Unearthing Exoplanets using Machine Learning

Prof. Chandrasekhar Mukherjee: Computer Engineering Department, San Jose State University, San Jose, CA- 95192

Gauravkumar Koradiya
Computer Engineering – CMPE 257
San Jose State University
San Jose, CA, USA
gauravkumar.koradiya@sjsu.edu

Saicharan Reddy Kadari
Computer Engineering – CMPE 257
San Jose State University
San Jose, CA, USA
saicharanreddy.kadari@sjsu.edu

Pratik Mali
Software Engineering – CMPE 257
San Jose State University
San Jose, CA, USA
pratikrajesh.mali@sjsu.edu

Mian Dudhatra
Computer Engineering – CMPE 257
San Jose State University
San Jose, CA, USA
milangirishkumar.dudhatra@sjsu.edu

Pravin Ramasamy Balachandran
Software Engineering – CMPE 257
San Jose State University
San Jose, CA, USA
pravin.ramasamybalachandran@sjsu.edu

***Abstract*— "That is a big question we all have: are we alone in the universe? And exoplanets confirm the suspicion that planets are not rare."— Neil deGrasse Tyson**

For thousands of years, people have looked up at the stars, recorded observations, and noticed patterns. An exoplanet is any planet beyond our solar system. Scientifically, it's very important for us to know whether there's another Earth-like world out there, and whether life could exist outside our solar system. Here, we intend to develop a machine learning model based on dataset from NASA and Caltech that can predict if an observation is a real candidate for an exoplanet or not. The data was collected by the* Kepler mission *that revealed thousands of planets out of our Solar System. Kepler was able to find planets by looking for small dips in the brightness of a star when a planet transits in front of it. It is possible to measure the size of the planet based on the depth of the transit and the star's size. There may be many exoplanets still unfound in Kepler data, and new ideas and techniques like machine learning will help fuel celestial discoveries for many years to come. To infinity, and beyond!

Keywords—exoplanet, life, Kepler, data

I. Motivation

The quest for unraveling the mysteries of the mighty universe (maybe multiple universes) always fascinates human minds through ages. It is believed that studying the formation of stars, planets and all other mighty cosmic objects scattered and hidden in the universe we know and seen through our theories and lenses will aid us and improve our knowledge about the formation and functioning of the universe. and one of the core reasons is to understand how things are made. To understand is to think, see and relate, or to see and relate. Exoplanet hunting is considered as one such process and a fundamental step in identifying such planets. NASA considers and describes this big challenge of search for potential life in other worlds and planetary systems as a multistage approach that starts with:

Identifying an exoplanet → checking if it is in a habitable zone → looking for Life Signs →The Future

Scientists and research enthusiasts of earlier times always relied on various techniques and methods to search for the exoplanets, like the Radial Velocity, Transit spectroscopy, Gravitational Microlensing and Direct Imaging, ranging from the earliest methods to the latest ones. Transits: Planets found in the dips in light are the focus area of this project and this is based on Data captured through the Kepler Space Telescope. It has Stayed in orbit for about 10 years and positioned in one view for 4 years each and gathered an Immense data of about 678GB which paved the way for a staggering 2662 confirmed exoplanets, half of all the confirmed exoplanets.

It was an Immense task for the scientists to find through the conventional method to see through the data of Kepler Objects of Interest (KOI's) on a case-by-case basis and try to extract the features that makes it an exoplanet. This is one such use case where the modern day technique and tools of machine learning can be employed to search for and match all the necessary features that make up a cosmic body an exoplanet. We try to identify this based on the specifications of the potential exoplanets which can be helpful in classifying and characterizing the exoplanets based on their size, distance from earth, age, composition, and locality in the universe

II. Liturature Survey

We evaluate the extent to which newly detected exoplanetary systems containing at least four planets adhere to a generalized Titus–Bode (TB) relation. We find that most exoplanet systems in our sample adhere to the TB relation to a greater extent than the Solar system does, particularly those detected by the Kepler mission. We use a generalized TB relation to make a list of predictions for the existence of 141 additional exoplanets in 68 multiple-exoplanet systems: 73 candidates from interpolation, 68 candidates from extrapolation. We predict the existence of a low-radius ($R < 2.5R\oplus$) exoplanet within the habitable zone of KOI-812 and that the average number of planets in the habitable zone of a star is 1–2. The usefulness of the TB relation and its validation as a tool for predicting planets will be partially tested by upcoming Kepler data releases.

III. Data Overview

We assess the degree to which newly discovered exoplanetary systems with at least four planets follow a generalized Titius-Bode (TB) relation. We discover that the majority of exoplanet systems in our sample, particularly those discovered by the Kepler mission, adhere to the TB relation to a greater extent than the Solar system.. We use a generalized TB relation to make a list of predictions for the existence of 141 additional exoplanets in 68 multiple-exoplanet systems: 73 candidates from interpolation, 68 candidates from extrapolation. We predict the existence of a low-radius ($R < 2.5R\oplus$) exoplanet within the habitable zone of KOI-812 and that the average number of planets in the

habitable zone of a star is 1–2. The usefulness of the TB relation and its validation as a tool for predicting planets will be partially tested by upcoming Kepler data releases.

	KepID	KOIName	KeplerName	ExoplanetArchiveDisposition	DispositionUsingKeplerData	DispositionScore	NotTransit-LikeFalsePositiveFlag	koi_fpflag_ss	Centroid
0	10797460	K00752.01	Kepler-227 b	CONFIRMED	CANDIDATE	1.000	0	0	
1	10797460	K00752.02	Kepler-227 c	CONFIRMED	CANDIDATE	0.969	0	0	
2	10811496	K00753.01	NaN	CANDIDATE	CANDIDATE	0.000	0	0	
3	10848459	K00754.01	NaN	FALSE POSITIVE	FALSE POSITIVE	0.000	0	1	
4	10854555	K00755.01	Kepler-664 b	CONFIRMED	CANDIDATE	1.000	0	0	

Figure 1Overview of Dataset

IV. METHODOLOGY

The following high-level diagram represents the function of a predictive ML system.

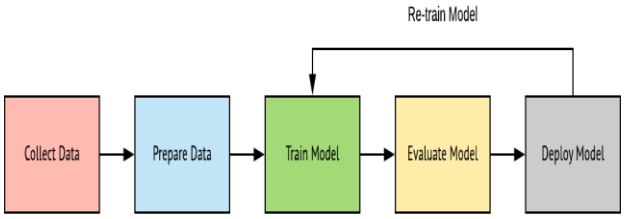


Figure 2: High level System Diagram

A. Data Extraction

Generally, Data collection is the critical process in the ML space; it would be internal or external based on the customer projection in the markets. It is perhaps the most important operation of the Extract/Transform/Load (ETL) process because it is the foundation for critical analyses and decision-making processes. It enables consolidation, analysis and refining of data so that it can be converted into meaningful information that can be stored for further use and manipulation. The extracted data can help in decision making, customer base expansion, service improvements, predicting sales and optimizing costs, among other things. The current dataset is structured in nature and static. However, the Exoplanet Archive data are accessed primarily through an Exoplanet Archive application programming interface (API). We transformed the JSON data from API to CSV using Excel to make it available for Machine Learning analysis.

B. Data Pre-Processing and Feature Engineering

The data cleaning process gives a better understanding of the features and the relationships between them, each entity/feature/attribute. During the step, extracting the essential variables and leaving behind/removing non-essential variables is a big challenge. For example, handling missing values or human error in the given data set would enhance the process and improve the modeling performance, and data quality would be improved considerably. Because this would lead us to unrealistic results, we either truncate the data above a threshold or transform the data using log transformation. So there are lots of methods in ML space to handle these situations. Converting categorical columns into numerical is one of the primary and critical approaches in most ML model developments since most algorithms need numerical features. Numbers are always significant!

Exploratory Data Analysis (EDA) is the process of visualizing and analyzing data to extract insights from it. In other words, EDA is the process of summarizing important characteristics of data to gain better understanding of the dataset. Plotting in EDA consists of Histograms, Box plot, Scatter plot and many more. It often takes much time to explore the data. Through the process of EDA, we can ask to define the problem statement or definition on our data set which is very important. Some techniques above might work better with some algorithms or datasets, while some of them might be beneficial in all cases. Based on current situation, we have performed:

1. Change the columns name
2. Dropped some columns
3. Transformed target variables
4. Handling Missing values
5. Remove Duplicate instances

We have used an open-source library named pandas-profiling. It extends pandas DataFrame with df.profile_report(), which automatically generates a standardized univariate and multivariate report for data understanding. For each column, the following information (whenever relevant for the column type) is presented in an interactive HTML report. It calculates and presents the following:

- Type inference: detect the types of columns in a DataFrame
- Essentials: type, unique values, indication of missing values
- Quantile statistics: minimum value, Q1, median, Q3, maximum, range, interquartile range
- Descriptive statistics: mean, mode, standard deviation, sum, median absolute deviation, coefficient of variation, kurtosis, skewness
- Most frequent and extreme values
- Histograms: categorical and numerical
- Correlations: high correlation warnings, based on different correlation metrics (Spearman, Pearson, Kendall, Cramer's V, Phik)
- Missing values: through counts, matrix and heatmap
- Duplicate rows: list of the most common duplicated rows
- Text analysis: most common categories (uppercase, lowercase, separator), scripts (Latin, Cyrillic) and blocks (ASCII, Cyrilic)

File and Image analysis: file sizes, creation dates, dimensions, indication of truncated images and existence of EXIF metadata.

Few snippets from the EDA report:

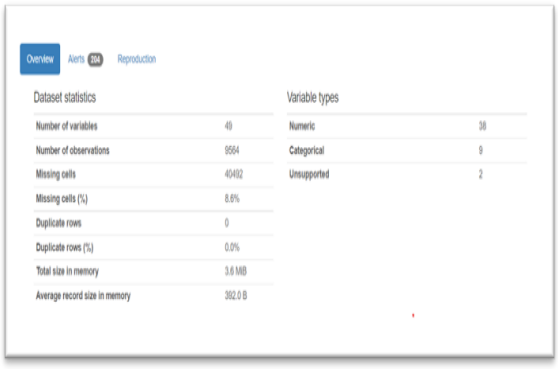


Figure 3: Overview of Dataset Statistics

Variable statistics

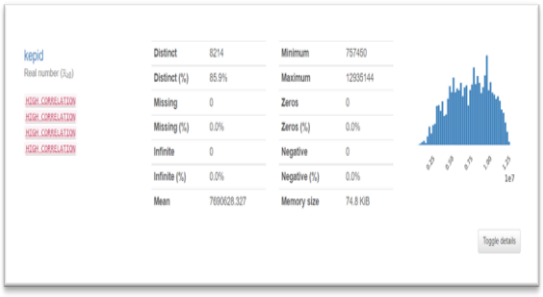


Figure 4:Variable Statistics-1

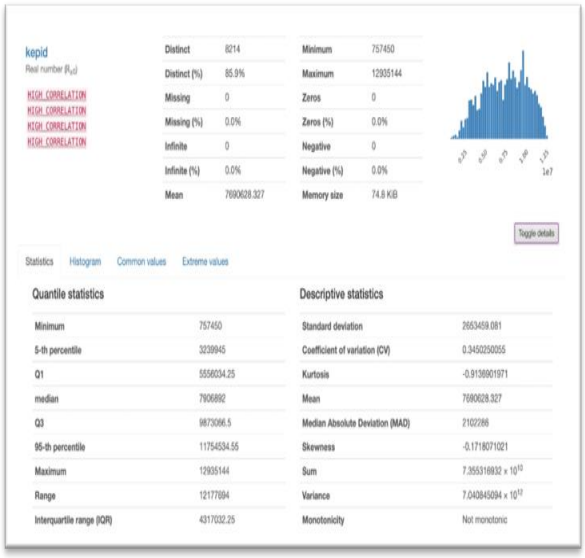


Figure 5:Variable Statistics-2

Figure 6:Variable Statistics-3

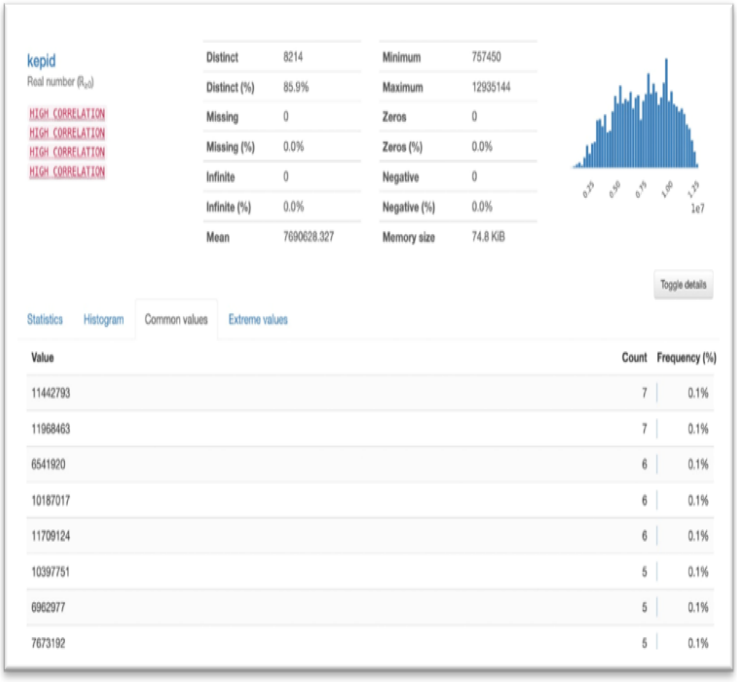
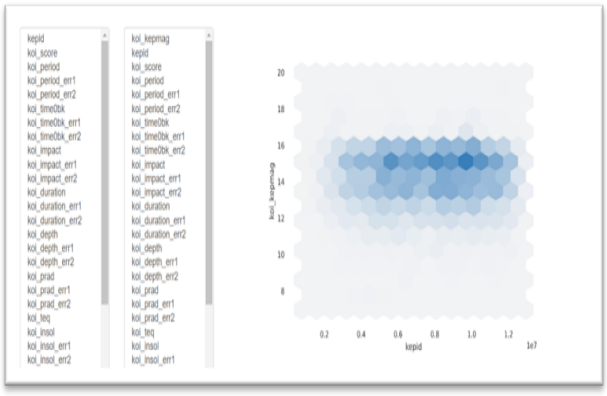


Figure 7:Variable Statistics-4

Interactions

Figure 8:Interactions



Correlations

Spearman’s p correlation

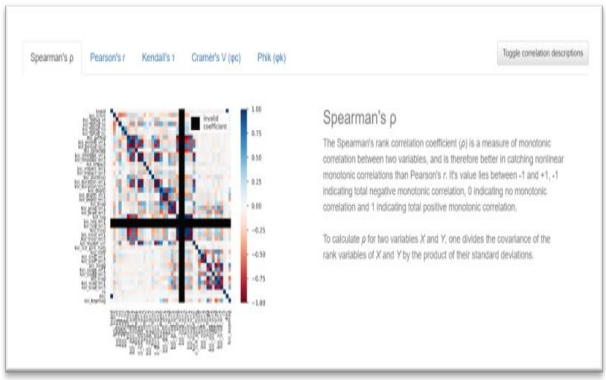


Figure 9:Spearman’s p correlation

Pearson’s r correlation

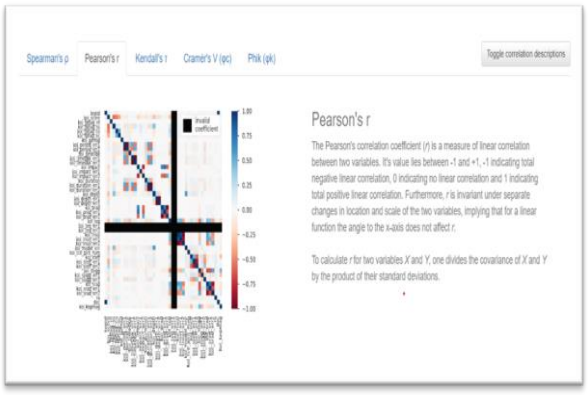


Figure 10:Pearson’s r correlation

Kendall’s T correlation

Figure 11: Kendall’s T correlation

Cramer’s V correlation

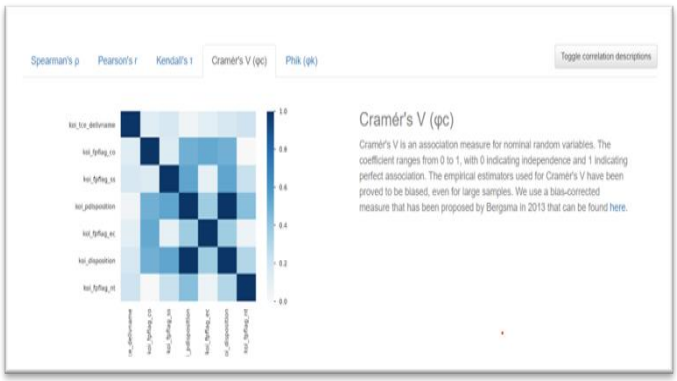


Figure 12: Cramer’s V correlation

Phik correlation

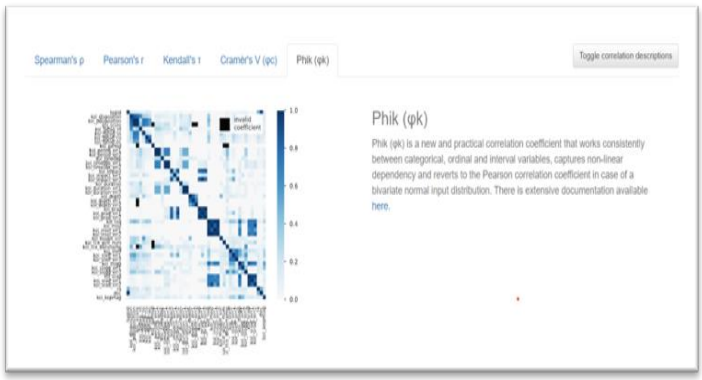


Figure 13: Phik's correlation

Feature engineering is an expensive and time-consuming process, but a necessary process to make data neat. Most of the time, feature engineering can be a manual process, but it can be automated. So that the model development can be expedited much better and efforts can be channelized into other activities, and other value-added things can be brought into the project and customer benefits perspective.

Sampling of training and testing model.

Generally, data used here is usually split into training data and test data. The training set contains a known output, and the model learns on this data in order to be generalized to other data later on. We have the test dataset (or subset) to test our model's prediction on this subset. To keep thing ideal, we allocate 80% of total data to training and rest of 20% of data for testing.

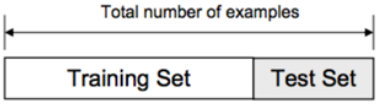


Figure 14:Division of Dataset

C. Training Model

Training a machine learning (ML) model is a process in which a machine learning algorithm is fed with training data from which it can learn. ML models can be trained to benefit businesses in numerous ways, by quickly processing huge volumes of data, identifying patterns, finding anomalies or testing correlations that would be difficult for a human to do unaided. Model training is the primary step in machine learning, resulting in a working model that can then be validated, tested and deployed. The model's performance during training will eventually determine how well it will work when it is eventually put into an application for the end-users. Both the quality of the training data and the choice of the algorithm are central to the model training phase. In most cases, training data is split into two sets for training and then validation and testing.

The selection of the algorithm is primarily determined by the end-use case. However, there are always additional factors that need to be considered, such as algorithm-model complexity, performance, interpretability, computer resource requirements, and speed. Balancing out these various requirements can make selecting algorithms an involved and complicated process. Training a model requires a systematic, repeatable process that maximizes utilization of available training data.

Classification is the process of differentiating and categorizing the types of information presented in the dataset into the discrete values. In other words, it is the "sorting out" part of the operation. First, The algorithm labels the data according to the input samples on which the algorithm was trained. It recognizes certain types of entities, looks for similar elements and couples them into relevant categories. The algorithm is also capable of detecting anomalies in the data.

1) Algorithm Choice

Model selection is a critical process since choosing one of the models as the final model to address the given business problem. A specific algorithm is selected for training to learn underlying data insights based on data and a given task. To yield maximum performance over test data, the model is iterated over a different combination of hyperparameters to identify the right set of hyperparameters. A wide range of supervised learning algorithms are available, each with its strengths and weaknesses. There is no single learning algorithm that works best on all supervised learning problems. There are four major issues to consider in supervised learning

a) Bias-variance tradeoff

A first issue is the tradeoff between bias and variance. Imagine that we have available several different, but equally good, training data sets. A learning algorithm is biased for a particular input x if, when trained on each of these data sets, it is systematically incorrect when predicting the correct output for x. A learning algorithm has high variance for a particular input x if it predicts different output values when trained on different training sets. The prediction error of a learned classifier is related to the sum of the bias and the variance of the learning algorithm.[4] Generally, there is a tradeoff between bias and variance. A learning algorithm with low bias must be "flexible" so that it can fit the data well. But if the learning algorithm is too flexible, it will fit each training data set differently, and hence have high variance. A key aspect of many supervised learning methods is that they are able to adjust this tradeoff between bias and variance (either automatically or by providing a bias/variance parameter that the user can adjust).

b) Function complexity and amount of training data

The second issue is of the amount of training data available relative to the complexity of the "true" function (classifier or regression function). If the true function is simple, then an "inflexible" learning algorithm with high bias and low variance will be able to learn it from a small amount of data. But if the true function is highly complex (e.g., because it involves complex interactions among many different input features and behaves differently in different parts of the input space), then the function will only be able to learn with a large amount of training data paired with a "flexible" learning algorithm with low bias and high variance.

c) Dimensionality of the input space

A third issue is the dimensionality of the input space. If the input feature vectors have large dimensions, learning the function can be difficult even if the true function only depends on a small number of those features. This is because the many "extra" dimensions can confuse the learning algorithm and cause it to have high variance. Hence, input data of large dimensions typically requires tuning the classifier to have low variance and high bias. In practice, if the engineer can manually remove irrelevant features from the input data, it will likely improve the accuracy of the learned function. In addition, there are many algorithms for feature selection that seek to identify the relevant features and discard the irrelevant ones. This is an instance of the more general strategy of dimensionality reduction, which seeks to

map the input data into a lower-dimensional space prior to running the supervised learning algorithm.

d) Noise in the output values

A fourth issue is the degree of noise in the desired output values (the supervisory target variables). If the desired output values are often incorrect (because of human error or sensor errors), then the learning algorithm should not attempt to find a function that exactly matches the training examples. Attempting to fit the data too carefully leads to overfitting. You can overfit even when there are no measurement errors (stochastic noise) if the function you are trying to learn is too complex for your learning model. In such a situation, the part of the target function that cannot be modeled "corrupts" your training data - this phenomenon has been called deterministic noise. When type of noise is present, it is better to go with a higher bias, lower variance estimator. In practice, there are several approaches to alleviate noise in the output values such as early stopping to prevent overfitting as well as detecting and removing the noisy training examples prior to training the supervised learning algorithm. There are several algorithms that identify noisy training examples and removing the suspected noisy training examples prior to training has decreased generalization error with statistical significance.

2) Gradient Boosting Modeling

The boosting algorithm creates new weak learners (models) and sequentially combines their predictions to improve the overall performance of the model. For any incorrect prediction, larger weights are assigned to misclassified samples and lower ones to samples that are correctly classified. Weak learner models that perform better have higher weights in the final ensemble model. Boosting never changes the previous predictor and only corrects the next predictor by learning from mistakes. Since Boosting is greedy, it is recommended to set a stopping criterion such as model performance (early stopping) or several stages (e.g. depth of tree in tree-based learners) to prevent overfitting of training data. The first implementation of boosting was named AdaBoost (Adaptive Boosting). Based on current data information, we have a limited number of features; therefore, a tree-based non-parametric classifier (i.e, Random Forest and Gradient Boosting Model) works well comparatively. We will still be looking to apply another algorithm to make the conclusion more assertive.

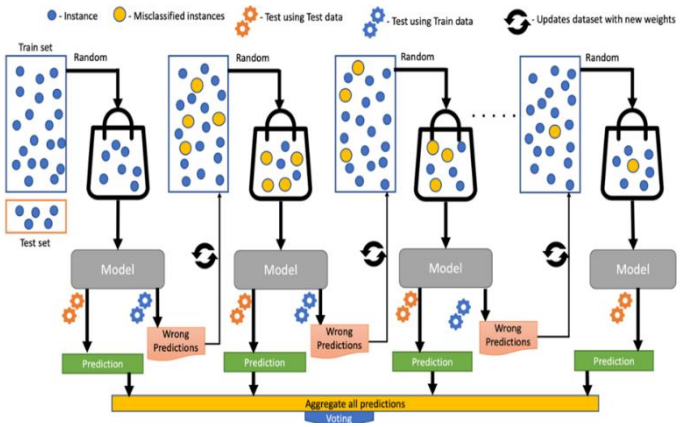


Figure15 15: Working Mechanism of Gradient Boosting Model

The boosting algorithm creates new weak learners (models) and sequentially combines their predictions to improve the overall performance of the model. For any incorrect prediction, larger weights are assigned to misclassified samples and lower ones to samples that are correctly classified. Weak learner models that perform better have higher weights in the final ensemble model. Boosting never changes the previous predictor and only corrects the next predictor by learning from mistakes. Since Boosting is greedy, it is recommended to set a stopping criterion such as model performance (early stopping) or several stages (e.g. depth of tree in tree-based learners) to prevent overfitting of training data. The first implementation of boosting was named AdaBoost (Adaptive Boosting).

$$F_i(x) = F_{i-1}(x) + f_i(x)$$

Gradient boosting is a special case of boosting algorithm where errors are minimized by a gradient descent algorithm and produce a model in the form of weak prediction models e.g. decision trees. The major difference between boosting and gradient boosting is how both the algorithms update model (weak learners) from wrong predictions. Gradient boosting adjusts weights by the use of gradient (a direction in the loss function) using an algorithm called Gradient Descent, which iteratively optimizes the loss of the model by updating weights. Loss normally means the difference between the predicted value and actual value. For regression algorithms, we use MSE (Mean Squared Error) loss as an evaluation metric while for classification problems, we use logarithmic loss.

$$w = w - \eta \nabla w$$

$$\nabla w = \frac{\partial L}{\partial w} \quad where\ L\ is\ loss$$

d

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

Loss function ↙ ↘ Regularization

Gradient boosting uses Additive Modeling in which a new decision tree is added one at a time to a model that minimizes the loss using gradient descent. Existing trees in the model remain untouched and thus slow down the rate of overfitting. The output of the new tree is combined with the output of existing trees until the loss is minimized below a threshold or specified limit of trees is reached. Additive Modeling in mathematics is a breakdown of a function into the addition of N subfunctions. In statistical terms, it can be thought of as a regression model in which response y is the arithmetic sum of individual effects of predictor variables x.

3) Benchmark

Benchmarking on training data is the process of evaluating a machine learning model's performance on the same data that was used to train the model. This is typically done to ensure that the model is not overfitting to the training data, which would result in poor performance on new, unseen data. By comparing the model's performance on the training data to its performance on a separate, "held out" dataset, it is

possible to get a sense of how well the model will generalize to new data. We have benchmarked our data on majority of algorithm.

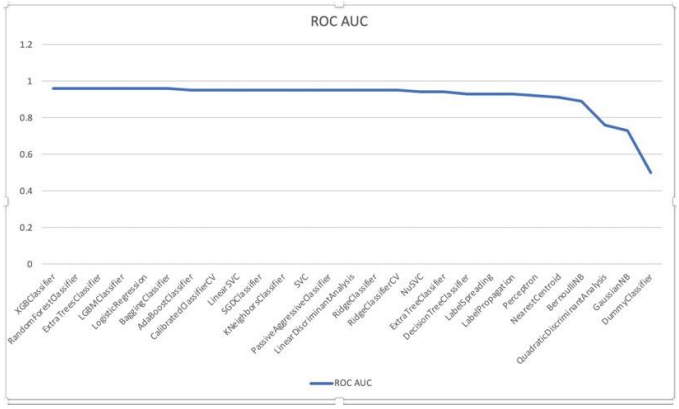


Figure 16: ROC AUC

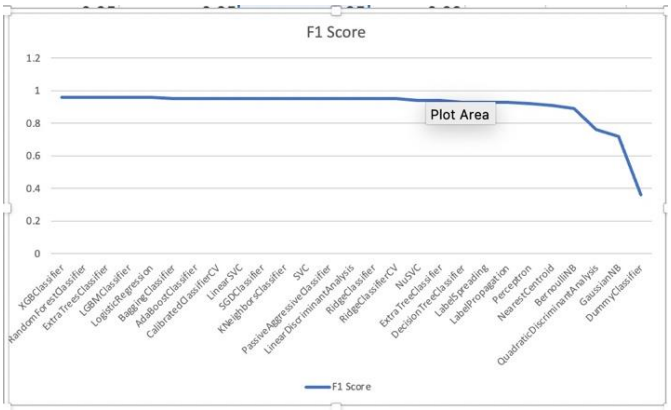


Figure 17:F1 Score

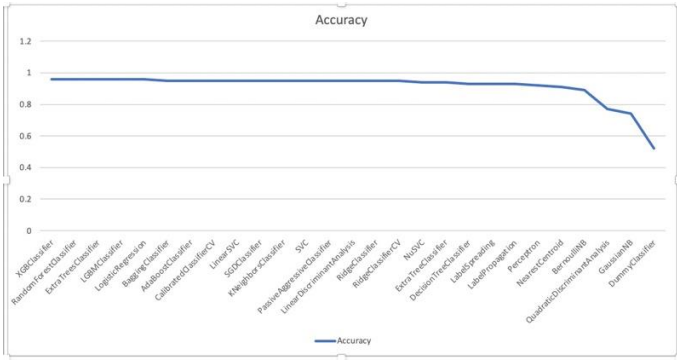


Figure 18: Accuracy

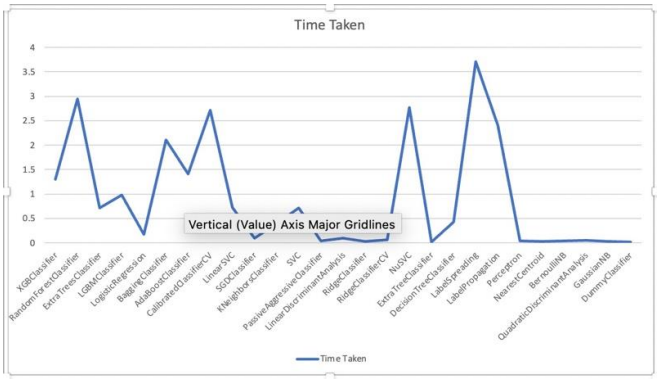


Figure 19: Time Taken

4) Hyper Parameter Tunning

Hyperparameter tuning is the process of adjusting the values of the hyperparameters for a machine learning model in order to improve its performance on a particular dataset. Hyperparameters are the settings that are not learned from the data during training, and they control the overall behavior of the model. Some examples of hyperparameters include the learning rate for a neural network, the depth of a decision tree, and the regularization coefficient for a linear model.

Tuning the values of the hyperparameters can be a challenging task, because there is often no straightforward way to know what the optimal values should be. One common approach is to use a grid search, in which a range of possible values for each hyperparameter is specified, and the model is trained and evaluated using all possible combinations of these values. This can be computationally expensive, but it can provide valuable insights into which combinations of hyperparameters work best for a given dataset. In general, hyperparameter tuning is an important step in the machine learning process, because it can have a significant impact on the performance of the model. By carefully adjusting the values of the hyperparameters, it is possible to improve the model's accuracy, precision, and other performance metrics.

Randomized cross-validation search is a method for hyperparameter tuning that involves training a model multiple times, each time with different values for the hyperparameters. The values of the hyperparameters are selected randomly, rather than using a grid or other predetermined values. This can be an efficient way to search for good values for the hyperparameters, because it avoids the need to train the model on all possible combinations of hyperparameters. To perform a randomized cross-validation search, the first step is to define a range of possible values for each hyperparameter. These values are typically chosen based on the model type and the characteristics of the dataset. For example, the range of values for the learning rate of a neural network might be from 0.01 to 0.001, while the range of values for the regularization coefficient of a linear model might be from 0.001 to 0.1.

Once the ranges of values for each hyperparameter have been defined, the next step is to train the model multiple times, each time with a different set of randomly chosen values for the hyperparameters. For each set of hyperparameter values, the model is trained on a portion of the data (the training set), and its performance is evaluated on a separate portion of the data (the validation set). This process is repeated for a specified number of iterations, and at the end, the performance of the model is averaged across all iterations. The randomized cross-validation search can help identify the values of the hyperparameters that result in the best performance on the dataset. This can be useful for improving the accuracy and other performance metrics of the model. Additionally, because the values of the hyperparameters are chosen randomly, rather than using a predetermined grid, the search is more likely to find good values that may not have been explored using a grid search.

Hyperparameters	Values
Colsample_bytree	[0.7, 0.3]
Gamma	[0, 0.5]
Learning Rate	[0.03, 0.3]
Max Depth	[2, 6]
Number of Estimators	[100, 150]
Subsample	[0.6, 0.4]

D. Evaluation Measure and Results

Model evaluation is an important step in the machine learning process, and it is especially important in classification tasks. The goal of model evaluation is to measure the performance of a machine learning model on a particular dataset, and to use this information to help improve the model and make more accurate predictions on new data. There are many different metrics that can be used to evaluate a classification model, and the choice of metric will depend on the specific characteristics of the dataset and the goals of the analysis. Some common metrics for evaluating classification models include:

Performance Measure	Value
Accuracy	0.96
Recall	0.96
Precision	0.96
F1 score	0.96

1. Accuracy: This is the number of correct predictions made by the model, divided by the total number of predictions. It is a simple and intuitive metric, but it can be misleading if the dataset is imbalanced (i.e., if there are more examples of one class than another).

$$\text{Accuracy} = \frac{(TP + TN)}{(TP + FP + TN + FN)}$$

2. Precision: This is the number of true positives (correct predictions of the positive class) divided by the total number of predicted positives. It is a measure of the model's ability to avoid false positives (incorrect predictions of the positive class).

$$\text{Precision} = \frac{TP}{(TP + FP)}$$

3. Recall: This is the number of true positives divided by the total number of actual positives. It is a measure of the model's ability to identify all instances of the positive class.

$$recall = \frac{TP}{TP + FN}$$

4. F1 score: This is the harmonic mean of precision and recall. It is a balanced measure of the model's performance that takes into account both precision and recall.

$$F - score = \frac{2}{1/precision + 1/recall}$$

In addition to these metrics, it is also common to use a confusion matrix to evaluate a classification model. A confusion matrix is a table that shows the number of true positives, true negatives, false positives, and false negatives for a model, and it provides a detailed breakdown of the model's performance on each class.

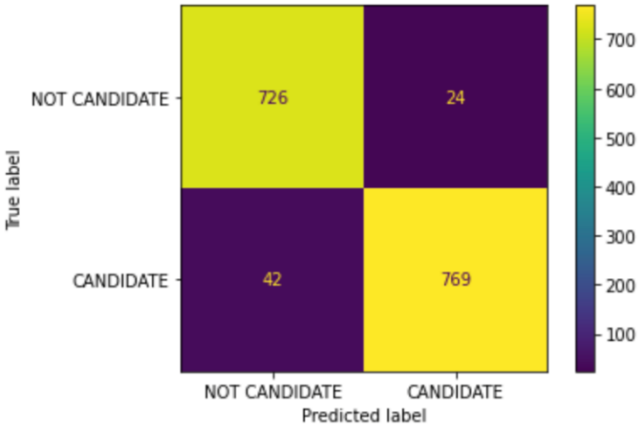


Figure 20: Confusion Matric of our Model

Overall, model evaluation is a crucial step in the machine learning process, and it is important to choose the appropriate evaluation metrics for your specific classification task. By carefully evaluating the performance of model, we can identify areas for improvement and make more accurate predictions on new data.

E. Model Explainability and Feature Importance

Model explainability and feature importance are closely related concepts in the field of machine learning. Model explainability refers to the ability of a machine learning model to provide an interpretation of its predictions, while feature importance refers to the degree to which each input feature (e.g., a particular predictor variable) contributes to the model's predictions.

Model explainability is important for a number of reasons. First, it can help people understand how a model is making its predictions, which can be valuable for building trust in the model and its outputs. Second, it can provide insight into the underlying relationships and patterns in the data that the model has learned, which can be useful for debugging and improving the model. Finally, it can help people identify potential biases or errors in the model, and take steps to address them.

There are many different techniques for explaining machine learning models, including visualizations, feature importance scores, and surrogate models. Visualizations, such as decision trees and partial dependence plots, can help people understand the structure of the model and the relationships between the input features and the output predictions. Feature importance scores, such as permutation importance and SHAP values, can provide a numerical measure of the contribution of each input feature to the model's predictions. Surrogate models, such as linear models and decision trees, can be used to approximate the behavior of a complex model, and provide a simpler, more interpretable explanation of the model's predictions.

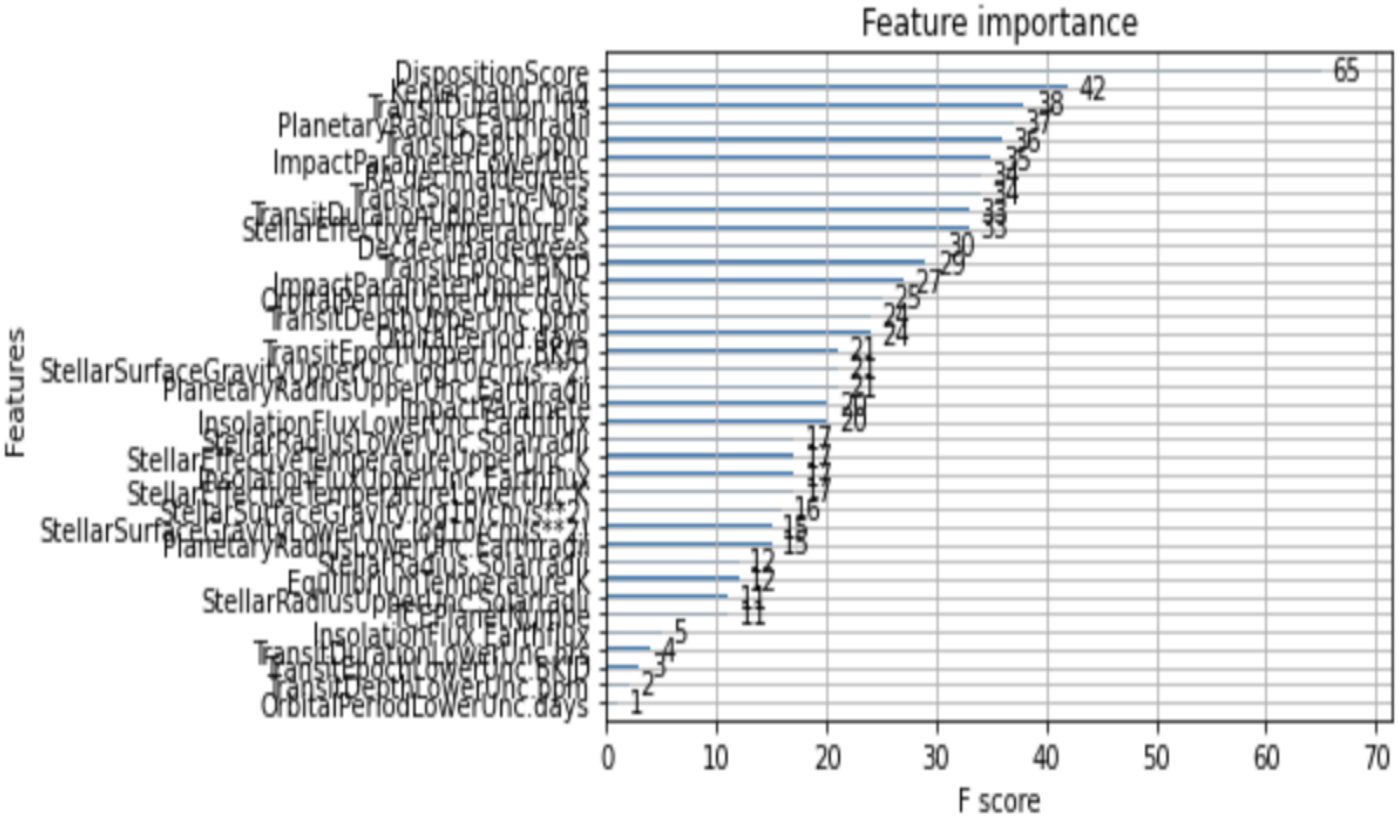


Figure 21: Feature Importance calculated by XG-Boost model

In summary, model explainability and feature importance are important aspects of machine learning that can help people understand and trust the predictions of a model and improve its performance. By providing an interpretation of the model's predictions, these techniques can help people make more informed decisions based on the outputs of the model.

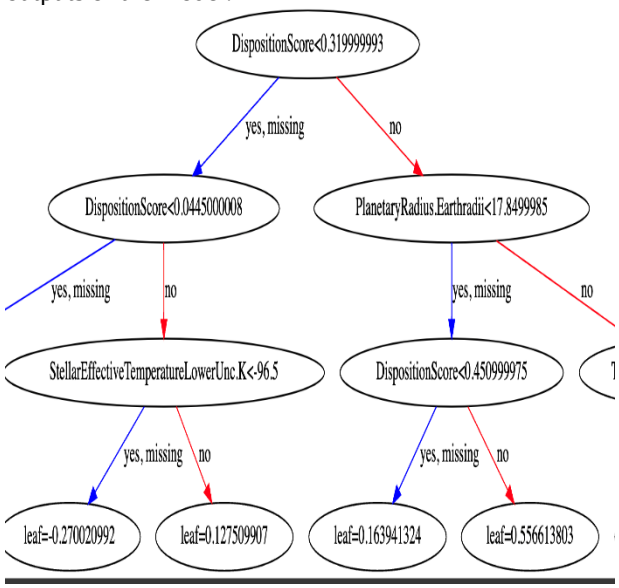


Figure 22: Explainable Decision Tree by XG-Boost

V. Relevance to the Course

Since we are attempting to categorize whether we have a potential exoplanet, we will use different Classifier models for this model and will use the one with the best result. Both estimators and the Gini Impurity index will be used. We will examine a few metrics for this model, but Accuracy will be our key concern. In order to solve this problem, we will use Supervised Machine Learning with two-Class Classifiers (exoplanet or not). Pre-processing of data is required, such as detecting null values, noise, and outliers. Following the cleaning of the data, data is divided into training and testing splits of 60% and 40%.

Machine learning algorithms from various families are considered and trained using training data, Using real information provided by NASA and Caltech or dataset on Kaggle or NASA's API to do some web scraping, and then we can train models to recognize potential exoplanet candidates.

VI. Impact

If we are able to determine whether or not one of our sightings was an exoplanet with greater than 90% accuracy. We can accomplish a lot with a model that is relatively easy to set up and rapid. With a dataset provided publicly, we worked on a project to identify exoplanets. Evidently, there are a few new worlds out there that we can all try to find.

ACKNOWLEDGMENT

This research and project will be more valuable in studying the various techniques and algorithms of machine learning and Mistakes in the documentation are our obligation, and they should not jeopardize the credibility of the respected experts. We would like to express our gratitude to Prof. Chandrashekhar Mukherjee and the Computer Engineering Department, San Jose State University for sharing their insights, reviews, and comments on various phases of the project

REFERENCES

[1] Timothy Bovaird, Charles H. Lineweaver, Exoplanet predictions based on the generalized Titius–Bode relation, Monthly Notices of the Royal Astronomical Society, Volume 435, Issue 2, 21 October 2013, Pages 1126–1138, https://doi.org/10.1093/mnras/stt1357

[2] https://exoplanetarchive.ipac.caltech.edu/docs/program_interfaces.html

[3] https://exoplanetarchive.ipac.caltech.edu/docs/API_keplerstellar_columns.html

[4] https://api.nasa.gov/

[5] https://api.nasa.gov/planetary/apod?api_key=”your_API_key”

[6] https://www.kaggle.com/datasets/nasa/kepler-exoplanet-search-results

[7] https://blog.google/technology/ai/hunting-planets-machine-learning/

[8] https://exoplanets.nasa.gov/what-is-an-exoplanet/overview/

[9] https://www.nasa.gov/sites/default/files/styles/full_width_feature/public/thumbnails/image/10-24_by_the_numbers_-_mission_stats.jpg

[10] https://exoplanets.nasa.gov/

[11] https://www.pnas.org/doi/10.1073/pnas.1304196111 use this

[12] https://exoplanetarchive.ipac.caltech.edu/docs/Kepler_KOI_docs.html