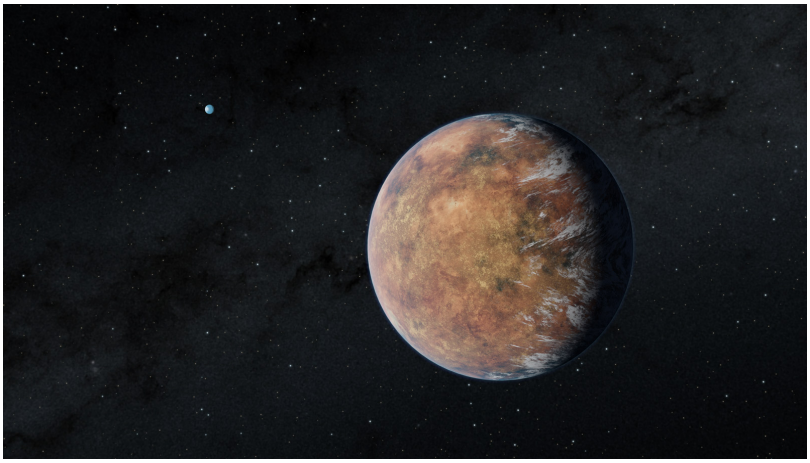
Kepler Exoplanet Exploration



Ayan Chowdhury, Anika Das, Gary Shetye, Grace Yang

Abstract

The Kepler space telescope, launched by NASA in 2009, had a mission to discover planets outside of our solar system, known as exoplanets. In our research, we used the flux data that measures light intensity collected from the Kepler telescope to classify if a distant star has an orbiting exoplanet or not. Identification of an exoplanet can then be used to run further experiments to determine the planet's characteristics and potential habitability. In order to identify exoplanets, we used k-Nearest Neighbors (KNN), logistic regression, and recurrent neural networks (RNN) to classify the given stars from NASA's public Kepler dataset. Data preprocessing involved employing feature engineering techniques such as scaling with Robust Scaler and synthetic minority sampling to balance the classes. We hyperparameter tuned each of the algorithms with a focus on achieving the highest prediction accuracy. Our KNN model performed the best on the test data, achieving 98.3% accuracy.

Introduction & Background

The original Kepler space telescope launched in 2009, and had a mission to collect flux intensity readings in order to detect transit events. A transit event occurs when an exoplanet orbits around its host star, passing between Earth and the host star. From previous space explorations, there is evidence for 3 main exoplanet distinctions: gas giants, hot-super Earths in short period orbits, and ice giants.

Historically, exoplanets have been difficult to study and identify due to their sheer distance from the solar system. However, machine learning algorithms working in tandem with Kepler data has helped astronomers identify stars with exoplanets for further explorations. Traditional exoplanet identification methods include the wobble method, direct imaging, and gravitational microlensing (Jin). The wobble method utilizes the Doppler shift in a star's light flux caused by its planets, direct imaging uses extraterrestrial telescopes to capture images of exoplanets, and the gravitational microlensing method analyzes background light distortion (Jin). However, limitations of these methods include significant time, money, and labor investments which machine learning can circumvent.

There have been prior attempts to perform identification of exoplanets using light flux data. Among the previous attempts, one paper titled “Identifying Exoplanets with Machine Learning Models: A Preliminary Study,” published in 2022, aimed to illustrate the efficiency of machine learning algorithms over traditional methods with the Kepler data. The algorithms used in the paper were decision trees, random forests, Naive Bayes, and neural networks with accuracies of 99.06%, 92.11%, 88.50%, and 99.79%, respectively (Jin). Apart from identification, machine learning has been used to sift out false positives using deep learning with neural networks. Among these attempts, one paper titled, “Identifying Exoplanets with Deep Learning 2: Two New Super-Earths Uncovered by a Neural Network in K2 Data” (Dattilo), explored the NASA Kepler data to identify exoplanets and to rule out false positive light flux signals mimicking transiting planet signals. The neural network created was a modified network previously used to identify exoplanets in a different Kepler mission campaign, and had an 98% accuracy in identifying exoplanets and false positives. Additionally, from this model, the algorithm was able to identify and validate two previously unknown exoplanets, which solidifies a previously mentioned point in which machine learning algorithms can streamline identification.

Data Analysis

Our data describes the change in flux of several thousand stars. In addition to light intensity information, each star in the dataset is assigned a binary label of “1” or “2”. A label of 1 indicates that the star does not have any exoplanets in orbit while a label of 2 indicates that the star is confirmed to have at least one exoplanet in orbit.

When the flux of a star is measured over several months or years, a periodic ‘dimming’ of the flux may be observed if there is an exoplanet orbiting around the star. Figure 1 illustrates this concept, where the star’s emitted brightness decreases as an exoplanet orbits its star. The training dataset used describes the light flux for over 5,000 stars with potential exoplanets where each feature of the data is the light flux at a specific time during Kepler's K2 Campaign 3 mission.

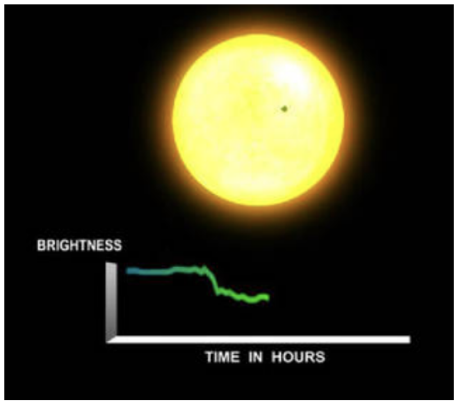


Fig. 1: Light emitted from a star will have a slight decrease when an exoplanet orbits around it. (Kepler:Overview)

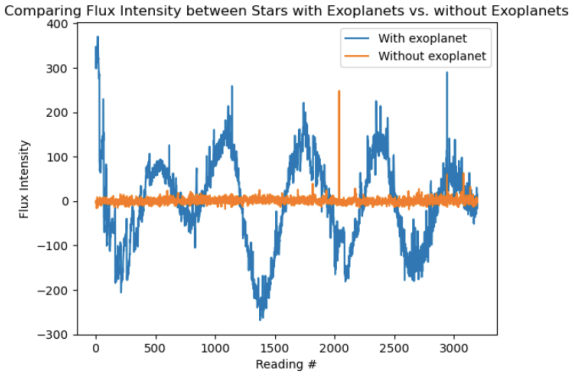


Fig. 2: Flux intensity data for arbitrary stars

Figure 2 visualizes the change in flux intensity over all the flux readings for two stars. One star has a label of 1, indicating that it does not have an exoplanet, while the other has a label of 2, indicating that it has an exoplanet. For the star with an exoplanet, the light intensity changes sinusoidally, illustrating how the exoplanet orbits its star. The light intensity dims as an exoplanet between the star and the telescope, and the intensity increases again when the planet moves out of the transit path.

Figure 3 illustrates the difference of the standard deviation of each class of star with outliers removed. Thus, removing outliers removes biases across the dataset. The standard deviation of the flux intensity for stars with exoplanets is 311.23 while the standard deviation for stars without exoplanets is 262.53, indicating that data for stars with exoplanets deviate further from the mean when compared to the stars without exoplanets.

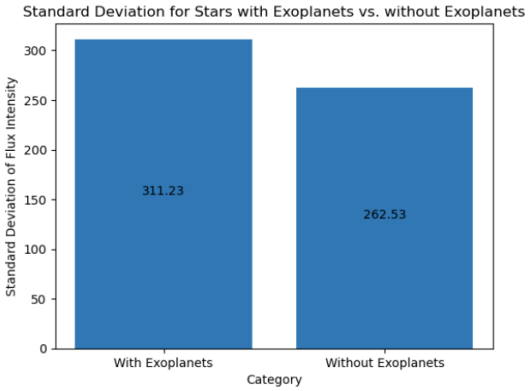


Fig 3: Standard Deviation of the data with outliers taken out

At a first glance, the data is imbalanced. The training data has 5,087 instances and 3,198 features. 37 out of 5,087, or 0.72% of stars in the training data have a label of 2, indicating that they have exoplanets. The testing data has 570 instances and 3,198 features. 5 out of 570, or 0.87% of the stars in the training data have exoplanets. Although this data is imbalanced, this distribution is expected as there are not many stars with confirmed exoplanets.

The first attempts to utilize the raw data in our KNN implementation yielded low accuracy because there are severe flux intensity outliers. For example, some stars had flux readings greater than 3,000,000 while the majority of the other stars' readings fell between 0 and 100. We deduced that this range is attributed to vast differences in the size of stars. Stars can vary in size from the smallest neutron stars with a 12 mile diameter to supergiant stars that are over a billion miles in diameter. Hence, we chose to normalize our data using the robust scaling technique. Robust scaling is useful for data with extreme outliers which may have distorted traditional scaling techniques. First, we transposed the dataframe so that each star was represented as a feature and the flux reading data were represented as columns. Then, we applied the robust scaler to each column. The scaler eliminates outliers from each feature, then calculates the median and IQR for the remaining values. Finally, the robust scaler scaled each flux reading. Reducing the data effectively made the data more normal for input into the algorithm. We then transposed the data back to reflect the original structure of the dataframe.

Next, we used SMOTE to create a more balanced training set. As stated above, the original training data set contained only 37 instances of stars with exoplanets and 5,050 stars without exoplanets. SMOTE selects an instance from the minority class of a given dataset, then finds that instance's 5 nearest neighbors, and finally creates a new instance of the given class with similar features. By utilizing SMOTE, we were able to create 5,013 new instances of stars with exoplanets, effectively creating a larger, more balanced training dataset. We leveraged SMOTE for our testing data as well, since there were only 5 instances of stars with exoplanets and 570 stars without. We created 563 new instances of stars with exoplanets to balance the test set as well.

The final data manipulation we performed was standardizing the instance labels. The original labels in the dataset were "2" and "1". 2s labeled stars with exoplanets and 1s labeled stars without exoplanets. For ease of coding, we relabeled the stars with exoplanets to "1" and stars without exoplanets to "0".

Methods

Feature Selection

The features used in the models are the light flux measurements for each star. For each of our models, we used the full 3,197 features for training and validation. In our RNN model, we created windows for each star instance to capture the time-series nature of the flux values. Each window was used as input to the Sequential layer, and then encoded into the Each window was grouped with its corresponding label and passed into the RNN architecture.

Machine Learning Models Employed and Rationale

We chose to employ KNN, logistic regression, and recurrent neural networks (RNN) as the machine learning classification algorithms. The KNN model was chosen as it is a robust algorithm when used for classification and does not rely on the assumption that the data is normally distributed. However, one assumption for the KNN model is that similar classifications exist in close proximity to each other, which is true for our data post-scaling.

We implemented our own version of KNN utilizing the Euclidean and Manhattan distance functions to measure the similarity between two vectors being tested at any given time. These were tested using hyperparameter tuning as detailed in the subsequent section. Our KNN implementation accepts a given instance from the validation dataset and finds the k nearest neighbors to that instance, as represented by a vector. The majority class from the k neighbors is then selected as the predicted class for the validation instance. By iterating through all instances in the validation datasets, we were able to generate a list of predicted classes, which we can then calculate performance metrics for. It should be noted that all training and validation was performed with k-fold validation with 5 folds.

The logistic regression model was implemented using scikit-learn, and was chosen as it is a statistical model that models the probability of binary outcomes. Assumptions for this model are that the observations are independent of each other and have no linear correlation. The scikit-learn logistic regression fit method is used to train the model on a given dataset. The process includes finding parameters for the logistic function that best fit the data. The first step involved in the fit method of logistic regression is to randomly initialize the models parameters (including the weights and bias terms) before training. Next, the predicted probabilities of an instance of the positive class (having exoplanets) are calculated by passing the input features to a sigmoid function. The model then calculates the loss using the logistic loss function. After that, an optimization algorithm (such as gradient descent) is used to update the model's parameters based on the gradient of the loss function. The updated parameters are used to calculate the predicted probabilities and the loss until convergence. Finally, the trained model is evaluated on the test dataset and its accuracy is assessed.

The scikit-learn logistic regression predict method can be used to make predictions on new data points after a logistic regression model on a given dataset. It takes in a set of input features and outputs a predicted class label, which is either a 0 or 1 in the case of binary classification. The general steps for predicting y labels in logistic regression are as follows. First, the predicted probability of the positive class is calculated for the new input features using the trained parameters of the logistic regression model. Then the probability is converted to a class label, by applying a threshold; if the predicted probability is

greater than the threshold, the input is classified as the positive class (1), otherwise it is the negative class (0).

For our RNN model, we looked to perform a binary sequence classification task based on windows of time-series flux values for each star instance. RNN's are particularly useful for sequential data as it maintains a memory of past inputs when predicting next outcomes, stored and updated in a hidden state. The model takes in an input vector representing the current flux value in an instance, and a hidden state vector that represents the memory of previous flux values. The RNN uses weights to transform the input into a new hidden state vector, which is passed to the next step of the training. This creates a network where past flux values are used to predict future ones. This lies in accordance with the transitory periods of planets, where the flux of a star during an exoplanet transit is time-dependent on the previous location of the planet.

As training RNN models take a long time, we used a random 50% of training data in our analysis. We looked to transform our training data into blocks of window size n, described further in the hyperparameter section. Thus, each star's flux values were split up into n windows, and their labels were padded such that each window still had its original label. After this preprocessing, we initialized the RNN architecture with the following layers. Firstly, the Sequential layer, which allows the RNN to process each instance one window at a time. Next, we initialized the long-term short-memory layer, or LSTM. This layer takes in a parameterized number of memory cells, which are used to retain information from previous steps in a sequence. The LSTM layer requires careful configuration of the input size of each window and time steps, but is vital to the underlying mechanism of treating our flux values as time series data. Finally, we add our Dense output layer, which connects every node in the previous layer to the final output. We use the sigmoid function in the Dense layer to return a value between 0-1 as we are performing a binary classification task. After this, we initialized a Stochastic Gradient Descent (SGD) optimizer with a parameterized learning rate, and then compiled the model using the binary cross-entropy loss function, SGD optimizer, and accuracy for metrics. Finally, we fit the model on the preprocessed features and outcomes.

Hyperparameter Tuning

Each algorithm we employed has parameters that can have significant effects on the prediction performance. For this reason, we trained each model with permutations of parameters in an attempt to maximize the prediction accuracy among all three algorithms.

The parameters that we tested in the KNN model were the distance function utilized and the number of neighbors compared against. The two distance functions tested were Euclidean and Manhattan distance. Euclidean distance is the square root of the sum of squared distances between points of the vectors being compared. Manhattan distance is the sum of the absolute value of the difference between points of the vectors being compared. The number of neighbors parameters is simply the number of nearest instances that the algorithm scores in order to vote among all k neighbors for the majority class. We tested 3, 5, and 7 neighbors for this algorithm. Models were trained and validated using all permutations of the parameters with a focus on maximizing accuracy. We determined that the best parameters for KNN are euclidean distance and 7 neighbors. The model predicted classes with a 75.2% accuracy with these two parameters.

The hyperparameters that we tested for the logistic regression model were the penalty term (L1 and L2) and solvers (Liblinear and Saga). We tried permutations of these penalty and solver methods. The penalty term is used to regularize the model and prevent overfitting. L1 regularization, also known as Lasso Regression, is a regression model that adds an "absolute value of magnitude" coefficient as a penalty term to the loss function. In this case, the penalty term is proportional to the sum of the absolute values of the model's weights. L2 regularization, also known as Ridge regularization, is a regression model that adds a "squared magnitude" coefficient as a penalty term to the loss function. The added term is proportional to the sum of the squared values of the model's weights. In terms of solvers, Liblinear implements a coordinate descent algorithm; it is most commonly used for problems with a large number of input features and recommended for solving large-scale classification problems with high dimension datasets. Stochastic Average Gradient descent (SAGA) solver is a modified version of the SAG method; the SAG method optimizes the sum of numbers of smooth convex functions. The SAG method's iteration cost is independent of the number of terms used in the sum, but by collecting a memory of previous gradient values, the SAG method accomplishes a faster convergence rate than other stochastic gradient methods. SAGA is a stochastic gradient descent algorithm which also supports non-smooth loss functions and has a better theoretical convergence compared to SAG. Therefore, SAGA is the optimum solver for sparse multinomial logistic regression.

For our RNN, we tuned the parameters of number of LSTM cells, number of training epochs, and the learning rate. First, the number of LSTM cells is variable, as more memory cells could mean overfitting of the data, but less wouldn't accurately capture the time-series nature of the fluxes. We also varied the number of training epochs to find the optimal balance between under/overfitting the data. Finally, we looked to manipulate the learning rate to consider the line between overshooting the optimal values and getting stuck at local minima. As the training of RNN's takes a considerable amount of time due to their use of the hidden memory layers, we tempered our hyperparameters to account for < 30 min training time.

Cross-validation Methods for Training

In order to avoid overfitting and represent our data better, we used 5-fold cross-validation. This involved splitting our training data into 5 folds, and using one of them as a validation set and the other 4 as training sets. We did this 5 times, and for each iteration, one of the 5 folds was used as the validation set, and the other 4 were used as training. We then calculated the average accuracy across each iteration to determine our overall validation accuracy. We used this 5-fold cross-validation method for logistic regression and KNN.

Validation and Test Set

Using our best validation hyperparameters, we ran our models on the testing data, as reported in the results section. Our best validation hyperparameters for KNN were

Results/Analysis

All training on logistic regression and KNN models were run using 5-fold cross validation. The training data consists of **10,100** rows (star instances) and **3,197** columns (flux values). The testing data consists of **1,130** rows (star instances) and **3,197** columns (flux values).

Training Results

Method	Hyperparameters	Best Hyperparameters + Average Accuracy on Validation	Testing Metrics Using Best Hyperparameters
K-Nearest Neighbors (KNN)	**Distance functions**: Euclidean, Manhattan **# of Neighbors**: 3, 5, 7	**Distance**: Manhattan **# of Neighbors**: 5 **Average Accuracy:** 77.2%	Running with Manhattan, 5 nearest-neighbors: **Accuracy**: 98.3% **Precision**: 96.7% **Specificity**: 96.6% **Sensitivity**: 100% **F1-Score**: 98.3%
Logistic Regression	**Penalty**: L1, L2 **Solver:** Liblinear, Saga	**Penalty**: L1 **Solver**: Liblinear **Average Accuracy**: 94.8%	Running with L1, Liblinear: **Accuracy**: 57.3% **Precision**: 78.6% **Specificity**: 94.5% **Sensitivity**: 20.1% **F1-Score**: 32.1%
Recurrent Neural Networks (RNN)	**Training Epochs:** [3, 5, 7] **Learning rate:** [0.01, 0.001] **LSTM cells:** [3, 7]	**Training epochs**: 5 **Learning rate**: 0.01 **LSTM cells**: 7 **Average Accuracy**: 59.1%	Running with 5 training epochs, learning rate of 0.01, 7 LSTM cells: **Accuracy**: 36.1% **Precision**: 36.7% **Specificity**: 36.2% **Sensitivity**: 36.0% **F1-Score**: 36.3%

After running KNN with the best hyperparameters, we achieved a high accuracy of 98.3%, a specificity of 96.6% and precision of 96.7% as well. Our sensitivity was at 100% and our F-1 score was 98.3%. Overall, our KNN did very well in terms of metrics. This may be because KNN works well with discrete value data with high dimensionality because it can handle categorical variables without requiring them to be encoded into a numerical representation. The KNN algorithm can compute the distance between categorical variables using different distance measures. In our case, Manhattan distance ended up being the best distance measure.

The Logistic Regression method did not perform as well as KNN. After running Logistic Regression on our best hyperparameters, we got an accuracy of 57.3%, a precision of 78.3%, a specificity of 94.5%, a sensitivity of 20.1%, and an F1 score of 32.1%. This poor performance may in part be due to the fact that logistic regression is prone to overfitting when dealing with high dimensionality data. Instead of selecting certain features to be used in the model, we are using all of the flux values. The model may be memorizing the specific pattern of each training instance, contributing to overfitting. Our high specificity scores were also a bit unexpected, this model was highly adept at minimizing false negatives (most stars with exoplanets were identified as such). This could be due in part to our use of SMOTE, which may have produced similar synthetic samples that our model could pick up on.

Finally, our RNN model did not perform to expectations. After running it on our best hyperparameters, we got an accuracy of 36.1%, precision of 36.7%, specificity of 36.2%, sensitivity of 36.0%, and F-1 score of 36.3%. This low accuracy could be attributed to a few things. For one, we did not include a Dropout layer in our model, which is commonplace in RNN's. This layer randomly sets some neurons' outputs to zero, forcing the model to learn different representations that are less sensitive to specific input patterns. Also, we did not experiment with changing the % of data used as well as the window size of each instance in our final reporting. Upon making both of these changes, we expect our RNN to perform better.

Conclusion

In this project, we conducted supervised learning on a dataset collected by the NASA Kepler Space Telescope. The dataset was used to predict exoplanets using light flux data using KNN, logistic regression, and RNN, which had accuracies of 98.3%, 56.7%, and 36.1% respectively. Our best model was the KNN, due to its ability to deal with high-dimensionality, continuous features without over/underfitting the data. The models were each properly tuned to have the highest average 5-fold accuracy. In the future, we would try using Kepler light curve data in tandem with the flux data to examine and analyze the transitory periods. Additionally, it would be interesting to focus on hyperparameter tuning the models to get the highest sensitivity without using the SMOTE function which we used to reproduce the imbalanced data. Overall, the application of machine learning in space exploration is extensive and its potential is tremendous in the field.

Author Contributions

AC took lead on development for all three models, contributed to the cross validation and results sections of the paper, and performed all data preprocessing like SMOTE and feature scaling in Python.

AD organized logistics by arranging meetings with Professor Rachlin to define our project's goals and contributed to the report and poster by creating the outline, writing the cross validation section, and researching and writing the rationale, hyperparameter tuning, and analysis sections for the KNN and Logistic Regression models.

GS performed data exploration, feature selection, and research on algorithms. GS created the visualizations used throughout and wrote sections pertaining to KNN.

GY laid out the frame of the report and wrote the majority of the paper. GY did not write the hyperparameter tuning, validation, and RNN rationale section. In addition, GY created and wrote the majority of the poster.

All authors reviewed and edited the research paper.

References

"Kepler: Overview." NASA, NASA, 13 Dec. 2019, https://www.nasa.gov/mission_pages/kepler/overview/index.html.

Dattilo, Anne, et al. "Identifying Exoplanets with Deep Learning. II. Two New Super-Earths Uncovered by a Neural Network in *k2* Data." *The Astronomical Journal*, vol. 157, no. 5, 2019, p. 169., https://doi.org/10.3847/1538-3881/ab0e12.

Jin, Yucheng, et al. "Identifying Exoplanets with Machine Learning Methods: A Preliminary Study." *International Journal on Cybernetics & Informatics*, vol. 11, no. 2, 2022, pp. 31–42., https://doi.org/10.5121/ijci.2022.110203.