

Identifying Exoplanets from Kepler Light Curves: Milestone 3

Michael Calderin*
University of Florida CAP5771
(Dated: April 4, 2025)

1. INTRODUCTION

Starting in 2009 and continuing for 9.6 years, NASA's Kepler/K2 missions set out to hunt for planets outside of our solar system [1]. A large part of the identification process was to record the flux (brightness) from stars in a small patch of our galaxy and detect when there is a dip in the flux. The dips are typically signs of a planet crossing the star. For a planet, these transits are periodic and can be fit to help estimate parameters such as the planet's size, distance from its star, etc. These fitting models also give better quantification for the transit depth (the amount that the flux falls during the transit) and other transit-related features.

However, not all transits are planets. Some stars come in pairs and can also have transits. These are known as eclipsing binaries. There are other false positives such as interference from the light of other stars. For a transit to be confirmed as a planet, there is typically a pipeline that requires additional observations and can take years. There are several planetary candidates that to this day have not been confirmed to be planets. NASA uses Robovetter, a decision tree, to automate the classification process. It distinguishes between candidates and false positives but does not make predictions for true planets, and even with automation, the full pipeline can be long. Detected transits have their classification stored in a "Kepler Objects of Interest" (KOI) table. Some of these classifications/dispositions are from automated processes like Robovetter and others are human-verified. These will be used to train classification models in an attempt to replicate NASA's exoplanet pipeline.

2. OBJECTIVE AND TECH STACK

The primary motive was to classify transits based off the light curves acquired during the Kepler mission and additional contextual data. A transit was classified as either a confirmed planet or a false positive. Features chosen for each model varied and were engineered to best suit the model. Generally, the flux and times for light curves of stars combined with features that provided additional context were used. It was ideal to use features that had strong predictive power but were also non-trivial and relatively simple to measure in practice. Features with strong predictive power that are difficult

or time-consuming to measure would not be beneficial to NASA's pipeline.

To answer project-related questions, an interactive conversational agent was used. The user interface for it was a Flask Python app. A logistic regression, Recurrent Neural Network (RNN), and Random Forest Classifier model was trained. SQLite was used for the bulk of the storage, meaning the SQLite3 module. Pandas, NumPy, and SciPy were used for data manipulation, and Matplotlib and Seaborn for visualizations. Scikit-learn and TensorFlow were used for preprocessing and building the models. Nbformat and Pdfplumber were used to convert the Jupyter Notebooks and Milestone 2 report to text files that could be fed to the chatbot as part of its prompt.

As for the agent, Open AI API was used. The code that contacted the API was hosted as a Google Cloud Run Function and Flask was used for the user interface. Google Cloud was used to keep the API key private, but also allowing possible changes to the chatbot in the future to immediately take effect for all users. As for the user interface, letting it run locally through Flask allows flexibility in terms of personalization for any developers wanting to adjust the HTML design elements.

3. TIMELINE

February 24, 2025 - March 9, 2025

The database was modified to use the star ID and time stamps of a light curve as the primary key which sped up queries. A decision tree classifier was trained so that its feature importances could help reduce the dimensionality of the data.

March 10, 2025 - March 16, 2025

Features were selected based on previous analysis. Samples were split and training/hyper-tuning began for the three supervised classification models, iteratively evaluating performance metrics.

March 17, 2025 - March 21, 2025

Final attempts for improvements were made along with an analysis of the strengths, weaknesses, and biases of each model. The report and GitHub were updated.

March 22, 2025 - March 30, 2025 Using the test set, the best models based on previous performance

*Electronic address: michaelcalderin@ufl.edu

were run. Insights and limitations were explored.

March 31, 2025 - April 13, 2025 The conversational agent was researched and explored. A Flask app was used on the client-side which contacted a Google Cloud Run Function that acted as the intermediary to the Open AI gpt-4.1-nano response.

April 14, 2025 - April 23, 2025

A video presentation, demo, and overall work was finalized.

4. DATA COLLECTION

All data used was publicly accessible through NASA’s API. There were no explicit licensing or usage restrictions, especially for the scope of this project. NASA’s exoplanet archive provided a “Cumulative Kepler Objects of Interest (KOI)” table in the form of a CSV which had summary information about each star’s transits. This is where the potential exoplanets’ dispositions were labeled as confirmed, candidate, or false positive. This was directly downloaded through their website [2] and saved as *KOI_cumulative.csv*. The light curve data was more complicated to fetch since there was 3 TB worth of light curves in their database. There was a section of the archive for bulk downloads that provided a script called *Kepler_KOI_wget.bat* [3]. It contained a *wget* command on each line that fetched light curve data for each star that was in the KOI table and would likely amount to more than 100 GB when fetched in its entirety. Each star’s light curve was its own dataset.

In a Jupyter Notebook titled *data_collection.ipynb*, the bat file was processed line by line. Single quotation marks had to be converted to double quotations to be able to run the commands on Windows. The commands were run through Python using its *subprocess* module. The data was saved in an SQLite database titled *light_curves.db* and due to the size of the data, only relevant features were kept and the process was stopped after collecting data for the first 2680 stars which is about a quarter of the stars in the KOI table and roughly 20 GB. Since transits are typically periodic, each planet had multiple dips in its corresponding star’s light curve; these repetitions were utilized to augment the data size. With this sample, SQL queries were already slow. In terms of downloading the data, it took two days to fetch these stars alone. It would have been ideal to use a random sample instead of going based on first come first serve, but due to computational constraints it would take too long to download a random sample at this time. The potential for bias was noted and tracked throughout the project.

5. DATA CONTENT AND PREPROCESSING

The data was analyzed in *data_processing.ipynb*. The KOI CSV was read in as a Pandas data frame but filtered so that it would only include stars that also appeared in the SQL light curve database. The KOI data had 3164 rows and 141 columns with features such as star ID, transit time, duration, etc. The SQL database had features such as star ID, time of measurement, flux, and quality of measurement. There were six columns and over 200 million rows.

With the exception of a few features, the features were mainly analyzed and not dropped at this stage. Those that were dropped were mainly due to being entirely null, constant, or in the interest of preserving the most useful data while having no null values. Basic information for each feature such as their null count, descriptive statistics, most frequent values, etc. were displayed. There were no duplicates. Some features that obviously needed data types conversions were handled. Pearson correlation coefficients were calculated for numerical features and chi-squared for categorical features. Histograms, frequency bar graphs, and box plots for each feature were saved to a folder. Outliers were detected but not removed at this stage. However, they were excluded for imputation purposes. Numerical features were scaled using the scikit-learn standard scaler. Scaling was done at this point to help with feature selection, but each model had its own pipeline that varied and will later be discussed. The cleaned KOI data was saved as *KOI_cumulative_cleaned.csv*.

The SQL database of light curves originally had slow query times so its exploration was limited and its content was left untouched until the feature selection phase; that is when a clearer picture was drawn for how the content of the database would be utilized. For more information, refer to the Jupyter Notebook which has markdown cells with details and insights.

6. EXPLORATORY INSIGHTS

Due to the large number of features, some key insights will be discussed but they are in no means exhaustive. To begin, Pearson correlations showed three predominant areas of high correlations (above 0.8): star/planet characteristics, equipment information, and errors. Meaning, features related to stellar or planetary data could likely be reduced to a few key features and the same goes for the other two categories. The chi-squared test is shown in Figure 1. Most dependent relationships are contextually obvious. For example, the disposition and false positive flags would be related because one of the classes of disposition is false positive and the flags are simply a more descriptive version of that. This gives strong indication that the categorical features could be condensed. Despite this, no features were dropped during the exploratory phase since that was better suited for the feature selec-

tion and engineering phase. The main emphasis during the exploratory phase was to understand the data and acquire enough evidence to justify feature selections.

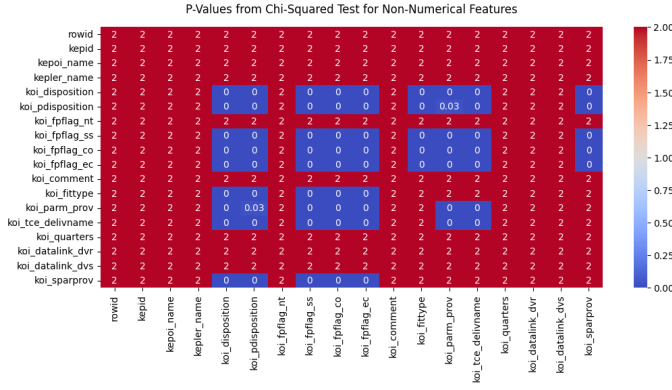


FIG. 1: The p-values after running the chi-squared test on non-numerical features. P-values of 2 are obviously nonphysical and indicate that the chi-squared would be inaccurate for that pair of features due to a small size in expected frequency which decreases reliability according to the scikit-learn documentation.

The target variable is "koi_disposition" and it was important to understand its distribution. It turns out there are about 700 candidates, 1000 confirmed planets, and 1400 false positives. The imbalance indicates that stratification might be useful for modeling. However, during feature selection/engineering, multiple transits per star were used so this distribution changed and made the number of false positives closely match the number of confirmed planets; this will later be revisited.

In Figure 2, higher scores for candidates and confirmed planets indicate greater confidence in the classification while for false positives, lower scores indicate greater confidence. Across the board, there is high confidence in each disposition. Still, there is a notable imbalance, specifically for candidates, between median and mode. Likely, low-confidence outliers are skewing this category. This could be due to false positives having more obvious patterns and confirmed planets having more rigorous processing in the pipeline, while candidates have less predictable trends and are somewhat in the middle between false positives and confirmed.

As shown in Figure 3, the current sample from the Kepler light curve data was the outer edges of the field of view. There are no points from the center. Given that Kepler only captured a small section of the sky and it was our own galaxy, the data is innately biased aside from sampling. It still would have been more representative of the data at hand to do randomized sampling, but the data is ultimately biased regardless and computational constraints prevented "fair" representation. There are also drawbacks to randomized sampling such as not getting enough data from neighboring stars which could lead to a lack of recognition of light interference type

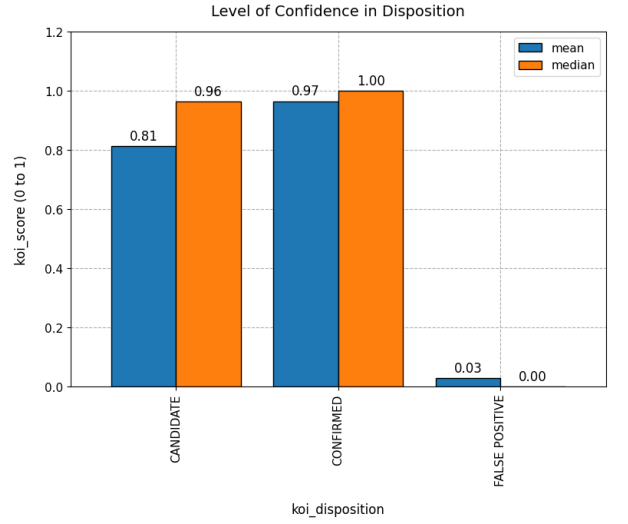


FIG. 2: The mean/median level of confidence in each disposition is displayed. These scores are generated by a Monte Carlo technique such that the score's value is equivalent to the fraction of iterations where NASA's automated classifier (Robovetter) outputs "CANDIDATE".

false positives. The problem at hand has many complex factors at play so optimized sampling would be a study of its own.

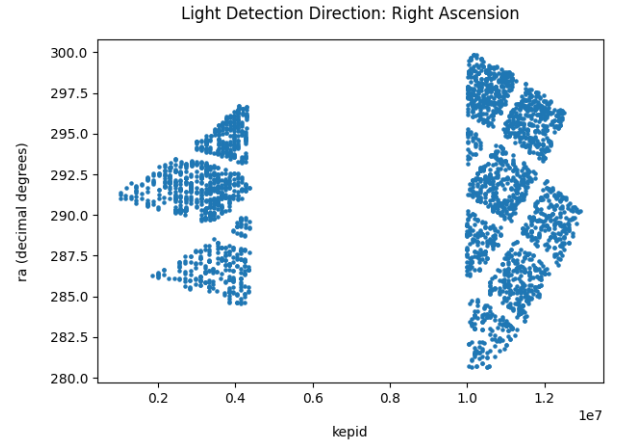


FIG. 3: "kepid" is a unique identifier for each star and is plotted against right ascension. The accompanying coordinate to right ascension, declination, varied from roughly 36 to 52 decimal degrees and was also missing stars in the middle. It provided no new information in terms of sample selection that this right ascension visualization did not encapsulate.

Figure 4 shows that the types of false positives are also imbalanced. Due to the size of the data, it is difficult to find all imbalances but clearly they are present so this should be noted.

In terms of outliers, a democratic method was employed between z-score, inter-quartile range, and median absolute deviation for each feature. Generally, using two-

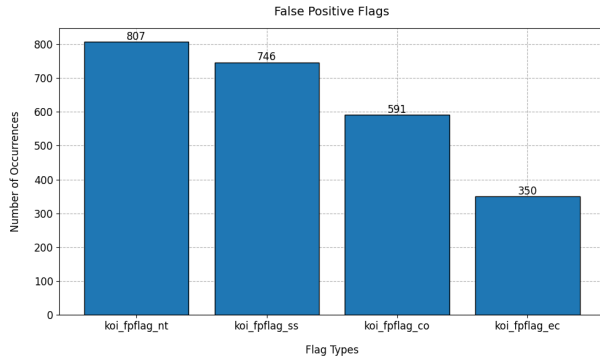


FIG. 4: This is the distribution of false positive flags. "nt" is not transit-like, "ss" is stellar eclipse, "co" is centroid offset (detecting light from a different, nearby star), and "ec" is ephemeris contamination (flux contamination or electrical crosstalk).

thirds agreement was considering too much of the data as outliers. This is not only inconvenient since training typically requires as much data as possible, but also disconnected from visual insights. Upon inspection, many of the "outliers" are generally part of the cluster of data. Unanimous outlier detection seems to be a better fit. Thus, values were considered outliers if there was unanimous agreement. These points were identified as outliers but not discarded. Discarding even one outlier would mean a large amount of light curve data is thrown out. It would also be strange to remove outliers since transits themselves are rare events compared to the number of data points in a light curve. For more specifics on summary statistics, distributions, etc., refer to the Jupyter Notebook which is documented step-by-step.

7. FEATURE ENGINEERING AND SELECTION

7.1. Encoding and Data Reduction

KOI_cumulative_cleaned.csv was used for feature selection purposes. The "kepoi_name" feature was a unique identifier for each transit so it had no predictive value and was dropped (along with similar identifiers). Note that although it was dropped for feature selection, it was still useful for identification purposes throughout the pipeline. Thus, a feature may have been discarded for a particular task such as feature selection or model training, but be reintroduced to provide additional context for performance. For example, planet size could be irrelevant for such tasks, but it might be interesting to see how model performance varies by planet size.

"koi_quarters" was a binary string where each bit represented whether data was collected in that quarter of the Kepler mission. Although this format is easy for a human to read, the models would likely benefit from a more intuitive form. It was engineered into two forms. The first was 32 new features, each with a binary digit

that represented whether data was collected in that quarter or not. The second form was one new feature that represented the number of quarters that data was collected in. Later, a decision tree classifier was used and its list of feature importances showed that other features were much more powerful. The chosen features and the rationale for choosing them will soon be discussed in greater detail, but note that even the engineered versions of "koi_quarters" were not used.

"koi_pdisposition" is the guess from NASA's automated system called Robovetter which is rule-based and not a machine learning model. Based off this description and its previous high correlation to the class labels, it would likely be a strong predictor. In spite of this, part of the interest of this project was to replicate Robovetter and see if improvements could be made to NASA's pipeline. It would be counterproductive to include data generated by Robovetter, other than the labels used for classification, so "koi_pdisposition" will not be used. Similar logic is used to exclude "koi_score", which gives Robovetter's confidence in its disposition, and the false positive flags (generated by Robovetter).

Features related to errors, such as the margin of error for a transit period, were dropped due to their high collinearity and contribution of noise to the data. These error-based features saturate the feature space which would likely prevent the decision tree classifier from accurately picking out the most relevant features.

The target label, "koi_disposition", was encoded using the ordinal encoder from scikit-learn. The false positive class was marked as 0, candidate was marked as 1, and confirmed was marked as 3. This artificial ordering was to imply the "closeness" that a sample was to being a planet. Few categorical features were left and they were relatively low cardinality so they were one-hot encoded; they also had no implicit ordering to justify an ordinal encoding.

At this point, the KOI data was clean enough to train a decision tree classifier that could help with feature selection. A tree was used because scikit-learn provides an accessible list of feature importances which represents the importance of the variable in making its splits/classifications; trees naturally capture non-linearity and NASA's pipeline is also rule-based when flagging false positives which is similar to a tree's behavior. The data from the light curve database was not used here because the emphasis was to reduce dimensionality; the light curve database had a much smaller feature space so seeing the most relevant features was much more obvious.

Figure 5 shows a snippet of code used for training the tree. When splitting the data into 80% for training and 20% for testing, stratification by the target label was used so that the ratios of the classes were preserved. The tree was hyperparameter tuned with K-Fold cross-validation (5 folds). Optimization was based on precision since the desired behavior of the models was to be confident in its predictions; discovering a new planet is a bold claim

and it would be disappointing to later find out it was not a true planet. The best tree had the following hyperparameters: "criterion" set to entropy, "max_depth" of 10, "min_samples_leaf" of 2, and "min_samples_split" of 200. For validation, the mean metrics across all folds were 0.77 for precision, 0.75 for recall, 0.76 for F1, 0.80 for accuracy, and 0.91 for AUC. Similarly for the training data, precision was 0.80, recall was 0.78, F1 was 0.79, accuracy was 0.82, and AUC was 0.93. Based on these metrics, there were minor indications of overfitting but not a concerning amount.

Looking at the tree's feature importances, many features were assigned zero weight. Out of the ones assigned non-zero importance, the following were chosen: "koi_ror" (0.26 importance), "koi_dikco_msky" (0.22 importance), and "koi_max_mult_ev" (0.12 importance). These were one of the most important, but not necessarily the top three. The decision was largely aided by domain knowledge and ensuring that the correlation matrix did not indicate high collinearity. "koi_ror" is the ratio of the planet radius to star radius and was chosen since it provides information about the size of the transiting object and star. "koi_dikco_msky" represents the difference between the observed position of a star and its cataloged position, so it gives insight to the uncertainty in positional alignment which helps detect false positives that are light interference. "koi_max_mult_ev" is the maximum signal to noise ratio which helps distinguish instrumental noise. These three are considered the "contextual" features; they supplement the raw brightness of a star during a transit and roughly cover all the types of false positives.

The light curves in the SQL database were transformed into features. For a given potential planet, 30 time steps centered around its first detected transit were used as columns in a CSV file; specifically 30 were used due to computational constraints and most samples having transit lengths within this window. The PDC SAP flux is the brightness of the star after being processed by NASA's pipeline to help remove instrumental effects while keeping the transits; it was included for each time step. Other variables were also added to the CSV for each time step but were ultimately unused. Transits that had less than 30 time steps were not included.

As these transits are periodic, the first four dips were used to augment the number of samples in the dataset; if an object had less than four transits, then the amount of transits it had were used. This process resulted in about 10,318 samples of transits and the table was saved as *transits.csv*. Originally, the classification was going to be for false positives, candidates, and confirmed planets, but due to the data augmentation, there were enough samples to train a binary problem predicting just false positives and confirmed planets. This was ideal since "candidates" are a gray area that worsen the performance of the models, and with binary decisions, these candidates could be classified into planets or false positives which is more useful to NASA's pipeline. Thus, the problem statement

became to classify transits as planets or false positives and the candidate class was not used for training. This resulted in approximately balanced classes and 7301 samples to use for training and testing. Roughly, each model was trained on 30 time steps with the associated brightness for each time, and the three contextual features. Adjustments were made depending on the model and this is explored in Section 7.2.

7.2. Model-Specific Engineering

RNNs can accept tensors as input which allows each time to be paired with its associated flux so the features did not have to be engineered other than reshaping into a compatible tensor; the specific shapes will be explained in Section 8 since they are part of the model architecture. However, logistic regression and random forest are limited to 2-dimensional input so instead of having the times and fluxes as inputs, they were engineered into 29 features that represent the slope of the flux. Each feature is the change in flux divided by the change in time between each pair of adjacent time steps. Time and its corresponding flux come in pairs for each time step, so this engineering was done to help the models recognize the pairwise relationship since they cannot process features sequentially like RNNs.

The features for the random forest were not scaled since it is an ensemble of decision trees which do not require scaling and could otherwise distort the data. The features for the logistic regression were normalized using the standard scaler from scikit-learn since it helps with convergence and prevents features with larger scales from dominating; the standard scaler was preferred over the min-max scaler because it is less sensitive to outliers. In contrast, the features for the RNN were scaled using the min-max scaler from scikit-learn; since transit dips can be small, it is likely best to remain in a consistent range for a model that can process the data sequentially. For all models, the target label was ordinally encoded with 0 representing a false positive and 1 representing a true planet; this is typically required by the libraries and models chosen, but the ordering also has the intended behavior of viewing class 1 as "more" of a planet and class 0 as "less" of a planet. Categorical features were previously used, but they were not needed or used for training since numerical features happened to be the most important based on feature selection.

8. DATA MODELING

8.1. Random Forest Classifier

As mentioned, the feature space was customized for each model, but those resulting sets were all split into 80% for training and 20% for testing. This split was done in an attempt to train on as much data as possible while

```

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Hyperparameter tune
scoring = {'roc_auc': make_scorer(roc_auc_score, needs_proba=True, multi_class='ovo', average='macro'),
           'precision': make_scorer(precision_score, average='macro'),
           'accuracy': make_scorer(accuracy_score),
           'recall': make_scorer(recall_score, average='macro'),
           'f1': make_scorer(f1_score, average='macro')}
params = {'criterion': ['gini', 'entropy', 'log_loss'], 'max_depth': [None, 1, 2, 5, 10, 50, 200], 'min_samples_split': [2, 10, 200, 500, 1000], 'min_samples_leaf': [1, 2, 3, 5, 20, 100]}
grid = GridSearchCV(estimator=DecisionTreeClassifier(random_state=42), param_grid=params, scoring=scoring, refit='precision', cv=5, n_jobs=-1, verbose=1, return_train_score=True)
grid.fit(X_train, y_train)

```

FIG. 5: This code was used to train the decision tree classifier that helped with dimensionality reduction and feature selection. The hyperparameters were varied through GridSearchCV from scikit-learn and are represented in the "params" dictionary. K-Fold cross-validation was used with 5 folds, optimizing for precision.

still setting samples aside to evaluate generalization to data that was unseen during training. Stratification was used for the target label even though it was relatively balanced to help ensure the training and testing performance could be directly comparable. Other imbalances could exist between the sets but an imbalance of the target label would likely be the greatest disturbance. Note that all models were saved as pickle files for future use in a "Models" folder.

The random forest classifier from scikit-learn was used and it classifies by averaging the predictions of an ensemble of decision trees, each adjusted with some randomness for reduced overfitting. It was hyperparameter tuned as shown in Figure 6. Hyperparameters such as criterion and max tree depth were varied with K-Fold cross-validation (5 folds), optimizing for precision. The best hyperparameters found were the following: gini for "criterion", 50 for "max_depth", 2 for "min_samples_leaf", and 450 for "min_samples_split". The mean validation scores across all folds were: 0.93 for precision, 0.91 for recall, 0.92 for F1, 0.93 for accuracy, and 0.98 for AUC. The mean training scores across all folds were: 0.94 for precision, 0.92 for recall, 0.93 for F1, 0.93 for accuracy, and 0.98 for AUC. There was minimal increased performance for the training data compared to the validation data.

The confusion matrix on the entirety of the training data is shown in Figure 7. There are fewer misclassifications for the non-planet class. Figure 8 shows the ROC curve with an AUC of 0.98, indicating a strong ability to separate the classes. Figure 9 shows the importance of each feature for the model to make its decisions. A large emphasis is placed on the three contextual features. Out of the features that represent the slope of the light curve, there are two peaks which correspond to the region near the center of a transit. This could indicate that the most important characteristics are slightly before the center of the transit and slightly after, indicating a need for depth perception. Another possibility is that non-planets sometimes have two dips and this could be an indication; samples with two dips might be a clear red flag.

```

# Tune
scoring = {'roc_auc': 'roc_auc',
           'precision': 'precision',
           'accuracy': 'accuracy',
           'recall': 'recall',
           'f1': 'f1'}

params = {'criterion': ['gini', 'entropy'],
          'max_depth': [2, 5, 10, 50, 100, 200],
          'min_samples_split': [350, 400, 450, 500, 550],
          'min_samples_leaf': [2, 3, 5, 20]}

grid = GridSearchCV(estimator=RandomForestClassifier(random_state=42),
                    param_grid=params,
                    scoring=scoring,
                    refit='precision',
                    cv=5,
                    n_jobs=-1,
                    verbose=1,
                    return_train_score=True)

grid.fit(X_train, y_train)

```

FIG. 6: This code was used to train the random forest classifier. The hyperparameters were varied through GridSearchCV from scikit-learn and are represented in the "params" dictionary. K-Fold cross-validation was used with 5 folds, optimizing for precision.

8.2. Logistic Regression

The logistic regression model from scikit-learn was used and it classifies by feeding a linear combination of the features into a sigmoid function bounded between 0 and 1. It was hyperparameter tuned as shown in Figure 10. Hyperparameters such as the penalty and solver were varied with K-Fold cross-validation (5 folds), optimizing for precision. The best hyperparameters found were the following: "C" of 5, "max_iter" of 100, "penalty" of l1, and liblinear as the "solver". The mean validation scores across all folds were: 0.88 for precision, 0.96 for recall, 0.92 for F1, 0.92 for accuracy, and 0.95 for AUC. The mean training scores across all folds were: 0.88 for precision, 0.97 for recall, 0.92 for F1, 0.92 for accuracy, and 0.96 for AUC. There was no significant difference between training and validation performance, likely due to regularization preventing overfitting.

The confusion matrix on the entirety of the training data is shown in Figure 11. There are fewer misclassifications for the true non-planet class. Figure 12 shows the ROC curve with an AUC of 0.96, indicating a strong ability to separate the classes. Figure 13 shows the mag-

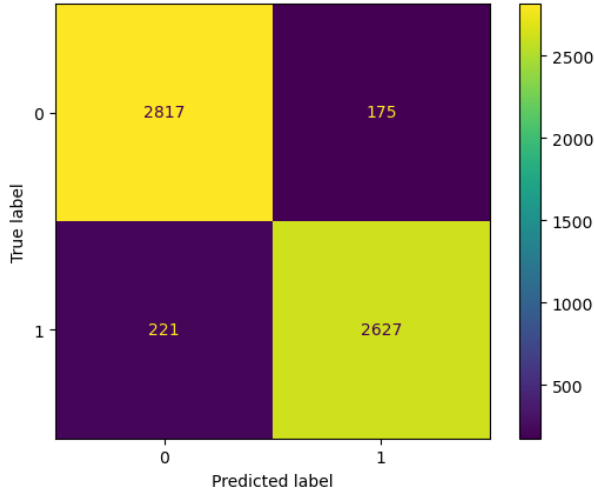


FIG. 7: Confusion matrix for random forest classifier on the training data, where 0 represents the false positive (non-planet) class and 1 is the true/confirmed planet class.

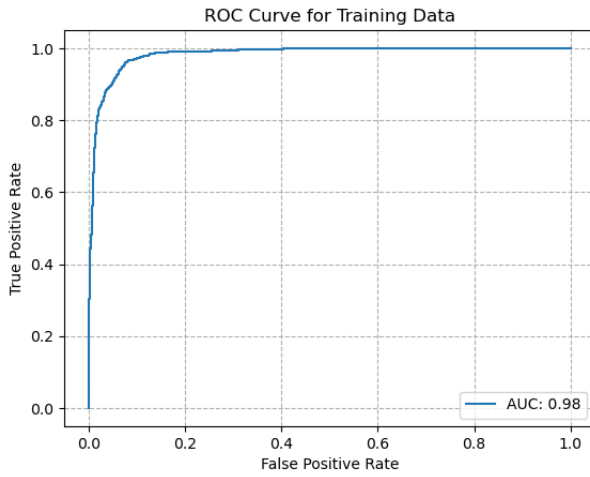


FIG. 8: Receiver Operating Characteristic (ROC) curve with its associated Area Under the Curve (AUC) for the random forest classifier.

nitude of the coefficients which is analogous to the feature importances of the random forest. The three contextual features are important, but there is much more variability in the time-based features. This could indicate a greater understanding of the nuances of the light curve, or difficulty picking out the main patterns.

8.3. Recurrent Neural Network

Figure 14 shows the structure of the RNN model, built using TensorFlow. The time input was a tensor in the following shape: (number of samples, 30, 2). "30" represented the number of time steps and "2" represented the time and PDC SAP flux for each time step. The

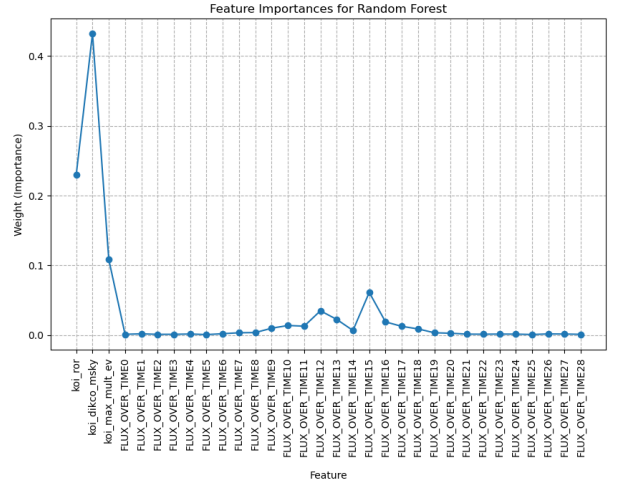


FIG. 9: Feature importances of the random forest classifier, as provided by scikit-learn.

```
# Tune
scoring = {'roc_auc': 'roc_auc',
           'precision': 'precision',
           'accuracy': 'accuracy',
           'recall': 'recall',
           'f1': 'f1'}

params = {'C': [0.1, 1, 5, 10, 15, 20],
          'penalty': ['l1', 'l2'],
          'max_iter': [100, 200, 500],
          'solver': ['lbfgs', 'liblinear', 'saga']}

grid = GridSearchCV(estimator=LogisticRegression(random_state=42),
                    param_grid=params,
                    scoring=scoring,
                    refit='precision',
                    cv=5,
                    n_jobs=-1,
                    verbose=1,
                    return_train_score=True)

grid.fit(X_train, y_train)
```

FIG. 10: This code was used to train the logistic regression model. The hyperparameters were varied through GridSearchCV from scikit-learn and are represented in the "params" dictionary. K-Fold cross-validation was used with 5 folds, optimizing for precision.

contextual input was for the three chosen contextual features and was a tensor of the following shape: (number of samples, 3,). The number of neurons and dropout shown were the best hyperparameters found for maximum validation precision. 20% of the training data was reserved for validation and 200 epochs were attempted per configuration with early stopping (patience of 5 monitoring the validation loss). Binary cross-entropy was chosen as the loss function due to its convexity which helps with convergence. TensorFlow's Adam optimizer was used. The output layer was a sigmoid but all other dense layers were a rectified linear unit (ReLU). When tuning, the option was given between either using Long Short-Term Memory (LSTM) or the Gated Recurrent Unit (GRU).

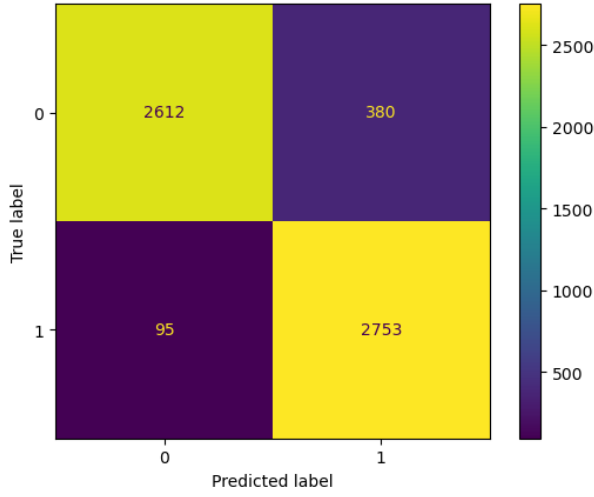


FIG. 11: Confusion matrix for logistic regression on the training data, where 0 represents the false positive (non-planet) class and 1 is the true/confirmed planet class.

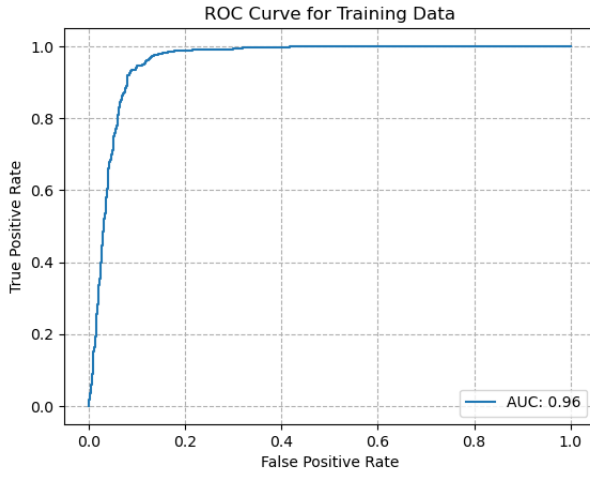


FIG. 12: Receiver Operating Characteristic (ROC) curve with its associated Area Under the Curve (AUC) for logistic regression.

The best model chose LSTM. Batch normalization was used to prevent distribution shifts and dropout was used to prevent overfitting. 50 configurations were run to find the best hyperparameters, randomly choosing the hyperparameters during each iteration. The options for the hyperparameters were the following: (1×10^{-3} , 1×10^{-4} , 1×10^{-5}) for learning rate, (16, 32, 64, 128, 256) for the number of neurons in a layer, (0, 0.1, 0.2, 0.3, 0.4, 0.5) for dropout, LSTM vs. GRU, and (16, 32, 64, 128) for batch size.

After the best model was found, it was trained on the entire training dataset for the full 200 epochs without early stopping. Learning rate and batch size were manually chosen to be 0.0001 and 32, respectively, based on the shape of the loss function across the epochs. A smooth

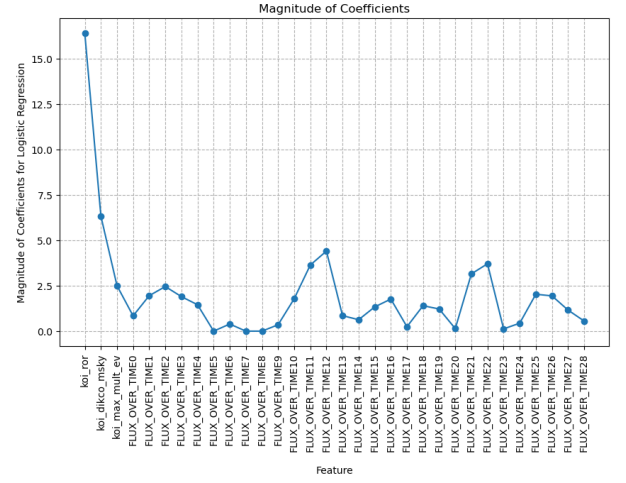


FIG. 13: Coefficients of logistic regression, as provided by scikit-learn, after retaining their absolute value.

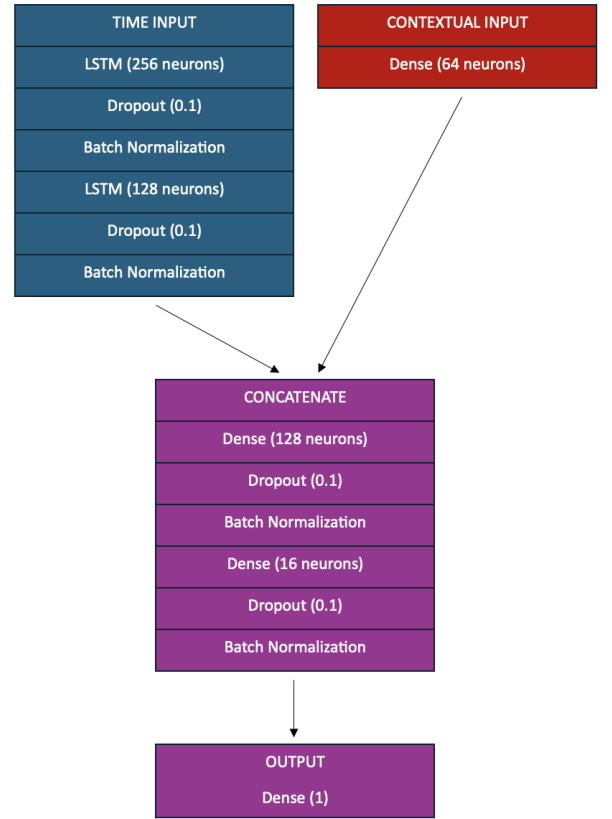


FIG. 14: Structure of the RNN model with the best hyperparameters shown for each layer, optimizing for precision.

curve and a flat-line indicating convergence was sought after and shown in Figure 15. Validation loss was typically lower than training loss which was an indicator that the model was not overfitting. On the last epoch, the training accuracy was 0.91, precision was 0.86, and recall was 0.95. The validation accuracy was 0.92, vali-

dition precision was 0.90, and validation recall was 0.94. Validation performance was better than the training performance so the model seems to generalize well.

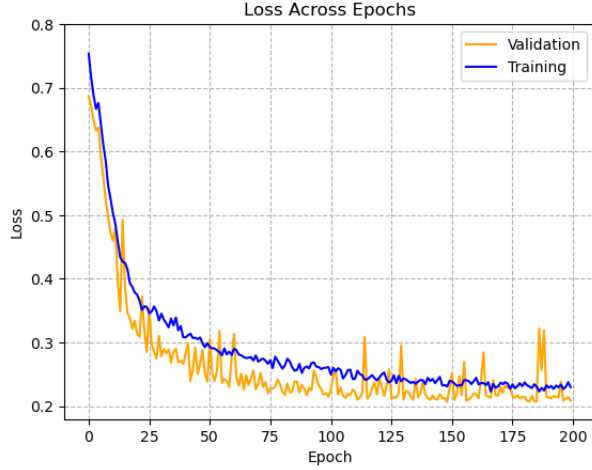


FIG. 15: Training and validation loss of the RNN across the 200 epochs it was trained over. The loss function used was binary cross-entropy.

Figure 16 shows the confusion matrix. There are fewer misclassifications for the true planet class. Figure 17 shows the ROC curve with an AUC of 0.97 which demonstrates strong discriminatory power.

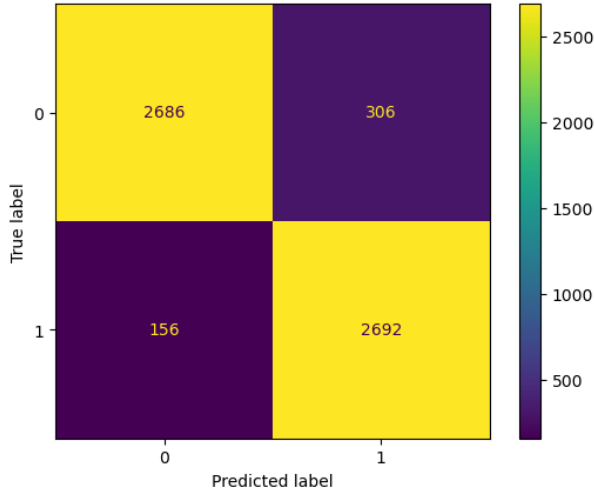


FIG. 16: Confusion matrix for RNN on the training data, where 0 represents the false positive (non-planet) class and 1 is the true/confirmed planet class.

8.4. Training Performance Comparison

Random forest had fewer misclassifications for the non-planet class which was in contrast to the other two models which had fewer misclassifications for the true planet

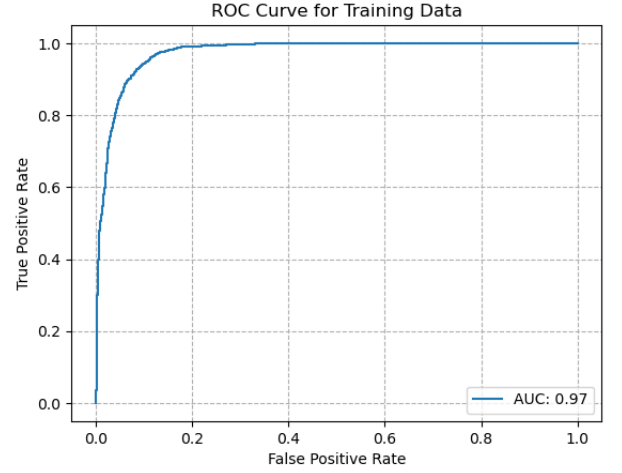


FIG. 17: Receiver Operating Characteristic (ROC) curve with its associated Area Under the Curve (AUC) for the RNN.

class. Based on AUC, the RNN and random forest had the strongest discriminatory power. Random forest also paid less attention to the time-based features compared to the logistic regression. This was likely because the random forest was able to pick out the most important aspects of the time series while the logistic regression became confused. However, we cannot rule out the possibility that logistic regression picked up on more complex patterns and was possibly not a strong enough model to decipher or utilize them properly.

Both random forest and logistic regression seemed to have an over-reliance on the contextual features, although less so for the logistic regression. In terms of metrics, the RNN and random forest were close competitors. Logistic was not far behind but there was certainly a gap compared to the other two. In general, random forest made the fewest misclassifications and had the greatest discriminatory power. It also had the highest validation precision which was the primary metric. Its complexity is also lower than the RNN so at this point, it was the best model, followed by the RNN and then logistic regression. It is interesting to point out that NASA's Robovetter created the false positive class labels. Recall that Robovetter is rule-based like a decision tree so random forest might have naturally been able to capture this behavior the best since it is tree-based. It could also have been bias since the feature importances of a decision tree were used to help with feature selections. RNNs typically do well with time series data so the fact its performance was so close to the random forest is interesting. There are several possibilities, including this project's methodology, the limited size of the training data, and the potential for the false positive class to have inaccuracies since NASA had an automated system generate them.

9. TEST SET PERFORMANCE AND POTENTIAL BIASES/LIMITATIONS

9.1. Random Forest Classifier

First note that when referring to precision, recall, and F1, it is implied that macro averages were used unless referring to individual classes. After making predictions on the test set, random forest was found to have a precision of 0.92, recall of 0.92, F1 of 0.92, and accuracy of 0.92. According to the confusion matrix, there were 700 true negatives, 650 true positives, 48 false positives, and 63 false negatives. This implies that there was slightly higher precision for true planets but higher recall for nonplanets. The AUC was 0.97 so in general, performance was similar to that of training and validation.

Out of the samples where the ground truth was the nonplanet class, the distribution of transit periods for correct and incorrect predictions showed similar behavior. Most correct and incorrect classifications occurred for shorter periods, shorter periods being the most common type of sample in the training set so it is natural to have more incorrect classifications here due to volume. It indicates there is no significant bias in terms of period. As for transit duration, correct and incorrect predictions usually pertained to short durations, but there were spikes of incorrect predictions from about 10 to 30 hours. Transits longer than 40 hours were rarely misclassified. Thus, long transits do not seem to be an issue but there is an intermediate region (10 to 30 hours) that seems to be unusually troublesome and likely cannot be explained based off volume as before.

Planet-star radius ratio (koi_ror), angular offset between observed star position and cataloged star position (koi_diko_msky), and the signal-to-noise ratio estimate ($koi_max_mult_ev$) were the contextual features used for modeling and they all showed a significant bias towards misclassifications of small values. Intuitively, the model is likely focusing on noise when there is a low signal-to-noise ratio which is not very insightful and leads to wrong predictions. The other two features likely struggle due to not having enough samples for the lower extreme of their range. In general, lower values for these features is the most common but the lower extreme can be just as rare as the upper bound. There can also be more complex patterns in the lower range where most samples are because outliers are often nonplanets, but where the bulk of the population lies, the distinction between planet and nonplanet is blurred.

Out of the samples where the ground truth was the true planet class, transit periods for correct predictions were similar to that of incorrect, but there were small spikes for periods greater than 100 days. After transit durations of about 7.5 hours, there were much more spikes in incorrect classifications compared to correct ones. This indicates the model was having trouble with recalling true planets with longer durations.

Planet-star radius ratios greater than 0.05 showed sig-

nificant spikes for incorrect predictions compared to correct ones. The two other contextual features showed similar behavior so there was a clear struggle in recall for true planets when these features had higher values. High signal-to-noise and large angular offset are likely seen as characteristics of nonplanets since they are extreme or unusual, so a true planet that has these characteristics is often misclassified. Figure 18 shows the distribution for the transit duration. Note the general similarity in distributions for correct and incorrect predictions but the spikes for the incorrect classifications after 7.5 days. Similar visualizations were used when discussing the other features so refer to this figure for an understanding of what is meant by "spikes" and other characteristics mentioned.

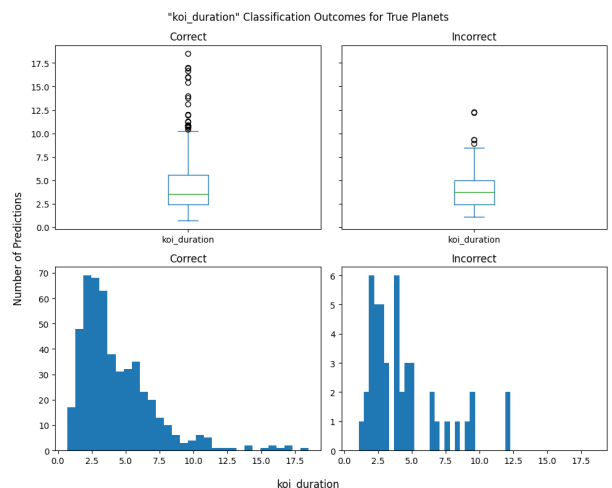


FIG. 18: For random forest, the distribution of predictions where the ground truth is the true planet class is shown. This is relative to the transit duration which is measured in days.

9.2. Logistic Regression

For logistic regression, the macro average for precision was 0.92, for recall it was 0.91, and for F1 it was 0.91. The accuracy was 0.91 and the AUC was 0.95. There were 654 true negatives, 681 true positives, 94 false positives, and 32 false negatives. There was a noticeable difference (about 0.07) between the precision of the planet class and the nonplanet class. Recall showed a similar difference. True planets had better recall but nonplanets had better precision.

Out of predictions that had a ground truth of nonplanet, the distributions showed similar behavior to that of random forest. The angular offset was an exception. There were notable incorrect predictions for angular offsets up to about 2.5 arcseconds, whereas the incorrect predictions for random forest were only a sliver of this range. Random forest only struggled with the extreme of the lower range, but logistic regression struggled with

a larger range of angular offsets. Out of predictions that had a ground truth of planet, the same patterns were seen as for random forest. Out of the contextual features and variables mentioned prior, there is a struggle with larger values. Samples in the high-end range for these variables are typically less abundant so perhaps oversampling would have been beneficial.

9.3. Recurrent Neural Network

The RNN had a macro average precision of 0.92, for recall it was 0.92, and for F1 it was 0.92. The accuracy was 0.92 and the AUC was 0.98. There were 671 true negatives, 667 true positives, 77 false positives, and 46 false negatives. Precision was higher for the nonplanet class and recall was higher for the true planet class. This is similar to logistic but in contrast to random forest where precision was higher for the planet class and recall was higher for the nonplanet class. The difference for both metrics is about 0.04. The difference for the random forest was slightly less dramatic (about 0.01-0.03).

Out of the predictions that had a ground truth of non-planet, the patterns were similar to logistic regression. Both suffered much more in the lower extreme of the variables compared to random forest. Out of the predictions that had a ground truth of true planet, the patterns were similar to the other models. There is usually greater misclassifications when there are less representative samples; this manifests itself in both lower extremes of the variables and in the higher end of their ranges.

9.4. Discussion

Random forest generally outperformed the other models. RNN was not too far off and logistic regression fol-

lowed it. The success of the random forest could potentially be due to bias from dimensionality reduction via a decision tree's feature importances and Robovetter which is rule-based and generated many of the labels, especially the false positive (nonplanet) labels. In general, the models suffered when there were not enough representative samples. They could have all benefited from oversampling key variables, even if they were not chosen as features but were still key for representation purposes. Compared to training and validation, there was minimal decrease in performance for the test set.

10. CHATBOT DEVELOPMENT

Open AI API was used to develop the chatbot. The model used was "gpt-4.1-nano" which is one of the less powerful ones, but it is cost effective and allows for a greater number of input tokens. All of the code and the second milestone report was converted to ".txt" files and then combined into one text file. Upon a user query (prompt or question), the full file is fed to the model as a string under a system message that also explains its role.

The full pipeline is as follows: a user runs code which opens a Flask app with a text box for typing their questions, they submit and a POST request is sent to a Google Cloud Run Function which reads from the text file and gets a response from Open AI's model, and this is sent back and that response appears on the Flask app. The chatbot is designed to answer questions about the project and tries to avoid going out of the scope. Note that it does not have access to the final report so queries related to the biases, limitations, final results, etc. will be the model's opinion rather than raw commentary from its knowledge base that it reads from.

-
- [1] NASA, *Kepler by the numbers*, <https://science.nasa.gov/resource/nasas-kepler-mission-by-the-numbers/> (2018).
 [2] NASA, *Kepler objects of interest*, <https://exoplanetarchive.ipac.caltech.edu/docs/data.html>,

- cumulative KOI data.
 [3] NASA, *Bulk data download*, https://exoplanetarchive.ipac.caltech.edu/bulk_data_download/, kepler KOI time series.