Efficiency Characterization of the Kepler Exoplanet Discovery Pipeline Using Machine Learning

DS 440 Capstone Final Report

Yuya Jeremy Ong
Penn State University
yjo5006@psu.com

Tomoki Takasawa
Penn State University
tmt5336@psu.edu

ABSTRACT

The Kepler mission, launched by NASA has been used to discover the possibilities for the existence of exoplanets beyond our solar system, providing massive amounts of data that we can use to speculate such candidates. However, to collect the massive amounts of data to identify new and potentially interesting planet candiates, researchers have developed several different methods, heuristics, and data pipelines to facilitate the discovery of new exoplanets. In this work, we characterize the efficiency of the Kepler missionâĂŹs Transiting Planet Search (TPS) system known as Robovetter, using the Robovetter's simulated light-curve pipeline data and provide insights towards building robust automated pipelines for detecting potentially new and interesting stellar objects and identify potential potential false positives within the samples. Furthermore, to facilitate the research and discovery process of this work, we also develop and propose a data-driven research pipeline specifically for our modeling purposes to provide an efficient workflow for researchers to easily design reproducible machine learning-based experimental workflows that both enables automation and flexibility in modeling approaches.

1 INTRODUCTION

The NASA's Kepler Telescope was deployed as a means to discover the possibilities for the existence of exoplanets beyond our solar system, providing massive amounts of data that we can use to speculate such candidates. Concretely, the data generated from these sensors provided researchers with the ability to determine the frequency of planets similar to Earth - however are very far from the detection ranges of the detection spans. Collectively, these telescopes have generated close to 200,000 high-precision photometric stellar data [16] [28] [31], even leading to new discoveries of planets and many candidate planets for further investigation and potential discovery [4] [6] [10] [39].

The majority of work performed on this Kepler data was based on the works of various statistical methods, especially earlier works of that by Borucki et al. [6] and Batalha et al. [4]. Both of these works relied primarily on utilizing "Threshold Crossing Events" (TCEs), which are periodic signals closely indicating transiting planets which were primarily based on manual heuristics based on the Kepler pipeline [27] [28]. These methods relied on researchers to manually false positive results by evaluating variability and instrumental anomalies from the data. Crowd-sourced solutions, such as those conducted by Fischer et al. [21] has shown great success in identifying new planets through manual evaluation of the data.

Even after the completion of its mission, the data processing pipeline for the Kepler mission has scaled to account for larger population of stellar candidates [22]. Thus, it has become no longer feasible for researchers to scale out such manual evaluations and heuristics and opt for data driven methods. This allowed researchers to discover a narrow subset of the Earth-like planets which they can focus much of their resources and attention towards, as opposed to evaluating a much larger search space of stellar candidates.

One process that allowed researchers to accelerate this development process was through the development of Robovetter, a decision-tree based heuristic method which emulated the underlying vetting process used to automatically reject false positive Transit Crossing Events [18]. However, there are also other methods that have began to emerge to automating such vetting process, which relied on a data-driven methodology. The Autovetter Project, which is based on a random forest model which utilized features that were generated by the Kepler Pipeline [11]. Recently, methods that utilize much more sophisticated algorithms such as Deep Neural Networks have emerged as methods to process directly over the photometric data, demonstrating state-of-the-art results in identifying potential new planet candidates [42].

Although many of these automated planet candidate vetting processes have been employed to narrow the potential subspace of planets that would need to be closely scrutinized, one of the fundamental challenges in this Kepler pipeline effort is to further reduce the rate of false positives (*i.e. the rate when a result returns positive to when the actual sample is deemed to be negative*) detected by the system and increase the rate of false negatives (*i.e. the rate when a result returns negative to the actual sample is deemed to be positive*). Various efforts have been established to characterize and quantify the efficiency of the pipeline's ability to detect and recover signals through a technique which involves simulated injections of both planet like objects as well as other non-planet like candidate objects directly to the raw light-curve data [15], [12].

In particular this has been tested through the method of utilizing a Monte Carlo experiment, where the raw calibrated pixels were directly injected using various data augmentation methods and was fed through the Robovetter's detection pipeline. The output distribution of these simulated injection was then analyzed further analyzing key parameters and detection results [13] [15] [12] [14].

In this project we, characterize the efficiency of the Kepler mission's Transiting Planet Search (TPS) system using synthesized data and develop an improved classification model for labeling threshold crossing events. In first phase of report, we evaluate the overall efficiency of the model in its ability to recover indicators for a Kepler Object of Interest based on various parameters through the TPS pipeline through a machine learning based heuristic for characterizing the various pipeline's underlying properties. By building these models, we can potentially identify key characteristics of the current pipeline by analyzing its strengths and weaknesses as well as the efficiency by which the pipeline identifies planets and classifies non planetary objects. In particular, the focus of the models that we produce are not only based on the identification of the planet candidates, but also seeing how the pipeline can potentially pick up on the different types of False Positives that are artificially generated from the original dataset. Although the focus of the project is based on building a machine learning based approach towards evaluating the Robovetter pipelines, our key focus is to utilize the learned parameters and utilize various techniques from model interpretability heuristics such as SHAP analysis [33], LIME analysis [37], and Anchors [38] to perform a comparative analysis between the baseline probabilistic models proposed by Christiansen et al. [12] and our generated models.

The organization of this partitioned into seven different sections, correspondingly containing a subsection of various components which correspond to each respective topics and areas we focus on within this comprehensive report. In the first, and current, section, we introduce and contextualize the problem that we are solving, addressing some of the known prior efforts that have been executed and worked on to build our current work on top of. The second section introduces our Data Processing Infrastructure, which describes the critical data pipeline process of our software as well as key data engineering design choices that we have considered in the context of the current problem we are attempting to address. The third section of this paper describes the first phase of our project which entails the modeling and analysis of the characterization of Robovetter's TCEs detection pipeline efficiency. The fourth section of this paper presents the second phase of our project which models Robovetter's ability to detect and classify various false positive objects based on the identification of the KOI (Kepler Object of Interests) Planet Candidates. The fifth section provides an in-depth analysis of the models that we have generated and built in sections three and four to contextualize, synthesize, interpret, and generate key observations from our generated models through numerous black-box-based model analysis techniques from Machine Learning, followed by a set of implications and research recommendations for future missions with similar efforts. The sixth section provides an in-depth discussion of the research methodology, collaboration efforts, lessons learned, as well as some suggestions and recommendations we would like to make towards the overall logistics of the capstone project process. In the seventh section, we provide our concluding remarks on the project and the overall contributions of our work.

2 DATA PROCESSING INFRASTRUCTURE

2.1 Introduction

In this section of the paper, we introduce our core Data Processing Infrastructure which has supported the development, iteration, documentation, experiment management, and distribution of data-driven experimentation of results. In any data science project, the fundamental life cycle and corresponding project management ecosystem plays a critical role in the methodologies utilized can dictate the integrity, efficiency, and efficacy of any data-driven workflow. Realizing the overall complexity of the problem scope and the methodology used to evaluate our modeling process, we have invested a heavy portion of our project dedicated to first build a core infrastructure which can facilitate many of the day-to-day data-ops process of a typical data scientist in a modular and scalable fashion, while being able to provide enough efficiency to make adjustments in any point of the data processing pipeline.

One of the underlying design philosophies of this pipeline attributes to the major challenges taken from a Software or Data Engineering standpoint, where the *accumulation of ad-hoc code bases and scripts can generate and incur massive* ***technical debt***, which ultimately leads to decreased efficiency and productivity in later stages or iterations of the project, when it either comes to code base, data, or model deprecation, reproducible models, and ultimately model deployment. In particular, the works of Sculley et al. [40] [41] brings to attention some of the critical flaws and traps of much of our modern machine learning modeling methodology, noting that the ratio of the actual *machine learning code base* is much smaller than that of the rest of the other code base dedicated to other data operations such as ETL (extract, transform, and load) modules, data integration, model evaluation, etc. Although much of the frameworks provided today address some components of these issues, the choices of frameworks limits many of the flexibility and domain specific needs that our project requires at both the micro and macro level of our data processing pipeline.

Furthermore, another core challenge which often plague a majority of the data science projects is the *reproducibility crisis.* This problem, however, is not specific to just data science, but almost any field which includes some stochastic component and variability [5]. Hence, this issue of trust and integrity of reported results also contributes to not only the reproducibility crisis, but also to the technical debt issues raised as well. In our work, this is particularly important to address, as we take a data-driven methodology involved with an inherently stochastic system that we are modeling our problem against. Therefore, developing an infrastructure which can facilitate an efficient and comprehensive manner to allow for easy persistence, dissemination, and execution of reported results in a trusted and scalable manner needs to be considered as part of our core design choice for our infrastructure.

Finally, another area within Data Science, in particular Machine Learning and especially Deep Learning, is the *model interpretability* issues, which are also incurred partially within the technical debt issues earlier. In general, modeling processes often play a core dilemma in between the trade-offs between model utility versus interpretability versus complexity dimensions, where researchers must leverage a careful conscience to search a critical sweet-spot which balances the needs of the particular model's usage context. However, we also argue that the scalability of interpretability must also be accounted for when being able to find this sweet-spot. The main question we asked ourselves when designing such system is: *how can we embrace the trade-offs of the various interpretability dimensions, while being able to identify model characteristics and properties from both a micro and macro level perspective - and do so within a scalable manner?*

In this project, we have attempted to address these three key issues as a data engineering problem and developed a fully comprehensive data-driven platform for our project, as a byproduct of the entire modeling methodology we have used to facilitate the development of our models. We introduce an end-to-end pipeline which we have utilized in our project that facilitates processes such as ETL, data integration, visualization to modeling, cross validation evaluation, logging/documentation and model artifact persistence all in a self-contained data pipeline infrastructure. We show, through our workflow process, that our platform enabled us to iteratively and efficiently test out new ideas, evaluate and debug models from various scales, all while persisting and documenting every experimental runs and maintaining a cache of every experimental design tested. Furthermore, we consider this section of the paper as a comprehensive documentation of our core source code which addresses some of the meta-level designs of our project.

We believe that this addresses many of the issues raised above as it: 1) cuts many of the overhead in the development cycle for redundant data pre-processing by using module driven development of data operations, hence minimizing the time spent in developing a lot of "glue-code" and reallocating them towards much more useful tasks where human-intervention is required; 2) we enable our pipeline for enough automation while maintaining a good balance for careful intervention-enabled experiment control; 3) provide a useful mechanism to log, persist, and document experimental configuration, data transformations, and even output metric logs for each unique experiment, therefore mitigating reproducibility and interpretability at-scale.

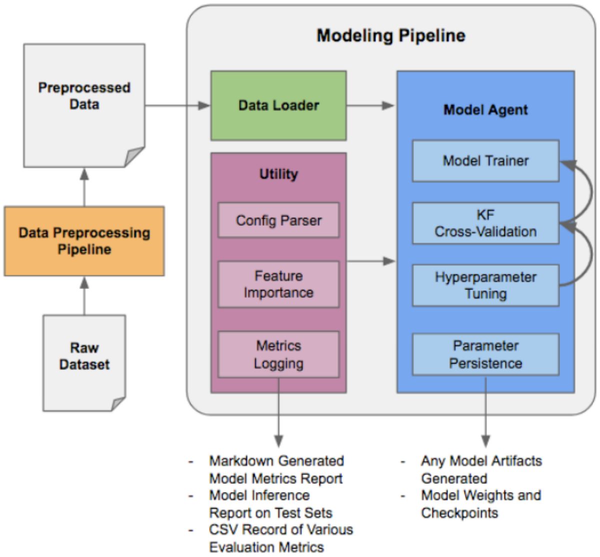
In the following sections, we first introduce our high-level data processing pipeline, where we describe the overview of the different main blocks of our pipeline. Then within subsequent sections, we point out some of the features, design choices, and rationale for each of the individual components. Finally in the last section, we describe some of the results of how these design choices have influenced our workflow process, and demonstrate some of the key use-cases for which highlights both the advantages and disadvantages of our data pipeline architecture.

2.2 Infrastructure Overview

In this section, we cover the high-level overview of our data pipeline infrastructure as illustrated in Figure 1. In particular, we describe the main design blocks of our data processing pipeline and the role they play within our infrastructure. Furthermore, we also include some key details, which are not explicitly illustrated within Figure 1, but are critical components from an operational perspective of the pipeline - these details include topics such as folder structures, Operating System environments, and project dependency management and integration platforms that we have utilized.

First the pipeline's objective is based on the premise of rapid iterative model design, where the input to the pipeline is the original raw and unprocessed dataset, and the outputs to the pipeline are the models (where the parameters are persisted as raw binaries) along with the byproducts such as the configuration and log files and the supporting metrics which report on the performance of the modeling process with respect to the given dataset and user-defined configurations. For this, we subsequently divide this data processing process into four key elements: 1) the data preprocessing phase, 2) data loader, 3) utility and auxiliary functions, and 4) model agent. Each of these components play a critical role which affects the design choices of the machine learning experiments that are performed, and hence are componentized in this given manner. We believe that initially breaking down our components by these critical operations will amount for coarse control of how the data is primarily transformed within the data pipeline. Hence, finer controls and effects for certain transformations and operations performed on the data are abstracted away within each of these components internally as individual modules which can be user-defined, which provides the flexibility for controlled factor-based analysis of how each subsequent operations can make a difference in the way the model performs. In subsequent sections, we will describe the components in detail, along with the sub-modular components which make up each of the components.

Figure 1: Data Processing Infrastructure


Modeling Pipeline
Preprocessed Data
Data Loader
Model Agent
Model Trainer
KF Cross-Validation
Hyperparameter Tuning
Parameter Persistence
Utility
Config Parser
Feature Importance
Metrics Logging
Data Preprocessing Pipeline
Raw Dataset
- Markdown Generated Model Metrics Report
- Model Inference Report on Test Sets
- CSV Record of Various Evaluation Metrics
- Any Model Artifacts Generated
- Model Weights and Checkpoints

2.2.1 *Development Environment.*

Our primary workflows were performed under UNIX-based operating systems (mostly Mac-OSX and Linux-based systems). However, we have not tested this environment directly under a Windows based environment, and hence, setup scripts that we have authored for general file structure generation and raw data download routines must be refactored (using a windows batch file), to be able to fully make use of our platform.

We developed our software using the latest version of Python 3.6.5 (at the time of writing this), with pip3 (Python's Package Management System) version 18.0. Although we have tried to some extent to ensure backwards compatibility with older versions of Python 2, we do not guarantee that it would be fully operational as we have not extensively tested this against previous versions of Python.

All packages and dependencies which we have used in our project is maintained in a simple requirements.txt file, which developers can subsequently use to feed this text file as an argument into pip3 to download and install all packages used within our project. However, there are some packages, such as the Rank Aggregation libraries [20], which were not included in the requirements folder. Retrospectively, it may have been best to utilize pipenv or Docker container to self contain some of the dependencies, along with an automated script to handle all dependency management, so that there would be much of a hassle for setting up the environment for new developers setting up this pipeline.

2.2.2 Project File System. To support and organize work flow in our project, we subsequently structured our folders within the following hierarchical file structure, as shown below. Much of the folders which appear below are primarily derived from the original code base and maintained within Github. However, some folders which are marked with asterisks are folders that are not being tracked - this recursively also includes some of the sub-folder structures within each of the folders as well.

```
\config
    \robovetter
    \tps
\data*
    \feat_eng
    \feat_rank
    \features
    \raw
\docs
\log*
\notebooks
\src
    \robovetter
        \models
        \utility
        main.py
    \tps
        \models
        \utility
        main.py
.gitignore
README.md
requirements.txt
setup.sh
```

Listing 1: Project File Structure

The first folder, \config, is where the underlying configuration folders for each of the experiment are maintained within each of their respective folders, for each phases of the project (\tps being our first phase and \robovetter being our second phase).

The \data folder is where all of the data that we use to perform our analysis are included. Primarily, the \raw folder contains the first, untouched versions of the dataset, while some of the other folders also include other artifacts from our pipeline such as feature importance outputs, feature importance rank aggregation outputs, and auto-generated feature output files. In hindsight, we would have wanted a much better system to maintain this organizational structure of the data that we have used. Furthermore, we would have liked to utilized a much more robust data and log file sharing mechanism similar to Github called, *Data Version Control* (DVC) [1]. Essentially, this is a version controlling system utilized for managing experiment data, model parameters, logs, and configuration files or artifacts from your workflow. We believe that the use of this would have helped us tremendously in better managing, maintaining, and sharing our models, logs, and data much more consistently between our team and our clients that will be utilizing this platform.

The \docs folder is where we place markdown files for quick notes and memos developers can exchange within the research platform. As opposed to the use of notebooks, these notes were simply used as a quick and ad-hoc area for maintaining idea lists, to-do lists, and resource listings for methods and techniques. Throughout our project, however, we did not make much use of this folder.

The \notebooks folder is where all of the Jupyter Notebooks we have used, primarily for Exploratory Data Analysis (EDA) and Model Interpretation Analysis. It was also used as a means for preemptively evaluating the data and working with the dataframes in a much more visual and closely hands-on manner. Much of the plots and visualizations we have generated in this paper are derived from the notebooks alone. One weakness of this structuring was the ability to not be able to cross reference models that were built within the pipeline - as we have essentially repeating much of the modeling in a serial manner within the notebook again (using the results we obtained from the pipeline, however). In retrospect, we would have wanted to be able to reuse the developed module and call an inference API to directly channel our model to generate predictions. However despite the overhead of redoing the modeling, the majority of the notebooks were self contained, so such dependencies to other code bases were not necessary for this particular use of the project.

However, the majority of the work in the codebase was done within the \src directory of the project, as this is where much of the pipeline components and their submodules reside. Within the \src folder, we have two pipeline (essentially duplicates), of the pipeline code for each of the respective phases in our project. We did this to self contain much of the development process, such that modules we build for one phase is restricted within that particular phasing. However, we did not see that much of an overlap in some of the modules which were used between the pipeline, hence this process have worked out well in our case. For future development, it would have been better to have two sets of \utility folders - both within and above each of the phases of the codebase. The higher level utility directory can be used to handle any high level auxiliary operations and data loading, configuration, and feature importance processing, while lower level utility functions can be used for phase specific data operations.

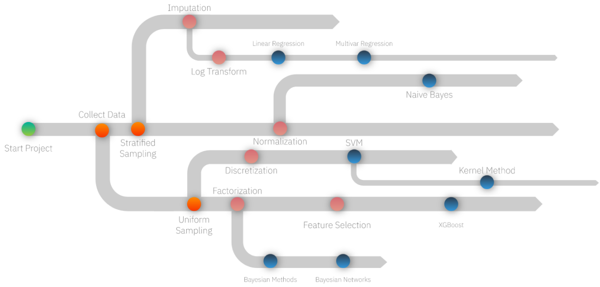
Furthermore, within each of the \srcś phase folders, we have a \models folder, where all of our model pipeline agents reside. In our project, we call the entire modeling process, we use what is known as an *agent*, which is responsible for the operationalization of the modules and integration of the pipeline in an end-to-end fashion. The code we have developed follows an OOP (Object-Oriented Programming) principle, where we have a base agent, which acts as an interface to how all subsequent agents are built. This acts as a guiding framework for us when designing the agents, as they normalize all function calls and operational process, such as training the model, evaluating the model, or loading and saving model weights. We did this to ensure that every model we have built can be fully customized for every specific model - but also designed our process in a uniform manner, such that our main.py, our pipeline's main entry point program can consistently access each of these operations with the assumption that these syntax are initially established uniformly.

2.3 Data Preprocessing Pipeline

In this section, we discuss the first major component of our modeling pipeline, which is the data preprocessing step. In our pipeline, we distinguish our preprocessing steps within two different types of operations: *offline vs online* data preprocessing. In offline preprocessing, we denote this as any type of operation which is not operationally repeated within the pipeline, where the raw data transformation only happens once and the resulting values are persisted in another file. On the other hand, online data preprocessing refers to any type of data operations which are initially applied to the dataset prior to directly being fed into the model, hence not persisted onto a flat-file for preservation. We made this decision, due to considerations in how the scalability of the platform was going to affect the way the model was being built - with respect to the complexity of the operations and transformations applied to the data, as well as the overhead time spent on performing these data transformations.

Primarily, a heuristic we utilized to determine whether something is considered an offline vs online based preprocessing operation was based on the different factor levels of the type of transformations that are possible. In other words, the more options there are in the ways we can perform a transformation on the data, we stronger case we would make to utilize an online process - as we can optimize for the best type of operation as a search-based procedure which can help to maximize our objective function, and vice versa.

Figure 2: The Cambrian Explosion of Models



By analogy, we consider this to be almost the *Cambrian Explosion* of models, as illustrated in Figure 2. We believe that, at every junction point of how the model changes (i.e. via operations, transformation, integration of the data) can have a huge impact in the way subsequent models are produced. The operations on the left can be considered as much more coarse evolution, while towards the right, we can consider them as finer operations, many branches of possible data preprocessing methods and modeling techniques used. Higher level changes, can influence mainly heavily on the optimized metrics, while finer level transformations and operations can influence marginal performance differences.

Concrete examples of how some of the operations that we considered to be coarse were the creation of new features - in particular the CCD channel augmentations. In our data preprocessing pipeline, we have operationalized those process offline and persisted those in a flat file for which subsequent modeling pipelines can make use of. On the other hand, finer data preprocessing methods included simple log-space transformations or missing-data imputations.

Considering scalability in mind, the original datasets that we are using for the model is not relatively large, as opposed to some of the big data problems (which goes into several hundred gigabytes or terabytes), we may potentially opt-in to utilizing Spark for some of the preprocessing and feature engineering steps - so that we can precompute those values and persist them as a file which can then be fed into the modeling process. However, in most cases we aim to develop our feature engineering process as a multi-threaded process which would concurrently run the preprocessing routines on numerous threads on the CPU.

2.4 Data Loader

In this section, we discuss the Data Loader component of our project, which its primary responsibility is to execute the ETL functionality of the pipeline during the runtime of the modeling process - which includes loading the dataset onto memory and performing any necessary operations or transformations over the dataset. Carrying on the discussion from the previous points raised in the data preprocessing phase, the data loader is responsible for not only applying micro-level operations which have a wider search-space of possible operational parameters that can be applied to the data. Hence, these transformation choices can also be applied over as key hyperparameter selections for when building models.

In our data loader function, we worked directly with the raw data through loading it on memory by using AstroPy [3], a package to handle data driven operations on astronomy or astrophysics data. The data was then converted into a Pandas dataframe, where further operations and transformations were made possible. Within the dataloader, we can define modular components which include operations, transformations, imputations, all as function calls which can be enabled or disabled using binary flags in the JSON-based configuration files. For instance, this can be things such as feature selection, instance selection, Principal Component Analysis, data imputation processes, and etc. The final output from this data loader function would be the dataframe containing the features (X) and the target values (y) of a typical Supervised Machine Learning problem, which can then be subsequently feed into train/split/test sets or through a cross-validation evaluation procedure.

2.5 Utility

In this section, we discuss some of the utility functions that are present within the Data Processing Infrastructure on our code base, which reside under the `\utility` folder, for each of the respective phases of the project. The utility components within the project are essentially a lose collection of both self-contained scripts and libraries that help the typical data-ops processes within the pipeline. For example, standalone scripts such as the feature importance and feature rank aggregation scripts reside within this folder. However, there are other dependencies and auxiliary functions which are required for the processing of the pipeline, such as reading and writing configuration files for the experiment designs, as well as the scoring metrics and evaluation report markdown generators. Here, we describe just some of the major modules that we have built that facilitated the majority of our data-op processes.

2.5.1 Configuration Files. The configuration file reader in our platform is based on a JSON-based structured file which can dictate an experimental design of the model that the developer would like to produce. We show an example of a sample (partial) experimenal configuration below. The purpose of this is to provide a systematic method of quickly testing various factor levels and experiments without needing any modifications to the direct source code, still while being able to provide the power and flexibility to add and remove lower-level operational modules of the code. Furthermore, another added benefit for the user to be able to preserve the configuration file enables reproducibility and documentation of the modeling process right out of the box.

```
{
    "model": "catboost",
    "out_dir": "log",

    "data": {
        "plti": "data/raw/tces/kplr_dr25_inj1_tces.txt",
        "noise": "data/raw/misc/DR25_DEModel_NoisyTargetList.txt",

        "features": ["period", "epoch", "Expected_MES", "NTran", "depth", "duration", "Rp", "Rs", "Ts", "logg", "a", "Rp/Rs", "a/Rs", "impact", "SNR_DV", "Sp", "Fit_Prov"],

        "feat_trans": true,
        "ifs": true,
        "pca": false,
        "drop_noise": false
    }
}
```

Listing 2: Sample Configuration File

As a typical workflow process, the researcher would first define a configuration that they would use by defining key parameters like the dataset, what type of preprocessing would be done, and any other hyperparameter changes that they want depending on the model that they were using. Then the user would simply call the main entry point of the program, `main.py` and feed in the configuration file as the only argument into the program, where the program will parse the configurations and construct the entire data processing pipeline in an end-to-end fashion. As a byproduct of the modeling process, the pipeline not only provides some of the logs and metrics derived during the modeling procedure, but also duplicates and persists a carbon-copy of the JSON-based configuration file within the output directory of the model artifacts. Therefore, another user or researcher can take this same exact JSON based file to essentially: 1) be informed about the summary of the experiment and understand very quickly how the experiment design was setup without having to read any papers or written documentation, and 2) be able to take the same configuration file and feed it as an argument into the entry point script again to obtain the *same exact* results as reported by the original metrics from where the results have been claimed by, and 3) provide a higher-order semantics for data-driven research practitioners to experimentally try new experimental designs from forking known working models.

Practically, the implementation of such design has help significantly reduced the overhead of our own workflow as we were able to efficiently use this data to formulate new experiments, perform meta-analysis over past experiments, and easily document our work without loss of records or having to re-find the exact parameters we used to obtain those reported results. This has helped save a lot of time in the long run and believe that this paradigm of meticulous experimentation logging should be a key practice when covering a large search space of datasets, parameters, operations, and models to try. In the foreseeable future, we see that this can also benefit meta-learning based models, which can make use of these meta data generated from each of the experiment, to understand and learn best practices for how to perform things like data transformation and data preprocessing over various types of machine learning models and parameters.

2.5.2 Feature Importance. One of the major modules, which has contributed a lot of the development of the modeling process was the feature analysis process. The primary objective in a feature analysis is to identify key features which are likely to improve predictive power of the model. This was a procedure that allowed us to select the best feature for the model to use in our modeling process, and subsequently helped to form the decisions we made as to which features were being selected for each modeling process. Although, we have noticed that these feature importance metrics were useful in identifying key features that we can utilize for modeling by feature selection, sometimes there were cases where multiple ranks with different metrics made it difficult to judge which features to utilize in our models. Hence as a pragmatic approaching this problem, we also included modules which helped to aggregate different ranks together as a single rank so that we can establish which features are useful and not in a quick and easy heuristic. More details on the algorithms and methods used to formulate these rank aggregations are detailed in subsequent sections of the paper.

2.5.3 Metrics Logging. The third most used component of our data processing pipeline was the metrics logging module which was responsible for the overall logging, persistence, and reporting of the metrics tracked for each experiment. One of the design principles behind this logging module was to provide both a user and machine readable format for presenting results for machine learning experiments. In particular, one of the emphasis of the experiment logging was to enable debugging of models by evaluating individual fold analysis of models for each k-fold iteration of the model. This was a key part in understanding the variability between each of the model's data distribution and has helped us checked to see if our models were potentially over-fitting on the data. To mitigate this, we opted to persist every single user-defined metric over each of the fold, in addition to the Area Under the Curve (AUC) data per fold and even persisted each of those within a CSV file. However in some cases, we did opted to keep only the average of the confusion matrix, as it was not as significant as the AUC was sufficient.

The final output from this pipeline results in a Markdown formatted tables with all of the metrics laid out and easy to read (with any markdown viewer). This enabled us to read any of the logs without much difficulty navigating the results and consolidated a summary report of the model's performance. From a pragmatic standpoint, we would hope to see better solutions for experimentation logging and management in a web environment, where users can easily search and sort through past experiments without hassle - such as a side project that has emerged as a byproduct of this original design principle that attempts to achieve this, called *monolog* (which was originally intended for Deep Learning) (https://github.com/yutarochan/monolog).

2.6 Modeling Agent

One of the most important core of the data-driven modeling platform that we have built is the Modeling Agent, which facilitates the setup, construction, and execution of the user (or machine, if using a meta-level configuration generator) defined models from the configuration file. For each different model variation used (i.e. Logistic Regression, Categorical Boosting, etc), each corresponding model object is encapsulated within an agent which is responsible for performing training, validation (whether by a simple split or by k-fold validations). All agents inherits a base class which provides the function for training, validation, hyperparameter tuning, loading, and persisting model objects.

The added benefit for this allowed us to configure custom modeling profiles for different types of experiments - in particular ensembles, where more than models are trained. Here we can import other existing agent's that have implemented the base model and devise a new model agent just for the ensemble model - which allows us for full code modularity. From a pragmatic standpoint, this has helped us organize our codebase in maintaining modular components which enabled us to add more models as we see fit, making it a scalable way to implement new models effectively and efficiently.

2.7 Conclusion

In this portion of the paper, we have presented our data-driven experimentation framework, along with key implementation details of how we attempted to address some of the key issues raised in the introduction. In particular, our key contributions to this project was this pipeline, which avoided some of the technical debt issues most machine learning projects have a tendency to fall under due to various organizational issues. With the allotted time and infrastructure in place, we have been able to effectively spend time to work on more interesting dimensions of the project, such as model interpretation and analysis. We cannot emphasize enough of the importance of how project organization can lead to efficient use in the overall data-driven research process.

Some of the main advantages of this pipeline are: 1) increased efficiency of the overall workflow due to how well the codebase, files, and logs are placed, 2) systematizing and automation can help decreasing overhead where repetition is present, while keeping areas of analysis and human-in-the-loop intervention relevant in the most key parts of the framework, and 3) allows for a robust framework that can enable reproducible workflows which can be shared across different teams working on such projects.

However, from a pragmatic standpoint, we found there were some disadvantages to how we approached this data engineering problem. During the process of modeling and data, there were some points in the project where we had to pivot the approach we took in the way we performed our data analysis methods. This pivot has influence and impacted the overall design choices due to the preemptive assumptions of the best way to approach a problem. In particular, this was a great problem when we realized that the focus of the project should include more interpretation and model reasoning than a optimization-centric approach in developing our capstone, we had to restructure our codebase in a way to accommodate for the ability to perform the various analysis. Hence, we noted previously that much of our work in the notebooks were simply reproduced from scratch and not called from existing codebase due to a flaw in our original experimental workflow design. Thus, this framework is only viable if one has a solid plan as to how they would approach a plan and sticks to that methodology from the beginning, then the utility gained from the platform would be high as possible. However, this is not to say that this framework would be appropriate for all problems, as it falls under the same fallacy of the No Free Lunch Theorem, where one approach would be able to address all different types of problems. We learn here that for a framework to be pragmatic enough to address a problem, we would need to design it within a balance between something which can dynamically fit the need of the problem while keeping it consistent and fixed for certain portions.

For future work, we hope to take this opportunity to take some of the lessons learned and develop a generalized framework that practitioners can make use of. We plan to release parts of this codebase as an open source solution for data scientists to be able make use of some of the components to facilitate code re-usability and ensure that replication of experimentation is possible to maintain robustness and integrity of the reported results.

3 PLTI ROBOVETTER INJECTION MODEL

3.1 Introduction

The Kepler mission was designed to discover Earth-size planets orbiting Sun-like stars. During the 4 years of its deployment, the Kepler collected the data for over 160,000 stars at a time, identifying over 4700 planet candidates through its multi-scale Bayesian Maximum A Priori (msMAP) method of narrowing down candidate planets [45]. Prior to the termination of its mission, the subsequent data processing pipeline for the Kepler mission has constantly been evolving to allow researchers to discover candidate Earth-like planets for further investigation. For this, NASA provides to the public a complete set of the dataset known as a "Data Release", which includes both the inputs, outputs, as well as the raw target pixel files for external researchers to conduct their own photometric analysis on the dataset. These data releases correspondingly were updated per cadences, which are the total period of time when the telescope captures, co-appends, and stores pixel-level telemetry. In this work we make use of the DR25 version of the data catalog released by NASA for our project.

To best characterize the efficiency of the Kepler Robovetter pipeline to successfully extract planetary or non-planetary objects from the raw pixel-level data, we build models which take in various features from the input data and predict whether the model is able to detect a Threshold Crossing Event (TCE). However, since the nature of the data comprises of sparse observations from the sky, telemetry and data acquisition is very difficult in order to obtain proper ground truth data. Therefore, simulation experiments where synthesized data are injected directly into the raw inputs are used to determine empirically whether the model is capable of detecting these TCEs. To simulate these TCE phenomena, NASA has performed primarily two categories of injection based experiments: Pixel-Level Transit Injection (PLTI) and Flux-Level Transit Injection (FLTI) [12]. In this project, we identify key characteristics of the current pipeline by analyzing its strengths and weaknesses as well as the efficiency by which the pipeline identifies planets and classifies non planetary objects based on the attributes provided in the DR25 files from both the PLTI and FLTI based simulated data.

Current work in this space to characterize the efficiency of the models are only based on a simplified statistical analysis of a single-feature, notably the Expected MES, to model the effectiveness of the pipeline [12], [18]. However, methods listed does not exploit key features from the Data Release inputs and other statistical and machine learning based approaches which can make use of these additional features. In this work, we have characterized the efficiency of the Transit Planet Search system pipeline, identified key features, parameters, and their relationships to the input parameters as a probabilistic model to determine whether the pipeline has successfully recovered the corresponding target signal. Effectively, this analysis will allow us to build our understanding of the Kepler Robovetter Pipeline and how each of the features play a critical role in determining whether a light curve is indeed a planetary object or not.

As a result of successfully building a predictive model on whether a TCE is considered as a planet or non-planet planets, this allows us to characterize new features and potential combination of attributes. Along with preliminary exploratory data analysis (EDA) process, model weight parameters and feature importance metrics, we can effectively characterize and reason through the data on factors and contributors from the data features to better understand how to detect new planets. Furthermore, from applying this model over unseen data with the same given features to potentially identify new candidate Kepler Objects of Interests - leading to discoveries of new planets and other non-planet based objects that were previously unidentified.

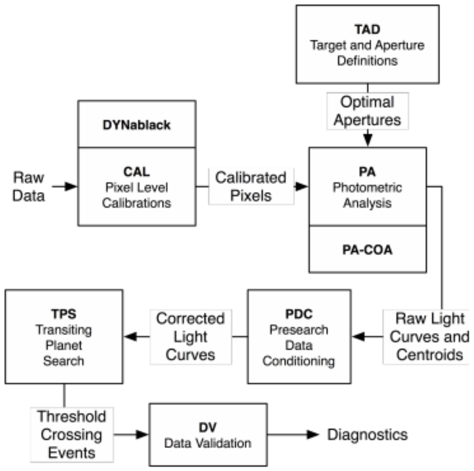
The objective of the this phase entails the development of a model for predicting the efficiency of the pipeline to detected whether a planet's Threshold Crossing Event (TCE) of the Transit Planet Search (TPS) system pipeline is able to recover signals of the injected data or not. This would entail developing a machine learning based model which would take various features provided by the Kepler dataset to devise a probabilistic model which would predict the overall efficiency from the data generated by the TPS pipeline system based on whether the signals recovered by the simulated injections were recovered or not.

We structure the first phase portion of the paper as follows: First, the current section introduces the first problem we are addressing through our modeling process. Then the second section evaluates the features found in the DR25 PLTI dataset used to model the efficiency by describing each of the distribution characteristics and providing contextual information for each of the features within the dataset. In the third section of the paper, the Feature Engineering section evaluates some of the methods we have utilized in attempting to derive new features which our models can exploit to model the efficiency of the PLTI pipeline. In the next section, we evaluate the Feature Importance of each of the described feature based on the evaluation of various key feature importance and selection metrics and methods of aggregation of feature importance ranks. The fifth section of the paper addresses and defines each of the algorithms we have utilized to build our probabilistic models, and describe briefly some of them with further references one can consult to for better understanding the modeling process of each algorithm. Within the modeling section, we also describe some of the key parameter selections for each models we have developed, which includes the original baseline models proposed by Christiansen et al. [12], ensemble strategies, and feature engineered methods. The sixth section describes the methods of how we perform our model evaluation, which introduces the metrics we utilize, as well as the cross validation methods we use to validate the model to assess the performance of the predictions empirically. The seventh section presents the results obtained from the our empirical evaluations, along with various remarks and comments on the performance based on any surprising or sensible characteristics from the results shown. We reserve the Analysis section of the paper in the fifth portion of this paper which provides an in-depth analysis of the top performing model and contextualizing the results within some of the results we have obtained from our model.

3.2 Dataset

The dataset used to model the first phase probabilities was based on the Data Release 25 file set, "kplr_dr25_inj1_plti.txt". The dataset was based off of the Pixel-Level Transit Injection experiment that was utilized to measure the efficiency of the detection of planets [12]. The pipeline was based on the SOC pipeline version 9.3 and DR25 Robovetter model processing uniformly across all of the light curves, based on the work of Thompson et al. [44], from the Q1-Q17 DR25 sets based on the stellar properties of the work in Mathur et al. [34]. The data set derived from this process, as shown in Figure 3 [29], is obtained from the "Transiting Planet Search" phase to produce a *Threshold Crossing Events*, where candidate data samples of interest (Kepler Object of Interest or KOI) are fed into a Data Validation phase for further evaluation.

Figure 3: Kepler Science Data Processing Pipeline



The features utilized in this dataset are based on the input are based on the following key variables which were permitted for use in modeling - note that majority of the parameters are based on the "input" values into the Kepler Robovetter pipeline: KIC_ID, Sky_Group, i_period, i_epoch, N_Transit, i_depth, i_dur, i_b, i_ror, i_dor, Expected_MES. In this section, we briefly describe each of the features in the data set and provide a close observation of each parameter. Furthermore for each parameter, we visualize each of the variable it's corresponding Sky_Group CCD channel. *Refer to Appendix D of Thompson et al. for the corresponding CCD channel configurations* [44].

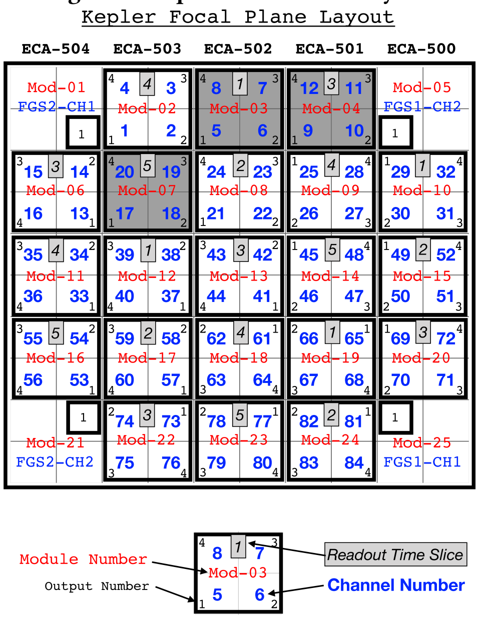
KIC_ID

The KIC_ID refers to the corresponding light curve of the Kepler Input Catalogue ID of the target star. We utilize this as a primary key to identify each unique sample during the modeling process.

Sky_Group

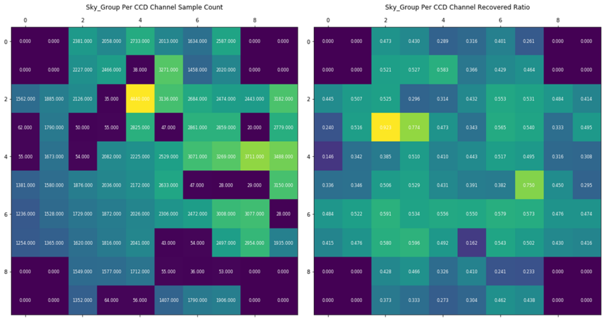
The Kepler telescope is based on a array-based sensor configuration of photometric sensors which are arranged in a specific orientation based on the Charged-Coupled Channel - a photometric instrument which utilizes p-doped Metal-Oxide Semiconductor (MOS) to capture pixel information from light sources. However, as the telescope captures images of spatially shifting images, each corresponding stars will fall on different CCD channels. The Sky Group variable is an integer based representation (from 1 to 84) which spatially groups together each of the stars in a time-invariant manner. The sensor configuration and orientation for each of the CCD channels are arranged in the following manner, as described in Figure 4.

Figure 4: Kepler Focal Plane Layout



In the subsequent data analysis process, we plot each of the distribution per CCD channel to gain an insight to the overall distribution of each parameter per channel. As shown in Figure 5, we visualize the overall sample count and Recovered Signal ratio for each CCD channel. Empirically, we can observe that there are certain regions where there are more signals present than other channels and also some channels where there are more recovered signals than other channels. This is because of the samples that specific channel areas were selected within the Data Catalog were there was a much less distribution of earth-like planets in certain regions more than others or there were some regularities with the way the data was collected due to noisy sensors. Note that each of the cell grids of the CCD chanels in the plot correspond to the values as shown in Figure 4.

Figure 5: Sample Count (Left) and Recovered Ratio (Right)



Input Period

The input period, or i_period, is the orbital period in days of the injected signals. The orbital period is a time period in which accounts for how long it takes an astronomical object to complete a single orbit around another planet. Mathematically, we can can define the orbital period by the following formulation below, according to Kepler's Third Law:

$$T = 2\pi\sqrt{\frac{a^3}{\mu}}$$

where T is the orbital period defined in seconds (we can subsequently convert this into days by dividing the equation by 1.15741×10^{-5}, and a is the orbit's semi-major axis, and $\mu = GM$, where μ is the standard gravitational parameter, which can be derived by multiplying G, the universal gravitational constant ($6.67408 \times 10^{-11} m^3 kg^{-1} s^{-2}$), and M, the mass of the more massive body. We later utilize this formulation to derive new features through domain knowledge of equations.

As shown in Figure 6, we see that the per channel distribution of the epoch appears to be mostly uniform due to the spread of sample values captured in a uniform manner. However, we can see that for some CCD channels, some of the distributions exhibits a pattern of sparsity in some regions. This plot also resonates somewhat similarly to the plots in Figure 5, as some of the high recovery ratio values are somewhat associated with the sparsity in the distributions of the sample spans. In this plot, the distributions are further segmented by whether the sample comes from a recovered or a non-recovered signal - where they are encoded as blue and orange, respectively.

Input Epoch

The input epoch, or the i_epoch variable, refers to the epoch value

Figure 6: Period Per CCD Channel Distribution

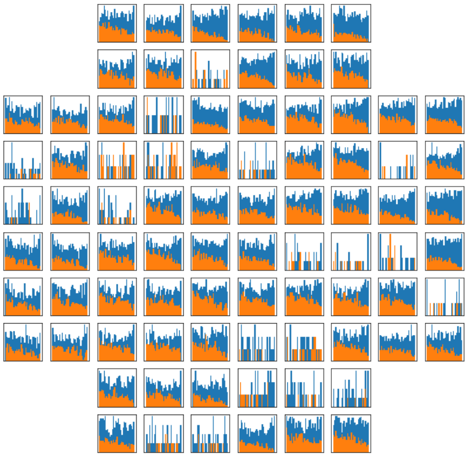
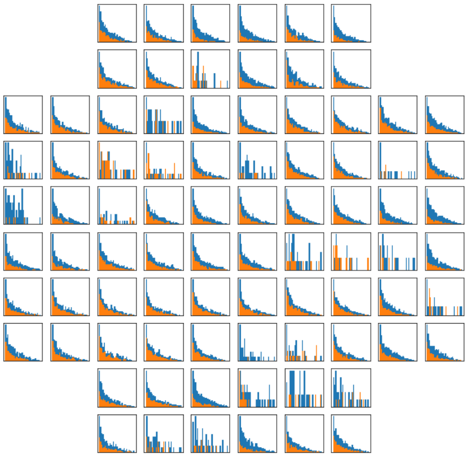


Figure 7: Epoch Per CCD Channel Distribution



used in the injection simulation. An epoch, in an astronomical context, refers to a span of time used as a reference point for a time-varying astronomical quantifier. The values for the epoch values are encoded as an eight byte, double precision, floating point number [45]. This value representation, also known as the Baycentric Julian Date (BJD), is used to represent 2.5 million years; however the original time encoding cannot be used due to its large size - and thus Kepler reports the value of BJD-2454833.0.

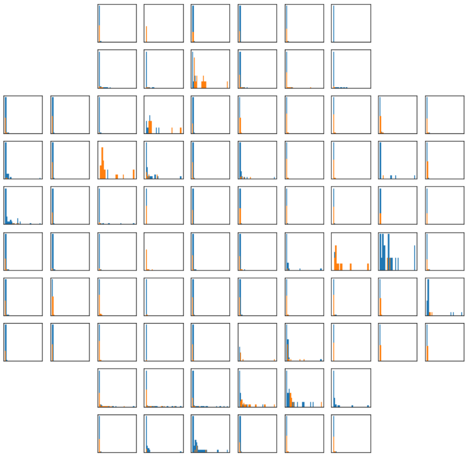
Based on Figure 7, we see that the signals in each channel approximately represents a Zipfian distribution. We can also observe a similar phenomena with regards to the signal sparsity present in some corresponding CCD channels, where the signals that are sparse are likely to have higher occurrences of the recovered signals, than those that are not so sparse.

N Transit

The N_Transit feature in the dataset quantifies the number of valid transits contributing to the Expected MES signal. According to [12], "valid" signals are classified based on the cadences, which are the sample rate of observations per time span weighted between 0 (for invalid) and 1 (for valid), are not de-weighted by less than 0.5. These weightings are applied based on whether there exists a data gap or an anomaly in the provided data.

Based on the plot shown in Figure 8, we see that for the majority of the signals are distributed closely towards 0, which indicates that there are fewer valid transits. However, as we have seen numerous times already, some signals that appear to be closely related to whether they are recoverable tends to have more signals that have higher valid transits. This indicates that signals that are of higher quality improves the likelihood of a signal recovery by the pipeline.

Figure 8: N Transit Per CCD Channel Distribution



Input Depth

The input depth, or i_depth, corresponds to the variable which indicates the central transit depth of the signal, measured in parts-per million (ppm). The central transit depth refers to the deepest point of the light curve where the two planets overlap one another. This value is measured after the signal correction has been set into place.

As shown in Figure 9, we see that the overall distribution of the dataset are heavily skewed towards the right. However, we see that for some of the CCD channels, we see the same emerging phenomena again with some signals being present with the same type of behavior, except the variance of the signal is much wider than those that do not have any recovered signals.

Input Duration

The input duration of the dataset, or the i_dur, indicates the transit duration of the signal. As shown in Figure 10, we see notice that the behavior of the signals are very similar to the one shown in Figure 9, where the signal distribution for some of the channels have a high amount of variance and the presence of a recovered signal is available in those cases.

Figure 9: Depth Per CCD Channel Distribution

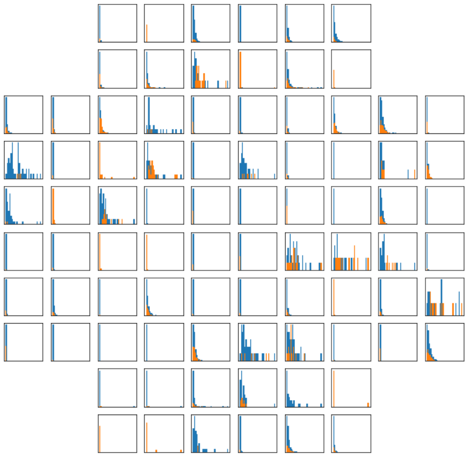
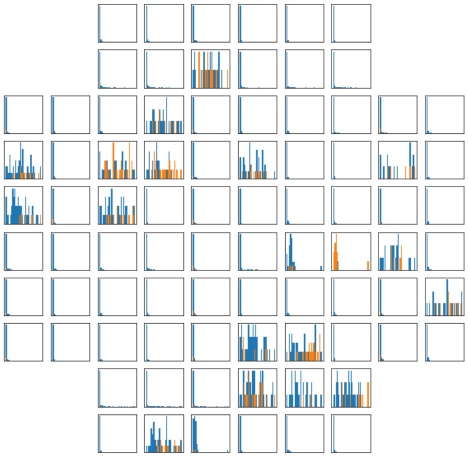


Figure 10: Duration Per CCD Channel Distribution



Impact Parameter

The impact parameter of the model, variable i_b, indicates the perpendicular distance between the path of a projectile and the center of the potential field $U(r)$, created by an object that the projectile is approaching. As demonstrated in Figure 11, we see that the figure demonstrates the same uniform distribution, except in few regions where the signals are very sparse.

Planet Radius and Stellar Radius Ratio

The parameter, i_ror, indicates the injected ratio between the Planet Radius and the Stellar Radius for the input signal. As demonstrated in Figure 12, the majority of all distributions are skewed towards the right, and most of the signals exhibit similar behaviors seen so far. Except, we should also take note that some of the signals in certain channels are dominated by signals that have either recovered all signals, and those that do not have any recovered signals.

Figure 11: Impact Parameter Per CCD Channel Distribution

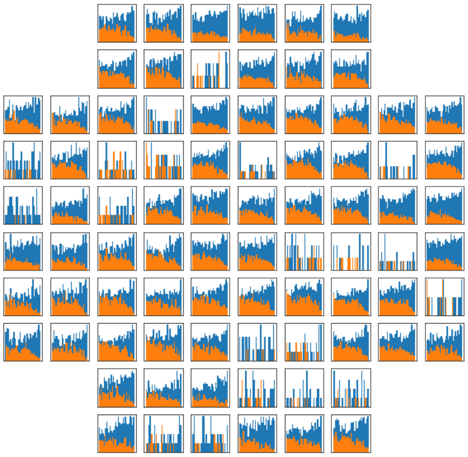
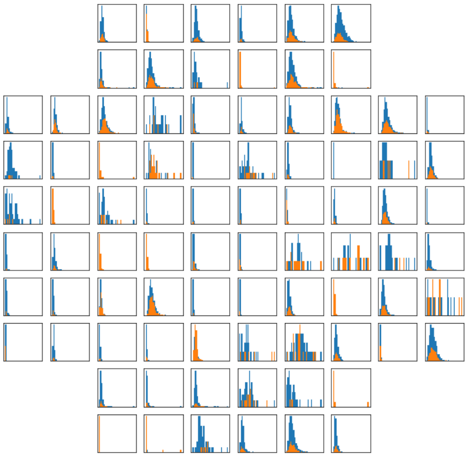


Figure 12: i_ror Per CCD Channel Distribution



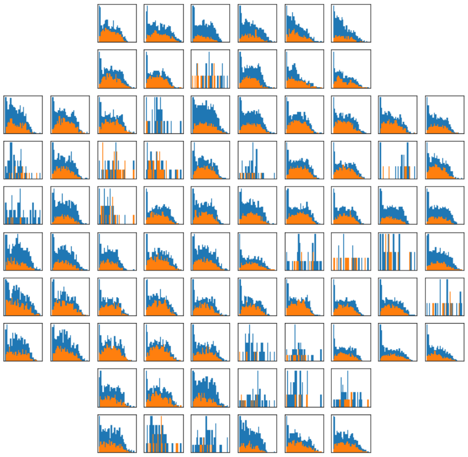
Semi-Major Axis and Stellar Radius Ratio

The parameter, i_dor, indicates the ratio of the planet's semi-major axis and the stellar radius ratio of the injection signal. As demonstrated in Figure 13, we see the data distribution for each channel is not as skewed as some of the other features in the data. However, the spread of the data is wider compared to some of the other variables, with the sparsity of the data present in the same channels as seen in other variables.

Expected MES

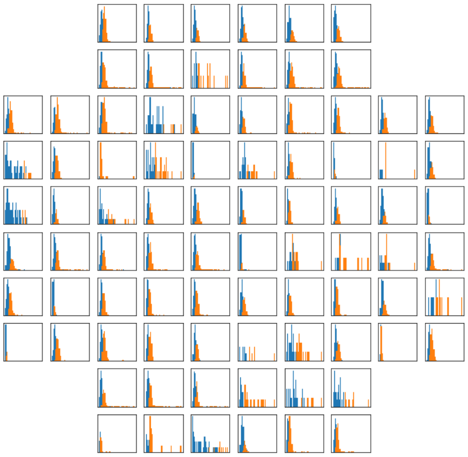
The Expected MES, or the Multiple Event Statistics, is a parameter which indicates the strength of the transit signal relative to the noise [45]. This signal was computed based on a Monte Carlo simulation experiment where Van et al. injected signals of transiting planet signatures into the calibrated raw pixels and evaluated the resulting distribution of the detected signals. Previous models [45] made use of this signal to generate a 1D-feature based prediction of how likely the signal can be recovered from the pipeline.

Figure 13: i_dor Per CCD Channel Distribution



As demonstrated in Figure [**?**], the variance of the signals for the expected MES is fairly small. Likewise with this plot, we see that the majority of the plots demonstrate a high rate of occurrence for recovered signals more so than non-recovered signals. This indicates that the presence of the Expected MES indicates the recoverable signal, which is also why it can be considered as a highly weighted feature, as later indicated by the feature importance analysis.

Figure 14: Expected MES Per CCD Channel Distribution



3.3 Feature Engineering

In this section we describe methods we have utilized to perform various feature engineering techniques. Notably, the primary feature engineering techniques we employ are primarily based on known domain knowledge-based expertise and deriving methods in using explicit knowledge and encoding them into the model as additional features of the model. We propose two different heuristics in our experiments using i) astrophysics equations and ii) Sky_Group CCD Channel Pooling. In our experiments, we use the second Sky Group based pooling operation, and reserve the first method of feature engineering as a proposal for a method for future modeling procedures.

3.3.1 *Astrophysics Heuristics.*

The effort of deriving the new features using the domain knowledge based on Astrophysics has successfully revealed some of the features which may result better performance of the models. By taking a top down approach, we can approach the method of generating new features based on known facts or simple principles. On the other hand, other methods of combining features includes the use of brute-force combinatoric approaches which would be ideal for exploratory feature-synthesis, which falls under a bottom-up approach for methodically approaching this problem. Although we will not make use of these heuristics for our model development, we propose some plausible methods that can potentially be used to encode known domain knowledge in a much more explicit manner and can be potentially useful features that can help to improve the predictive power of such models.

In our modeling process, we make use of only the Data Release 25 file set, "kplr_dr25_inj1_plti.txt", to derive features. However, we decided to extend our reach and utilize another dataset from Kepler Input Catalog [2], which includes the effective temperature of the stellar objects. The effective temperature is the measurement of the total amount of electromagnetic radiation. Below, we will demonstrate the derivation of the Stellar radius of the objects (S):

$$S = \sigma * T^4$$

In this formula, T stands for the object's surface temperature while σ denotes the Stephan-Boltzmann constant. As shown, the Stellar radius can be represented as a function of the surface temperature. The newly acquired attribute, effective temperature, is often used as an estimator of object's surface temperature. Using such domain knowledge, we are now able to estimate the stellar radius.

In the given dataset, another feature that can potentially be exploited for further augmentation was i_ror, which was defined as the ratio of stellar radius to the planet radius. Using this attribute along with the known stellar radius, we could now derive planet radius. The radius can be used to further explore the physical characteristics of the objects such as cross sectional area, volume, and the density. In addition, we can derive the semi-major axis in a similar manner. The i_dor was defined as the ratio of stellar radius to the semi-major axis of orbit.

We believe that the data of the star which the objects orbit, can also provide helpful insight for distinguishing the Kepler Object of Interests. Since estimations for semi-major axis are acquired, we can now compute the estimated mass of the star using the formula for the period of orbit. The following equation demonstrates the derivation of star mass using known attributes:

$$T = \sqrt[3]{\frac{GMT^2}{4\pi^2}}$$

Since M is a mass of the larger object, we can derive the estimated value for the mass of the star. The dataset includes the attributes called i_depth, which represents the central transit depth. This variable is used to calculate the size of planet using the known size of star, and its value is equal to the ratio of the cross sectional area of the planet to the cross sectional area of the star. Because we already estimated the cross sectional area of the planet, we can also estimate the cross sectional area of the star using i_depth.

By including one attribute from another dataset in the Kepler Input Catalog, we could derive or estimate many of previously unknown attributes including radius, surface area, volume, and the surface temperature of the planets as well as some useful data for the stars. Having these features can help to uncover potentially confounding or lurking variables for predictions and can better ensure that we capture the causality attributed for each predictions.

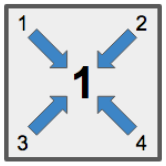
3.3.2 Sky Group CCD Channel Pooling. When investigating the Sky Group, we can immediately notice that there are certain spatial properties we can extract and modify within the representation of the Sky Group parameters. With this, we aimed to devise and produce new transformation mapping functions which can help to reduce the feature domain through binning the data in a way which incorporates these spatial properties.

For this, we propose three new transformation mappings to reduce the overall space of the data down to some smaller set of categories from the raw integer representation: i) Center Pooling, ii) Corner Pooling, and iii) Orientation Pooling.

Inspired by the techniques proposed in Convolutional Neural Networks [17], we look to the pooling operation which reduces the spatial dimensionality of the dataset by some heuristic function. In CNN architectures, this is often used to help the model to generalize across many pixel level variances across the data within a spatial context. Just like in CNNs, we have methods like average pooling or max pooling, where we utilize certain methods to selectively filter out and combine information to reduce the dimensionality of the image. For our purpose, we utilize a similar heuristic in which we attempt to reduce the number of categorical features of the Sky Group parameter, however, in utilizing different pooling methods to better bin the data using the spatial domain knowledge provided by the schematics of the Kepler CCD Sensor Panels.

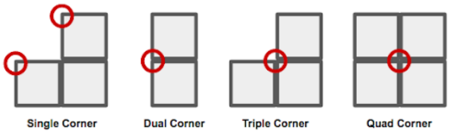
Center Pooling: In each panel, there are 4 different CCD channel sensors that are attached to it, and they total up to 24 different panels with 84 sensors. The first method, Center Pooling, is a method which essentially groups the four CCD channel per panel as a single entity, as demonstrated in Figure 15. This method yields in the reduction of the data by up to 21 different categories, rather than the 84 different CCD channels originally provided in the Sky Group.

Figure 15: Sky Group Center Pooling



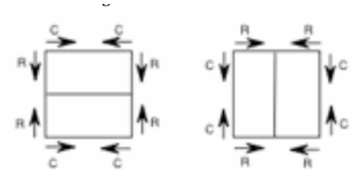
Corner Pooling: The second method of pooling that we have devised is the Corner Pooling method. Similar to how we pool each channel to a fixed point, we consider the fixation of the pooling to a particular intersection of the different panels or the corners of the spatial arrangement of the CCD panels. In the same notion that convolution operators have filters (rough heuristics for matching each sub-region), we identified four different filter-types in which we perform our pooling operations on, as demonstrated in Figure 16. The Corner Pooling operation simply groups together adjacent corners of the panels together and fixates them into a single value. Intuitively, this pooling operation can produce a higher bound of the number of category as it does not handle some of the corner values and focuses on grouping inner CCD channel values.

Figure 16: Sky Group Corner Pooling Heuristics



Rotational Pooling: The last type of pooling operation, known as the Rotational Pooling operation, looks at the rotational variation of the panels. One thing to observe closely from Sky Group property, is that the panels are not oriented in the same direction. This is described by the directional lines and the way the channel numbers are ordered on each of the panels - which boil down to four different types, as shown in Figure 17. We consider this operation to produce a lower bound on the categorical feature sets that we can use as one of our features.

Figure 17: Sky Group Panel Orientation Heuristics



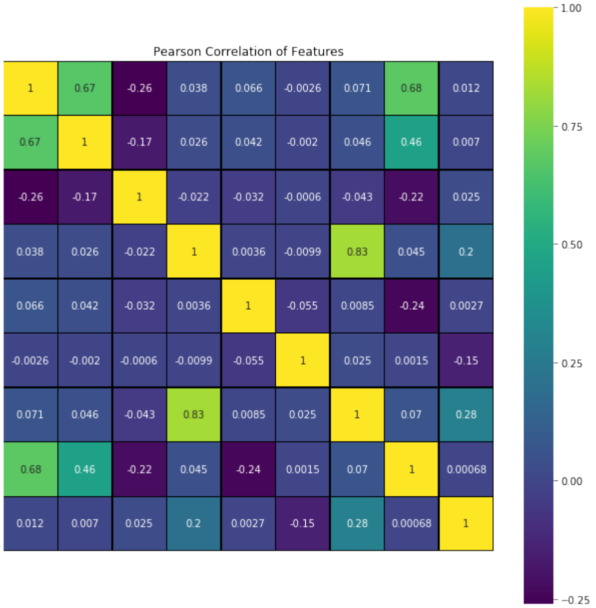
As demonstrated above, we introduced three different types of Pooling based operations, with each providing a different level of spatial pooling strengths. Using these newly generated features, we can determine which of the representations above can yield a much more informative signal to the models through using the feature importance metrics, which we will define in the next section.

3.4 Feature Importance Analysis

Prior to building models, one of the first steps we have taken towards looking at the dataset carefully was through evaluating the feature importance or feature selection. We performed this because we saw that not all of the data was considered to be very useful or there may be issues due to collinearity between features, and thus, we needed some methodical way to formulate an understanding for the relative ranking for each of the features available in the dataset. Furthermore, as we introduce new features, such as the one formulated in the previous section, we also evaluated the effectiveness of these newly generated features to test their relevance against existing feature sets.

3.4.1 Correlation Analysis. Prior to evaluating the features through various feature importance evaluation metrics, we will first evaluate the correlations between each of the variables to identify potential collinearities between any of the given features. This in principle will provide us with an understanding of the features and if there are independence between the features. We generate the corresponding plot as shown in Figure 18, with the features in the order of "i_period", "i_epoch", "N_Transit", "i_depth", "i_dur", "i_b", "i_ror", "i_dor", "Expected_MES".

Figure 18: Pearson Correlation of Features



1	0.67	-0.26	0.038	0.066	-0.0026	0.071	0.68	0.012
0.67	1	-0.17	0.026	0.042	-0.002	0.046	0.46	0.007
-0.26	-0.17	1	-0.022	-0.032	-0.0006	-0.043	-0.22	0.025
0.038	0.026	-0.022	1	0.0036	-0.0099	0.83	0.045	0.2
0.066	0.042	-0.032	0.0036	1	-0.055	0.0085	-0.24	0.0027
-0.0026	-0.002	-0.0006	-0.0099	-0.055	1	0.025	0.0015	-0.15
0.071	0.046	-0.043	0.83	0.0085	0.025	1	0.07	0.28
0.68	0.46	-0.22	0.045	-0.24	0.0015	0.07	1	0.00068
0.012	0.007	0.025	0.2	0.0027	-0.15	0.28	0.00068	1

Based on the Pearson Correlation Matrix we have generated, some notable correlations are between the input period and input epoch, input period and i_dor, and the strongest one of them is between input depth and i_ror.

3.4.2 Methods of Analysis. In our analysis, we implemented various algorithms which evaluates and ranks each feature based on some model based heuristics. In particular, we implemented Random Forest, AdaBoost, Extra Trees, Gradient Boosting, Lasso Cross Validation, Random Trees Embedding, and Chi Squared Feature Selection, which provided us with a ranked listing of all features in the dataset. Each algorithms are able to provide us with a numerical value which indicates the relative feature importance with respect to the target Recovered value. In this section, we briefly describe the methods we have utilized in devising the feature importance metrics for each variable.

The Random Forest Classifier [8] method from the Scikit Learn library [36] provides us with a feature importance ranking for each of the features. For this ranking, we trained the entire dataset with 1000 estimators and with a balanced class weight setting. The rest of the hyperparameters were left with the default values dictated by the Scikit Learn framework.

The AdaBoost Classifier [23] method from the Scikit Learn library [36] also provides us with a feature importance ranking method as well based on its base classifier. We generate 1000 estimators using a learning rate of 0.5, with all remaining parameters set as default values dictated by the Scikit Learn framework.

The Extra Trees Classifier [25] method from the Scikit Learn library [36] also provides us with a feature importance ranking method. We likewise train a model with 1000 estimators using a max feature of 0.90, with all remaining parameters set as the default values set by the Scikit Learn framework.

We train a Gradient Boosting Classifier [24] with 1000 estimators, with a max feature of 0.90 and a max depth of 6, with the rest of the values set as default parameters from the Scikit Learn library [36]. We subsequently extract feature importance values from the model prior to training it on the full dataset.

We also train a Random Trees Embedding [25] [35] with 1000 estimators on the entire dataset, with the rest of the hyperparameters set as default by the Scikit Learn library [36]. We subsequently utilized the feature importance values from the model to obtain the feature importance ranks.

In addition to the Tree and Boosting based models, we have also utilized other analytical methods, such as Chi Squared analysis which selects the best K by training a classifier which takes the square of the feature as the input and the target as the Recovered signal. Based on this model, we can derive a score for each of the features in the dataset to determine which features are the best based on a ranking based heuristic.

Finally, the last method we used for feature selection was based on the Lasso model, where we trained a Lasso Cross Validation model with the hyperparameters: n_alphas of 200, alpha value of 0.01, max iteration of 2000, using a cross-validation of k = 5. Given the support value for each feature selection, we obtain a binary value of 0 or 1, which indicates whether to not keep or keep the feature respectively.

3.4.3 Feature Rank Aggregation. Despite having many different feature importance and feature selection rankings, each with its own metrics and values which are defined within the ranges of the ranks, the use of having too many ranks may not be a feasible and pragmatic way to approach this as some of the features may be ranked in a relatively different manner. For example, we may have one feature which is ranked the highest in one algorithm, while in the other algorithm it may be slightly lower than the first algorithm we used. Although it is interesting to evaluate these differences from an analytical standpoint, we would have to make pragmatic decisions to understand what the best feature is overall and would need some method to derive a consensus amongst all of the different feature importance metrics. Hence, along with each of the individual rankings, we will use this method to derive a single list of rank-based list of the features.

Therefore, to solve this problem, we can frame this as a rank aggregation problem, where we have multiple set of ranks of some number of elements - each with a particular quantity indicating its importance or significance. To mitigate this problem, we can borrow a heuristic from Information Retrieval ranking methods, such as the method proposed by Dwork et al. [20] to essentially combine the rankings together as a single list using a method known as local Kemenization.

3.4.4 Feature Evaluation. Given the various feature importance metrics and evaluation methods we have presented above, we will utilize these methods to perform various feature importance analysis to see which features will most likely provide a higher amount of predictive power for the machine learning models. This preliminary analysis using the entire dataset as a means to extract the most relevant features allows us to evaluate some of the potential features that may be deemed useful versus some feature that we can completely remove when performing our modeling process. In particular, we believe that such analysis will help us to inform if we can make use of any useful features or perhaps derive new features from weaker features that were rated inadequate by the feature importance algorithms.

To evaluate the feature importance of the dataset, we have also decided to include several other experimental features to see whether or not they would be deemed useful by the models. As an experimental feature, we will include the analysis of a modified Sky Group feature, presented in the previous section of the paper. This analysis will help us gauge the relevancy of the features to the original dataset to see how well these models will likely to perform when used in training a model.

3.4.5 Result of Analysis. Based on the results demonstrated in Table 1, we can notice several different rankings show several different relative measures for each of the defined features. However, as mentioned in the original PLTI Documentation from NASA, most of the feature importance and selection algorithms mostly agree that Expected_MES is a very important feature that can contribute greatly to the prediction of a TCE. However, we note that AdaBoost ranks the period of the planets as the highest, Random Trees ranks the N_Transits as high, and the Chi Square analysis show that the input duration is significantly important.

Performing the feature importance and selection analysis over the Kepler CCD channel feature extraction and pooling methodologies, we fed the new augmented dataset into our feature importance evaluation pipeline to evaluate the importance using various metrics. Based on this analysis shown in Table 2 we found that, in general, most feature importance algorithms did not really consider this to be a very important feature (as compared to the Expected MES). However, we have found that panel orientation was less informative than the Sky Group feature, however, both the center pooling and corner pooling was greater in importance than the Sky Group feature - especially the corner pooling variable was better. This confirms our intuition on the basis of how we previously justified the corner feature being a close-to-upper-bound feature based on the overall reduction ratio being slightly greater than the center pool, but less than the original range of the feature size.

Based on these new key insights, we build a model which would use the corner pooling feature and the orientation type feature. We decided to use these two because we also found that pooling by either center or corner will cause us to lose rotational information, since we group the corners irrespective of the original orientation. Thus requiring us to explicitly encode this knowledge as an additional feature. In addition to this, we also consider trying to remove Sky Group, as it is not as relatively important as some of the other features found in our dataset.

3.5 Models

In this section, we concretely define the problem objective and the methods we aim to employ, both for developing, optimizing, and evaluating the model. We describe in detail the methodology we employ, including hyperparameter selections and the formulation for each model we have built. The aim for this section is to provide a detailed description which aims towards reproducible results based on the methods we have described.

3.5.1 Proposed Model Definition. The fundamental formulation of the model that we will be developing is based on a probabilistic model formulated as, the probability that the given set of features - whether coming from the pipeline modeling for planetary TCE detection or another model considering non-planetary objects, results in the probability that the TPS pipeline has successfully detected the planet or not.

For the first phase of the project, we devise a model that defined as a binary classification model which predicts a target value as a probability function based on the "Recovered" feature of the PLTI Data Release files, which is indicated by a 1 for successful or 0 for unsuccessful recovery of the signal. For the input features, we utilize several key variables, in particular, "Expected_MES", "Sky_Group", "i_period", "i_epoch", "N_Transit", "i_depth", "i_dur", "i_b", "i_ror", and other derived features based on feature engineering methods.

In this evaluation, we build four different model variations: i) based on the given ten baseline features, ii) based on the engineered Sky Group features, iii) based on omitting the Sky Group related features entirely, and iv) modified feature sets based on the feature importance analysis.

3.5.2 Data Preprocessing. Prior to building the classification models presented above, we preprocess the data accordingly through the following standard procedures. For all missing numerical values (notable Expected MES, which has a small number of missing records), we correspondingly impute the values with -999. Then we apply a standard scaler to normalize and center the values for each of the parameters. Furthermore, given that the target value of the Recovered Flag is not necessarily binary (i.e., 0, 1, & 2), we augment values that are labeled as 2 to 1, which is based on the following justification brought by Dr. Ford, stating that: "Those correspond to injected events where the pipeline 'detected' something, but instead of finding a planet with an orbital period close to the period of the injected planet, it thinks it found a planet with an orbital period that is roughly an integer alias of the true orbital period (e.g., a measure an orbital period twice or three times the injected period)." Hence, in this case, since the proportion of values which are marked by 2 is significantly small, we perform the suggested transformation.

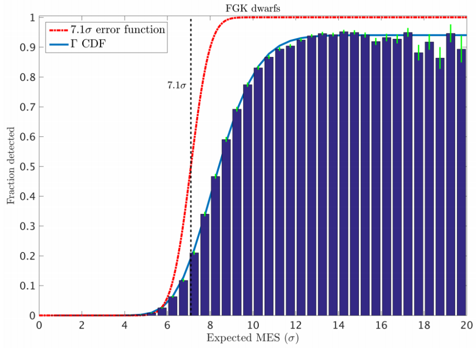
Note that for the baseline model, described in the next section, it does not use any data preprocessing methods due to the nature of how the parameters were fitted to the data and works off of the original values found in the dataset directly (i.e. Expected MES).

3.5.3 Baseline Model. As a preliminary model, we will first implement the baseline model described in Section 5 of the work done by [45]. This model will serve as a baseline benchmark to compare the performance of any subsequent models we develop and will be used as a relative benchmark bracket. The model is formulated by a Gamma Cumulative Distribution Function shown below:

$$p = F(x|a,b,c) = \frac{c}{b^{a}\Gamma(a)} \int_{0}^{x} t^{a-1} e^{-t/b} dt$$

where the parameters for the cumulative distribution function is a = 30.87, b = 0.271, c = 0.940, and x is the input expected MES of the pipeline's input. We can visualize the distribution of the Expected MES by the fraction detected as shown in Figure 19 [45].

Figure 19: Expected MES vs Fraction of Simulated Transits



3.5.4 Predictive Models. In this subsection, we describe in brief some of the base/straw man models we have trained, and also provide some of the relevant literature pertaining to the models (as some may not be familiar with certain modeling methods). In particular, we emphasize any hyperparameter usage that we have specified when training the model - however much of the detailed implementation can be found in the configuration files provided in our experimental result sets.

We train a Logistic Regression [30] with the default parameters provided by the Scikit Learn library [36]. We also train a train an AdaBoost Model [23] with 1000 estimators with the rest of the model's parameters set as default values. We also train a Categorical Boosting [19] model with a hyperparameter of 1000 estimators with no other alterations to the default parameters. Subsequently, we also train a Decision Tree [9] with default parameters provided by the Scikit Learn library [36]. Furthermore, we also train a Random Forest Classifier [8] with 1000 estimators with the rest of the parameters being set by the default values in Scikit Learn [36]. We also train a Bagging Classifier [7] based on using a K-Neighbor Classifier as the base classifier with max samples of 0.5 and max features of 0.5 and all hyperparameters left as default values. We train a Multi-Layer Perceptron (Artificial Neural Network) [26] with the default parameters provided by Scikit Learn [36], except for the solver using lbfgs and using a learning rate of alpha = 1.0×10^{-5} with an adaptive learning rate. Furthermore, we train a Naive Bayes model [47] based on the Gaussian distribution and the Bernoulli Distribution. Another model we trained was a Random Forest [8] using 1000 estimators and a default parameter provided by the Scikit Learn library [36]. We also train a Stochastic Gradient Descent Classifier model provided by the Scikit Learn library [36], using the log loss as our loss function with an l2 pentalty and a maximum iteration of 100 epochs. Finally one of the last base models that we trained was an XGBoost Classifier model [24], using the following hyperparameter values: eta = 0.01, max depth = 6, min child weight = 1, subsampling = 0.80, with an objective function of a binary with a logitraw value, colsample_bytree = 0.50, scale_pos_weigth = 2, and the base score being set as the mean of the target values ($\hat{y}$).

3.5.5 Ensemble Models. In this work, we build on top of the models that we have constructed by performing a model level fusion of the results. To combine the predictive power of the models, we utilize two different ensemble strategies to attempt to obtain the highest results. For this, we pick the top three and top five models which we have deemed the best from our baseline models, as described above, and use them to generate a combined model.

The first method we employ is the Voting Ensemble, where we take the top results from each of the model and derive a majority vote from the classifiers. This allows us to obtain the best set of results based on the predictions generated by each of the models.

The second strategy we apply is the stacking ensemble method. In this model, we take the top k models and use the results from the model to train a meta-classifier on top - in this case we utilize a Logistic Regression Model to train the meta-classifier for its simplicity.

3.6 Evaluation

In this section, we define our evaluation methodology for empirically testing the effectiveness of how well our models perform. First, we describe some of the evaluation metrics we will be utilizing through explicit definitions of each of the metrics. We note these definitions explicitly in our paper, as there are some discrepancies in the semantics across different literature in the astronomical sciences, and hence we clarify and connect equivalent definitions of the metrics in terms of the metrics utilized in statistical machine learning. Then, we will devise our experimental evaluation methodology in how we employ k-fold cross-validation as a primary method for testing the performance of our models.

3.6.1 Evaluation Metrics. First, we define the metrics we used to evaluate our models. We use various different metrics as each of them have their respective merits and disadvantages when it comes to the interpretation assessment of a model's performance.

Root Mean Squared Error (RMSE)

The Root Mean Squared Error (RMSE) is a metric used to measure the value difference between the actual sample values and the predicted value by a model. It is a measure of the variance of the residuals are between the two samples, indicating how well our predictions are close to the actual values. It is defined by the square root of the mean squared error:

$$\sqrt{MSE(\hat{y})} = \sqrt{\mathbb{E}[(\hat{y} - y)^{2}]}$$

where y is our actual value and $\hat{y}$ is the predicted value by our model.

Logarithmic Loss

The Logarithmic Loss function is a metric (in conjunction with the

Cross-Entropy function from Information Theory) which quantifies the performance of a given probabilistic model's prediction. In machine learning, we often want to minimize this function - hence, a perfect model would have a log loss of 0, where as a model that diverges from the actual prediction would deviate away would have values above 0.

For a binary classification based log loss, we define the formula as follows:

$$-(ylog(p) + (1 - y)log(1 - p))$$

where y is a binary value (0 or 1) of the actual class label of a given sample and p is the model's predicted probability.

Accuracy
Accuracy, is a measure of how much statistical bias is present, or in other words, what the degree of closeness our predictions are to the actual target value. In the case of the classification based model (which we use for this metric), it evaluates to see how much of the predicted labels are matched exactly by the ground truth of the dataset. It is defined by the following formula:

$$\frac{TP + TN}{TP + TN + FP + FN}$$

where TP is the True Positive, TN is the True Negative, FP is the False Positive, and FN is the False Negative values.

Precision
Precision is the measure of the positive predicted values, or in other words, the ratio between the number of true positives to the sum of the number of true positives and the number of false positives. This can be concisely expressed as follows:

$$\frac{TP}{TP + FP}$$

Recall
Recall is the measure of the positive rate or sensitivity, or in other words, the ratio between the number of true positives to the sum of the number of true positives and the number of false negatives. This can be concisely expressed as follows:

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

F-Measure (F1)
The F1-Measure is the harmonic mean (approximately the geometric mean divided by the arithmetic mean) of the precision and recall metric, which is defined as follows:

$$2 \times \frac{PR \times RC}{PR + RC}$$

where PR denotes the precision and RC denotes the recall. This metric is often utilized as an aggregate measure between the two metrics defined above.

Area Under the Curve (AUC)
The Area Under the Curve quantifies the relationship between the False Positive Rate and the True Positive Rate of the model as a plot, respectively on the x and y axis. The score is computed by computing the underlying area generated by the curve for each of the values under the False Positive Rate to the True Positive Rate. The score ranges between 0 and 1, where 1 would indicate a perfect model and 0.5 would indicate randomly guessing, while 0 would indicate a poorly performing model. For this metric, we would like to maximize it as close to 1 as possible. In our work, we use this to quantify the overall performance of the model's FPR and TPR.

Kappa Statistic
The Kappa Statistic (or canonically known as Fleiss' Kappa) is a statistical measure for assessing the *reliability of agreement* between a number of multiple parties for either rating or classification. It is often a generalization of the Cohen's Kappa value which computes only for two raters, however can be applied in the case of two parties as well in this case. It is defined by the following formula:

$$\kappa = \frac{\bar{P} - \bar{P_\epsilon}}{1 - \bar{P_\epsilon}}$$

where $\bar{P} - \bar{P_\epsilon}$ is the real degree of agreement defined above chance alone and $1 - \bar{P_\epsilon}$ is the degree of agreement that is possible above chance. This measure ranges between 0 and 1 inclusive. Hence, if the measure is closer to $\kappa = 1$, then this indicates that there is complete agreement, while 0 indicates no agreement, though other than by chance.

3.6.2 Cross Validation. To ensure that our models are robust enough to generalize against any unseen data, we utilize the K-Fold Cross Validation method, or sometimes known as the out-of-sampling test. As opposed to the traditional methods of using a simple training and test split, this method of validation allows us to evaluate the performance of the model against all subsample partitions of the dataset.

In the case of K-Fold Cross Validation, we initially randomly shuffle the ordering of the sample data and partition it into K subsamples, where K is initially selected. Then, we would take the first K subset and consider that as our validation piece, while the remaining non-selected subset would be used to train our model. We repeat each of this process for all K subset of the data and correspondingly derive K different models.

We evaluated models using the metrics discussed above in each fold of 10-Fold Cross Validation in order to evaluate the robustness of the models. In this process, we partitioned the training data into 10 groups and correspondingly developed 10 separate models from the subset permutations. For each model, we evaluated our model against the previously listed metrics and tracked them for each fold individually. Given that we have the individual scores from the corresponding metrics, we also take the average to compute a single metric representing our model. We also persist each of the AUC curves per fold and plot out each of the model's performance and derive an average AUC plot as well.

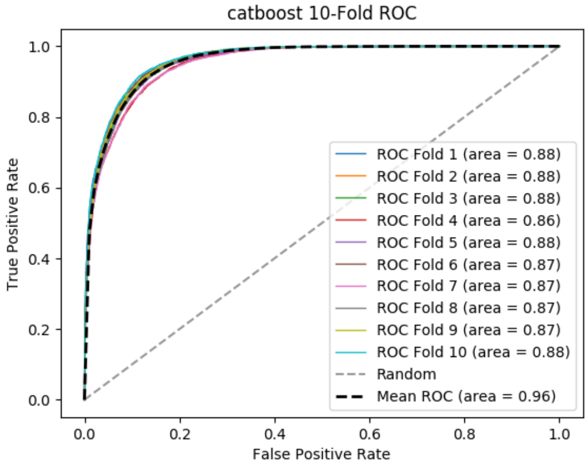
3.7 Results

In this section, we present our results from the models that we have trained based on the proposed experimental designs in the previous section. We empirically break down the results from our trained models and report our metrics, which can be found in the Appendices Section A of this documentation. We correspondingly provide the Area Under the Curve (AUC) Plot for each of our best performing model. Furthermore, we provide some comments and observations on each of the model results and provide some insights as to how they have performed, and note any interesting results that may be worth evaluating in future analysis.

3.7.1 NASA Baseline Model. First, we present the baseline model that was originally proposed by Van et al. [45], we see that it performs with an accuracy of 0.69 and and AUC of 0.50, which only amounts to randomly guessing. In particular, the F1-measure indicates that the model has a hard time to discriminate the signal effectively.

3.7.2 Baseline Models. Based on our results, shown in Table 3, we have found that the Categorical Boosting model performed the best. However, we notice that other models are also tend to have very similar metrics close to the results of the Categorical Boosting model. As shown in Figure 20, we see that the variance between each of the model is very small with regards to the Area Under the Curve. However, comparing some of the other models from Table 3, we see that the Naive Bayes Gaussian model performed significantly worse than Van et al.'s baseline model, however, has the highest recall with a significantly low recall.

Figure 20: Baseline Categorical Boosting AUC



3.7.3 Sky Group Based Feature Model. Based on our results, as shown in Table 4 and our AUC plot in Figure 21, we see that the Sky Group based feature overall performed slightly worse than that of the previous baseline model, although much of the scores are relatively the same. However, one interesting observation to note from our results are that the Recall and the AUC has slightly gone up as a result of this new feature being used (although the improvement difference is quite marginal).

3.7.4 Sky Group Features Omitted Models. Based on our results, as shown in Table 5 and from our AUC plot in Figure 22, we see that dropping Sky Group has significantly improved the results of our model. However, we have found this to be not surprising, as the feature importance metrics we have derived indicated to us that the informative power of the Sky Group feature itself was not useful at all. In short, we see that the Categorical Boosting model from dropping the Sky Group variable performed the best out of all the models we have trained thus far.

Figure 21: Sky Group Feature Categorical Boosting AUC

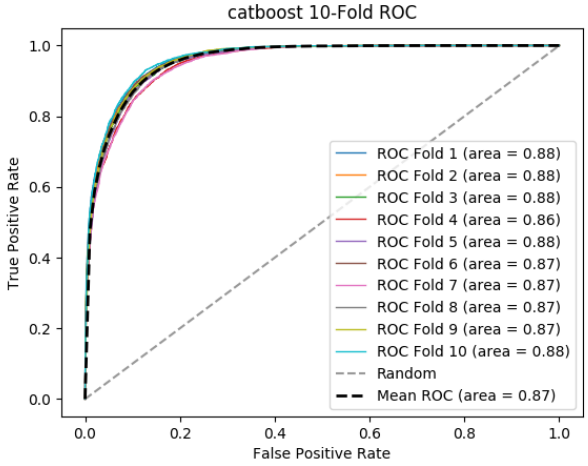
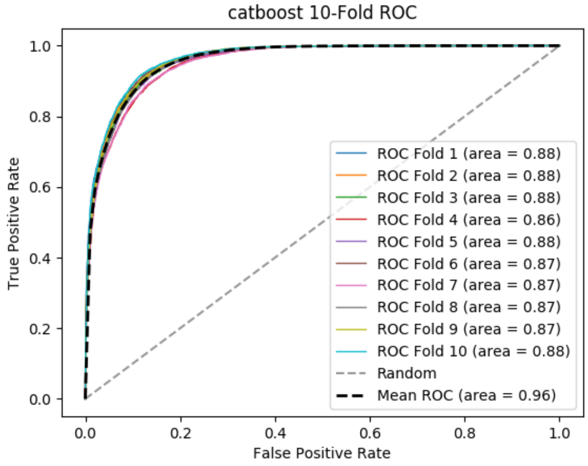


Figure 22: Sky Group Omitted Categorical Boosting AUC



3.7.5 Ensemble Models. Based on the results found on the table, we find that the results from the Stack Ensemble performed very similarly to the baseline Categorical Boost model in Table 6. We see that this model does not, however, exceed the performance of the dropped Sky Group model, and therefore we find that the bottleneck of this ensemble model is attributed to the Categorical Boosting model itself.

4 TCES DETECTION MODEL

4.1 Introduction

In the previous portion of the paper, our work has so far discussed the process of identifying Threshold Crossing Events (TCEs) based on the raw pixel level-based data of the stellar light curve data obtained by Kepler's telemetry systems, as outlined by Jenkins et al. [29]. Through this pipeline, we are able to identify potential objects which are classified by Kepler Objects of Interests (KOIs). Given the list of KOIs identified by the pipeline, one of the other problems of interests within many researchers is to further identify between the Planet Candidates (PC) and the various types of False Positives that can potentially be misidentified by the pipeline, as described by Coughlin et al. [18].

Within the class of False Positive flags, there are four different types of signals that are possible which include: Not Transit-Like (NTL), Stellar Eclipse (SS for Significant Secondary), Centroid Offset (CO), and Ephemeris Match (EM). Based on these attributes, we can specifically look at the efficiency of the Robovetter pipeline with respect to the simulated input values. Our objective in developing models to characterize each of these false positive types will entail an in-depth analysis of uncovering the attributes for each of these false positives and gain insight on the key parameters for each of these simulations.

In particular, Coughlin et al. is interested in quantifying two key metrics which they hope will help to characterize the identification of the Planet Candidates and the False Positives by the Kepler Robvetter Pipeline, which are *completeness* and *effectiveness*. According to Thompson et al. [43], they define completeness as *"the fraction of transiting planets detected by the pipeline that are classified as planet candidates by the Robovetter"*, while they define effectiveness as *"the fraction of false positives detected by the pipeline that are classified as false positives by the Robovetter"*. To understand this question at a fundamental level, the works such as those by Christiansen et al. [12] focused on various injection transformations to the original light curve data, which can simulate some of these false positive phenomenon.

In this portion of the paper, we will discuss the work we have performed with regards to the second phase objective of our project. The secondary objective of this problem is to utilize simulated data from Kepler (including those that inject non-planetary data points) to develop a predictive model to also characterize the efficiency of the model that determines whether a Threshold Crossing Event is considered as a planet or a non-planets and helping to identify factors which contribute to a specific type of the False Positives. Similar to the first problem, we define another set of probabilistic models which aims to understand the efficiency of the model - however, this time evaluating the efficiency of detecting non-planetary objects and how the pipeline performs on the simulated injection data. In particular, the focus of the models that we produce are not only based on the identification of the planet candidates, but also seeing how the pipeline can potentially pick up on the different types of False Positives that are artificially generated from the original dataset.

However, different from our previous efforts to build a predictive model which can help to identify these false positives as accurately as possible as a strongly optimized problem. The second phase focuses more so on a analytical and interpretative mode of analysis, focusing more on the qualitative aspects of the model and providing a much more in-depth analysis and interpretation of the models we have developed. The analysis and interpretation will be further provided in detailed within the next portion of the paper, where we combine the use of black-box machine learning analysis tools to help pick apart feature and sample based analysis of the models to draw conclusions and recommendations for future work. In this portion of the paper, we address the prior step of building the model which will be used in the subsequent analysis phase - similar to the structural approach we've introduced in the previous portion of the paper.

We structure the second phase portion of the paper as follows: First, the current section introduces the first problem we are addressing through our modeling process. Then the second section evaluates the features found in the TCEs dataset used to model the efficiency by describing each of the distribution characteristics and providing contextual information for each of the features within the dataset. In the next section, we evaluate the Feature Importance of each of the described feature based on the evaluation of various key feature importance and selection metrics and methods of aggregation of feature importance ranks. The next section of the paper addresses and defines each of the algorithms we have utilized to build our probabilistic models, and describe briefly some of them with further references one can consult to for better understanding the modeling process of each algorithm. Within the modeling section, we also describe some of the key parameter selections for each models we have developed. Then we present the results obtained from the our empirical evaluations, along with various remarks and comments on the performance based on any surprising or sensible characteristics from the results shown. As mentioned previously, we reserve the complete Analysis section of the paper in the fifth portion of this paper which provides an in-depth analysis of the top performing model and contextualizing the results within some of the results we have obtained from our model.

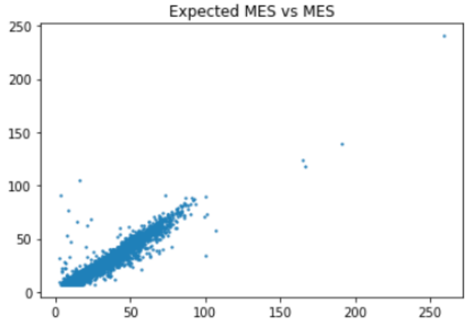
4.2 Dataset

The dataset used to model the first phase probabilities was based on the Data Release 25 file set, "kplr_dr25_inj1_tces.txt". The dataset was based off of the pixel-level data injection experiments conducted by Christiansen et al. [12], where they emulated signatures of transiting planets through artificially augmenting the signal patterns. The pipeline was based on the SOC pipeline version 9.3 and DR25 Robovetter model processing uniformly across all of the light curves, based on the work of Thompson et al. [44], from the Q1-Q17 DR25 sets based on the stellar properties of the work in Mathur et al. [34].

In addition to the provided TCEs data set from the data catalog, there was another data set file, "kplr_dr25_inj1_robovetter_input.txt", which included key stellar properties and other parameters from the experimental data processing pipeline. However, we did not make use of this dataset, due to the limited timeframe we had in performing further modeling efforts with a much larger dataset and also due to the increasing complexity of understanding a large interaction of features was going to be a non-trivial task when it comes to the analysis portion of the paper.

In the TCEs dataset, which contains 45,377 data samples, the features that we had available to use were: "TCE_ID", "KIC", "Disp", "Score", "NTL", "SS", "CO", "EM", "period", "epoch", "Expected_MES", "MES", "NTran", "depth", "duration", "Rp", "Rs", "Ts", "logg", "a", "Rp/Rs", "a/Rs", "impact", "SNR_DV", "Sp", and "Fit_Prov". However, based on the data specifications listed in Coughlin et al. [18], we do not make use of the features such as "TCE_ID", "KIC", "Score", "MES", "Fit_Prov" were most likely candidates for removal due to being outputs from the pipeline. Features such as the Expected MES and MES were first obvious choices for removal, as they both include some form of the MES and have some linear relationship between each other, as shown in Figure . However, we opted to utilize the Expected MES as that is what we have utilized in the previous section.

Figure 23: Scatter Plot of Expected MES vs MES



Upon evaluating the dataset, we found there are couple of areas that we should be mindful and careful of when developing our models. First, we performed a co-occurrence analysis between the number of FP types (only of two types) and found that there are some categories of FP that occur together simultaneously. This observation has raised some flags with regards to the independence assumption of the injections - which informs how we may have to formulate our modeling strategy. As shown in the table below, some FPs may have a potential conflict with other FP types and must be very careful when formulating our modeling structure.

_	NTL	CO	SS	EM
NTL	5303	196	0	2
CO	196	1021	13	1
SS	0	13	576	0
EM	2	1	0	23

4.3 Feature Importance

Prior to developing our base models, it is important to identify key features that can be useful and likewise features that should not be used to model - in particular variables that may be linked to the output of the system. Since we did not obtain a clear indication of which variables we are permitted to utilize, this analysis will help us to rule out any key features which are used as indicators for the labels.

4.3.1 Evaluation Methods. For this, we utilized the same pipeline as described in the previous portion of this report (*see Section 3.4*) Each algorithms are able to provide us with a numerical value which indicates the relative feature importance with respect to the target "Disp" value.

In addition to the individual ranks of the various feature importance algorithms, we found that we would need to find a single method which would correspondingly rank the features based on a aggregate score of all the possible feature importance scores. This analysis can help us to make an uninformed decision, in addition to the independent analysis of the feature importance, to allow us to set a straightforward method for feature selection. For this, we utilize the method proposed by Dwork et al. [20] which proposed a method for rank aggregation for web search results. However, in our context, we utilize the ranks provided by each of the feature importance weighted by various different metrics. The final output of this algorithm will yield a ranked list of the different features, for then which we can run a threshold to select the top-K features for use in modeling.

4.3.2 Feature Importance Results. Based on the feature importance techniques we have applied based on our proposed approaches, we first implement the analysis on the entire dataset, using the Disp as the primary target as shown in Table 7. However, not surprisingly, we noticed that there are key variables which dominate the majority of the importance metrics which nearly suppresses the importance of other variables. In particular, Score, NT, SS, CO, and EM were some of the most dominant features that were highly important according to the analysis.

However, we believe that these features obtained a higher ranking since these values are directly tied to the output. Therefore, this suggests that we should remove these features from our analysis and perform the feature importance analysis again with these features removed, as shown in Table 8. In addition to removing the listed features above, we would also consider the removal of MES, as it conflicts with the Expected MES parameter and that in our first phase experiments, we have used only the Expected MES feature.

Finally, using the proposed methods derived from Dwork et al. [20], we derive the complete rank of features that our models utilize as listed in the rank below:

TECs Feature Ranking

(1) SNR DV
(2) Expected MES
(3) Rp
(4) logg
(5) Period
(6) Duration
(7) Epoch
(8) a/Rs
(9) a
(10) Rp/Rs
(11) Rs
(12) Depth
(13) Fit Prov
(14) NTran
(15) Ts
(16) impact
(17) Sp

We see that from our overall analysis, the features that we should keep are those within the top 9 as there are some redundant features within the variation of Rs, Rp, and a. Also, compared to the previous feature importance analysis that we have performed, we also note that features such as impact and depth are fairly weak in comparison and also is strongly demonstrated within here.

4.4 Models

In this section, we concretely define the problem objective and the methods we aim to employ, both for developing, optimizing, and evaluating the model. We describe in detail the methodology we employ, including hyperparameter selections and the formulation for each model we have built. The aim for this section is to provide a detailed description which aims towards reproducible results based on the methods we have described. We utilize the same modeling methodology and evaluation described in previous sections (see Section 3.5 and 3.6).

4.4.1 Proposed Model Definition. The fundamental formulation of the model that we will be developing is based on a probabilistic model formulated as, the probability that the given set of features - whether coming from the pipeline modeling for pipeline has successfully detected a Planet Candidate or a False Positive.

For the second phase of the project, we devise a model that defined as a binary classification model which predicts a target value as a probability function based on the "Disp" feature of the TCEs dataset files, which is indicated by a 1 for successful or 0 for unsuccessful recovery of the signal. For the input features, we utilize several key variables, in particular, SNR DV, Expected MES, Rp, logg, Period, Duration, Epoch, a/Rs, a, Rp/Rs, Rs, Depth, Fit Prov, NTran, Ts, impact, Sp.

In this evaluation, we build four different model variations: i) based on the selected features we have proposed initially and ii) based on simple log-based transformations. Our focus here is based on the principle of building a sufficient enough model such that we can have some baseline to discuss further analysis in later stages of the modeling process.

4.4.2 Data Preprocessing. Prior to building the classification models presented above, we preprocess the data accordingly through the following standard procedures. There were no known missing data from our dataset, hence missingness imputation was not required. Then we apply a standard scaler to normalize and center the values for each of the parameters.

Another feature preprocessing method we apply over the data is performing log transformations on certain features. Based on our exploratory data analysis performed on the dataset from the previous weeks, we have evaluated the possibility of transforming these features through simple transformation which can adjust the scale of the data. By scaling the data through other heuristics than the default scaler methods used in Scikit Learn, we hypothesize that separability of the data distribution will make it easier for the model to classify between certain classes. Although it has been shown that the majority of transformations operations are quite trivial with respect to the performance, we believe that even a small increase in performance in this problem can be significant.

To systematically evaluate which transformations are appropriate, we will only focus on the logarithmic transformation. Based on our Exploratory Analysis, we find that there are some inconsistencies within the scaling of the data, which can make it very difficult for the model to perform well. Hence, the choice of logarithmic data allows us to normalize data whose distribution are, for instance, zipf-like or have a tendency for outliers. To perform this evaluation, we build a simple Logistic Regression generated from the previous week to iteratively test individual log transformations with the same pipeline used in the previous week to evaluate our baseline models. As per our pipeline, we evaluate the dataset over a 10-Fold Cross Validation method to evaluate the performance and closely look at the accuracy metrics and compare it against our original baseline metrics from the previous week. We repeat this process for each of the different features individually - note that we only perform these transformation and evaluations based on the assumption that these feature are independent (for the sake of simplicity to disregard interactions of transformations). Ideally, we would have performed a much more comprehensive analysis, such as ANOVA or related evaluation methods, but due to time restrictions, we only consider this analysis as our main method of evaluation. From this analysis, we determine what the best transformations are and subsequently combine those together and reevaluate our models for the entire set.

Based on the first experimental analysis of our baseline logistic regression model, evaluated over a 10-fold cross validation evaluation procedure, we found the following results based on the accuracy metrics of our model, as shown below. We rank them based on the rounded scores that they have obtained from the logistic regression models (note that some are tied together for the same place). We do not show any features that are lower than the original baseline without any transformation due to space constraints.

(1) Ts (0.859)
(2) SNR_DV (0.8580)
(3) Depth (0.8576)
(4) NTran, Rs (0.8575)
(5) a/Rs, impact (0.8574)
(6) Baseline (0.8573)

Given the above empirical accuracy results, we see that the transformation of Ts and SNR_DV appears to have the highest impact based on the transformation as it has they each have made approximately 0.001 to 0.002 increments in performance. Although this appears to be a trivial amount, we have decided to execute both of our data transformations on two of the features and run them across all of our models.

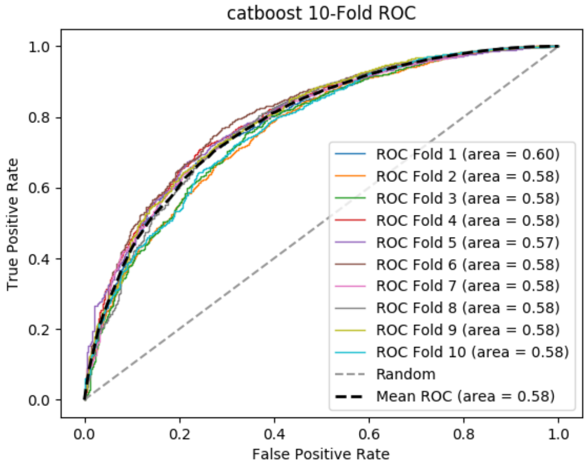
4.5 Results

In this section, we present our results from the models that we have trained based on the proposed experimental designs in the previous section. We empirically break down the results from our trained models and report our metrics, which can be found in the Appendices Section A of this documentation. We correspondingly provide the Area Under the Curve (AUC) Plot for each of our best performing model. Furthermore, we provide some comments and observations on each of the model results and provide some insights as to how they have performed, and note any interesting results that may be worth evaluating in future analysis.

4.5.1 Baseline Models. Based on the results we have obtained, as shown in Table 9, we see that the performance for detecting Planet Candidates and False Positives - as shown by the general accuracy score and how even the AUC of the model is approximately around

0.5. Similar to the first set of models we have produced in the first phase of the project, we find that the same boosting algorithms such as Categorical Boosting and Random Forest tends to perform better in general. However, we still see that all of the models are struggling to identify and characterize between the two different classes, and therefore, would require much more significant work in terms of data preprocessing, feature engineering and feature selection to improve the performance of the models.

Figure 24: Baseline Categorical Boosting AUC



4.6 Log Transformed Features

Based on the transformation we have performed, as shown in Table 10, we have shown that the results have slightly improved. In particular, our best performing model, Categorical Boosting, has shown the greatest improvement - even from the top performing model in the previous baseline and in the feature based accuracy metric in Figure 2. This provides enough evidence that our transformation has made a significant impact in performance.

5 MODEL ANALYSIS AND INTERPRETATIONS

In the following section, we perform a post-modeling analysis of the features based on the results we have obtained from the model. This post analysis specifically looks at how each of the parameters in the model has helped to contribute towards building a predictive model. In lieu of building interpretable models, we perform a *SHapley Additive exPlanations* (SHAP) Analysis based on the best model from the results we have obtained.

5.1 SHAP Analysis

The SHAP values technique was proposed in recent papers by Lundberg et al. [32] [33], which allows us to explain the result of machine learning models by visualizing how each of attributes contributed to the given outcome. It is based on Shapley values, which is a methodology applied in game theory to evaluate how much each player has contributed in a collaborative game towards its success. In this circumstance, we define the success as whether the feature contribute positively or negatively towards recovering of the signal within the pipeline. This framework enables us to examine how the increments and decrements of a particular features impact the result for any machine learning models.

Figure 25: Log Transformed Categorical Boosting AUC

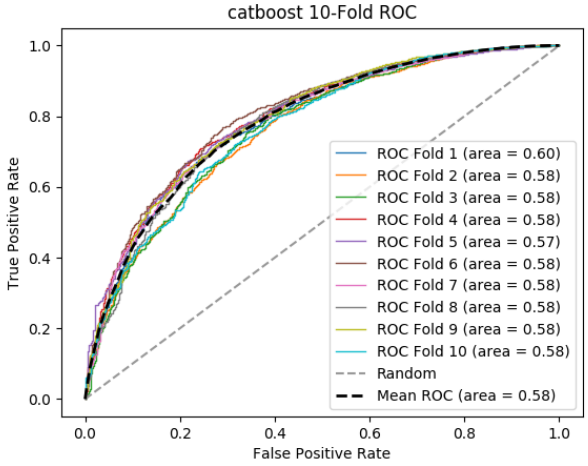
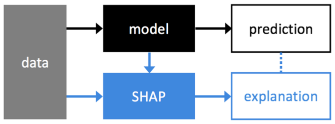


Figure 26: SHAP model visualization



The Figure 26 above demonstrates how the SHAP analysis can be retrofitted into any modeling pipeline to easily integrate interpretability within any context of the modeling process. Thus, this offers explainability to any model that we build through this method.

Using such framework, we identified the importance of each attribute in prediction of the outcome of our best model, categorical boosting. Previous work has already shown that the expected MES has, by far, the largest impact on the result. Therefore, we attempt to get the importance of features other than the Expected MES in this project to see how other features and variables can play a role in the models.

This Figure 27 shows us that the N_transit, i_ror, i_dur has relatively large influence on the outcome of the model. To further analyze the impact of the increments and the decrements of features, we also generated another summary plot.

Figure 27: Mean SHAP Magnitude Summary

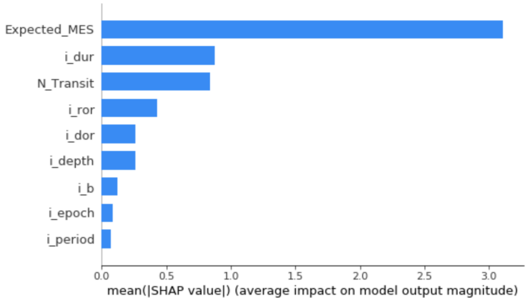
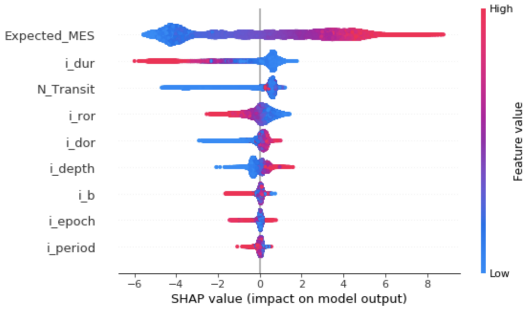
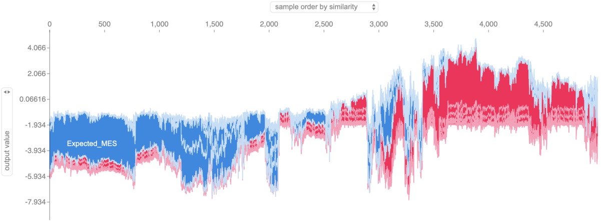


Figure 28: Signed summary plot



In the Figure 28 above, X axis represents the SHAP value, and the color represents the sign of impacts. Red represents the positive impact, and blue represents the negative. Based on this graph, we can observe that the N_transit and i_ror have positive impacts. On the other hand, i_dur and i_b have negative impacts.

Figure 29: Stacked SHAP Values



The Figure 29 above shows explanations for the entire dataset of the given model. Each layer represents how much one feature contributes to push the model outcome from the base value. This graph adds all such layers to demonstrate how every features of input impacted the result.

It is very obvious that the expected MES has contributed the most to the outcome of the result. Because of such a significant impact on the outcome of the result, it becomes difficult to observe the impact of other features.

5.2 Dependencies to SHAP value

Knowing how a single attribute influence the output of the model is very important in order to gain more insights of the problem. SHAP value is made to illustrate a feature's responsibility for a change in the outcome of the model. Therefore, we can use this value to observe the change of the output, recovered flag, as each of features change. We attempt to do this by plotting the SHAP value of feauture and the value of the feature itself for every attributes of this dataset.

Figure 30: dependency plot for N_transit

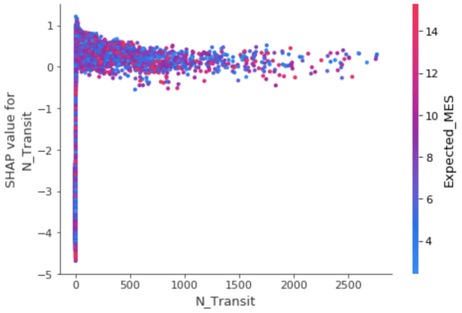


Figure 31: dependency plot for i_ror

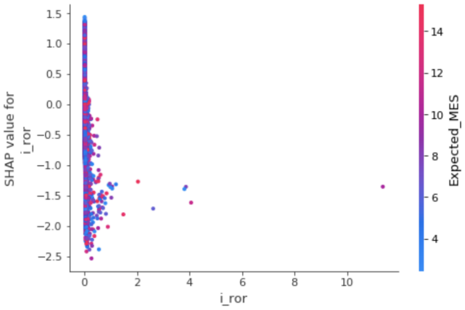


Figure 32: dependency plot for i_dur

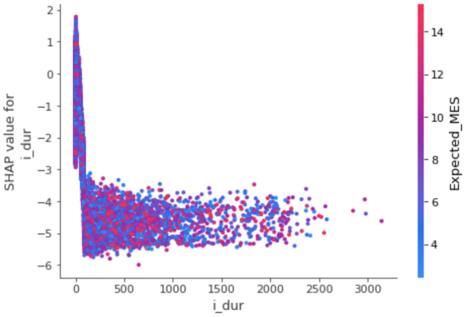


Figure 33: dependency plot for i_b

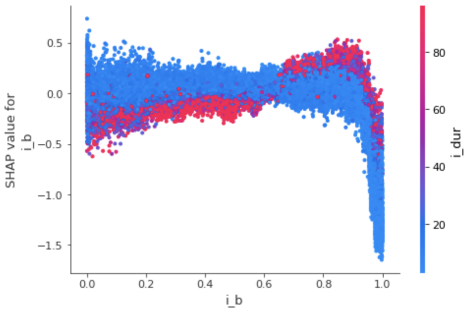


Figure 34: dependency plot for i_depth

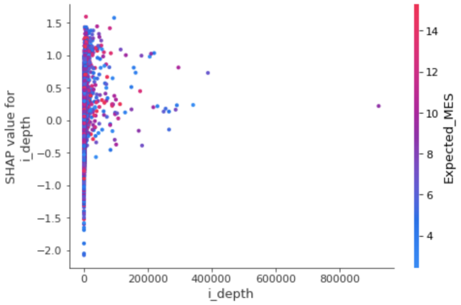


Figure 35: dependency plot for i_dor

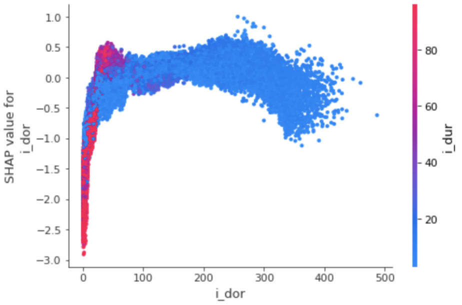
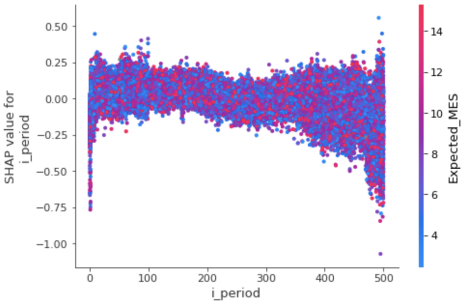


Figure 36: dependency plot for i_period



Some of the graphs above show interesting insights to this problem. As appears in the Figure 30, N_transit seem to impact the outcome of the result in consistent manner. Some features such as i_dur and i_b influence the result negatively. We can observe this characteristic based on the SHAP values of these variables. It is also interesting to note how i_b starts influencing the outcome only when it exceeds 0.6. We can also observe that all the features other than N_transit and i_ror has very little effect on the result. Especially, i_period, i_epoch, and Sky Group have their SHAP values hover around 0 most of the time.

5.3 Additional Analysis

Furthermore, we look at our results and reflect some of the core observations and key questions noted by Christatien et al [12]. We believe that answering some of these fundamental questions below can lead us to important insights which can help NASA inform some key characteristics about their own pipeline system.

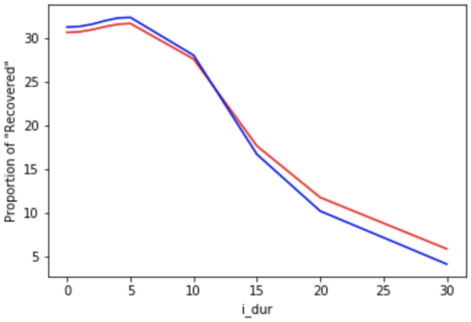
Transit Durations Longer than 15 Hours

According to Jenkins et al. [29] the pipeline performs searches over the range of durations ranging between 1.5 to 15 hours with respect to the transit duration. For this, they note that the durations that are greater than 15 hours are explicitly omitted from the pipeline analysis and were not considered within the original analysis performed by NASA. Hence, in the analysis performed by Christatien et al. [12], they have restricted their analysis to only durations of up to 15 hours and did not consider those beyond 15 hours.

To confirm this claim, we used SHAP analysis and we looked at the impact of transit duration to the outcome of the prediction with respect to its value. Figure 26 shows the transit duration and its SHAP values. The negative SHAP values means that the value of the feature is negatively contributing to the decision of the object being classified as Kepler Object of Interest. Careful evaluation on the graph reveals that the positive SHAP values only occur on the left end of the graph. Therefore, it seems that the cutting off the objects with large transit duration seems like a reasonable idea.

However, SHAP analysis does not clearly show whether or not the cut off for the duration at 15 hours was actually appropriate. Therefore, we decided to see more closely at how the model works when the transit duration exceed 15 hour.

Figure 37: plot for i_dur



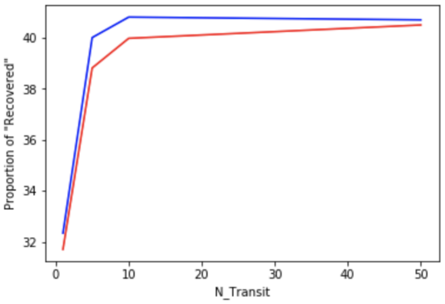
The Figure 37 shows the proportion of the data points marked as recovered. The red line represents the proportion of the data points classified as "Recovered" by our best model, Categorical Boosting. The blue line represents the proportion of actual "Recovered" objects. At transition duration of 15, about 17% of the data points are still classified as "Recovered". However, at duration of 30 hours, less than 5% of the data falls into the category of "Recovered". Our raw data has the data points of transition duration up to 3000 hours. However, more than 95% of the "Recovered" data points turned out to be located under the transition duration of less than 30 hours. From this, we conclude that the decision made by the previous studies were reasonable, but the threshold could be raised to 30 hours instead of 15 hours. There are 684 data points that are classified as "Recovered", compared to the total of 45691 "Recovered" data points. It means that setting a threshold at 30 hours will only cut on 1.49% of the recovered data.

Number of Valid Transits

According to the Christatien et al. [12], the valid transit parameter used in the data analysis pipeline is utilized as a threshold value to determine a cutoff point for which transits are considered valid and invalid. Based on their analysis they show that for different transit values, they yield a different set of detection efficiency ratios. The recommendation from NASA suggests that if we were to consider a completeness period below 300 days, a smaller number should be used as a threshold value for the number of valid transits.

In order to conduct analysis, we took deeper look at the SHAP summary plot in the Figure 28. In SHAP plots, the color of points represents the feature values, red representing high, and blue representing low. The X axis shows the SHAP value of the features. On i_dur, transit duration, we can observe that all data points with high feature value (red) goes onto the negative side on X axis. This matches with what we discussed in the earlier section. However, the number of valid transit, N_Transit, does not have such pattern. On the other hand, the data points with low feature value has negative SHAP value, meaning it negatively contributes to classifying the data points as "Recovered". The Figure 30 also displays such pattern.

Figure 38: plot for N_Transit



The Figure 38 represents the proportion of "Recovered" data points with respect to the value of valid transit. Red line correspond to the result from our best model, Categorical Boosting model. Blue line represents the actual data. From this plot, we can find similar insight to what we found in the SHAP analysis.

The proportion of the data points that are classified as "Recovered" is very low when the feature value is low. Our assumption is that the data points with low number of transit might have been captured by error. Therefore, it makes sense to have the cut-off for the data points with very low number of valid transits.

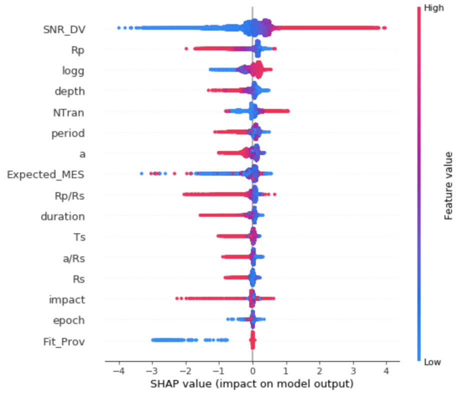
Analysis on Robovetter

We also conducted analysis on our best model that characterizes the efficiency of the Kepler Robovetter pipeline. Our best model, Categorical Boosting, is a machine learning that is based on the gradient boosting. While gradient boosting often provides high accuracy in the result, it is very difficult to observe what the model does to classify the data points. Therefore, it often ends up in "black box" situation. In order to seek the insights for the problem, we utilized SHAP technique. As we explained in earlier section, this analysis visually reveals how each of the features affect the result of prediction. Similarly to the TCL pipeline, the paper by Coughlin et al. [18], also raises couple of insightful questions to be answered.

First question asked is how often does the Robovetter classify an injected planet as a planet candidate. We have approached this problem in very straightforward manner. In order to answer this question, we counted the number of data points that are classified as PC, indicating that it is a planet candidate, and the number of total data points. As a ground truth, we have executed this method on the raw data. According to this, Robovetter classifies 85% of the data points as the planet candidate.

Second question raised was whether or not the Robovetter completeness significantly vary with signal-to-noise ratio, period, or other parameters. The following graph represents how each of the features affected the prediction for our Categorical Boosting model.

Figure 39: SHAP analysis summary plot



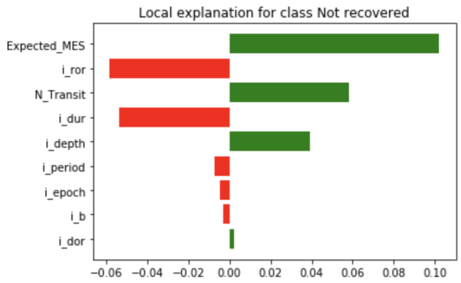
This plot in the Figure 39 reveals some of the interesting facts about how our model comes up with the prediction. The most influential feature SNR_DV has clear relationship to the outcome of the model. In SHAP summary plot, the red points represent high feature value, and the blue represent low. As clearly illustrated, the majority of red points for the SNR_DV reside on the positive side the summary plot and the majority of blue points reside on the negative side the summary plot. This means that this feature will push the model to classify the data points to PC as SNR_DV feature gets high. The second influential feature, Rp, has opposite tendency. We observe that, unlike SNR_DV, this feature has it blue data points on the positive side and the red data points on the negative side. This means that this feature is inversely influencing the prediction of the model. In addition, the span on x-axis of each features represent the magnitude of influence to the result of the outcome. In similar manner, one can observe how each of the attribute contributes to the outcome of the model.

Alternative Approaches

While SHAP analysis is extremely useful tool to conduct analysis on the classification models, it does not reveal how the model comes up with the outcome at each instances of the data. However, it is very important that we are able to explain how our model comes up with the decision at each instance for this model to be used in any astronomy research. Therefore, we suggest the use of two techniques called LIME and Anchor.

First technique, LIME, is a short name for local Interpretable Model-agnostic Explanations. This method of analysis allow us to visually understand what features classification model looks at and how it affected the result at any given data points. By providing the reasoning for the result of classification model, the astronomy researchers can use their domain knowledge to seek if such result is believable. Therefore, this type of analysis can become the connecting bridge between machine learning and the experts.

Figure 40: LIME analysis example



The Figure displays how our best model, Categorical Boosting, came up with the prediction at one randomly selected data point. This method of analysis is expected to be very useful when this model is being used against unexplored dataset.

Similarly, the second technique called Anchor, is also useful to observe how model come up with its prediction at each of the input data. Unique benefit of using this method of analysis is the ability that this method can explain the coverage of features to the model's behavior. For example, if we pick a random instance of the dataset, id = 0, we can examine what features the classification model uses in order to come up with the prediction as well as its corresponding precision and coverage. For the example of data id=0, the combination of the conditions - N_Transit <= 3.04, i_ror > 0.02, and i_dor <= 132.02 is sufficient for our Categorical Boosting Model to classify the event as "Recovered" with the precision of 98% for this particular instance.

The importance of model interpret-ability is raising significantly in recent years. Being able to explain what machine learning model does and understanding how it comes up with the prediction is key for machine learning models to be useful in the scientific research. These analysis will allow machine learning models to be more useful for the research of other fields including astronomy. In order to avoid "Black Box" machine learning models, we highly recommend the use of these analytical techniques for the future works.

6 DISCUSSION

In this section, we take a retrospective perspective on the overall capstone experience by providing meta-level commentary on our project, process, and overall experience. In particular, we reflect on some of the positive, negative, and areas that both we can learn from and develop our own skills, as well as some ways we believe that the whole capstone project experience can be improved for future iterations of students who will be taking this course.

6.1 Data-Driven Research Methodology

In this project, one of the most interesting things we have experienced was the process of applying data sciences towards a particular scientific problem, in a way that addressed a specific need. As opposed to the traditional top-down approach of specifying a hypothesis and iteratively finding methods to test out some claim, the methodology we utilized primarily relied on a bottom-up heuristic to discover new and interesting directions for researchers to tread on with future work. Therefore, this would essentially lead to the feedback loop to confirm some of the claims we have found in a top-down manner.

Our discussions from one class regarding the problem life cycle has made us think about how Data Sciences can be used to work together with a hypothesis driven approach to solving problems. Based on our experiences, we found that such methods can be used as a compass which can help guide scientists and researchers to find patterns and new research directions - essentially providing them with a *way to find the right questions to ask*. Sometimes by working with the data, we see that we can uncover new patterns or new correlations, which can then be confirmed through a statistical process to test those claims. In short, this exercise of using Data Science in a way to drive research has helped us experience and find a workflow that we can apply to solve new problems in the future.

6.2 Project Strategy

The main success behind how we executed our project relied on three key principles we followed, which allowed us to execute successfully. They are: 1) Resource Management, 2) Goal and Objective Setting, and 3) Collaborative Workflows.

6.2.1 Resource Management. Prior to working on the project, one of the first things we have done to ensure that we understood the problem scope was to collect as much information as possible - whether that was relevant literature, code, data, and any possible hints and expertise. To quote Andrew Ng, *"AI is akin to building a rocket ship. You need a huge engine and a lot of fuel. The rocket* *engine is the learning algorithms but the fuel is the huge amounts of data we can feed to these algorithms."* Thus, before we even had our proposal for the project, we invested close to one week on finding every relevant literature through Google Scholar, IEEE, NASA, and ArXiv and other sources for literature review.

Another method we used to collect useful information was finding existing codebases and relevant work that other people have done on Github. We performed specialized searches by using the source-code search feature on Github to find repositories that used the same datasets to narrow down repositories and developers. Then from there, we analyzed watchers and stargazers of those repositories to see if they worked on similar problems and collected a repository set of Github link collections that we have referenced for inspiration on approaches and methods of analysis. We provided a list of the relevant Github sources we identified under the `README.md` file in our Github repository.

The last resource that we have made use of was some of the codebases that we have developed in our past courses or personal projects. By reusing existing methods, tools, and utility scripts, we were able to quickly assemble and piece together our proposed Data-Driven Research pipeline that we have proposed in the second section of this paper. This saved us a significant amount of time, which we allocated towards understanding the problem as one of the major challenges in this project was the lack of domain expertise and knowledge necessary to work and understand the problem enough to start working with the dataset. Therefore, given the existing infrastructure setup, we allocated alot of our time and invested a significant amount of time to read through over 30 papers and journal sources pertaining to the subject and attempt to have a working knowledge of astrophysical data analysis and the Kepler Project. We realized that without the deep domain knowledge, it would have been very difficult to address some of the questions in the process - *working with data out of context would be one of the most dangerous things, since it would lead to erroneous conclusions and fallacies with our logic and reasoning process.*

6.2.2 Goal and Objective Setting. In any project, having a structured plan and execution course is a critical step towards understanding what you need in order to succeed. However, when you don't have much of a background domain knowledge to understand the problem in a complete way, this is particularly challenging in various ways - thus affecting the way we set our goals and objectives on the problem.

We believe that this problem was in particular challenging because of the following two ways:

First, due to the lack of domain knowledge this made it difficult to scope out the landscape of the problem - therefore made it difficult to estimate whether some objective or approach we wanted to try was feasible or something that would take months of dedicated work to complete. Since we did not have background in astrophysics or astronomical data analysis, we felt that we often overestimated what was possible and a constant shift in the scope of the project was required to ensure that we can have some deliverable met every week.

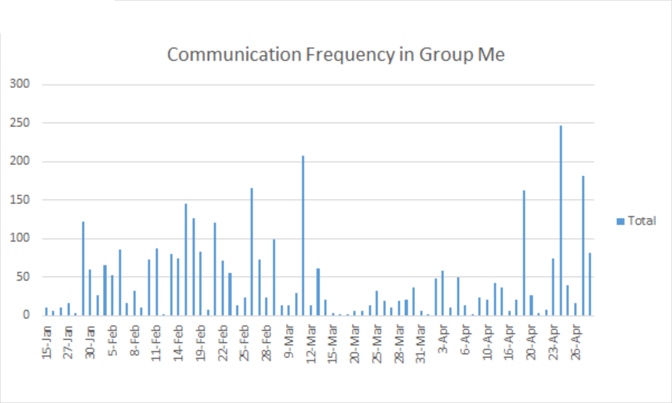
Upon discussion with Dr. Ford on this matter, we believe that the project's problem scope was not considered to be well defined. Although having the initial project presentation was useful to some degree, it still left us with much more questions than answers as to what exactly did we need to perform. Although the scope was left in sort of an open-ended manner, this allowed us some flexibility to explore different questions - however, it was difficult to really find relevant questions to answer due to the nature of the field as we did not have enough experience to know what we are looking for. Hence, we resorted to spending much of the first earlier phases of the project towards reading and researching existing materials before we were able to get some tangible results out.

Given that it was hard to pin down specific goals and objectives, the formulation of a concrete plan was especially difficult. Therefore, for many of our reports we did not include a Gantt Chart as there was no guarantee that we would follow the schedule as planned. However, having a weekly plan of project objectives listed out was sufficient in just knowing where we needed to be in the project, so that we can pace ourselves within a pragmatic manner.

6.2.3 Collaborative Workflows. In this project, team work was a very important process that also contributed greatly with the way we performed and executed our objectives. In particular, there were two dimensions of teamwork that we would like to emphasized that helped us greatly with this project, which are: 1) collaboration tools and 2) cross-team collaboration.

First, to be able to effectively coordinate our efforts, especially in a remote setting, constant communication and updating were necessary. For this, we relied on Group Me to consolidate all of our communication needs and project updates. To evaluate usefulness of this tool, we have performed a meta-analysis of our communication logs that we have obtained from Group Me. To show the efficacy of the communication tools, we plot the temporal distribution of our daily communication frequency, as shown in Figure 41. We note that the more frequently we communicated, the more significant progress we made more efficiently. Therefore, we learned that frequently communicating and updating each other on individual progress was critical in being informed of what the other partner was doing.

Figure 41: GroupMe Communication Frequency



In addition to using GroupMe for communication, we used GitHub to facilitate the exchange and maintenance of codebase resources. This was particularly useful, since the entire workflow pipeline was very complicated and multilayered, and thus a good version control management tool for our work was important. Furthermore, we also made use of other tools such as Google Drive and PSU's Box Drive for large file exchanges and reference literature that we have collected. Through all of these tooling platforms, we were able to efficiently work through our project.

Another dimension which lead to the success to the project was facilitating the communication and collaboration beyond the assigned partners for the project. To better increase the amount of information, data, and project directions that people were pursuing, we created a class-wide GroupMe for student groups that were working on the same project as we did. We firmly believed that the cross-collaborative nature of the GroupMe benefited a lot of the teams in understanding common problems that we all faced with the data, compare benchmark results, and share literature and data resources effectively. This philosophy of cross-team collaboration was based on the practice in the Netflix Challenge, where taking three separate approaches to solving one common problem led to a ensemble approach in deriving a better solution than the individual parts of the total sum of the work produced. The conversations within the group chat were mostly productive and was a good pool of inspiration for each other and certainly recommend this as an approach to better facilitate communication within a larger scope.

6.3 Lessons Learned

While this project was quite successful in delivering various materials and results, we believe that there are many areas for improvement and lessons learned. We summarize our lessons learned from famous principles inspired by Machine Learning principles, which are: Use Occam's Razor, There is No Free Lunch, and Don't Get Stuck in the Local Minima.

Use Occam's Razor

Occam's Razor is a principle which expresses that one should not make more assumptions than the minimum needed. In other words, we learned that we should always start from simple solutions and gradually build up in complexity. This is important not just in modeling, but also in how we approached the problem for our project, when we initially did not have a concrete direction that we wanted to pace towards. Starting something very simple from the very start enabled us to see what worked and what did not work and immediately get valuable feedback to try new things or continue spending time on the problem to see if we can obtain gains in our progress.

There is No Free Lunch

The No Free Lunch Theorem states that "We have dubbed the associated results NFL theorems because they demonstrate that if an algorithm performs well on a certain class of problems then it necessarily pays for that with degraded performance on the set of all remaining problems" [46]. We saw this as a crucial flaw when we were trying to design our data-driven research platform, as we realized that we needed to design our system in such a way that we retained enough automation for certain processes, we still had to work around some areas to add and remove modules to accommodate for specific modeling tasks. In other words, there are no one-size-fits-all type solutions that can be applied in Data Science.

Don't Get Stuck in the Local Minima

In optimization, one of the most common problems we are often faced with is getting stuck in some solution that is not considered optimal - or as they would say, getting stuck in the local optima. Retrospectively, we believe that we may have encountered this during our research efforts in using the Sky Group as a feature that may be useful. In hindsight, we realized that we should have quickly pivoted in orthogonal directions from our current efforts, and tried other solutions and methods that may have likely have better outcomes, than succumb to trying to squeeze out something that cant́ really offer any benefit. In this case, we would need to carefully consider the trade-offs between exploitation of investigating one particular solution and going for more exploratory approaches in finding something novel to try.

6.4 Suggestions and Improvements

After going through the Data Sciences Capstone project experience, we found that this experience has really helped us to understand and experience what a real-world application of what a data sciences project was. However, we saw that there could also be some ways to further improve the whole experience in ways that enables future students that will be undergoing this journey to better take advantage of the experience. We list some of the improvements and the rationale behind these suggestions:

- Make weekly reports and deliverable bi-weekly, and instead either facilitate a quick 1-min presentation of the current progress and difficulties that people are going through in class. This provides students with the opportunity to prepare much more sufficiently for the deliverable and yields much higher quality reports. Furthermore, having class time to present their work allows students to practice presenting their work to peers and also allows them to get feedback from other students for addressing some of the issues they may be facing.

- Make use of Piazza for cross-team collaboration, and include both clients as well as all students. Manage discussions based on some tags that they can use to specifically address questions for general (i.e. debugging, modeling tips, etc), to domain knowledge specific questions which may be directed to the client (which another student who is working on the project may potentially have an answer to). Although, this may be hard to execute if the client is from a third party company or where the nature of the data or problem being solved is sensitive in nature (for this maybe enforce some rule or policy to explicitly disclose what can and cannot be posted?)

- One point that Dr. Ford has raised during his time in class was the reciprocated communication between students and the domain expert. However, despite that conversation being held, we felt that this was not really followed through as there was a lack of directly feedback and communication between the client due to the availability of the client and the bottle neck of communication being held at one point for questions. Instead of having questions being sent in a batch manner, we would suggest having multiple areas of contacts that we can rely on. In the case of the Astronomy project, this could have been other professors or graduate students who are willing to answer questions or help us with some domain knowledge related topics. This would ensure that the average turnaround time for a question would be answered faster, and therefore provide much more efficiency in how students focus on the important problems and get better feedback to improve the quality of the project.

We realize that the logistical organization of this project is a very tremendous effort and we understand that it was a huge effort especially for Dr. Yen to organize much of the projects concurrently for us. We truly appreciate the time and effort spent to organize the project and hope that these suggestions would better improve the experience for future students.

7 CONCLUSIONS

In this paper, we present our method for utilizing machine learning methods to characterize the detection efficiency of the Kepler Robovetter's Exoplanet's ability to detect and identify exoplanets. In particular our contributes offers some key directions that can be taken in the methodology for how we can both identify and model these complex heuristics based pipelines and identify entry points for future work through how we can evaluate the robustness of these data pipelines.

In particular, our contributions to this project were as follows: i) we devised a Data-Driven Research Platform which allows researchers to develop Supervised Machine Learning models efficiently and effectively through a simple, scalable, flexible and iterative process which is semi-automated and reproducible; ii) we also developed a model for predicting the efficiency of the pipeline to detected whether a planetâĂŹs Threshold Crossing Event (TCE) of the Transit Planet Search (TPS) system pipeline is able to recover signals of the injected data or not; iii) furthermore, we also developed a model for a predictive model to also characterize the efficiency of the model that determines whether a Threshold Crossing Event is considered as a Planet Candidate or a False Positive result; iv) given the models derived from part (ii) and (iii), we perform an in-depth analysis using various machine learning based interpretability tools and techniques to obtain insights from the model, attempted to answer key questions raised by NASA, and provide some future directions that are worth exploring.

ACKNOWLEDGEMENT

First, we would like to thank both Dr. John Yen and Yiming Liao for the logistsical organization of this capstone project, as well as the various insights, discussion, suggestions, guidance, and supportive encouragements to further refine our project. We would also like to acknowledge Dr. Eric Ford for the contribution of this capstone project, which was very challenging and eye-opening towards the field of astronomical data analysis. In particular, we would like to acknowledge the following individuals, Daniel Walker and John Reid, for the insightful conversations, exchange of information, codes, and papers, as well as cross-referencing information on the project. We would also like to thank also the IT Services for their provision and technical support for the cluster and compute infrastructure. Finally, we would like to thank the entire College Information Sciences and Technology (IST) department, including staff, faculty, and professors for the opportunities, courses, and the dedicated support they have provided. Without the culmination of all these pieces, this project would not have manifested in its entirety.

REFERENCES

[1] [n. d.]. Data Science Version Control System. ([n. d.]). https://dvc.org/

[2] [n. d.]. Kepler Data Products (Updated February 15 2018). ([n. d.]). https://archive.stsci.edu/kepler/data_products.html

[3] Astropy Collaboration, T. P. Robitaille, E. J. Tollerud, P. Greenfield, M. Droettboom, E. Bray, T. Aldcroft, M. Davis, A. Ginsburg, A. M. Price-Whelan, W. E. Kerzendorf, A. Conley, N. Crighton, K. Barbary, D. Muna, H. Ferguson, F. Grollier, M. M. Parikh, P. H. Nair, H. M. Unther, C. Deil, J. Woillez, S. Conseil, R. Kramer, J. E. H. Turner, L. Singer, R. Fox, B. A. Weaver, V. Zabalza, Z. I. Edwards, K. Azalee Bostroem, D. J. Burke, A. R. Casey, S. M. Crawford, N. Dencheva, J. Ely, T. Jenness, K. Labrie, P. L. Lim, F. Pierfederici, A. Pontzen, A. Ptak, B. Refsdal, M. Servillat, and O. Streicher. 2013. Astropy: A community Python package for astronomy. 558, Article A33 (Oct. 2013), A33 pages. https://doi.org/10.1051/0004-6361/201322068 arXiv:astro-ph.IM/1307.6212

[4] Natalie M Batalha, Jason F Rowe, Stephen T Bryson, Thomas Barclay, Christopher J Burke, Douglas A Caldwell, Jessie L Christiansen, Fergal Mullally, Susan E Thompson, Timothy M Brown, et al. 2013. Planetary candidates observed by Kepler. III. Analysis of the first 16 months of data. *The Astrophysical Journal Supplement Series* 204, 2 (2013), 24.

[5] Brett K Beaulieu-Jones and Casey S Greene. 2017. Reproducibility of computational workflows is automated using continuous analysis. *Nature biotechnology* 35, 4 (2017), 342.

[6] William J Borucki, David G Koch, Gibor Basri, Natalie Batalha, Alan Boss, Timothy M Brown, Douglas Caldwell, Jørgen Christensen-Dalsgaard, William D Cochran, Edna DeVore, et al. 2011. Characteristics of Kepler planetary candidates based on the first data set. *The Astrophysical Journal* 728, 2 (2011), 117.

[7] Leo Breiman. 1999. Pasting small votes for classification in large databases and on-line. *Machine learning* 36, 1-2 (1999), 85–103.

[8] Leo Breiman. 2001. Random forests. *Machine learning* 45, 1 (2001), 5–32.

[9] Leo Breiman. 2017. *Classification and regression trees.* Routledge.

[10] Christopher J Burke, Stephen T Bryson, F Mullally, Jason F Rowe, Jessie L Christiansen, Susan E Thompson, Jeffrey L Coughlin, Michael R Haas, Natalie M Batalha, Douglas A Caldwell, et al. 2014. Planetary candidates observed by Kepler IV: Planet sample from Q1-Q8 (22 Months). *The Astrophysical Journal Supplement Series* 210, 2 (2014), 19.

[11] Joseph H Catanzarite. 2015. *Autovetter Planet Candidate Catalog for Q1-Q17 Data Release 24.* Technical Report.

[12] Jessie L Christiansen. 2017. Kepler Planet Detection Metrics: Pixel-Level Transit Injection Tests of Pipeline Detection Efficiency for Data Release 25. (2017).

[13] Jessie L Christiansen, Bruce D Clarke, Christopher J Burke, Jon M Jenkins, Thomas S Barclay, Eric B Ford, Michael R Haas, Anima Sabale, Shawn Seader, Jeffrey Claiborne Smith, et al. 2013. Measuring transit signal recovery in the Kepler pipeline. I. Individual events. *The Astrophysical Journal Supplement Series* 207, 2 (2013), 35.

[14] Jessie L Christiansen, Bruce D Clarke, Christopher J Burke, Jon M Jenkins, Stephen T Bryson, Jeffrey L Coughlin, Fergal Mullally, Susan E Thompson, Joseph D Twicken, Natalie M Batalha, et al. 2016. MEASURING TRANSIT SIGNAL RECOVERY IN THE KEPLER PIPELINE. III. COMPLETENESS OF THE Q1–Q17 DR24 PLANET CANDIDATE CATALOG WITH IMPORTANT CAVEATS FOR OCCURRENCE RATE CALCULATIONS. *The Astrophysical Journal* 828, 2 (2016), 99.

[15] Jessie L Christiansen, Bruce D Clarke, Christopher J Burke, Shawn Seader, Jon M Jenkins, Joseph D Twicken, Joseph D Catanzarite, Jeffrey C Smith, Natalie M Batalha, Michael R Haas, et al. 2015. MEASURING TRANSIT SIGNAL RECOVERY IN THE KEPLER PIPELINE. II. DETECTION EFFICIENCY AS CALCULATED IN ONE YEAR OF DATA. *The Astrophysical Journal* 810, 2 (2015), 95.

[16] Jessie L Christiansen, Jon M Jenkins, Douglas A Caldwell, Christopher J Burke, Peter Tenenbaum, Shawn Seader, Susan E Thompson, Thomas S Barclay, Bruce D Clarke, Jie Li, et al. 2012. The derivation, properties, and value of KeplerâĂŹs combined differential photometric precision. *Publications of the Astronomical Society of the Pacific* 124, 922 (2012), 1279.

[17] Dan Claudiu Ciresan, Ueli Meier, Jonathan Masci, Luca Maria Gambardella, and Jürgen Schmidhuber. 2011. Flexible, high performance convolutional neural networks for image classification. In *Twenty-Second International Joint Conference on Artificial Intelligence.*

[18] Jeffrey L Coughlin. 2017. Planet Detection Metrics: Robovetter Completeness and Effectiveness for Data Release 25. (2017).

[19] Anna Veronika Dorogush, Vasily Ershov, and Andrey Gulin. 2018. CatBoost: gradient boosting with categorical features support. *arXiv preprint arXiv:1810.11363* (2018).

[20] Cynthia Dwork, Ravi Kumar, Moni Naor, and Dandapani Sivakumar. 2001. Rank aggregation methods for the web. In *Proceedings of the 10th international conference on World Wide Web.* ACM, 613–622.

[21] Debra A Fischer, Megan E Schwamb, Kevin Schawinski, Chris Lintott, John Brewer, Matt Giguere, Stuart Lynn, Michael Parrish, Thibault Sartori, Robert Simpson, et al. 2012. Planet Hunters: the first two planet candidates identified by the public using the Kepler public archive data. *Monthly Notices of the Royal Astronomical Society* 419, 4 (2012), 2900–2911.

[22] François Fressin, Guillermo Torres, David Charbonneau, Stephen T Bryson, Jessie Christiansen, Courtney D Dressing, Jon M Jenkins, Lucianne M Walkowicz, and Natalie M Batalha. 2013. The false positive rate of Kepler and the occurrence of planets. *The Astrophysical Journal* 766, 2 (2013), 81.

[23] Yoav Freund and Robert E Schapire. 1997. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences* 55, 1 (1997), 119–139.

[24] Jerome H Friedman. 2001. Greedy function approximation: a gradient boosting machine. *Annals of statistics* (2001), 1189–1232.

[25] Pierre Geurts, Damien Ernst, and Louis Wehenkel. 2006. Extremely randomized trees. *Machine Learning* 63, 1 (01 Apr 2006), 3–42. https://doi.org/10.1007/s10994-006-6226-1

[26] Geoffrey E Hinton. 1990. Connectionist learning procedures. In *Machine learning.* Elsevier, 555–610.

[27] Jon M Jenkins, Douglas A Caldwell, Hema Chandrasekaran, Joseph D Twicken, Stephen T Bryson, Elisa V Quintana, Bruce D Clarke, Jie Li, Christopher Allen, Peter Tenenbaum, et al. 2010. Initial characteristics of Kepler long cadence data for detecting transiting planets. *The Astrophysical Journal Letters* 713, 2 (2010), L120.

[28] Jon M Jenkins, Douglas A Caldwell, Hema Chandrasekaran, Joseph D Twicken, Stephen T Bryson, Elisa V Quintana, Bruce D Clarke, Jie Li, Christopher Allen, Peter Tenenbaum, et al. 2010. Overview of the Kepler science processing pipeline. *The Astrophysical Journal Letters* 713, 2 (2010), L87.

[29] Jon M Jenkins, Peter Tenenbaum, Shawn Seader, Christopher J Burke, Sean D McCauliff, Jeffrey C Smith, Joseph D Twicken, and Hema Chandrasekaran. 2017. Kepler Data Processing Handbook: Transiting Planet Search. *Kepler Science Document, KSCI-19081-002, Edited by Jon M. Jenkins.* (2017).

[30] Rie Johnson and Tong Zhang. 2013. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in neural information processing systems.* 315–323.

[31] David G Koch, William J Borucki, Gibor Basri, Natalie M Batalha, Timothy M Brown, Douglas Caldwell, Jørgen Christensen-Dalsgaard, William D Cochran, Edna DeVore, Edward W Dunham, et al. 2010. Kepler mission design, realized photometric performance, and early science. *The Astrophysical Journal Letters* 713, 2 (2010), L79.

[32] Scott M Lundberg and Su-In Lee. 2017. Consistent feature attribution for tree ensembles. *arXiv preprint arXiv:1706.06060* (2017).

[33] Scott M Lundberg and Su-In Lee. 2017. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems.* 4765–4774.

[34] Savita Mathur, Daniel Huber, Natalie M Batalha, David R Ciardi, Fabienne A Bastien, Allyson Bieryla, Lars A Buchhave, William D Cochran, Michael Endl, Gilbert A Esquerdo, et al. 2017. Revised stellar properties of Kepler targets for the Q1-17 (DR25) transit detection run. *The Astrophysical Journal Supplement Series* 229, 2 (2017), 30.

[35] Frank Moosmann, Bill Triggs, and Frederic Jurie. 2007. Fast discriminative visual codebooks using randomized clustering forests. In *Advances in neural information processing systems.* 985–992.

[36] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python . *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[37] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining.* ACM, 1135–1144.

[38] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2018. Anchors: High-precision model-agnostic explanations. In *Thirty-Second AAAI Conference on Artificial Intelligence.*

[39] Jason F Rowe, Stephen T Bryson, Geoffrey W Marcy, Jack J Lissauer, Daniel Jontof-Hutter, Fergal Mullally, Ronald L Gilliland, Howard Issacson, Eric Ford, Steve B Howell, et al. 2014. Validation of Kepler's multiple planet candidates. III. Light curve analysis and announcement of hundreds of new multi-planet systems. *The Astrophysical Journal* 784, 1 (2014), 45.

[40] D Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, and Michael Young. 2014. Machine learning: The high interest credit card of technical debt. (2014).

[41] David Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. 2015. Hidden technical debt in machine learning systems. In *Advances in neural information processing systems.* 2503–2511.

[42] Christopher J Shallue and Andrew Vanderburg. 2018. Identifying exoplanets with deep learning: A five-planet resonant chain around kepler-80 and an eighth planet around kepler-90. *The Astronomical Journal* 155, 2 (2018), 94.

[43] Susan E Thompson, Jeffrey L Coughlin, Kelsey Hoffman, Fergal Mullally, Jessie L Christiansen, Christopher J Burke, Steve Bryson, Natalie Batalha, Michael R Haas, Joseph Catanzarite, et al. 2018. Planetary candidates observed by Kepler. VIII. A fully automated catalog with measured completeness and reliability based on

data release 25. *The Astrophysical Journal Supplement Series* 235, 2 (2018), 38.

[44] Susan E Thompson, Dorothy Fraquelli, Jeffrey E Van Cleve, and Douglas A Caldwell. 2016. Kepler Archive Manual. (2016).

[45] Jeffrey E Van Cleve, Jessie L Christiansen, Jon M Jenkins, Douglas A Caldwell, Thomas Barclay, Stephen T Bryson, Christopher J Burke, Jennifer Cambell, Joseph Catanzarite, Bruce D Clarke, et al. 2016. Kepler Data Characteristics Handbook. *Kepler Science Document, KSCI-19040-005, Edited by Doug Caldwell, Jon M. Jenkins, Michael R. Haas, and Natalie Batalha* (2016).

[46] David H Wolpert, William G Macready, et al. [n. d.]. No free lunch theorems for optimization. ([n. d.]).

[47] Harry Zhang. 2004. The optimality of naive Bayes. *AA* 1, 2 (2004), 3.

Appendices

A PHASE 1: PLTI ROBOVETTER INJECTION REFERENCE

A.1 Feature Importance Ranking Results

Table 1: Feature Importance/Selection with Base Features

	AdaBoost	Extra Trees	GBM	Lasso	Random Forest	Random Trees	Chi Squared
Expected_MES	0.154	**0.4783**	**0.6553**	1	**0.3323**	0.0912	179.6021226
i_b	0.099	0.0659	0.0174	1	0.0645	0.0871	26.32305742
i_depth	0.083	0.0434	0.014	0	0.0634	0.1215	130.6990697
i_dor	0.107	0.0636	0.0267	1	0.0766	0.0849	48.73736431
i_dur	0.09	0.0663	0.1396	1	0.0936	0.09	**249.1171421**
i_epoch	0.11	0.0413	0.0189	0	0.063	0.0957	41.21114159
i_period	**0.186**	0.0529	0.0154	1	0.0911	0.0942	24.70575607
i_ror	0.073	0.0537	0.0212	0	0.0596	0.1134	119.2648146
N_Transit	0.077	0.0954	0.0833	0	0.1052	**0.1426**	23.67576581
Sky_Group	0.021	0.0391	0.0083	0	0.0505	0.0795	1.682131843

Table 2: Feature Importance/Selection with Base Features + Sky Group Feature Engineering

	AdaBoost	Extra Trees	GBM	Lasso	Random Forest	Random Trees	Chi Squared
Expected_MES	0.146	**0.4683**	**0.6487**	1	**0.3176**	0.08	179.6021226
i_b	0.102	0.0611	0.017	1	0.0608	0.0674	26.32305742
i_depth	0.078	0.0395	0.0138	0	0.0599	0.1037	130.6990697
i_dor	0.106	0.0589	0.0266	1	0.074	0.0683	48.73736431
i_dur	0.08	0.0613	0.1369	1	0.0867	0.0688	**249.1171421**
i_epoch	0.115	0.0365	0.0184	0	0.0587	0.0748	41.21114159
i_period	**0.182**	0.0481	0.0159	1	0.0841	0.0761	24.70575607
i_ror	0.077	0.0494	0.0204	0	0.0577	0.0921	119.2648146
N_Transit	0.075	0.0896	0.0883	0	0.0982	**0.1224**	23.67576581
sg_center_pool	0	0.0208	0.002	0	0.0237	0.0607	3.224649261
sg_corner_pool	0.018	0.0279	0.0056	1	0.0327	0.0638	8.03443174
sg_ori_pool	0	0.0162	0.0011	0	0.0091	0.0546	1.053355337
Sky_Group	0.021	0.0225	0.0054	0	0.0369	0.0673	1.682131843

A.2 Model Results

Table 3: Baseline Model Results

Model	RMSE	Log Loss	Accuracy	Precision	Recall	F1	AUC	Kappa
CatBoost	0.1081	3.7331	0.8919	0.8246	0.8274	0.8259	0.8739	0.7473
Adaboost Classifier	0.1112	3.8391	0.8888	0.8075	0.8425	0.8246	0.8758	0.7430
XGBoost	0.1120	3.8670	0.8880	0.8144	0.8276	0.8209	0.8712	0.7392
Random Forest	0.1200	4.1449	0.8800	0.8303	0.7704	0.7991	0.8496	0.7135
Logistic Regression	0.1306	4.5101	0.8697	0.8261	0.7338	0.7770	0.8320	0.6852
MLP	0.1320	4.5599	0.8690	0.8215	0.7379	0.7773	0.8329	0.6847
NB Bernoulli	0.1446	4.9946	0.8554	0.7311	0.8436	0.7832	0.8521	0.6753
SGD Classifier	0.1511	5.2184	0.8377	0.7995	0.6920	0.7141	0.7953	0.6042
Decision Tree	0.1562	5.3952	0.8447	0.7492	0.7498	0.7494	0.8184	0.6366
K-NN Classifier	0.1684	5.8178	0.8415	0.8299	0.6178	0.7043	0.7808	0.6008
Baseline Gamma	0.3083	10.6496	0.6919	0.7851	0.0092	0.0182	0.5040	0.0111
NB Gaussian	0.5902	20.3863	0.4098	0.3389	0.9503	0.4992	0.5585	0.0781

Table 4: Sky Group Based Features Baseline Results

Model	RMSE	Log Loss	Accuracy	Precision	Recall	F1	AUC	Kappa
CatBoost	0.1082	3.7374	0.8918	0.8240	0.8279	0.8259	0.8740	0.7471
Adaboost Classifier	0.1104	3.8139	0.8896	0.8082	0.8445	0.8258	0.8769	0.7448
Random Forest	0.1107	3.8228	0.8893	0.8226	0.8200	0.8212	0.8700	0.7408
XGBoost	0.1121	3.8703	0.8879	0.8142	0.8276	0.8208	0.8711	0.7390
Logistic Regression	0.1293	4.4645	0.8707	0.8262	0.7383	0.7796	0.8341	0.6883
MLP	0.1298	4.4841	0.8702	0.8241	0.7385	0.7785	0.8336	0.6867
SGD Classifier	0.1321	4.5616	0.8679	0.8236	0.7353	0.7737	0.8309	0.6808
NB Bernoulli	0.1446	4.9948	0.8554	0.7308	0.8441	0.7833	0.8522	0.6753
Decision Tree	0.1560	5.3884	0.8440	0.7477	0.7490	0.7483	0.8175	0.6349
K-NN Classifier	0.1604	5.5387	0.8396	0.8298	0.6053	0.6960	0.7749	0.5914
Baseline Gamma	0.3083	10.6496	0.6919	0.7851	0.0092	0.0182	0.5040	0.0111
NB Gaussian	0.5900	20.3794	0.4100	0.3390	0.9501	0.4992	0.5586	0.0782

Table 5: Sky Group Omitted Baseline Results

Model	RMSE	Log Loss	Accuracy	Precision	Recall	F1	AUC	Kappa
CatBoost	0.1076	3.7157	0.8924	0.8251	0.8289	0.8269	0.8747	0.7486
Adaboost Classifier	0.1104	3.8134	0.8896	0.8080	0.8448	0.8259	0.8770	0.7449
XGBoost	0.1112	3.8422	0.8888	0.8151	0.8297	0.8222	0.8723	0.7410
Random Forest	0.1115	3.8521	0.8885	0.8216	0.8179	0.8197	0.8688	0.7387
MLP	0.1138	3.9305	0.8862	0.8164	0.8164	0.8163	0.8668	0.7336
Logistic Regression	0.1292	4.4617	0.8708	0.8275	0.7367	0.7793	0.8337	0.6882
SGD Classifier	0.1297	4.4794	0.8703	0.8273	0.7350	0.7783	0.8330	0.6869
K-NN Classifier	0.1368	4.7233	0.8632	0.8202	0.7159	0.7641	0.8228	0.6683
Decision Tree	0.1569	5.4184	0.8431	0.7457	0.7490	0.7473	0.8170	0.6332
NB Bernoulli	0.1712	5.9127	0.8288	0.7264	0.7186	0.7223	0.7985	0.5983
Baseline Gamma	0.3083	10.6496	0.6919	0.7851	0.0092	0.0182	0.5040	0.0111
NB Gaussian	0.5700	19.6886	0.4300	0.3465	0.9469	0.5070	0.5722	0.0976

Table 6: Ensemble Models

Model	RMSE	Log Loss	Accuracy	Precision	Recall	F1	AUC	Kappa
Stack Ensemble LR (T3)	0.1081	3.7331	0.8919	0.8246	0.8274	0.8259	0.8739	0.7473
Vote Ensemble (T3)	0.1093	3.7737	0.8907	0.8207	0.8287	0.8246	0.8734	0.7450
Vote Ensemble (T5)	0.1103	3.8089	0.8897	0.8268	0.8152	0.8209	0.8690	0.7410
Stack Ensemble LR (T5)	0.1209	4.1741	0.8791	0.8280	0.7702	0.7980	0.8489	0.7116
Baseline Gamma	0.3083	10.6496	0.6919	0.7851	0.0092	0.0182	0.5040	0.0111

B PHASE 2: TCES ROBOVETTER REFERENCE

B.1 Feature Importance Ranking Results

Table 7: Feature Importance/Selection with Base Features

_	**AdaBoost**	**Extra Trees**	**GBM**	**Lasso**	**Random Forest**	**Random Trees**	**Chi Squared**
Score	0.05	0.2141	0.7984	1	0.4602	0.0502	293.54075
NTL	0.059	0.6136	0.127	1	0.3243	0.0233	443.73056
SS	0.458	0.0702	0.0349	1	0.0447	0.015	502.77279
CO	0.043	0.0991	0.0376	1	0.0674	0.0152	499.63318
EM	0.005	0.0026	0.001	1	0.0013	0.0113	511.14551
period	0.021	0	0.0002	0	0.0089	0.0336	23.5613
epoch	0.023	0	0	0	0.0024	0.0378	37.80158
Expected MES	0.01	0	0	0	0.0041	0.0742	33.94005
MES	0.003	0	0.0003	0	0.0173	0.0837	42.83105
NTran	0.034	0.0001	0.0002	0	0.012	0.0489	41.43745
depth	0.002	0	0	0	0.0022	0.0418	127.89693
duration	0.027	0	0.0001	0	0.0038	0.0341	103.73231
Rp	0.012	0	0	0	0.009	0.054	204.49666
Rs	0.027	0	0	0	0.0041	0.0488	143.35453
Ts	0.007	0	0	0	0.0019	0.0428	29.81339
logg	0.007	0	0	0	0.0037	0.0454	99.85262
a	0.052	0	0.0001	0	0.0107	0.0392	22.03401
Rp/Rs	0.011	0	0	0	0.0025	0.0467	189.60708
a/Rs	0.083	0	0	0	0.0028	0.0442	63.0083
impact	0.025	0	0	0	0.0021	0.0406	2.3774
SNR_DV	0	0	0	0	0.0121	0.0718	38.17656
Sp	0.041	0.0001	0.0001	0	0.0023	0.0364	224.63078
Fit_Prov	0	0	0	0	0.0001	0.061	417.75156

Table 8: Feature Importance/Selection with Removed Features

	AdaBoost	Extra Trees	GBM	Lasso	Random Forest	Random Trees	Chi Squared
period	0.07	0.0702	0.0607	1	0.0657	0.0431	23.5613
epoch	0.063	0.0618	0.0633	1	0.0563	0.0431	37.80158
Expected MES	0.077	0.076	0.0937	0	0.0774	0.1042	33.94005
NTran	0.049	386	0.0457	0	0.0468	0.0542	41.43745
depth	0.063	0.0536	0.0491	0	0.0532	0.045	127.89693
duration	0.068	0.062	0.0637	0	0.0606	0.047	103.73231
Rp	0.065	0.0653	0.0956	0	0.0776	0.0706	204.49666
Rs	0.054	0.0569	0.0395	0	0.0568	0.0493	143.35453
Ts	0.039	0.0567	0.049	0	0.0552	0.0506	29.81339
logg	0.075	0.0661	0.0401	1	0.0578	0.0586	99.85262
a	0.073	0.072	0.0604	0	0.0716	0.0468	22.03401
Rp/Rs	0.045	0.0546	0.0488	1	0.0525	0.0618	189.60708
a/Rs	0.058	0.0572	0.0622	1	0.0551	0.0493	63.0083
impact	0.038	0.0583	0.053	0	0.0552	0.0483	2.3774
SNR_DV	0.112	0.0936	0.1272	1	0.1069	0.1078	38.17656
Sp	0.046	0.0478	0.046	0	0.0506	0.0393	224.63078
Fit_Prov	0.005	0.0093	0.0021	1	0.0008	0.0679	417.75156

B.2 Model Results

Table 9: Baseline Model Results

Models	RMSE	Log Loss	Accuracy	Precision	Recall	F1	AUC	Kappa
AdaBoost	0.1398059	4.8288213	0.8601941	0.8314928	0.8601941	0.8269183	0.5790176	0.2215057
CatBoost	0.1348033	4.6560388	0.8651967	0.84279	0.8651967	0.8302303	0.5811168	0.2324654
Decision Tree	0.2157695	7.4524965	0.7842305	0.7922355	0.7842305	0.7880072	0.5903936	0.1740383
Random Forest	0.1349797	4.6621302	0.8650203	0.8422015	0.8650203	0.8303425	0.5815998	0.2329431
K-Nearest Neighbors	0.1441032	4.9772619	0.8558968	0.8325828	0.8558968	0.7979631	0.5209794	0.0681392
Logistic Regression	0.1426707	4.927781	0.8573293	0.8312126	0.8573293	0.8052391	0.5329934	0.1041091
MLP	0.1470783	5.0800218	0.8529217	0.7471708	0.8529217	0.7873895	0.5048019	0.0154006
Naive Bayes (Bernoulli)	0.1823825	6.299358	0.8176175	0.7962029	0.8176175	0.8052425	0.5816848	0.1839217
Naive Bayes (Gaussian)	0.149855	5.17591	0.850145	0.8112269	0.850145	0.8150524	0.561126	0.1692185
Random Forest	0.1363902	4.7108471	0.8636098	0.8391171	0.8636098	0.8281957	0.577799	0.2228036
SGDC	0.2032387	7.019724	0.7967613	0.7733614	0.7967613	0.7394661	0.508971	0.0103582
XGBoost	0.1361036	4.7009512	0.8638964	0.8426802	0.8638964	0.8245737	0.5686242	0.2024474

Table 10: Log Transformed Feature-Based Models

Models	RMSE	Log Loss	Accuracy	Precision	Recall	F1	AUC	Kappa
AdaBoost	0.13983	4.82958	0.86017	0.83144	0.86017	0.8269	0.579	0.22145
CatBoost	0.1348	4.65604	0.8652	0.84279	0.8652	0.83023	0.58112	0.23247
Decision Tree	0.2152	7.43271	0.7848	0.7929	0.7848	0.78865	0.59178	0.17664
Random Forest	0.13595	4.69562	0.86405	0.84012	0.86405	0.82905	0.57955	0.22732
K-Nearest Neighbors	0.14093	4.86765	0.85907	0.84005	0.85907	0.80744	0.53623	0.11436
Logistic Regression	0.13844	4.78164	0.86156	0.84653	0.86156	0.81288	0.54518	0.14106
MLP	0.1378	4.75956	0.8622	0.83864	0.8622	0.82118	0.56263	0.18591
Naive Bayes (Bernoulli)	0.17921	6.18975	0.82079	0.79758	0.82079	0.80711	0.58207	0.18742
Naive Bayes (Gaussian)	0.14792	5.10893	0.85208	0.8132	0.85208	0.81456	0.55744	0.16278
Random Forest	0.13608	4.70019	0.86392	0.83981	0.86392	0.82859	0.57842	0.22464
SGDC	0.2079	7.18062	0.7921	0.8299	0.7921	0.74776	0.55428	0.15526
XGBoost	0.1361	4.70095	0.8639	0.84268	0.8639	0.82457	0.56862	0.20245