tag{26}$$

In Eq. 26, 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:

$$\begin{aligned}w_j^* &= -\frac{G_j}{H_j + \lambda} \\ \text{obj}^* &= -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j + \lambda} + \gamma T.\end{aligned} \tag{27}$$

5.8.4 Learning structure score

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

$$\text{Gain} = \frac{1}{2}\left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda}\right] - \gamma. \tag{28}$$

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 γ, the corresponding branch should not be added.

5.9 Other classifiers

The other classifiers that we have tried are those of tree-based classifiers, probabilistic classifiers, SVM, and

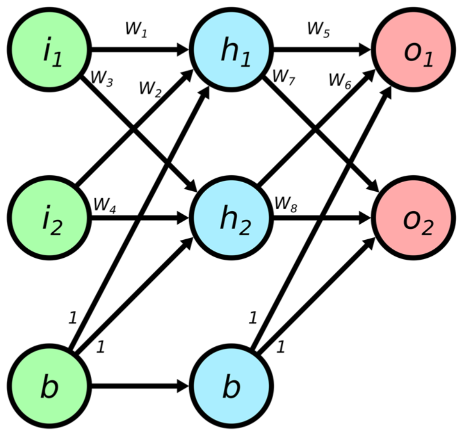


Fig. 3 The basic structure of a neural network. Here, the nodes are represented by the circles. The nodes labeled i are the input, h are hidden and o are output nodes

KNN. However, the restricted feature set used as a part of this work is too complicated for conventional machine learning methods to handle, so their results are presented in Table 1 with a brief discussion. The method that we focus on are that of neural networks.

6 Introduction to novel neural networks-based classifiers

Neural networks have always been a popular choice of classification methods since long time. [51]. These networks are made up of processing neurons that are connected to each other in layers. The connection across layers are termed as synapses that transmits signals across neurons. The connection strength between neurons is decided by weights assigned by the network. A network can be seen as a combination of layers of neurons arranged in input, hidden and output layers. Neurons at input layer are responsible for feeding the input to the network, hidden layers are responsible for propagating the signals and output layer conveys the output from the network. In the initial stages, since the weights are randomly chosen, output from the network does not match with the expected output. Hence, the network needs to be trained and this is achieved by stochastic gradient descent optimizer, [52] that incorporates computing error gradient at every layer and optimizing the weights in each node in such a way that the error gets minimized. An example of a neural network is shown in Figure 3. A detailed explanation of back propagation is explored in Appendix A.3.

As a part of the current work, we have used neural networks in three novel architectures, namely replicated weight neural networks (RWNN), generative adversarial networks and Fusion net, and these are explained in detail in Sects. 6.1, 6.2 and 6.3 respectively.

6.1 Introducing replicated weight neural network (RWNN)

An artificial neural network comprises of computing units arranged in layers of input, hidden and output neurons, as shown in Fig. 3. Input neuron sends signals to neurons of hidden layers that are internally equipped with a special function called activation function. The function appropriately squashes the signal and send it further to other layers with the help of connection synapses. Each synapse is associated with a weight. The signal moves from the input layer to hidden and lastly to the output and this constitutes a forward pass. The output received at the output layer is compared with the desired output and difference between the two is framed as unexpected error. Here, the network begins a backward pass, wherein the measured error is propagated back to the network in the form of an weight update, in order to facilitate error minimization during the subsequent passes. The process continues until the error is minimised and the optimized weights are preserved for testing new samples or instances. In this operation, the entire data set is divided into training and testing samples and the network is trained multiple times before being subjected to test new samples. The training set in turn, is divided into multiple small subsets and network picks up a smaller set at random and begins training its weights for minimizing error. Each time the network picks up a new set for training, it initializes connection weights with random values. Here, we propose a novel scheme, RWNN (Fig. 4) to train network in a more optimized manner. The conceptual idea behind RWNN is avoiding the random values for weight initialization and using the optimized weights from previous training set. The working of RWNN is as follows. We begin by assigning random weights for initialising the network. We divide training data into smaller subset. One random subset of training sample undergoes forward and backward pass, thus constitute a single epoch. After multiple epochs are run on the same subset, the error gets minimized due to the effect of gradient descent optimization, and weights acquired in current iteration (multiple epochs on one subset adds up to one iteration) are employed to train the next subset of training sample. Moving further, when the next subset of samples are picked up, the network uses the optimized weights which it receives from previous iterations and again runs multiple epochs before converging. The process continues until all the subsets are utilized for training the complete network. The idea behind using the optimized weights is that the network stabilizes at these weights giving minimum error for a subset of samples. It also indicates that the network configuration has inferred the features composition and would likely to converge faster if a similar subset of sample is applied as input in next cycle. All neural network use random initialization of weights. So as explained, instead of random re-initialization, RWNN makes use of optimized weights. In other words, the method attains convergence by avoiding re-initialization in every iter-

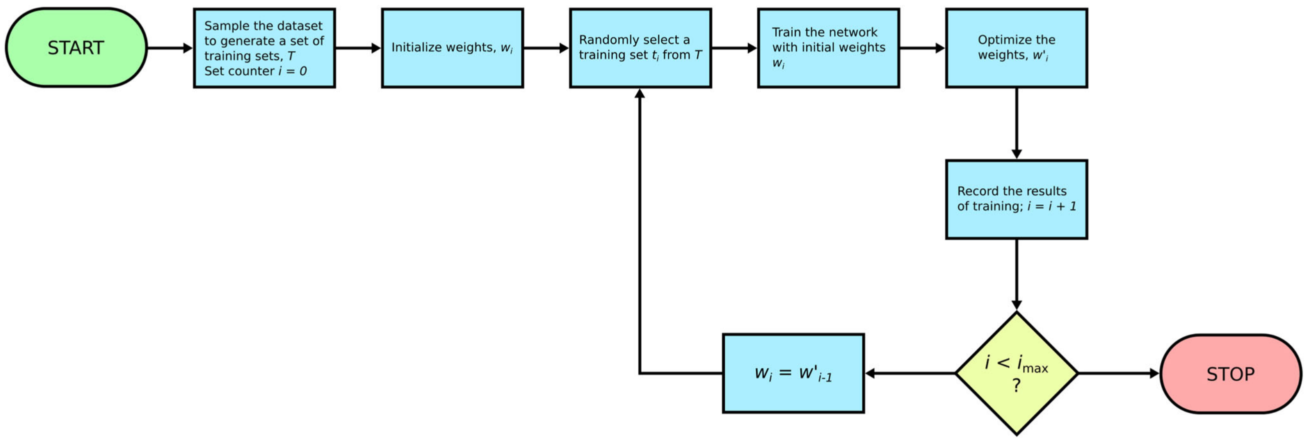
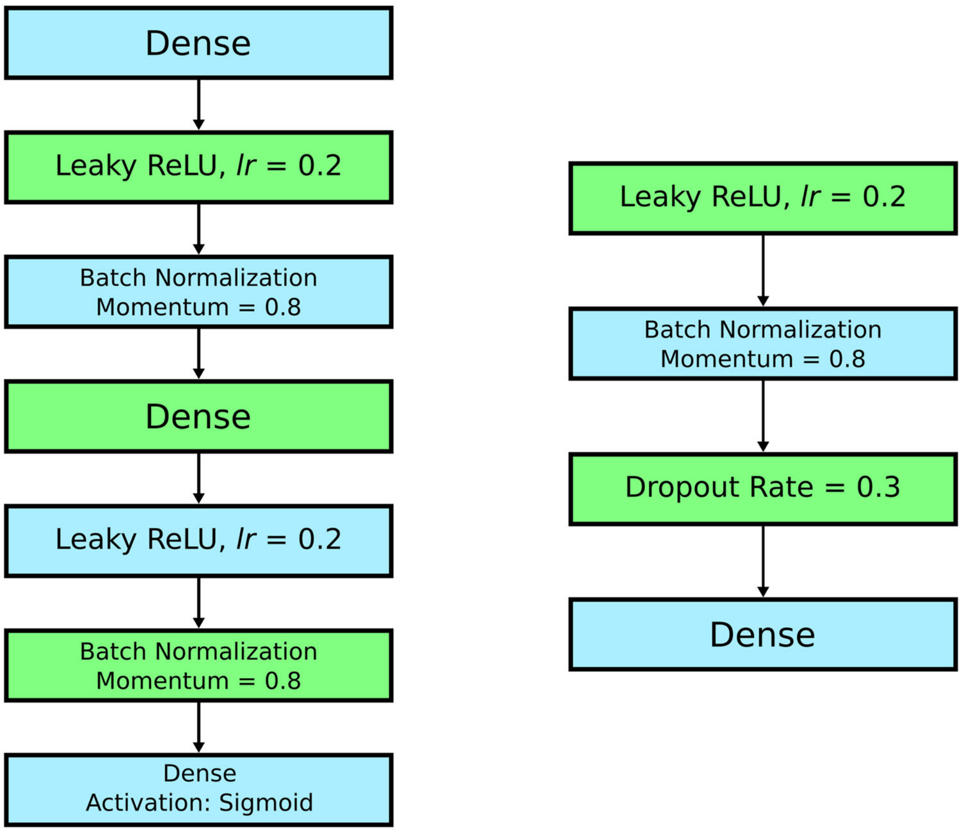

START
Sample the dataset to generate a set of training sets, T Set counter $i = 0$
Initialize weights, w_i
Randomly select a training set t_i from T
Train the network with initial weights w_i
Optimize the weights, w'_i
Record the results of training; $i = i + 1$
$i < i_{max}$?
STOP
$w_i = w'_{i-1}$

Fig. 4 Architecture pipeline for replicated weight neural network

Fig. 5 Layers of the generator and discriminator in our GAN model with batch-size of 128


Dense
Leaky ReLU, lr = 0.2
Batch Normalization Momentum = 0.8
Dense
Leaky ReLU, lr = 0.2
Batch Normalization Momentum = 0.8
Dense Activation: Sigmoid
Leaky ReLU, lr = 0.2
Batch Normalization Momentum = 0.8
Dropout Rate = 0.3
Dense

ation and setting a threshold on iterations rather than on error (MSE).

6.2 Introduction to generative adversarial networks

Generative adversarial networks (GANs) are frameworks comprising of two powerful components: the generator and the discriminator that gets trained by adversarial process. On one hand, a generator learns to generate synthetic data [53] or images with some component of noise included in samples, on the other hand, a discriminator is trained to differentiate synthetic data from the real one. GANs are popularly used for tasks of image generation and for obtaining high-resolution images from low-resolution ones. The concept of adversarial networks comes into play from the fact that the generator is trained to generate images to maximise the error in the contrary, the discriminator, on the other side, is trained to minimise the error.

Additionally, GANs can be used as semi-supervised learner where in, generator uses labeled data to generate unlabeled samples and discriminator on the other hand, learns the target function to classify the new set of points using data generated by generator. Traditionally, GANs are not used for text or numeric data, in this manuscript, we have tried using GANs for PHL-EC dataset. The reason behind choosing this approach is the conservative set of features described in Sect. 2, and one can see that features belonging to all three classes are cluttered. Moreover, if estimated surface temperature and associated attributes are removed, the demarcation between classes is completely lost which makes the classification of planets quite challenging.

6.3 A novel neural network pipeline using RWNN and GAN

A novel pipeline is developed using architectures proposed in previous section that consists of RWNN and GAN connected in series to achieve step-by-step classification of the aforementioned three categories. The pipeline works in two stages-

1. *Stage 1*—Stage 1 begins with performing classification of non-habitable, mesoplanets, and psychroplanets. Here, RWNN is deployed to distinguish the three classes of planets on PHL-EC dataset. Even though a high classification accuracy is visible, it is evident from the results that there is no clear demarcation between mesoplanets and psychroplanet.
2. *Stage2*—Classification of mesoplanet, and psychroplanet is accomplished at this stage. We begin by eliminating samples of non-habitable planets from PHL-EC data set. Next, the two classes of planets are fed to GAN, which internally uses a combination of generator and discriminator functions to first generate noise induced data samples for two classes and later, to perform classification on them.

The configuration of the GAN used in the current work is shown in Fig. 5, and the entire pipeline of the fusion network is shown in Fig. 7.

6.4 Training the GAN for categorical semi-supervised learning

Traditionally, GANs are used using discriminator functions as classifiers that determines whether a sample is from real data or from the data generated by generator. Alternatively, discriminator assigns class label y to each sample x using standard back propagation for training. The following objectives should be met in order to use GAN as categorical classifier:

1. Discriminator must return class assignments for all samples, instead of computing probability of every sample being real or generated.
2. Generator must make sure that every sample generated need to be of single class with high degree of certainty; Generator must produce synthetic samples in order to maintain balance between classes [54].

Both generator and discriminator in the experiment are implemented using Convolutional Neural networks and a soft-max layer is applied at the discriminator output so that the probability of a sample belonging to mutually exclusive k classes can be obtained. As already explained, generator produces samples by inducing random noise, which are then combined with real samples following which the new batch of samples is utilized to train the discriminator. The idea behind this is to necessitate the generator to recreate the original training data to improve the classification. The discriminator in turn gives feedback to the generator for improving the generation of samples, and eventually the discriminator works as a classifier that discriminates the real and newly generated data batch-wise. The discriminator identifies individual examples either as real or generated ones, and use the other examples in the minibatch to enhance the coordination between gradients [55]. The layered architecture of GAN is shown in Fig. 5.

In this manuscript, we not only implemented the fusion-net architecture on the data, but also tried GANs as classifiers in a separate independent experiment. We first merged samples of mesoplanet and psychroplanet together (and called them habitable), and later used GAN to classify planets as habitable and non-habitable. To describe the big picture of our manuscript, we developed RWNN architecture, the fusion-net architecture and GAN which constitute Phase I of experiments. Phase II begins by investigating the limitations of popular activation functions during back-propagation and explores the usage of SBAF for classification of different feature sets of PHL-EC data. The outcome of this phase is particularly interesting in the context of the problem.

7 Neural network architectures, experimentation conducted in two phases

7.1 Phase I: Exploration of RWNN, GAN and Fusion net

Before we begin this section, we want to mention that, in the experiments where we have shown the classification between habitable and non-habitable classes, we have considered the non-habitable class to be class 1, and the habitable class to be class 0; and in the experiments where we have tabulated the results of classification between the psychroplanet and mesoplanet classes, we have taken psychroplanet as class 0 and mesoplanet as class 1.

7.1.1 RWNN

As a first step in the endeavor to explore how well network for relevant classification tasks in the exoplanets domain, neural networks were trained without surface temperature. This network consisted of 44 neurons in input layer, 11 in hidden and 3 in output layer and is trained against 44 featured dataset. The resulting accuracy achieved after 400 epochs and at learning rate of

Table 2 Result of Neural network classifier shows that classification with sufficiently good accuracy is achieved without estimated surface temperature being used as parameter

Class	Accuracy %	Precision	Recall	Sensitivity	Specificity	Fscore
Non-habitable	99.8	0.99	1.0	1.0	0.99	0.99
Mesoplanet	99.8	0.997	1.0	1.0	0.998	0.99
Psychroplanet	99.7	1.0	0.994	0.994	1.0	0.997

Table 3 Results: The increase in accuracy was observed till learning rate reached 0.07

S. no.	Iter	L Rate	Accuracy of vanilla neural network%			Accuracy of RWN%		
			Non-habitable	Mesoplanet	Psychroplanet	Non-habitable	Mesoplanet	Psychroplanet
1	10	0.01	95.1	93.1	95.0	96.2	95.3	95.0
2	10	0.02	96.5	95.7	95.8	96.2	96.6	95.7
3	10	0.03	97.1	96.2	96.1	97.4	97.9	97.5
4	10	0.04	97.9	96.3	96.7	97.6	98.5	97.9
5	10	0.05	98.1	96.8	97.0	97.9	98.6	98.4
6	10	0.06	98.5	95.1	96.5	97.9	98.6	98.6
7	10	0.07	98.9	96.3	95.1	98.1	99.2	99.1
8	20	0.07	98.9	96.9	95.7	99.2	99.2	99.1
9	25	0.07	98.9	95.6	95.7	99.3	99.2	99.4
10	30	0.07	98.9	95.4	94.4	99.3	99.6	99.6

Later, the number of iterations were increased keeping learning rate constant. The accuracy (for meso and psychro) achieved using the vanilla neural network suggests that the dataset has not been efficiently utilized. As a consequence method, two were executed

Table 4 Classification result of potentially habitable planets where the sample wise accuracy of almost all planets is 100%, except trappist -1 b and trappist -1 c (93%). Trappist -1 d was always classified as a psychroplanet

Planet	Habitable class	Accuracy (%)
Proxima Centuri B	Psychroplanet	100
TRAPPIST-1 b	Non-habitable	93
TRAPPIST-1 c	Non-habitable	93
TRAPPIST-1 d	Mesoplanet	0
TRAPPIST-1 e	Psychroplanet	100
TRAPPIST-1 f	Psychroplanet	100
TRAPPIST-1 h	Non-habitable	100

0.19 was found to be encouraging as reported in Table 2.

After a thorough scrutiny of the usefulness of this exercise, more parameters are removed from the dataset that are found to be related to estimated surface temperature. The updated dataset then added-up to nine features which were P. Min Mass (EU), P. Mass (EU), P. Radius (EU), P. Esc Vel (EU), P. Mean Distance (AU), S. Mass (SU), S. Radius (SU), S. Teff (K) and S. Luminosity (SU). When the dataset with these features is fed for classification on network made up of nine neurons in input, five in hidden and three in output layers, classification accuracy dropped down and it looked challenging to train neural network to use these parameters and categorize the three planets into their respective classes. A skewed distribution of samples in the dataset was handled by bootstrap aggregation, wherein equal samples from three class are randomly drawn up and later used to train the network. It is at this point that the RWNN architecture was tried. We experimented with this set of features as a precursor to the experimental setup described in Sect. 2. The accuracy of these two approaches in the training iterations are shown in Table 3.

Once the accuracies for the entire dataset were computed, we wanted to see how RWNN performed while classifying specific samples in the data. The results of this are shown in Table 4. For this task, the neural network was run for 45 iterations at learning rate of 0.009 at 1000 epochs. The mean square error plotted against the epochs are shown in Fig. 6.

Following this, we developed the idea for the experiments that are presented in the main part of this paper. Since the results are generally good, we do not repeat the exercise of classifying specific samples. Finally, the algorithm for the RWNN neural network is described in Algorithm 1.

7.1.2 GAN

This section explores the results of GAN that solved the 2 class classification of habitable and non-habitable planets. Simultaneously, a synthetic data set was built using Parzen Window Estimation that generate samples for mesoplanets and psychroplanets. The result of using GAN for discriminating habitable and non-habitable classes using original and synthetic data cre-

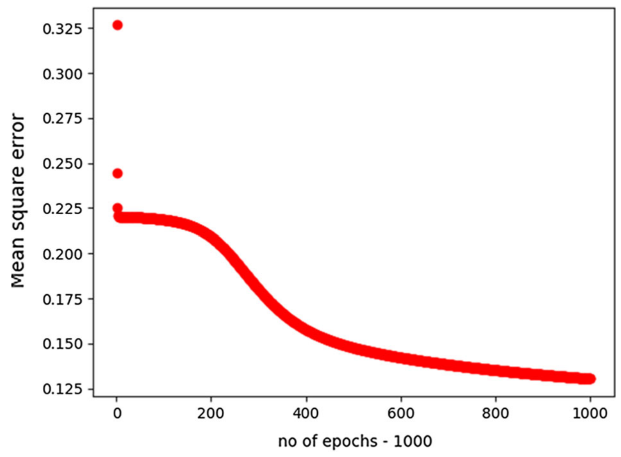


Fig. 6 MSE plot captured during one of the 45 iterations, each iteration was run for 1000 epochs. After 800 epochs, the MSE was seen to be stabilized

Algorithm 1 Replicated Weights Neural Network

```
1: Input: Dataset, D    Output: Accuracy, precision, recall
2: Initiate sampling process to generate training sets (T)
3: $T \leftarrow D$
4: procedure INITIALIZATION
5:     Initialize weights, iteration counter i, select training set $t_i$ from T
6:     $w_i \leftarrow randomnumber$
7:     $t_i \leftarrow T$
8:     $i \leftarrow 0$
9: end procedure
10: procedure TRAINING
11:     Use $t_i$ as training set, $w_i$ as initialized weights,
12:     Train the network using back propagation
13:     Let $w_i'$ be the optimized new weights
14:     Return $w_i'$
15: end procedure
16: procedure REPLICATION
17:     Record Accuracy, precision, recall of the current iteration, i
18:     $i \leftarrow i + 1$
19:     $w_i \leftarrow w_i'$
20:     Select another training set $t_i$ from T
21:     $w_i' \leftarrow Training(t_i, w_i)$
22:     $IF \quad i < I_{max} \quad GOTO \quad Replication$
23:     Compute average accuracy, precision and recall by dividing their sum by $I_{max}$
24: end procedure
```

ated by Parzen window estimation are shown in Tables 5 and 6 respectively.

7.1.3 Fusion-net approach

In this experiment, the RWNN approach was able to classify the non-habitable samples with an accuracy of $> 96\%$, with values of precision and recall as 1 (in a one-to-many class comparison). The results of GAN as a classifier are shown in Table 7. Since the mesoplanet and psychroplanet classes are hugely dominated by the non-habitable class, they are naively upsampled here by copying the samples in the classes by 40 times. Another execution of proposed fusion net approach (Fig. 7) involves samples after upsampling by Parzen window estimation shown in Table 8. The plot of the root mean square error of GANs for this task is shown in Fig. 9.

7.2 Phase II: A deeper analysis

The motivation for further exploration in to deep neural networks [56] is driven by an urge to improve the performance metrics of the classifiers discussed above. The focus is based on pruned feature sets described in Sect. 7.2.2 and the classifiers where both RWNN and GAN failed to produce reasonably accurate performance metrics for the three-class classification problem. We removed all features related to estimated surface temperature (in contrast to results described where only surface temperature is removed). Once this is done, the problem is more meaningful and challenging since the attempt is to predict thermal classification labels without using attributes related to thermal characteristics of the exoplanets. We accomplish this by proposing an activation function and discussing its merit over the standard one (sigmoid) used more frequently. As

Table 5 Results of GAN used for classifying habitable and non-habitable samples from the original dataset

Planet attributes	Accuracy	F-Score	Precision	Recall
Radius	85.4	85.8	85.4	86
Mass	85.0	85.0	85.0	85.0

Table 6 Results of GAN used for classifying habitable and non-habitable samples from the original dataset after upsampling by Parzen window estimation

Planet attributes	Accuracy	F-Score	Precision	Recall
Radius	92.9	95.07	95.14	86.8
Mass	88.5	88.18	89.15	87.23
Min mass and P. Distance	97.52	98.08	99.17	97.02

Table 7 Results of fusion net approach used for classifying mesoplanet and psychroplanet samples

Planet attributes	Accuracy	*F*-Score	Precision	Recall
Radius	77.8	82.6	78.8	86.8
Mass	79	79	73	89
Min mass and P. Distance	100	100	100	100

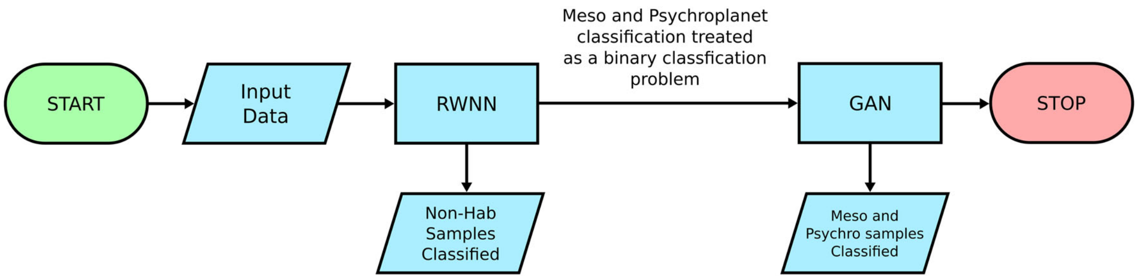

Meso and Psychroplanet classification treated as a binary classfication problem
START
Input Data
RWNN
GAN
STOP
Non-Hab Samples Classified
Meso and Psychro samples Classified

Fig. 7 Fusion neural-net approach for classification: First tier of the network classifies between habitable and non-habitable planets; second tier architecture uses GAN to discriminate among mesoplanets and psychroplanets (classes within the habitable class): the fused network produces perfect classification when Min Mass, P.distance and star properties are used as features

Table 8 Results of fusion net approach used for classifying mesoplanet and psychroplanet samples after upsampling by Parzen window estimation

Planet attributes	Accuracy	*F*-Score	Precision	Recall
Radius	78.5	80.00	77.61	82.5
Mass	86.03	86.52	84.66	88.46
Min mass and P. Distance	97.07	96.90	100.0	94.0

Table 9 Performance analysis of two Activation functions used in the study

Different feature sets	Performance	Different activation functions used in the study					
		Sigmoid			SBAF		
		Non-H	Meso	Psychro	Non-H	Meso	Psychro
	Accuracy	1.	0.975	0.975	1.0	0.99	0.994
All features (45)	Precision	1.0	0.943	0.932	1.0	1.0	0.2
(Case 1)	Recall	1.0	0.895	0.965	1.0	0.55	1.0
Restricted	Accuracy	0.758	0.741	0.798	0.987	0.854	0.847
Features (6)	Precision	0.719	0.525	0.808	1.	0.681	0.643
(Case 2)	Recall	0.864	0.583	0.384	0.978	0.223	0.943
Without surface	Accuracy	0.998	0.998	0.997	0.997	0.924	0.927
Temperature (44)	Precision	0.994	0.995	1.	1.	0.484	0.783
(Case 3)	Recall	1.	1.	0.994	0.996	0.5	0.783
Planet MinMass	Accuracy	0.888	0.825	0.750	0.922	0.922	0.864
Mass and	Precision	0.974	0.534	0.547	1.	0.3	0.456
Radius (4) (Case 4)	Recall	0.807	0.397	0.814	0.904	0.130	0.912

It is worth noting that, SBAF did not need parameter tuning, at all. Also, when these feature sets (case 2 onward) were fed to previously defined architectures, RWNN and GAN, the accuracy was between 40 and 65%

shown by Saha et al. [57], sigmoid activation has a few limitations which impede its ability to reduce error in backpropagation some times. This includes not having a critical point and, therefore, being handicapped with the absence of global optima. Presence of a point of inflection augments this problem further and this leads to "flat" MSE over iterations. We mitigate this problem by proposing SBAF [58]. SBAF, defined with parameters k, α (estimated and not tuned while training) and input x (the input), produces output y as follows:

$$y = \frac{1}{1 + kx^{\alpha}(1-x)^{1-\alpha}},$$

where $0 \leq \alpha \leq 1, k > 0$. It can be shown that SBAF has a critical point and its second derivative is positive implying SBAF is endowed with global minima. This helps in reducing MSE with a sharper downward slope as compared with sigmoid. The immediate impact of this property is reflected in much better accuracy metrics when the classification problem is faced with the difficulty of handling pruned feature sets with the obvious hard marker, 'estimated surface temperature and all attributes related to it' are removed. We present several cases of pruned data sets with reduced features below and the results of classification are shown in the subsequent section.

The functional form of the activation function is usually fixed, ignoring the possibility that different forms of activation functions of neurons could emerge as nonlinear, periodic solutions arising from second-order, linear ordinary differential equations (ODE) with coefficients (parameters). The chosen model is often used to describe Mechanical/Electrical systems. The model is represented as $ay''(x) + by'(x) + cy(x) = u(x)$, where $u(x) = 0, x \leq 0, 1$ otherwise. It is interesting to note that when $a = 0, b = 0, c = 1$, the solution is $y(x) = \sigma(x)$ and when $a = 0, b = 1, c = 0$, the solution is $y(x) = xu(x)$ under zero bias. Let us migrate to the more general case and consider the solutions to the following:

$$ay''(x) + by'(x) + cy(x) = 0; \quad y(0) = 0, \quad y'(1) = \eta.$$

Observation I—Keeping $c = 0$ does not change the form of the solutions obtained as above i.e

$ay''(x) + by'(x) = 0; y(0) = 0, y'(1) = \eta$ will yield similar solutions. **Observation II**—Suitable choice of parameters $(k, k_1$ etc) shall yield different activation functions. Set $a = x(1-x), b = 0, c = y - 1$, we recover SBAF

$$y = \frac{1}{1 + k * x^{\alpha} * (1-x)^{(1-\alpha)}}.$$

7.2.1 Relevant data: all cases

To evaluate the performance of SBAF we began by solving classification problem on a 45-feature data set (original data). Furthermore, we hand-picked features from original set and generated eight different data sets for execution on SBAF (shown as case 1–case 8 in the tables). For each of these cases, 80% of samples were used for training set and 20% for testing. Eliminating the need for parameters tuning, SBAF parameters were computed from the fixed point plots taken from [57] and were set to $\alpha = 0.5$ and $k = 0.91$. The results of SBAF are compared with sigmoid and presented in Tables 9 and 10. Accuracy, precision and recall for both the functions and for all the eight data sets is reported in these tables. The python code is available at github repository.[3] As observed, SBAF is able to classify better than sigmoid for all the cases mentioned above. Better performance metrics adds to the confidence of classifier in handling the problem with pruned feature sets given that we are attempting the difficult task of predicting thermal habitability labels without using any attributes related to thermal features. Note, for the sake of completion, we have included all cases. We begin by the all inclusive feature set (estimated surface temperature, etc.) and gradually reduce the features case-wise. This renders the progression/variation in performance quite succinctly.

7.2.2 Chosen feature sets

As mentioned previously, a case by case investigation of feature sets build for the sake of performance comparison is presented here. Not just the planetary features, other crucial parameters related to the architecture of neural networks like number of hidden neurons, learning rate, momentum, epochs are also reported for fair evaluation of both activation functions. Performance matrices used for the correspondence of two functions includes class-wise accuracy, precision and recall and are reported in table 9 and 10. We have also compared the two functions with regard to execution time, CPU utilization and memory usage at specific learning rate, momentum and epochs (Table 11).

1. (Case 1) Initially, all 45 features are utilized as input to neural network thus accounting for 45 neurons in input and 3 in output layer. Inherently, each activation function may behave in a unique way, thus leading to believe that for same data set applied as input, each may use different number of neurons in hidden layer for reaching convergence. With Sigmoid, hidden neurons used were 20 and with SBAF, 11 hidden neurons were put in. The best accuracy witnessed for sigmoid was at 500 epochs whereas SBAF converged at 100 epochs, while other parameters like learning rate and momentum were kept same (learning rate—0.015 and momentum—0.001).
2. (Case 2) Next, we choose a feature set (which we call restricted features) that comprises of Planet Minimum Mass, Mass, Radius, SFlux Minimum, SFlux Mean, SFlux Maximum. These features are applied to network that used 4 hidden neurons and we observed that Sigmoid converged at learning rate of 0.1 and momentum of 0.01. Likewise, SBAF ran till 300 epochs, at learning rate of 0.08, with momentum at 0.004. Table 9 reveals that the network could classify even the minority classes (Mesoplanets and Psychroplanets) with fairly decent accuracy.
3. (Case 3) Estimated Surface Temperatures (ST) is a commonly used planetary feature to differentiate habitable and non-habitable planets. Hence, the next feature set excludes ST from the set of 45

[3] https://github.com/mathurarchana77/A-RELUandSBAF.

Table 10 Performance comparison for cases 4–8

Different feature sets	Performance	Different activation functions used in the study					
		Sigmoid			SBAF		
		Non-H	Meso	Psychro	Non-H	Meso	Psychro
Radius and other star features (Case 5)	Accuracy	0.848	0.753	0.608	0.721	0.738	0.754
	Precision	0.856	0.668	0.440	0.617	0.669	0.152
	Recall	0.655	0.515	0.645	0.962	0.738	0.37
Planet mass and other star features (Case 6)	Accuracy	0.846	0.706	0.58	0.859	0.868	0.881
	Precision	0.764	0.55	0.337	0.914	0.442	0.307
	Recall	0.78	0.65	0.27	0.906	0.628	0.177
Minimum mass and other star features (Case 7)	Accuracy	0.85	0.683	0.7	0.850	0.8007	0.857
	Precision	0.711	0.75	0.532	1.0	0.190	0.520
	Recall	0.925	0.07	0.85	0.798	0.2666	0.8837
Minimum mass P. Distance and star features (Case 8)	Accuracy	0.958	0.8	0.808	0.858	0.77	0.716
	Precision	1.	0.785	0.649	0.911	0.35	0.38
	Recall	0.875	0.55	0.925	0.846	0.106	0.746

An anomaly is seen for case 8 where sigmoid performed marginally better than SBAF. It is important to note, that these feature sets (case 2 onward) when fed to different architectures, RWNN and GAN, the accuracy was between 40 and 65%

features, essentially to ascertain that, even though ST is removed, two functions can perform classification. The hidden neurons are 12, learning rate—0.01, momentum—0.001 and epochs—300 for both the functions. Hence, input features are Zone Class, Mass Class,Composition Class, Atmosphere Class, Min Mass, Mass, Radius, Density, Gravity, Esc Vel ,SFlux Min, SFlux Mean, SFlux Max, Teq Min (K) , Teq Mean (K),Teq Max (K), Surf Press, Mag, Appar Size, Period, Sem Major Axis, Eccentricity, Mean Distance , Inclination, Omega , HZD, HZC, HZA, HZI, ESI and Habitable. The features of parent Star that belong to feature set are Mass, Radius, Teff, Luminosity, Age, Appar Mag, Mag from Planet, Size from Planet, Hab Zone Min and Hab Zone Max. For SBAF, the 44 featured data set is tuned at learning rate = 0.01, momentum = 0.001 and epochs = 300. The number of units in the hidden layer is 12 for the 2 activation functions. Description of these features are available at Table 12 in Appendix. Here, sigmoid performs marginally better than SBAF.

4. (Case 4) Another attribute set comprising of Minimum Mass, Mass, Radius and Composition class are used for experimentation, purely to validate whether the two functions can classify using planetary mass and mass-related features. This could be challenging since planetary categorization is never being done using its mass. The network uses 3 neurons and works fairly well at learning rate-0.2 and momentum −0.3 converging at 400 epochs.
5. (Case 5) Furthermore, a unique combination of features from parent star are used along with planetary features to investigate whether this kind of blending can help in improving classification accuracy. Case 5 uses 6 star features (Mass, Radius, Teff, Luminosity, Hab Zone Min and Hab Zone Max) along with planet's radius. The network uses four hidden neurons, and performs classification at the learning rate of 0.09 and momentum of 0.001. Results are achieved at 300 epochs for both the activation functions.
6. (Case 6) Another combination involves features of parent's star with planet's mass are used as input features to the network comprising of four hidden neurons. Learning rate, momentum and epochs are kept same as Case 5.
7. (Case 7) This time planet's minimum mass is used as planetary feature along with features of parent star. Other parameters remains the same.
8. (Case 8) Two planet features—mass and minimum distance computed from parent star—are used as planetary features along with the same set of features of parent star. It is evident from the table that sigmoid performs better classification than SBAF.

7.2.3 Results of experiments phase II

The performance metrics shown in Tables 9 and 10 shows SBAF is more responsive in terms of classifying the three categories of planets corresponding to different feature sets. The comparison is also performed using few more critical parameters like CPU utilization training time, and memory requirements which is reported in Table 11. Comparison shows SBAF converges in shorter

Table 11 Comparison of training time, CPU utilization and Memory usage for Sigmoid and SBAF at epochs = 500. The reported values are averaged over all eight executions mentioned above. The SBAF has the lowest training time followed than Sigmoid. SBAF has marginally better CPU utilization from the parent activation function sigmoid

	Activation functions (Epochs =500)	
	Sigmoid	SBAF
Learning rate, momentum	0.01, 0.001	0.01, 0.01
Time (s)	409.33	281.97
CPU utilization (%)	54.6	53.6
Memory usage (MB)	67.7	67.8

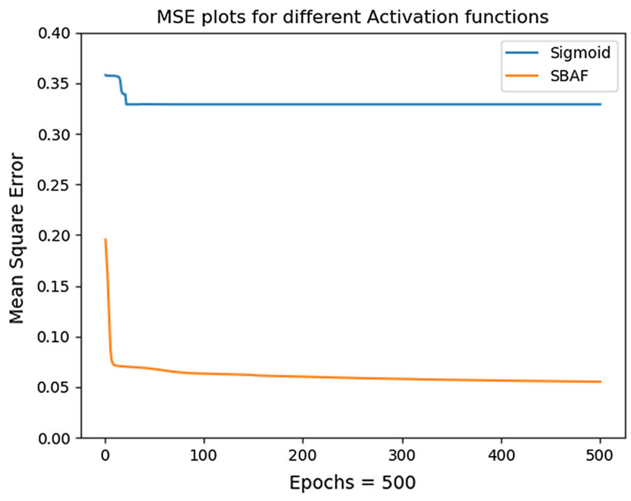


Fig. 8 MSE plots for Sigmoid and SBAF executed for 500 epochs on Restricted features of PHL-EC data: Sigmoid attains saturation at an early MSE value and fails to learn effectively when compared with SBAF. This fact may be interpreted as the representational power of the activation function (since identical network architecture was used). If the activation function is more expressive, it will, in general, get a lower MSE when trained sufficiently long enough. Therefore, it is not surprising that SBAF has the lower MSE (and thus higher representational power), since its shape can easily be changed. Even though SBAF has more parameters compared to sigmoid, its faster convergence is striking

time, with minimum CPU utilization for same number of epochs. However ,the memory utilization is at par for both the functions. MSE plots of the two functions are shown in Fig. 8

If we compare the performance of some of the statistical machine learning algorithm such as Gaussian Naive Bayes, Linear Discriminant Analysis, SVM, RBF-SVM, KNN, DT, RF and GBDT [59] with that of the proposed activation function, we could clearly see that none of these methods come close to 75% accuracy class wise, even after removing the "hard marker" features (i.e. for the Cases 2, 4, 5–8, "six" out of the total "eight" cases considered for our experiments) from feature set.

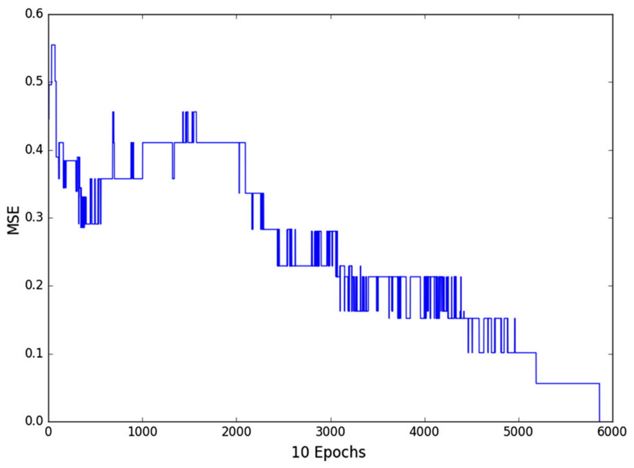


Fig. 9 Perfect separation between two classes of habitable planets depicted by zero mean-squared error

8 Discussion

8.1 The need for training classifiers on balanced datasets

In the PHL-EC dataset, the non-habitable category is over 1000 times as large in terms of the number of samples compared to the mesoplanet and psychroplanet classes. Upon inspection of the confusion matrices of the classification done using all the features and the datasets without balancing, we can see that the results are biased and almost every sample will be classified as a non-habitable sample. Hence, in our work, we use a combination of upsampling methods for the smaller classes and downsampling method for the non-habitable class to improve the accuracy of classifiers.

8.2 A recapitulation

In Tables 2 to 11, we have presented the results of all the classification runs. We see that the accuracy of the classifiers are reasonably high for some cases. In general, the results of classification of samples into the habitable and non-habitable classes renders better accuracies. This is understandable as the physical properties of any exoplanet are determined by many factors put together, and the conjunction of the mesoplanet and psychroplanet classes into the habitable class results in the properties of of the combined class having a larger range for each feature value. Additionally, the non-habitable class is a lot more distributed than the mesoplanet and the psychroplanet classes along each feature. Thus, their combination makes the distribution of values along each feature a little more broad and thus allows the classifiers to generalize better.

The results achieved by the RWNN and GAN architectures for classifying between the non-habitable, mesoplanet, and psychroplanet classes results good accuracies but when the Min. Mass and P. Distance features are used, and reasonable accuracies with the radius, and the mass features—the accuracies are further improved when the synthetic oversampled data from Parzen window estimation is used. This implies that a careful balancing of samples can improve the results of classifiers.

9 Conclusion

This paper presents the effectiveness of using machine learning to explore the problem of habitability classification of exoplanets [18]. The novelty of the work lies in the appropriate exploration of the PHL-EC dataset and usage of automated classification methods that was hitherto not investigated in the existing literature. The work is a detailed investigation on exploratory data analysis involving algorithmic improvisations and machine learning methods applied to the PHL-EC dataset, bolstered by a comprehensive understanding of these methods (as documented in the appendices). The inferences drawn fortify the usage of these methods.

The dataset is unique and is a combination of both derived and fundamental planetary parameters. The derived features provide a rich representation of the exoplanets' characteristics, which is useful at this point in time. As more exoplanets are discovered and as more follow-up missions observe different characteristics of exoplanets, we will eventually come to have a lot of real data—the methods that we have described can then be trained yet better. The accuracy of neural networks used on the PHL-EC dataset have been computed and tabulated, and this has been done for the three-class problem of separating non-habitable, meso and psychroplanets, as well as the two-class problem of separating non-habitable and habitable planets. Despite the sample bias due to the non-habitable class, we were able to achieve remarkable accuracies with classification algorithms by performing undersampling and synthetic data augmentation on the data. This goes to show that a careful study of the nature and trends of the data is a must.

Exoplanets are frequently discovered and categorizing them manually is an arduous task. As data are collected from different missions, gradually adding to the volume of existing data, automatic annotation methods, along with viable strategies for discovery [60], would eventually provide an advantage in terms of the required processing time and effort. Thus, in the 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.

As stated in the introduction section, the Earth is considered an anomaly in the family of planets. Anomaly detection is a relevant and useful technique used in Industrial data sets, employee login data, etc. [61] . Anomaly detection is best performed using unsupervised clustering methods when the reliability of training data is not certain. Since the Earth is conjectured as an anomaly, there is a scope to apply such methods and analyze cluster base anomalies formed out of the Exoplanet data base, PHL-EC. This means we do not have to consider training labels of PHL-EC. This promises to be an interesting future study where clusters of planets formed via anomaly detection method could be cross-matched with the class labels present in the PHL-EC. The set of meso and pychroplanets could be considered as anomalous as the number of such planets is considerably less in comparison with the non-habitable ones.

We end this section with a note on the correct usage of ML and artificial intelligence in habitability classification of exoplanets. We do not intend to inspire any notions of automata taking over subtle and sensitive tasks such as discovering habitable worlds in the universe. Rather, we emphasize that the current work should be used to facilitate and reinforce the process of discovery and the inferences drawn be used to bolster our knowledge in the areas of exoplanet discovery.

Acknowledgements 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.

Data Availability Statement This manuscript has associated data in a data repository. [Authors' comment: The data set used during the conduction of experiments is available at https://github.com/mathurarchana77/Habitability-PHL-EC-Dataset.]

Appendix A

Effects of imbalance in the dataset

In Table 1, we have presented the results of classification done using conventional machine learning classifiers. These were trained on the features of mass and radius, as well as on the entire feature set (that includes estimated surface temperature). However, upon considering the experimental setting described in Sect. 2, their performance drops. Even while considering all these feature sets, we see that all the classifiers do not always perform well. In this setting, we use both mass as well as radius, however, in Sect. 7.1, we have achieved yet better results without their conjunction. The underlying takeaway is that when more meaningful features are cascaded, even within neural networks, the performance generally improves; and if a classifier can perform better with a restricted feature set than a method that takes an extensive set of features, then they may perform yet better when considering an extensive features as well, but the features need to selected carefully and must have relevance.

Details of data augmentation methods

In Sect. 4.2.1, a technique to estimate the probability density function of a sequence of random variables, called KDE (using Parzen-window estimation), was briefly described. We elaborate on the method and validate the method by performing a density estimation on random numbers generated from several known analytic continuous and discrete distributions. As a part of this exercise, we have presented goodness-of-fit scores for the estimated distributions. In addition to that, we have also plotted the graphs of the data points for a visualization of the density (generated using the standard analytic distributions as well as a Parzen-window estimate of those distributions).

We provide evidence of the working of Parzen-window estimation for different standard continuous and discrete probability distributions.

An analytical exposition of density exploration

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:

$$\phi(u) = \begin{cases} 1 & u_j \leq \frac{1}{2} \quad \forall \quad j \in \{1, 2, \ldots, d\} \\ 0 & \text{otherwise.} \end{cases} \tag{29}$$

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

$$\begin{aligned} l_j &= \min\{(a-b)^2 \quad \forall\ a, b \in f_j\} \\ u_j &= \max\{(a-b)^2 \quad \forall\ a, b \in f_j\}, \end{aligned} \tag{30}$$

the edge length h_j is given by,

$$h_j = c\left(\frac{u_j + 2l_j}{3}\right), \tag{31}$$

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:

$$u_j = \frac{x_j{}' - x_{ij}}{h_j} \qquad \forall \quad j \in \{1, 2, \ldots, d\} \tag{32}$$

The density estimate is then given by the following relationship:

$$p(x') = \frac{1}{n \prod_{i=1}^{d} h_i} \sum_{i=1}^{n} \phi(u). \tag{33}$$

Generating synthetic samples from the estimated empirical distribution

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 Eq. (33), the probability is approximated to:

$$Pr(r) = p(r) \prod_{j=1}^{d} h_j. \tag{34}$$

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 Eq. (34), then the sample is accepted. Otherwise, it is rejected.

For the PHL-EC dataset, synthetic data was generated for the mesoplanet and psychroplanet classes using this method by taking $c = 4$ for mesoplanets and $c = 3$ for psychroplanets (in Eq. (33)). 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 85% of the features by importance were considered to estimate the probability density; the values of the remaining features were copied from the naturally occurring data points and shuffled between the artificially augmented data points. 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 a little more complex than assuming a standard probability distribution in the data. Nonetheless, this is an inherently unassuming method and can accommodate distributions in data which are otherwise difficult to describe using the commonly used methods for describing the density of data.

Neural networks

Weight update rule used back-propagation in Neural Networks

The basic structure of the neural network consists of input layer, hidden layer and output layer. Let us assume the nodes at input layer are i_1, i_2 , at hidden layer h_1, h_2 and at output layer o_1, o_2 . The goal is to optimize the weights so that the network can learn how to map from inputs to outputs. Considering Figure 10. as the network structure, the forward pass involves computation of the following parameters.
Calculate the total input for h_1.

$$\begin{aligned} h_{1_{net}} &= w_1 \cdot i_1 + w_2 \cdot i_2 + b_1 \\ h_{2_{net}} &= w_3 \cdot i_1 + w_4 \cdot i_2 + b_1 \end{aligned}. \tag{35}$$

Use sigmoid activation function to calculate the output for h_1,

$$\begin{aligned} y &= \frac{1}{1+e^{-x}} \\ h_{1_{\text{out}}} &= \frac{1}{1+e^{-h_{1_{\text{net}}}}}. \\ h_{2_{\text{out}}} &= \frac{1}{1+e^{-h_{2_{\text{net}}}}} \end{aligned} \tag{36}$$

This process is repeated for output layer neurons and compute the values of $o_{1_{\text{net}}}$, $o_{2_{\text{net}}}$, $o_{1_{\text{out}}}$ and $o_{2_{\text{out}}}$.

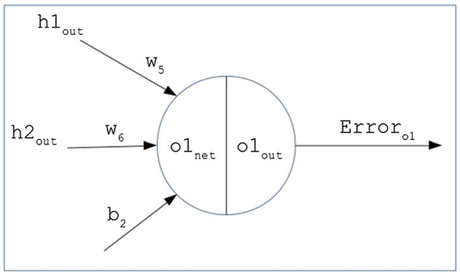


Fig. 10 Basic structure

Calculating the errors,

$$\begin{aligned} E &= \mathrm{E}_{o1} + \mathrm{E}_{o2} \\ E_{o_1} &= \frac{1}{2}\left(o1_{\text{target}} - o1_{\text{out}}\right)^2 \\ E_{o_2} &= \frac{1}{2}\left(o2_{\text{target}} - o2_{\text{out}}\right)^2 \end{aligned} \tag{37}$$

In the backward pass, the gradients are computed and weights are updated, so that the actual output is closer to the target, thus the error is minimized. In the output layer, consider w_5 to be the weight that needs to be updated. The error gradient can be computed as:

$$\frac{\partial E_T}{\partial w_5} = \frac{\partial E_T}{\partial o1_{\text{out}}} \cdot \frac{\partial o1_{\text{out}}}{\partial o1_{\text{net}}} \cdot \frac{\partial o1_{net}}{\partial w_5} \tag{38}$$

Each component in the RHS is computed individually:

$$\begin{aligned} E_T &= E_{o_1} + E_{o_2} \\ E_T &= \frac{1}{2}\left(o1_{\text{target}} - o1_{\text{out}}\right)^2 + \frac{1}{2}\left(o2_{\text{target}} - o2_{\text{out}}\right)^2 \\ \frac{\partial E}{\partial o1_{\text{out}}} &= 2 \cdot \frac{1}{2}\left(o1_{target} - o1_{\text{out}}\right) \cdot (-1) + 0 \\ \frac{\partial E}{\partial o1_{\text{out}}} &= -\left(o1_{\text{target}} - o1_{\text{out}}\right). \end{aligned} \tag{39}$$

Using Sigmoid activation function:

$$\begin{aligned} o1_{\text{out}} &= \frac{1}{1+e^{-o1_{\text{net}}}} \\ \frac{\partial o1_{\text{out}}}{\partial o1_{\text{net}}} &= o1_{\text{out}}(1 - o1_{\text{out}}) \end{aligned} \tag{40}$$

Finally,

$$\begin{aligned} o1_{\text{net}} &= w_5 \cdot h1_{\text{out}} + w_6 \cdot h2_{\text{out}} + b_2 \\ \frac{\partial o1_{\text{net}}}{\partial w_5} &= h1_{\text{out}}. \end{aligned} \tag{41}$$

Substituting Eqs. 39, 40, and 41 in $\frac{\partial E_T}{\partial w_5}$, the weights are updated w_5 as:

$$w_5 = w_5 - \eta \frac{\partial E_T}{\partial w_5}. \tag{42}$$

Computing the worst case error bounds for N samples

We consider the case where there is no weight update. Clearly, a combination of arbitrary weights in the activation function will be bounded above by the same function with optimized weights (since we have used sigmoid activation which is one-to-one). Therefore, an error bound for the simple vanilla feed forward neural network would suffice. We show that it is not possible to improve upon the error bound when the training on all the samples is complete.

Initially, an illustrative example of error bound computation of nth sample is shown. Let the network is trained for N samples of the data set, let i_1^n and i_2^n be the neuron inputs of nth sample and E_1^n and E_2^n are the error at output neuron. Let E_T^n be the total error, which is given by:

$$\begin{aligned} E_T^n &= E_1^n + E_2^n \\ E_T^n &= \frac{1}{2}\left(o_{1_n}^t - o_{1_n}^{\text{out}}\right)^2 + \frac{1}{2}\left(o_{2_n}^t - o_{2_n}^{\text{out}}\right)^2. \end{aligned} \tag{43}$$

$o_{1_n}^t$ is the expected (target) output of the first neuron of the nth sample. It is a constant whose value is independent of any input or output neuron values. $o_{1_n}^{\text{out}}$ is value of first neuron in the output layer.

$$\begin{aligned} o_{1_n}^{out} &= \frac{1}{1+e^{-(h_{1_n}w_5 + h_{2_n}w_6)}} \\ h_{1_n} &= \frac{1}{1+e^{-(i_{1_n}w_1 + i_{2_n}w_2)}} \\ h_{2_n} &= \frac{1}{1+e^{-(i_{1_n}w_3 + i_{2_n}w_4)}}. \end{aligned} \tag{44}$$

Substituting the values of h_{1n} and h_{2n} in o_{1n}^{out}, we get:

$$o_{1_n}^{\text{out}} = \frac{1}{1+e^{-\left[\left(\frac{w_5}{1+e^{-(i_{1_n}w_1+i_{2_n}w_2)}}\right)+\left(\frac{w_6}{1+e^{-(i_{1_n}w_3+i_{2_n}w_4)}}\right)\right]}} \tag{45}$$

The value of o_{2n}^{out} can be computed similarly.[4] This error expression is for the n^{th} sample and generalizing the expression for all N samples, the expression becomes:

$$\begin{aligned} E_T &= E_1^o + E_2^o + \cdots + E_N^o \\ E_T &= \frac{1}{2}\left(o_{1_n}^t - o_{1_n}^{out}\right)^2 + \frac{1}{2}\left(o_{2_n}^t - o_{2_n}^{out}\right)^2. \end{aligned} \tag{46}$$

The above expression 46 is valid when there are 2 neurons in output layer. It can be extended for architecture that supports more than 2 neurons and thus, the general equation when there are p neurons in output is as follows:

$$N \times E_T = \frac{1}{2}\sum_{j=1}^{p}\left(O_{ji}^t - O_{ji}^{out}\right)^2. \tag{47}$$

Here, O_{ji}^t is a constant $f(K)$ whose value is devoid of the weights in neurons in hidden or output layer. Thus, for all N samples over the entire set of neurons,

$$N \times E_T = \frac{1}{2}\sum_{j=1}^{p}\left(f(K) - O_{ji}^{\text{out}}\right)^2 \tag{48}$$

[4] The error bound of the replicated weights can be computed in a similar manner.

We observe from (20), O_i^{out} is an expression consisting of weights and input values and can be written as:

$$O_i^{\text{out}} = \frac{1}{1 + e^{-}(w_i, \text{inp}_{(.,i)})} = \sigma(w_i, inp_{(.}, i)) \tag{49}$$

This implies the Cumulative Error E,

$$E_T = \frac{1}{2N} \sum_{j=1}^{p} \Big(f(K) - \sigma(w_i, \text{inp}_{(.,i)} \Big)^2 . \tag{50}$$

We observe, $0 < \text{sig}(w_i, \text{inp}_{(.,i)} < 1$. $\big(f(K) - \sigma(w_i, \text{inp}_{(.,i)})\big)^2 \leq f(K)^2 - 2 \cdot f(K) \cdot \sigma(w_i, \text{inp}_{(.,i)}) = f(K)(f(K) - \sigma(w_i, \text{inp}_{(.,i)}) = E(N)$ where $\sigma(,)$ is the activation unit used to move between output, hidden and input layers. From Eq. (5.8.2), we can see that cumulative error is a function of number of samples and is therefore bounded by the following inequality:

$$\begin{aligned} &E(j) \leq E(N) \qquad \text{where } 1 \leq j \leq N \\ &\quad E(N) = \frac{1}{2N}(f(K)(f(K)- \\ &\quad \text{sig}(w_i, \text{inp}_{(.,i)}); \text{summed over the set of p neurons.} \end{aligned} \tag{51}$$

In the above expression, $E(j)$ is the sum of errors of j samples that is less than the total error of N samples. Mathematically, the expression, $\exp(w_i, inp_{(.,i)})$ is a small number with value less than 1. Hence, we conclude that the error is bounded by the training accuracy obtained on the maximum number of samples, N. Additionally, we infer, as the number of training samples per class increases, overall class-wise error diminishes.

Appendix B: Elastic gini in decision tree

Decision trees have remained a popular candidate for various classification tasks in data mining and machine learning. A greedy decision tree is built by the recursive partitioning of the feature space of input data based on a splitting criterion. Traditionally, the most popular splitting criteria, namely the Gini impurity and the Shannon entropy, have symmetric and concave functional forms. Neither criterion is believed to be better than the other, and it has been proved that in most cases, the Gini impurity and the Shannon Entropy choose the same split in a node.

Non-concave functions have remained largely unexplored as splitting criteria. An asymmetric splitting criteria is shown to have a better performance than the Shannon entropy; the efficacy of the tsallis entropy function and the necessary conditions of the parameters of the function are discussed in various literature. The primary reasons for the popularity of the Gini and entropy splitting criteria are they are symmetric about $p = 0.5$, thus equally considering the impurity induced in a node of a decision tree by the presence of samples of any class in a dataset; and because they are concave, with the minima at $p = 0$ and $p = 1$, and maxima at $p = 0.5$, where p is a probability variable with $p \in [0, 1]$, thus representing the largest impurity in a node when equal numbers of samples of each class is present, and the least impurity when samples of only one class is present [45]. These properties facilitate the greedy induction aspect of decision trees.

We introduce a new splitting criterion, which we call the elastic Gini impurity criterion. It is a generalization of the functional form of the Gini impurity and we define it as follows [62]:

$$I = 1 - \sum_{i=1}^{k} (p(c_i|t))^{\alpha_i}, \tag{52}$$

where I represents the impurity in a node t; $\forall i \in \{1, \ldots, k\}$ $c_i \in C$, where $C = \{c_1, c_2, \ldots, c_k\}$ is the set of classes in the learning sample L; $p(c_i|t)$ represents the conditional probability that a learning instance or object in t belongs to class c_i. We empirically demonstrate the efficacy of the elastic Gini for two-class datasets. We use the simpler notation: $p_1 = p(c_1|t)$ and $p_2 = 1 - p_1$, where $p_1, p_2 \in [0, 1]$, and without a loss of generality represent the conditional probabilities that a learning instance or object in t belongs to class-1 and class-2 in the data respectively. The notation of the elastic Gini for a two-class problem is as follows:

$$I = 1 - p_1^{\alpha_1} - p_2^{\alpha_2}. \tag{53}$$

These notations will be used throughout the paper.

The asymmetric entropy measure for decision trees developed by Marcellin in 2008 takes in a parameter w from the user to enforce a skewness in the splits made. Drawing inspiration from this, the elastic Gini incorporates parameters corresponding to each class, called *elasticities*, which control the asymmetricity and convexity.

The advantage that our criterion provides over the asymmetric entropy measure is that we use multiple control variables that can determine the convexity of the splitting criteria, thus determining the impurity induced by each class in the dataset. Additionally, we use values of α_1 and α_2 such that the functional form of the elastic Gini is non-convex and non-concave in $p_1 \in [0, 1]$, but may be convex or concave in certain ranges of p. By inducing convexity in the splitting criteria, the splits do not correspond to a maximum impurity decrease. However, the results presented in this paper indicate that such a splitting mechanism may globally perform better in some cases, especially when the number of samples of each class in a data set are not balanced (note the habitability classes), or when one class inherently has a greater priority to be classified over the other class.

An example

Consider the scenario of a classification task for detecting a particular disease based on the proportion of a particular type of marker cells (say, type A) in a cytometer. It is possible that these cells can be wrongly classified as belonging to type B (and vice versa) which is not a marker for the disease. Now, in a such a scenario, we do not want the misclassification rates for type A and type B to be the same. In fact, we can tolerate a higher misclassification rate for type B, but not for type A. By selecting values of elasticities appropriately, we ensure that the accuracy of the system towards classifying instances of A increases or the misclassification rate for type A reduces. Such asymmetry can be modeled by the elasticity parameters, and we demonstrate this aspect of improvement (in this paper, without a loss of generality, we cast the more important class as the true class and try to improve the true positive rate of the classifier) with a discussion of the results.

Properties of the elastic Gini splitting criteria

Functional form and similarity to Gini impurity criterion

For two classes, the Gini impurity is defined as follows:

$$I = 1 - p_1^2 - p_2^2 = 1 - p_1^2 - (1-p_1)^2 \tag{54}$$

The limiting case $\alpha_1 \to 2, \alpha_2 \to 2$, of the elastic Gini impurity criterion is the same as the Gini impurity criterion. In this condition, both impurity criteria are strictly concave and symmetric about $p_1 = 0.5$ in the range $p_1 \in [0,1]$, with a maxima at $p = 0.5$. However, in this condition, the splitting criteria is not sensitive to imbalances that exist in the sample size of each class, and assumes that the misclassification costs of all the classes are equal. In real-world test cases, these assumptions usually do not hold true.

Conditions for concavity and convexity

We first construct the Hessian matrix of the elastic Gini as a function of variables p_1 and $p_2 = 1 - p_1$. The first and second partial derivatives of the elastic Gini function are:

$$\begin{aligned}
\frac{\partial I}{\partial p_1} &= -\alpha_1 p_1^{\alpha_1 - 1}\\
\frac{\partial I}{\partial p_2} &= -\alpha_2 p_2^{\alpha_2 - 1}\\
\frac{\partial^2 I}{\partial p_1^2} &= -\alpha_1(\alpha_1 - 1) p_1^{\alpha_1 - 2}\\
\frac{\partial^2 I}{\partial p_1 \partial p_2} &= 0\\
\frac{\partial^2 I}{\partial p_2^2} &= -\alpha_2(\alpha_2 - 1) p_2^{\alpha_2 - 2}.
\end{aligned} \tag{55}$$

Thus, the Hessian matrix is:

$$H = \begin{bmatrix} -\alpha_1(\alpha_1-1)p_1^{\alpha_1-2} & 0 \\ 0 & -\alpha_2(\alpha_2-1)p_2^{\alpha_2-2} \end{bmatrix}. \tag{56}$$

Let D_1 and D_2 represent the determinants of the 1^{st} and 2^{nd} leading primary minor of H, respectively. Thus, D_1 and D_2 are:

$$\begin{aligned}
D_1 &= -\alpha_1(\alpha_1-1)p_1^{\alpha_1-2}\\
D_2 &= \alpha_1\alpha_2(\alpha_1-1)(\alpha_2-1)p_1^{\alpha_1-2}p_2^{\alpha_2-2}.
\end{aligned} \tag{57}$$

The signs of D_1 and D_2 determine if the matrix H is positive definite, negative definite or indefinite, and in turn, determine if the elastic Gini is concave, convex, or non-concave and non-convex based on the values of α_1 and α_2.

Lemma 1 *The elastic Gini is convex in $p \in [0,1]$ when $\alpha_1 \leq 1$ and $\alpha_2 \leq 1$*

Proof For a function to be convex, the Hessian matrix of the function needs to be positive semi-definite. H can be positive semi-definite *iff* $D_1 \geq 0$ and $D_2 \geq 0$. For this condition to be satisfied for D_1, α_1 must be less than or equal to 1. Given that $\alpha_1 \leq 1$, for this condition to be satisfied for D_2, α_2 must also be less than or equal to 1. □

Lemma 2 *The elastic Gini is concave in $p \in [0,1]$ when $\alpha_1 \geq 1$ and $\alpha_2 \geq 1$.*

Proof For a function to be concave, the Hessian matrix of the function needs to be negative semi-definite. H can be negative semi-definite *iff* $D_1 \leq 0$ and $D_2 \geq 0$. For this condition to be satisfied for D_1, α_1 must be greater than or equal to 1. Given that $\alpha_1 \geq 1$, for this condition to be satisfied for D_2, α_2 must also be greater than or equal to 1. □

Corollary 1 *The elastic Gini is non-convex and non-concave in $p \in [0,1]$ when*

1. $\alpha_1 < 1$ and $\alpha_2 > 1$; or
2. $\alpha_1 > 1$ and $\alpha_2 < 1$

Proof Since it is known from lemmata 1 and 2 that the elastic Gini is convex when when $\alpha_1 \leq 1$ and $\alpha_2 \leq 1$, and concave when $\alpha_1 \geq 1$ and $\alpha_2 \geq 1$, it immediately follows that for all other combinations of ranges of values of α_1 and α_2, the elastic Gini function would be non-concave and non-convex. Thus, the elastic Gini function is non-concave and non-convex when $\alpha_1 < 1$ and $\alpha_2 > 1$, or when $\alpha_1 > 1$ and $\alpha_2 < 1$. A generalization of the above properties for k classes follows shortly. □

Reflective property

Lemma 3 *Interchanging α_1 and α_2 reflects the elastic Gini function about $p = 0.5$.*

Proof At $p_1 = 0.5$, $1 - p_1 = 0.5$. Thus, upon interchanging the values of α_1 and α_2 in the model, the impurity at $p_1 = p_2 = 0.5$ remains the same. For the general case, we find the values of p_1 and p_2 where the impurities are the same upon swapping α_1 and α_2:

$$\begin{aligned}
&1 - p_1^{\alpha_1} - p_2^{\alpha_2} = 1 - p_1^{\alpha_2} - p_2^{\alpha_1}\\
&\quad\Rightarrow p_1^{\alpha_1} - p_1^{\alpha_2} = p_2^{\alpha_1} - p_2^{\alpha_2}\\
&\quad\Rightarrow p_1^{\alpha_1}(1 - p_1^{\alpha_2-\alpha_1}) = p_2^{\alpha_1}(1 - p_2^{\alpha_2-\alpha_1})\\
&\quad\Rightarrow p_1^{\alpha_1}(1 - p_1^{\alpha_2-\alpha_1}) = (1-p_1)^{\alpha_1}\left(1 - (1-p_1)^{\alpha_2-\alpha_1}\right)
\end{aligned} \tag{58}$$

Thus from the steps in Equation 58, we see that the value of impurity that is observed at any $p \in [0,1]$ is the same as that observed at $1 - p$ when α_1 and α_2 are interchanged. This corresponds to a reflection of the function about $p = 0.5$. □

Positive and negative regions

The elastic gini impurity function should have a positive region and a negative region. This can be considered to be a consequence of the non-convex and non-concave nature of the function. Typically, we define an interval Q of p_1 where $I(p_1) \leq 0$, that includes $p_1 = 0$ or $p_2 = 0$, but not both; $\forall p_1 \in Q'$, which is the complement of Q, $I(p_1) \geq 0$. In addition to that, $I(p_1)$ should be continuous $\forall p_1 \in [0,1]$.

Lemma 4 *There must exist a point $0 < p_T < 1$ where $I(p_T) = 0$.*

Proof Since $I(p_1) \leq 0$, $\forall p \in Q$, $I(p_1) \geq 0$, $\forall p \in Q'$, and $I(p_1)$ is continuous in $Q + Q'$, it follows that there must exist a point $p_T \in Q$ and $p_T \in Q'$ where $I(p_1 = p_T) = 0$. If such a point does not exist, then $I(p_1)$ would not be continuous in p_1. This contradicts the premise that the elastic Gini should be used as a continuous function in $p_1 \in [0, 1]$.

Having a value p_T such that $I(p_T) = 0$ allows us to define the negative and the positive regions. We can thus set a range $[0, p_T]$, or $[p_T, 1]$, which corresponds to the negative region, and we can accordingly find the values of α_1 and α_2.

The negative values allow the criteria to make splits that are sub-optimal compared to the Gini impurity criteria. However, the optimal splitting often leads to overfitting of decision trees. To reduce the effect of overfitting, tree pruning algorithms and bootstrap aggregation methods are used. □

Split when node impurities are positive

While using the Gini impurity criteria, a node is not split when the value of the impurity is zero. In the case of elastic Gini, a node may have a negative value of impurity if the probability variable p_1 falls in the negative region of the function; a split is made only when the value of impurity is positive. Doing this allows us the decision tree algorithm to prefer splits that are not at $p_1 = 0$ and $p_1 = 1$.

Elastic Gini as a means of cost-sensitive learning

Very often, classes in real-word data are imbalanced in terms of the number of instances in them. In some critical cases, it would be effective to prioritize the accuracy of one class over the other. For instance, in the breast-cancer dataset, class-1 represents benign tissues, whereas class-2 represents malignant tissues. A diagnosis program might want to be more accurate towards the malignant class as wrongly diagnosing this could post a greater risk to a patient's life. Similarly, in the case of the credit-card dataset, class 1 represents the class of non-defaulting clients, and class 2 represents the class of defaulting clients; class 2 could typically cost a credit card company greater losses, and they may want to automatically identify defaulters early in order to minimize loss. By adjusting the values of elasticities in our experiment iterations, we try to improve the recall or the true positive rate of the trees with elastic Gini as a way of incorporating a class-sensitive measure.

Formal comparison of Gini impurity criterion with elastic Gini impurity criterion

Raileanu in 2004 developed a framework for the comparison of two splitting criteria and to analyse how the same split may be selected or rejected in either criteria. They formally compare the Gini impurity measure with Shannon entropy.

An exhaustive formal comparison with the general form of the elastic Gini will be difficult because its properties are determined by the values of the elasticities that we choose for it. Hence, such a comparison can be made for particular values of elasticities. This would be a useful exercise as it could give us insights into when different splits are selected.

Selecting values of elasticities

In Sect. 1, we have mentioned that the elastic Gini function should be zero at a point $p_1 = p_T$ by selecting α_1 and α_2 appropriately. This corresponds to the following:

$$1 - p_T^{\alpha_1} - (1 - p_T)^{\alpha_2} = 0, \tag{59}$$

where $0 < p_T < 1$. However, note that Eq. 59 does not have a closed-form solution as there are multiple values of α_1, α_2 and p_T that can satisfy it.

We elaborate three methods of selecting α_1, α_2 and p_T that satisfy Eq. 59.

Case 1: *Choose a value of p_T and find pairs of α_1 and α_2.*

If a value of p_T is fixed and α_1 and α_2 have to be determined that satisfy Eq. 59, a metaheuristic algorithm such as NSGA-II may be used to find various feasible pairs of α_1 and α_2.

Case 2: *Choose a value of p_T and α_1, and find α_2.*

In this case, we can find α_2 using arithmetic:

$$\begin{aligned} &1 - p_T^{\alpha_1} - (1 - p_T)^{\alpha_2} = 0 \\ \Rightarrow &\alpha_2 = \frac{\log(1 - p_T^{\alpha_1})}{\log(1 - p_T)}. \end{aligned} \tag{60}$$

In a similar way, p_T and α_2 can be fixed and α_1 can be found.

Case 3: *Without fixing any values of elasticity or p_T, find combinations of α_1, α_2 and p_T.*

An entire set of solutions of p_T, α_1 and α_2 can be determined using metaheuristic algorithms. The set of solutions can be tested on a dataset with the elastic Gini splitting criteria to find the values of p_T, α_1 and α_2 that correspond to the best performance of the decision tree classification.

An important property

We define the elastic Gini splitting criteria for k classes as:

$$I(p_1, p_2, \ldots, p_k) = 1 - \sum_{i=1}^{k} p_i^{\alpha_i} \tag{61}$$

The second partial derivative of the elastic Gini impurity as:

$$\frac{\partial^2 I}{\partial p_j \partial p_i} = \begin{cases} -\alpha_i(\alpha_i - 1)p_i^{\alpha_i - 2}, & \text{if} \quad i = j \\ 0, & \text{if } i \neq j. \end{cases} \tag{62}$$

The Hessian matrix of $I(p_1, p_2, \ldots, p_k)$ consequently is:

$$H(I(p_1, p_2, \ldots, p_k)) = \begin{bmatrix} -\alpha_1(\alpha_1 - 1)p_1^{\alpha_1 - 2} & 0 & \ldots & 0 \\ 0 & -\alpha_2(\alpha_2 - 1)p_2^{\alpha_2 - 2} & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & -\alpha_k(\alpha_k - 1)p_k^{\alpha_k - 2}. \end{bmatrix} \tag{63}$$

The principal diagonal elements are computed as:

$$\begin{aligned} D_1 &= -\alpha_1(\alpha_1 - 1)p_1^{\alpha_1 - 2} \\ D_2 &= -\alpha_2(\alpha_2 - 1)p_2^{\alpha_2 - 2} \\ D_3 &= -\alpha_3(\alpha_3 - 1)p_3^{\alpha_3 - 2} \\ &\vdots \\ D_k &= -\alpha_k(\alpha_k - 1)p_k^{\alpha_k - 2}. \end{aligned} \tag{64}$$

Lemma 1 *The elastic Gini is convex in $p \in [0,1]$ when $\alpha_i \leq 1 \ \forall i$*

Proof For a function to be convex, the Hessian matrix of the function needs to be positive semi-definite. H can be positive semi-definite *iff* $D_i \geq 0 \ \forall i$. For this condition to be satisfied, α_i must be less than or equal to 1 $\forall i$. □

Lemma 5 *The elastic Gini is concave in $p \in [0,1]$ when $\alpha_i \geq 1 \ \forall i$*

Proof For a function to be concave, the Hessian matrix of the function needs to be negative semi-definite. H can be negative semi-definite *iff* $D_i \leq 0 \ \forall i$. For this condition to be satisfied, α_i must be greater than or equal to 1 $\forall i$. □

Corollary 1 *The elastic Gini is non-convex and non-concave in $p \in [0,1]$ when $\alpha_i < 1$ for a set A of i and $\alpha_i > 1$ for A'.*

Proof Since it is known from lemma 1 and 2 that the elastic Gini is convex when when $\alpha_i \leq 1 \ \forall i$, and concave when $\alpha_i \geq 1 \ \forall i$, it immediately follows that for all other combinations of ranges of values of α_i, the elastic Gini function would be non-concave and non-convex. □

Appendix C: Description of feature set

Table 12 Description of features of PHL-EC dataset

Features	Description
P. Zone Class	Planet habitable zone classification (hot, warm, or cold)
P. Mass Class	Planet Mass Class (mercurian, subterran, terran, superterran, neptunian, or jovian)
P. Composition Class	Planet Composition Class (iron, rocky-iron, rocky-water, water-gas, gas)
P. Atmosphere Class	Planet Atmosphere Class (none, metals-rich, hydrogen-rich)
P. Habitable Class	Planet Habitable Class (mesoplanet, thermoplanet, psychroplanet, hypopsychroplanet, hyperthermoplanet, non-habitable)
P. Min Mass	Planet minimum mass
P. Mass	Planet mass. Most of the values were estimated from minimum mass
P. Max Mass	Planet maximum mass
P. Radius	Planet radius. Most of these values were estimated for confirmed planets
P. Density	Planet density
P. Gravity	Planet gravity
P. Esc Vel	Planet escape velocity
P. SFlux Min	Planet minimum stellar flux
P. SFlux Mean	Planet mean stellar flux
P. SFlux Max	Planet maximum stellar flux
P. Teq Min	Planet minimum equilibrium temperature (at apastron)
P. Teq Mean	Planet mean equilibrium temperature
P. Teq Max	Planet maximum equilibrium temperature (at periastron)
P. Ts Min	Planet minimum surface temperature (at apastron)
P. Ts Mean	Planet mean surface temperature
P. Ts Max	Planet maximum surface temperature (at periastron)
P. Surf Press	Planet surface pressure
P. Mag	Planet magnitude as seen from a Moon-Earth distance (Moon = $-$ 12.7)
P. Appar Size	Planet apparent size as seen from a Moon-Earth distance (Moon = 0.5°)
P. Period	Planet period
P. Sem Major Axis	Planet semi major axis
P. Eccentricity	Planet eccentricity (assumed 0 when not available)
P. Mean Distance	Planet mean distance from the star
P. Inclination	Planet inclination (assumed 60° when not available, and 90° for Kepler data)
P. Omega	Planet omega
S. Teff	Star effective temperature
S. Luminosity	Star luminosity
P. HZD Planet	Habitable Zone Distance (HZD)
P. HZC Planet	Habitable Zone Composition (HZD)
P. HZA Planet	Habitable Zone Atmosphere (HZD)
P. HZI Planet	Habitable Zone Index (HZI)
S. Hab Zone Min	Star inner edge of habitable zone
S. Hab Zone Max	Star outer edge of habitable zone

References

1. A. Cassan, D. Kubas, J.-P. Beaulieu, M. Dominik, K. Horne, J. Greenhill, J. Wambsganss, J. Menzies, A. Williams, U.G. Jorgensen, A. Udalski, D.P. Bennett, M.D. Albrow, V. Batista, S. Brillant, J.A.R. Caldwell, A. Cole, C. Coutures, K.H. Cook, S. Dieters, D. Dominis Prester, J. Donatowicz, P. Fouque, K. Hill, N. Kains, S. Kane, J.-B. Marquette, R. Martin, K.R. Pollard, K.C. Sahu, C. Vinter, D. Warren, B. Watson, M. Zub, T. Sumi, M.K. Szymanski, M. Kubiak, R. Poleski, I. Soszynski, K. Ulaczyk, G. Pietrzynski, L. Wyrzykowski, Nature **481**, 167 (2012)
2. W. Bains, D. Schulze-Makuch, Life 6, 25 (2016). https://doi.org/10.3390/life6030025
3. N.M. Batalha, Proc. Natl. Acad. Sci. **111**, 12647–12654 (2014)
4. L.N. Irwin, D. Schulze-Makuch, (Springer-Praxis, New York, 2010), pp 45–67. https://doi.org/10.1007/978-1-4419-1647-1_3
5. J.F. Kasting, Science **259**, 920 (1993)
6. L. Irwin, A. Méndez, A. Fairén, D. Schulze-Makuch, Challenges **May**, 159–174 (2014)
7. D. Schulze-Makuch, A. Méndez, A.G. Fairén, P. von Paris, C. Turse, G. Boyer, A.F. Davila, M.R. de Sousa António, D. Catling, L.N. Irwin, Astrobiology **December**, 1041–1052 (2011)
8. K. Bora, S. Saha, S. Agrawal, M. Safonova, S. Routh, A. Narasimhamurthy, Astron. Comput. **17**, 129–143 (2016)
9. S. Snehanshu, S. Jyotirmoy, D. Avantika, N. Nandita, M. Anand, R. Ranjan, J. Cloud Comput. **5**, 1–23 (2016)
10. G. Gouri, S. Snehanshu, M. Archana, V. Sukrit, V. Sujith, N. Anand, D.B.S. Sagar, Scientometrics **108**, 1479–1529 (2016)
11. G. Gouri, S. Snehanshu, B. Chitra, R.S. Harsha, A. Mathur, B.S. Dayasagar, M.N. Anand, Proceedings of the Fourth National Conference of Institute of Scientometrics, SIoT, (2015)
12. S. Saha, J. Sarkar, A. Dwivedi, N. Dwivedi, A.M. Narasimhamurthy, R. Roy, J. Cloud Comput. Adv. Syst. Appl. **5**, 1 (2016)
13. M. Safonova, J. Murthy, Y.A. Shchekinov, Age aspects of habitability. Int. J. Astrobiol. **15**, 93–105 (2016)
14. Y.A. Shchekinov, M. Safonova, J. Murthy, Astrophys. Space Sci. **346**, 31–40 (2013)
15. D. Schulze-Makuch, A. Méndez, A.G. Fairén et al., Astrobiology **11**, 1041 (2011)
16. S. Agrawal, S. Basak, S. Saha, K. Bora, J. Murthy, EPrint:1804.11176, (2018)
17. C.J. Shallue, A. Vanderburg, Astron. J. **155**, 94 (2018)
18. S. Saha, S. Basak, K. Bora, M. Safonova, S. Agrawal, P. Sarkar, J. Murthy, Astron. Comput. **23**, 141–150 (2018)
19. J.H. Friedman, Ann. Stat. **29**, 1189–1232 (2000)
20. T. Chen, C. Guestrin, XGBoost, Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining—KDD, (2016)
21. A.C. Barr, V. Dobos, L.L. Kiss, Astron. Astrophys. **613**, A37 (2017)
22. J. de Wit, H.R. Wakeford, N.K. Lewis, L. Delrez, M. Gillon, F. Selsis, J. Leconte, B.-O. Demory, E. Bolmont, V. Bourrier, A.J. Burgasser, S. Grimm, E. Jehin, S.M. Lederer, J.E. Owen, V. Stamenković, A.H.M.J. Triaud, Nat. Astron. **2**, 214–219 (2018)
23. A. Méndez (2011) http://phl.upr.edu/library/notes/athermalplanetaryhabitabilityclassificationforexoplanets. Accessed 12 Oct 2020
24. S. Elfwing, E. Uchibe, K. Doya, Neural Netw. **107**, 3–11 (2018). https://doi.org/10.1016/j.neunet.2017.12.012
25. S. Haykin, *Neural Networks, A Comprehensive Foundation, Reviewed by R Lippmann* (World Scientific Pub Co Pvt Ltd, Singapore, 1994)
26. S. Makhija, S. Saha, S. Basak, D. Mousumi, Astron. Comput. **29**, 300–313 (2019)
27. A. Méndez (2018) http://phl.upr.edu/projects/habitable-exoplanets-catalog/data/database. Accessed 12 Oct 2020
28. PHL's Exoplanet Catalog of the Planetary Habitability Laboratory @ UPR Arecib (2017). http://phl.upr.edu/projects/habitable-exoplanets-catalog/data/database. Accessed 28 Sep 2020
29. S. Sailesh, S. Azhar, S. Saha, Y. Rahul, S. Sriparna, *Proceedings in International Joint Conference on Neural Networks* (2020)
30. I. Rish, *IJCAI 2001 Workshop on Empirical Methods in Artificial Intelligence, Vol3, No 22* (IBM, New York, 2001)
31. K. Mohanchandra, S. Saha, K. Srikanta Murthy, G.M. Lingaraju, Int. J. Intell. Eng. Inform. **3**, 313 (2015)
32. N. Vladimir, V. Alexey Ya Chervonenkis, Autom. Remote. Control. **1**, 103–109 (1964)
33. C. Corinna, V. Vladimir, Mach. Learn. **20**, 273–297 (1995)
34. B.E. Boser, I. Guyon, V.N. Vapnik, COLT 1992 (1992)
35. R.O. Duda, P.E. Hart, D.G. Stork, *Pattern Classification* (Wiley, New York, 2001)
36. J.R. Quinlan, Mach. Learn. **1**, 81–106 (1986)
37. L. Breiman, R. Forests, Mach. Learn. **45**, 5–32 (2001)
38. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, J. Mach. Learn. Res. **12**, 2825–2830 (2011)
39. M. Rosenblatt, Ann. Math. Stat. **27**, 832–837 (1956)
40. E. Parzen, Ann. Math. Stat. **33**, 1065–1076 (1962)
41. S.E. Sale, Mon. Not. R. Astron. Soc. **452**, 2960–2972 (2015)
42. N.B. Peng, Y.X. Zhang, Y.H. Zhao, Sci. China Phys. Mech. Astron. **56**, 1227–1234 (2013)
43. N.V. Chawla, K.W. Bowyer, L.O. Hall, W.P. Kegelmeyer, J. Artif. Intell. Res. **16**, 321–357 (2002)
44. D.A. Zighed, *Ritschard, Gilbert, Marcellin, Simon, Advances in Intelligent Information Systems* (Springer, Berlin, 2010), pp. 27–42
45. L. Breiman, Mach. Learn. **24**, 41–47 (1996)
46. N.M. Ball, R.J. Brunner (2011) http://ned.ipac.caltech.edu/level5/March11/Ball/Ball2.html. Accessed 12 Oct 2020
47. R. Heller, J. Armstrong, Astrobiology **14**, 50–66 (2014)
48. M. Welling, *Fischer Linear Discriminant Analysis* (University Of Toronto, Department Of Computer Science, Toronto, 2005)
49. C.W. Hsu, C.C. Chang, C.J. Lin, A Practical Guide to Support Vector Classification, Technical Report, Department of Computer Science and Information Engineering, University of National Taiwan, Taipei, 2003, pp. 1–12

50. Y.L. Cai, J. Duo, D. Cai, *Proceedings of NTCIR-8 Workshop Meeting*, pp. 336–340 (2010)
51. J.S. Denker, Phys. D Nonlinear Phenomena **22**, 216–232 (1986)
52. S. Amari, Neurocomputing **5**, 185–196 (1993)
53. A. Méndez (2011) http://phl.upr.edu/library/notes/syntheticstars. Accessed 12 Oct 2020
54. J.T. Springenberg, Unsupervised and Semi-supervised Learning with Categorical Generative Adversarial Networks (2015). arXiv:1511.06390
55. S. Tim, G. Ian, Z. Wojciech, C. Vicki, R. Alec, C. Xi, Proceedings of the 30th International Conference on Neural Information Processing Systems, pp 2234–2242 (2016)
56. R. Yedida, S. Saha, arXiv:1902.07399.
57. S. Saha, N. Nagaraj, A. Mathur, R. Yedida, Evolution of Novel Activation Functions in Neural Network Training with Applications to Classification of Exoplanets. arXiv:1906.01975 (2019)
58. S. Saha, M. Archana, B. Kakoli, B. Suryoday, A. Surbhi, International Conference on Advances in Computing, Communications and Informatics (ICACCI) (2018)
59. S. Basak, S. Kar, S. Saha, L. Khaidem, S.R. Dey, North Am. J. Econ. Finance **47**, 552–567 (2019)
60. D. Schulze-Makuch, W. Bains, Nat. Astron. **2**, 432–433 (2018)
61. S. Saha, J. Sarkar, S. Sarkar, S. Das https://www.researchgate.net/publication/344449525_BTAIBinary_Tree_Based_Anomaly_Identification_Algorithms_for_Industrial_Devicesas. Accessed 12 Oct 2020
62. B.E. Rhoades, Trans. Am. Math. Soc. **226**, 257–313 (1977)