Machine Learning in Exoplanet Exploration
Capstone Project

Alexandros Kapeletzis

April 2019

1 Definition

1.1 Domain Background

Exploring the unknown and understanding the Universe and our place in it has been a strong desire for mankind for thousands of years. Looking at the stars in the night sky fills us with fascination but also awe and wonder. How many stars are there in the Universe? Are there any planets similar to Earth or is our planet unique? Are we alone or is there life in other places of our galaxy? Thanks to our curiosity and the continuous advancements in science and technology we have been able to start answering many of these questions. It is estimated that the Milky Way, the galaxy that our Solar System is part of, contains between 200 and 400 billion stars. It is also believed that there are trillions of galaxies in the Universe, meaning that the total number of stars is likely to be of the order of 10^{24} [1]. This breathtaking calculation gives rise to a very important question: are these stars also orbited by planets, like our Sun?

Planets outside the Solar System are called exoplanets. The first such planet was discovered in 1995 through what is known as the radial-velocity method [2]. In 2009, NASA launched the Kepler Space Telescope into space. The spacecraft's mission was to detect Earth-sized exoplanets in our galaxy using a different approach known as the transit method [3]. The telescope watched a particular area of the sky containing about 150,000 stars for over nine years. It monitored each of the stars and aimed at detecting small dips in the brightness of their light, as planets pass in front of them during their orbit. This has led to the detection of more than 9,000 candidates or "Kepler Objects of Interest" (KOI), out of which 2,337 have been classified as confirmed exoplanets [4]. The Kepler mission ended in 2013 due to a severe malfunction of the telescope. However, NASA engineers managed to find a way of restabilising the spacecraft using sunlight pressure. This made it possible for the telescope to continue exploring space, giving birth to a new mission, named K2 [5].

The search for exoplanets is an open problem, so NASA has wisely made the data collected by both the Kepler and K2 missions publicly available. One can download the so called flux time-series data for each star that the telescope has monitored from the NASA Exoplanet Archive [6]. The data essentially consists of measurements of the light intensity of a star at regular time intervals (29.4 minutes for so-called long cadence (LC) targets and 1 minute for short cadence (SC) targets [7]). From now on we will refer to this type of data as flux data or flux time-series (where flux is equivalent to the light intensity). Scientists have manually analysed a lot of these flux time-series and determined whether the corresponding stars have exoplanet candidates or not. NASA also provides tables including all the candidates or "Kepler Objects of Interest" for both Kepler [8] and K2 missions [9].

Classifying a star as having exoplanets or not based on the Kepler measurements currently involves a lot of time-consuming, manual analysis and validation by astrophysicists. It is therefore believed that machine learning techniques could be utilised to make this process more efficient and aid the exoplanet exploration efforts.

1.2 Problem Statement

As mentioned above, scientists have to go through the data collected by the Kepler telescope to identify potential candidates and then decide whether each of them is an exoplanet or not. False positives can arise due to binary star systems (where another star is transiting in front of the target star), background objects or astrophysical stellar variability [10]. This classification process is to a large extent manual, which is obviously expensive in terms of time and resources. The K2 mission has collected flux data from hundreds of thousands of stars so far, many of which have not been classified yet. Moreover, the mission is still active, monitoring more and more stars. For this reason, it is imperative that sophisticated algorithms that can analyse the measured flux time-series and accurately determine whether a star is orbited by exoplanets or not are developed. This will be the primary problem explored in this capstone project.

To summarise the above, the objective of the project will be to develop a machine learning tool that identifies exoplanet candidates directly from the raw flux time-series data collected by NASA's Kepler telescope. It will be a binary classifier, where a positive output signifies that a star is likely to have an exoplanet orbiting it and a negative output means that there are no exoplanets. Depending on its performance, this could act as a first screening step, reducing scientists' workload and potentially discovering new exoplanets.

The solution will consist of three main steps. Firstly, a number of preprocessing techniques will be applied to the raw flux data. This will be followed by a feature engineering step, whose aim will be to extract a number of meaningful features from the time-series. Finally, these features will be used to train and test a binary classification algorithm.

1.3 Evaluation Metrics

The primary goal of the binary classifier is the detection of as many exoplanets as possible. However, only a very tiny subset of the thousands of target stars have been found to have an exoplanet (or at least an unconfirmed exoplanet candidate). Hence, accuracy would not be an appropriate evaluation metric for the model, due to the large imbalance of the dataset. In contrast, recall would be a measure of particular importance, since it indicates how many of the total exoplanets have been captured by the model. It is defined as $\frac{TP}{TP+FN}$ where TP is the number of True Positives (exoplanet candidates that have been correctly detected by the classifier) and FN the number of False Negatives (exoplanet candidates that have not been detected and have therefore been classified as negatives).

Another goal of the proposed solution is to reduce the amount of time that scientists spend to analyse the data collected by Kepler. Therefore, it will also be desirable to minimize the number of false positives that are incorrectly classified as exoplanet candidates by the model. This is measured by the precision metric, defined as $\frac{TP}{TP+FP}$, where TP is the number of True Positives and FP the number of False Positives (i.e. objects incorrectly classified as exoplanet candidates).

It follows that an evaluation metric that combines both precision and recall would be ideal for our problem. This can be obtained using a F_β measure, which is defined as:

$$F_\beta = (1+\beta^2)\frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall}$$

It is important to remember that our primary goal is the detection of exoplanets and thus recall should have a greater weight than precision. Therefore, it has been decided to use a F_β score with a high β value equal to 2 as the evaluation metric for this application.

2 Analysis

2.1 Data Exploration

Obtaining a suitable dataset to train and test the star classifier was a critical task that took a significant amount of time at the beginning of the project. As described above, the data used is of the form of so-called flux time-series (i.e. measures of the light intensity of stars at different points in time). Moreover, as the proposed solution to the problem is a supervised learning model, each of the time-series must be labelled (positive or 1 if the corresponding star has an exoplanet candidate and negative or 0 if there are no exoplanets orbiting the star).

The first approach was to explore an existing dataset available on Kaggle [11]. It includes a collection of labelled flux data from stars monitored in the K2 mission. In particular, it is made up of 5,657 sample points split into a training set (5,087 samples) and a test set (570 samples). Each time-series in the dataset is labelled as 2 or 1, depending on whether or not the corresponding star has been classified as being orbited by at least one exoplanet, respectively. Moreover, each time-series consists of 3,197 sample points spanning a period of approximately 80 days.

One of the first observations about this dataset was related to its large imbalance. There are only 37 and 5 positive samples in the training and test sets respectively. This equates to as little as 1% of the entire dataset, meaning that the dataset is largely imbalanced.

Class imbalance is a very common problem in machine learning. When one class is represented by much fewer instances than the other, most binary classification algorithms tend to have a poor performance on the minority class. In extreme cases, this may cause everything to be classified as the majority class, which is of course undesirable in this application where we are actually interested in identifying the minority of stars that have exoplanet candidates. Therefore, it was decided to produce a new flux dataset from scratch rather than using the highly imbalanced Kaggle dataset, as a first measure against this problem.

Luckily, the dataset described above only includes a small subset of the data recorded by the Kepler telescope. In particular, the samples are collected primarily from Campaign 3 of the K2 mission. The mission has completed 20 campaigns (0-19), each of which scans a different field or patch of stars [12]. NASA provides the flux data for each monitored star to the public [13] in the form of FITS (Flexible Image Transport System) files, which is a common file type in astronomy.

There are thousands of such files per campaign and therefore a choice had to be made with regards to which campaigns will be downloaded and used. The following chart shows the exoplanet candidates identified so far per campaign and their classification ('Confirmed', 'Candidate' or 'False Positive') according to the K2 candidates table [9].

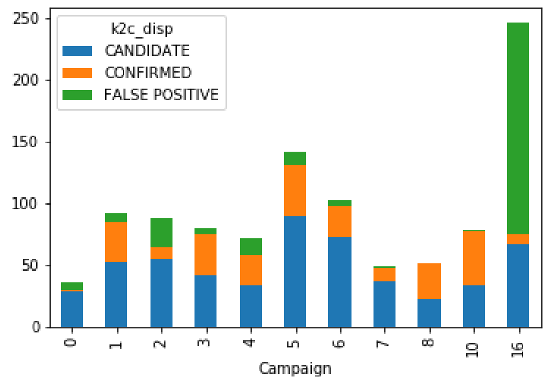


Figure 1: Exoplanet candidates by campaign.

Based on this, campaigns 1,3, 5 and 10 have been identified as the most useful ones as they include the largest number of confirmed exoplanets and candidates. The flux data for these 4 campaigns were then downloaded and compiled into a single dataset. The FITS files were handled using the Astropy package, which provides a set of tools for performing astronomy and astrophysics tasks on Python [14]. Each sample was labelled as 1 if the corresponding star is included in the K2 candidates table and 0 otherwise.

A first observation was that the time-series do not have the same length (i.e. number of measurements) for all stars. It varies from 2,983 to 3,555 data points, with a mean of 3,392. There are two options to overcome this issue. The first is to try and fill or pad the smaller time-series, so that every sample will have exactly 3,555 time-series points (the maximum). This would however alter the properties of the time-series, so it is deemed unfavourable in this case. The second option is to choose an appropriate threshold length and then truncate longer time-series to this length and discard all the time-series with fewer points than the chosen minimum. Setting the threshold length to a very high value means that many samples will be discarded, so many of the collected datapoints/stars will be lost. On the other hand, a very small threshold length will mean that longer time-series will be truncated even more, which may lead to loss of some useful information. In this case, the effect that we are trying to detect (i.e. the transit of planets in front of stars) is periodic and should therefore exist throughout the time-series of positive samples. This means that truncating some of the series should not have a big impact on the dataset's information.

Taking the above into consideration, it was chosen to use campaign 3's mean length (3,162) as the threshold. The flux data in campaign 10 generally have shorter lengths and were therefore discarded completely. This led to a dataset of 60,645 samples, including 279 exoplanet candidates, with 3,162 flux measurements each. The final step that was used in order to obtain a more balanced dataset is known as under-sampling; the time-series of all stars with exoplanet candidates were kept, but only a given number of randomely selected non-exoplanet samples were included in the dataset. This number was chosen to be 3,721, so that we end up with a final dataset of 4,000 samples and the following class split:

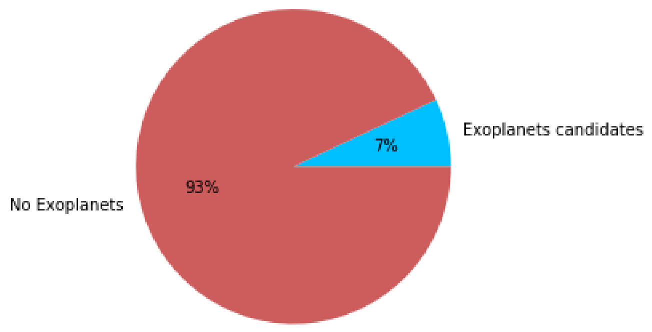


Figure 2: Distribution of classes in final dataset.

The positive samples now account for 7% of the dataset, which is definitely an improvement compared to the Kaggle dataset. Further measures against class imbalance will be employed at later stages of the classifier's training process.

2.2 Exploratory Visualisation

Now that the final dataset has been created, it is useful to try to understand what the flux data look like. Since each sample is associated with 3,162 measurements, the only way to make sense of the dataset is through some form of visualisation. To this end, the following charts show the flux time-series data for 3 randomly selected positive samples (stars with exoplanet candidates) and 3 randomly selected negative samples (non-exoplanet stars):

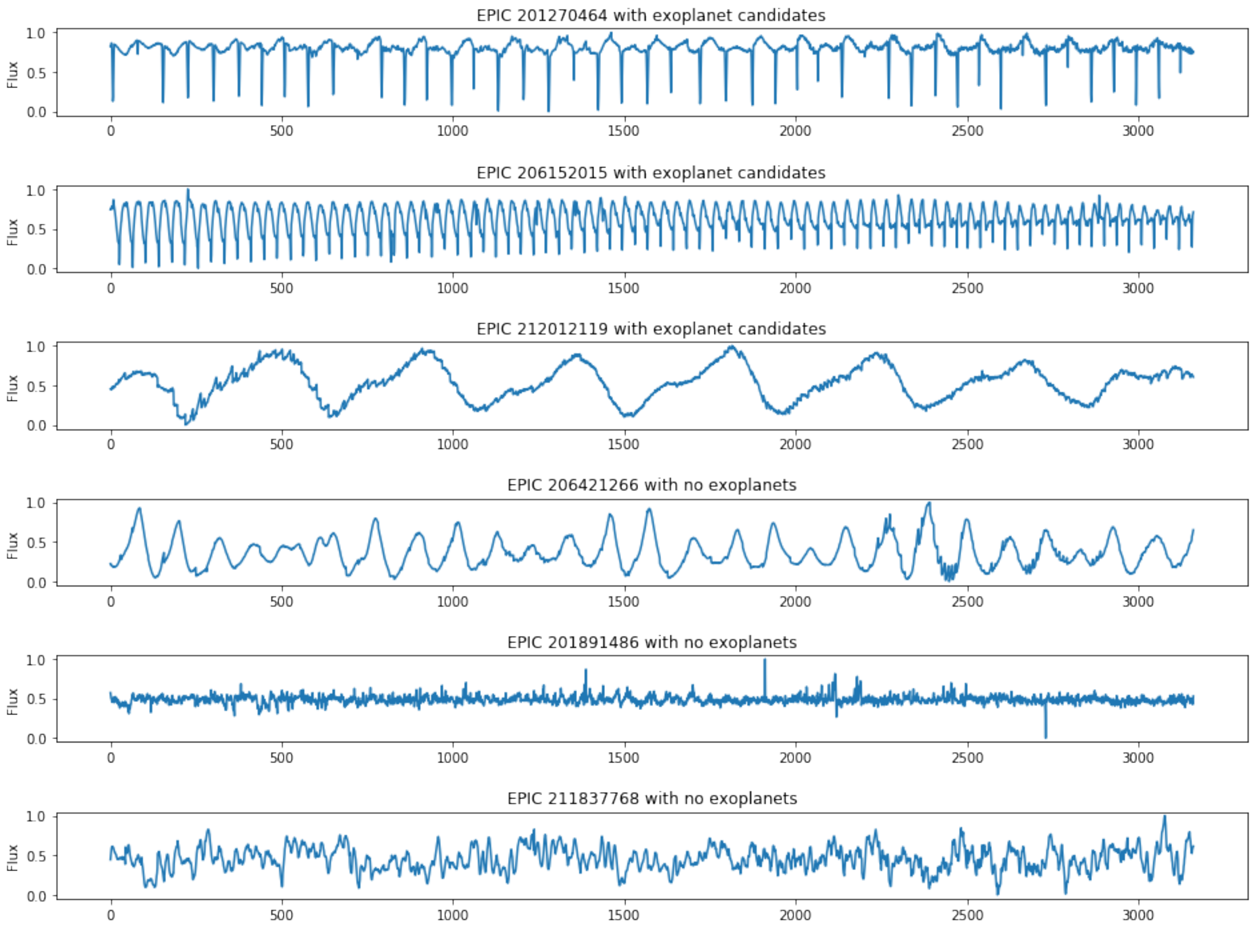


Figure 3: Flux time-series (normalised) visualisation.

It can be seen that the flux series of the 3 stars with exoplanet candidates clearly exhibit a periodic reduction of light intensity as the planet passes between the star and the telescope. However, one could argue that the 4th plot, which corresponds to a non-exoplanet star, also has a similar pattern. This is just one example of the complexity of the problem and the reason why exoplanet researchers need to perform a thorough and time-consuming analysis of the flux time-series before a planet is detected and confirmed. Moreover, the fluxes of the last two non-exoplanet stars also seem to vary or oscillate with time. This leads us to another initial observation that the data is quite noisy, indicating that some preprocessing will be required. Noise can arise from a variety of sources such as instrument noise from the photometry devices on board the Kepler spacecraft, cosmic radiation, as well as intrinsic stellar noise [15]. It is worth mentioning that the chart shows the min-max normalised data rather than the raw flux values. Normalisation is discussed in detail in section 3.1.

2.3 Algorithms and Techniques

As shown above, the flux data are characterised by a significant level of noise. This is why a signal smoothing or filtering technique will form a critical preprocessing step of the solution. A detailed description of the specific smoothing methods used can be found in the Data Preprocessing section below.

The next technique used is known as feature engineering. Rather than feeding all 3,162 datapoints per sample to the classifier and expecting it to learn the key features and temporal structures that distinguish the flux of a star with exoplanets, it is believed that better results may be obtained if a finer set of relevant features is first extracted from the time-series and then used to train the model. For example, some of the basic characteristics of a time-series are statistical measures such as the mean, maximum, minimum, variance, median, etc. of its values.

Ideally, we would want to be able to extract features that are relevant and important for the particular problem. This of course requires some domain knowledge. Fortunately, there is a python library called Feature Analysis for Time Series (FATS), which was developed with a primary focus on analysing and extracting features from astronomical time-series and in particular light-curve data [16]. The package makes it possible to extract a number of features that are currently used by researchers to analyse and classify light curves. As explained in the package's documentation, the number of features that can be extracted depends on the variables in the dataset. In this case, our dataset only has one variable - the magnitude of the flux - which makes it possible to extract 22 features, including some common statistics and some more advanced measures. In alphabetic order these are: Amplitude, Anderson-Darling statistic, Autocorrelation Length, Con Index, Flux percentile ratios (mid20, mid 35, mid 50, mid 65 and mid 80), GSkew, Mean, Mean variance, Median absolute deviation, Median buffer range percentage (MedianBRP), Pair slope trend, Percent amplitude, Percent difference flux percentile, Q_{3-1}, Range of a cumulative sum (R_{cs}), Skewness, Small Kurtosis and Standard deviation. A description of each of the features listed above is provided in the Appendix.

The main part of the solution is obviously the training of a binary classifier, using the 22 features extracted from the time-series. This is going to be a supervised learning algorithm, since all the samples in the dataset have labels specifying whether the associated star has an exoplanet candidate or not. A spot-checking exercise was first undertaken in order to quickly evaluate and compare the performance of various supervised learning algorithms, in their basic hyperparameter configurations. This is described in more detail in the Implementation section. The following paragraphs provide a description of Gradient Boosting - the algorithm that was found to have the best performance and was then further refined to obtain the final set of results. In particular, it was found to perform better than a logistic regression classifier, a decision tree classifier, a set of Support Vector Machine (SVM) classifiers, a Naive Bayes learner as well as the Adaboost, Bagging and Random Forest algorithms. The decision tree, bagging and Random Forest approaches severely overfitted the training data and therefore showed poor predictive performance on the test dataset. Logistic regression and SVMs had a moderate recall score but poor precision, resulting in low F_2 and accuracy scores. The Naive Bayes algorithm seemed to classify almost all the test samples as positive and therefore had a very low accuracy. Finally, Adaboost, which also is a boosting algorithm, had a similar but slightly lower F_2 score than Gradient Boosting.

It is important to try to understand why Gradient Boosting has this superior performance in this particular application. Let us first discuss how the algorithm works. Gradient Boosting belongs to the category of ensemble methods. More specifically, as the name indicates, it is a boosting algorithm. The principal concept of ensemble techniques is that they combine a number of weak learners in order to obtain a stronger learner with superior performance. Weak learners are simple classifiers that perform poorly, but better than random guessing. Many different algorithms can be used for this purpose, but most commonly (and also in this application) decision trees are picked as the weak classifiers.

A decision tree splits the data by first choosing a feature which would result in the best split (i.e. greatest information gain) and then applying a condition on this feature to split the data into two branches. This process can then be repeated using other features until a final split/classification is obtained. A common disadvantage of decision trees is that they tend to easily overfit the training data, especially as they become more and more complex. Ensemble methods such as boosting help to mitigate this issue through the use of a number of shallow decision trees, whose minimal complexity results in high bias but much lower variance. Combining a number of these weak learners improves the bias of the model, while keeping variance at low levels.

The basic characteristic of boosting algorithms compared to other ensemble methods is that they build every new weak learner sequentially. When a new learner is added in each iteration, it is trained putting particular emphasis on the samples that were incorrectly classified by the previous weak classifiers. In gradient boosting in particular, this is done by trying to minimize a differentiable loss function, such as the root mean square error (RMSE), using gradient descent. In each iteration, the coefficients of the new weak learner are fitted by taking small incremental steps towards the direction of the descending gradient of the loss function. The output of this learner is then added to those of the previous learners. Finally, the loss function is updated by comparing the model's prediction to the actual labels. This additive, sequential process gradually minimises the prediction error, resulting in a strong and robust classifier.

Through this approach of putting more effort on the classification of observations that are more difficult to classify, Gradient Boosting manages to achieve a very strong performance on a large set of problems. This is the primary reason why it is considered to be a good option for this particular problem. As discussed above, the stars with exoplanet candidates are the minority in the flux dataset and are therefore naturally the most difficult ones to classify correctly. By putting more weight on these samples while training each successive weak learner, Gradient Boosting methods are able to deal with class imbalance. Furthermore, another advantage of Gradient Boosting is that it is capable of learning complex, non-linear decision boundaries in datasets with a large number of features. Finally, some potential disadvantages that will have to be considered carefully during the implementation of this algorithm are that it can overfit (even if it is less prone to overfitting than decision trees), it may be slow to train and correctly tuning its many hyperparameters can be difficult.

A final technique that will be adopted in order to combat the class imbalance is what is known as over-sampling. The fundamental idea of over-sampling is that if a dataset does not have an adequate number of samples of a given class, one can increase this number by artificially generating more samples of this class that are similar to the existing ones. The most common over-sampling technique - which will be used in this application - is known as Synthetic Minority Over-sampling technique or SMOTE [17]. SMOTE generates a synthetic data point by taking the vector between an existing minority instance and its nearest same-class neighbors, multiplying it by a random number between 0 and 1 and then adding the result to the existing point. This technique will be used in order to enforce an equal ratio of positive (exoplanet candidate) and negative (non-exoplanet) samples in the training dataset. However, it is important that the test set and any cross-validation sets used for parameter tuning remain untouched.

2.4 Benchmark

Before developing a machine learning model, it is important to define a suitable benchmark against which the performance of the model will be compared. Initially, it was attempted to find similar solutions in the literature and use their results as a benchmark. However, as this is still an open and relatively new problem, it has not been easy to find many published papers with clearly stated results using evaluation metrics similar to those that will be used in this project. Moreover, even if available, such results should be first validated rather than being taken for granted. For example, it has been attempted to validate the very promising results claimed by a solution on Kaggle [18]. However, re-shuffling the training and test datasets resulted in an F-score that was considerably inferior to the one that is posted on the website. For these reasons, it was decided to develop a benchmark model from scratch.

A very basic benchmarking approach would be to use a naive classifier that randomly labels samples as positive or negative. However, due to the imbalance of the dataset this would probably have very poor results. For this reason, a slightly more sophisticated but still quite simple model has been developed. In particular, a logistic regression classifier was trained using the raw flux time-series data. 3/4 of the dataset were used to train this simplistic model, which produced the following results when tested on the remaining 1/4 of the data:

Metric	**Value**
F_2 score	1.42%
Recall	1.18%
Accuracy	90.40%

Table 1: Benchmark model results.

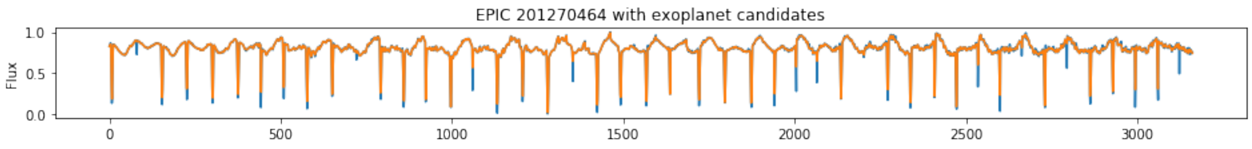
As expected, the performance of this simple approach is not satisfactory. The logistic regression classifier fails to learn the features that make a star likely to have an exoplanet (such as the periodic variation of flux). Almost all the samples are classified as negative - the majority class - and only 1.18% of the exoplanet candidates are detected. While one could argue that these results are too poor to be used as a comparator, the approach used to obtain them is actually representative of the very first experiment that a machine learning engineer would conduct - i.e. a simple model trained using the raw, unprocessed data. Therefore, it is believed that this is a reasonable benchmark model. The utilisation of the various preprocessing steps, feature extraction, over-sampling and the more advanced Gradient Boosting algorithm will aim at surpassing the performance of this classifier and developing a much more robust and useful model.

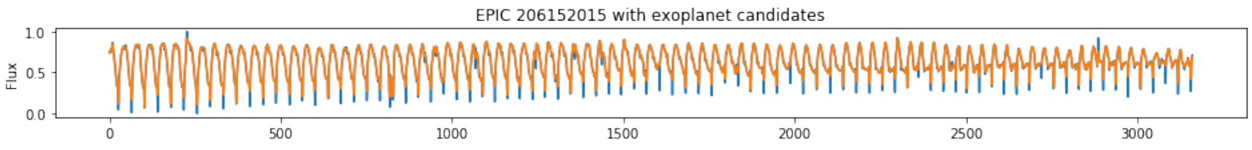
3 Methodology

3.1 Data Preprocessing

As shown in Figure 3 above, the flux data are quite oscillatory and noisy. Noise and outliers can have a detrimental effect on the results of classification algorithms. For this reason, the first pre-processing step that was used is what is known as data smoothing. Smoothing is commonly employed in order to improve the signal-to-noise ratio of time-series data. In simple terms, it works by passing the time-series through a filter which removes noise, outliers and other unwanted information, effectively smoothing the signal.

The smoothing technique used in this application is known as median filtering. The median filter replaces the value of a data point with the median of all the values in a given window around the point. It has been implemented using the *medfilt* function provided in the SciPy library and the default filter window of 3. The effect of applying this filter to the flux data is shown in the following chart, which compares the raw and filtered time-series for the 6 randomly selected samples visualised above.





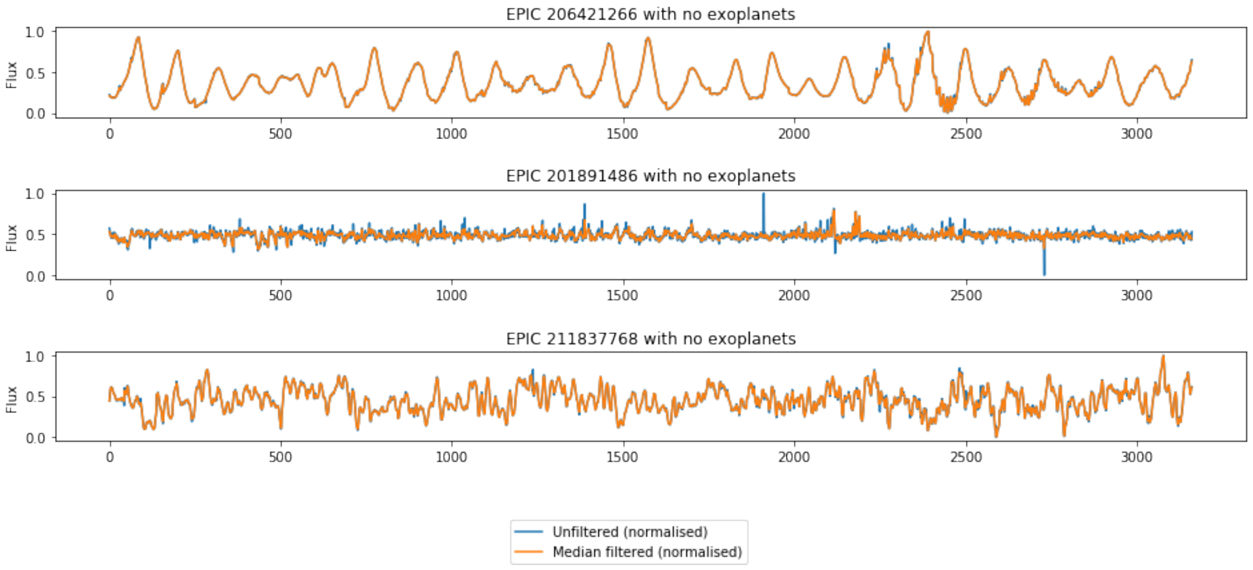


Figure 4: Raw and median filtered time-series (normalised).

It can be seen that the orange lines, which correspond to the median filtered signals, are considerably smoother than the blue, raw flux time-series lines. The median filter seems to have removed a number of large spikes and unwanted noise. This is particularly evident in the second to last graph, but a similar effect is found in the filtered time-series of all the 4,000 samples in the dataset.

The *FATS* package introduced in section 2.3 above, was then used to extract 22 relevant features from the filtered time-series. One of the very first observations made when looking through the flux dataset, was that the range of flux values in the time-series of different stars vary widely - in some cases by as much as several orders of magnitude. As a a result, a large variation is also found in the values of each of the extracted features. This can have a significant negative effect on the performance of some machine learning algorithms, in particular those that are based on calculating distances between sample points. A common pre-processing technique used to mitigate this issue is feature scaling. Its goal is to rescale the values in a dataset, preserving the differences between them. Since a number of different algorithms will be tested, as described in the Implementation section below, it was decided to use feature scaling to make sure that none of them is affected by the large variation of the flux values in the dataset. More specifically, two different techniques have been tested; min-max normalisation and standardisation. The former simply rescales the values of each time-series in the range 0 to 1, through the following operation:

$$\vec{x}_{normalised} = (\vec{x} - \vec{x}_{min})/(\vec{x}_{max} - \vec{x}_{min})$$

where $\vec{x}$ is the set of values of all the samples for a particular feature and $\vec{x}_{max}$ and $\vec{x}_{min}$ are the maximum and minimum values of this set respectively.

Standardisation, on the other hand, results in a new set of values with a mean of 0 and a unit variance, using the following operation:

$$\vec{x}_{standardised} = (\vec{x} - \vec{x}_{mean})/(\vec{x}_{std})$$

where $\vec{x}_{mean}$ and $\vec{x}_{std}$ are the mean and standard deviation of each feature's values respectively.

The two methods were applied separately on each of the 22 extracted flux features. The resulting, rescaled datasets were then both used to train a set of different algorithms in order to determine which one leads to the greatest performance improvement, compared to using the unscaled features' values.

In summary:

1. A median filter was first applied to the raw flux data of each of the stars in the dataset.
2. 22 features were then extracted from each of the filtered time-series using the FATS package.
3. Finally, the values of these features were scaled using two techniques: min-max normalisation and standardisation.

The two resulting datasets were subsequently used for the classification task.

3.2 Implementation

All datasets, from the raw flux time-series to the features obtained after the various pre-processing steps described above, were handled using Pandas DataFrames. The labels were detached from the rest of the data and stored in a separate DataFrame. In all the DataFrames, rows corresponding to each sample were indexed using the stars' names (as documented in the Kepler database). Finally, all the arithmetic operations on the datasets, such as normalisation and standardisation, were performed using the NumPy package.

A spot-checking pipeline was created in order to test a number of supervised learning algorithms, compare their results and determine the most suitable one for this problem. All the algorithms used were taken from the Scikit-learn library [19]. The default settings provided by scikit-learn for each algorithm were used, with just a few exceptions:

- Where available, the *class_weight* parameter was set to 'balanced'. This assigns a larger weight to samples of the minority class during training, which helps in mitigating the effect of class imbalance.
- The *n_estimators* parameter, which controls the number of weak learners used, was set to 50 for all the ensemble methods tested. This was because it was found that all of them produced considerably better results with this value of *n_estimators* rather than the default (=100).

The following table lists all the classifiers that were tested and the parameters used for each one of them. The default values were used for all the parameters that are not mentioned in the table.

Classifier	**Scikit-learn class**	**Parameters**
Logistic regression	LogisticRegression	class_weight='balanced'
Decision Tree	DecisionTreeClassifier	class_weight='balanced'
SVM with linear kernel	SVC	kernel='linear', class_weight='balanced'
SVM with cubic kernel	SVC	kernel='poly', class_weight='balanced'
SVMs with RBF kernel	SVC	kernel='rbf', class_weight='balanced', C=[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1]
Naive Bayes	GaussianNB	-
Bagging	BaggingClassifier	n_estimators=50
Random Forest	RandomForestClassifier	n_estimators=50, class_weight='balanced'
Gradient Boosting	GradientBoostingClassifier	n_estimators=50
AdaBoost	AdaBoostClassifier	n_estimators=50

Table 2: Classifiers tested.

A value of 42 was set as the random seed for all the classifiers, using Scikit-learn's *random_state* parameter, in order to ensure that the same results are obtained every time each algorithm is run.

As mentioned above, a pipeline was created to easily test and compare these algorithms. The first step in the pipeline is to split the dataset and the associated labels into a training and a test subset. This is done using Scikit-learn's *train_test_split* function. The training set consists of 75% of the total samples in the dataset and the remaining 25% forms the test set. The split of samples is done randomly, using a random seed of 42.

The next step involves over-sampling the training dataset, using the SMOTE technique that was discussed above. In particular, this is implemented using the *SMOTE* class of the imbalanced-learn Python library [20]. A *sampling_strategy* equal to 1 is used, meaning that the minority class is re-sampled so that we end up with a training set containing an equal number of samples of the two classes. Once again, a random seed was specified so that results are repeatable.

This over-sampled training dataset and the corresponding labels are then used to train the chosen classifier. This is done using the *fit* method of the classifier's class. Subsequently, the *predict* method is used to predict the label of each sample in the test set using the trained model. This is also done for the first 500 points in the training set, in order to evaluate the classifier's performance on the training data.

The classifier's predictions for both the test set and the subset of the training set are then compared to the actual test and training labels respectively. In this way, the F_2, recall and accuracy scores of the algorithm on the test and training sets are calculated using the *fbeta_score*, *recall_score* and *accuracy_score* classes of Scikit-learn respectively.

The classifiers listed in Table 2 were stored in a Python dictionary and were tested one by one using the pipeline specified above and a simple loop. This was repeated using both the min-max normalised and standardised features extracted from the filtered flux time-series. It was found that training the algorithms using the normalised dataset resulted in a better predictive performance. Hence, it was decided to use this dataset from this point onwards. Comparing the scores of each model, particularly those calculated on the test set, indicated that Gradient Boosting produces the best classification results for this application. In particular, it was found to have an F_2 score of 39.47%, a recall of 52.94% and an accuracy of 77.50% on the test set. These results are much better than those of the benchmark logistic regression model, even prior to further refinement of the classifier. This shows the importance of the various pre-processing and feature engineering steps performed before training the model. The next section describes how the various parameters of the Gradient Boosting algorithm were tuned to improve the classification results further.

3.3 Refinement

After identifying Gradient Boosting as the most suitable algorithm for this application, the next step is to try to tune its hyperparameters in order to refine it. This was done using the *GridSearchCV* class of Scikit-learn. Grid search performs an exhaustive search in the hyperparameter space to find the set of parameters that maximise a specified performance metric. A grid of parameters to be tested is first defined and then *GridSearchCV* builds a model for every possible combination of parameters in this grid. Each time, the chosen metric or score is evaluated by testing the algorithm on a cross-validation set. This is a randomly selected subset of the training set that is left out, so that it is not seen by the model during training. In this case, the evaluation metric was chosen to be the F_2 score, which as explained above is a suitable metric for this application.

Refining the Gradient Boosting algorithm is not straightforward because there is a large set of hyperparameters that need to be tuned. In particular, Gradient Boosting has a number of boosting related parameters as well as a number of decision tree/weak-learner related parameters. The approach used to overcome this problem was splitting the grid search process into several steps. These are described in more detail in the following paragraphs.

Firstly, the boosting related parameters *learning_rate* and *n_estimators* were tuned. The former dictates how big the steps towards the direction of the descending gradient are. In other words, it controls how big the impact of each successive weak learner added to the model is. It is also known as shrinkage. Smaller shrinkage values help towards regularising the model in order to avoid over-fitting. Jerome Friedman, in his "Stochastic Gradient Boosting" paper [21], has stated that "empirically, it was found that small values ($\nu \leq 0.1$) lead to better generalization error". However, it is important to note that the smaller the value gets the slower the training of the model becomes. The *n_estimators* parameter determines how many weak learners are used. Usually, increasing the number of learners improves the predictive performance of the ensemble method, but at some point over-fitting starts becoming an issue. The range of values tested are $[0.0001, 0.001, 0.01, 0.1, 1]$ and $[20, 50, 100, 200, 300, 400, 500]$ for *learning_rate* and *n_estimators* respectively. The best F_2 score was obtained using *learning_rate* $= 0.1$ and *n_estimators* $= 20$. This number of trees is relatively small, but was kept as is at this stage.

It is worth mentioning that a 5-fold stratified cross-validation strategy was used for this and all the following grid searches, as it was found to produce the best results on the test set, as well as on the cross-validation sets. Moreover, the SMOTE algorithm discussed above was used to over-sample the training set (but not the cross-validation set) in all the grid searches. This was achieved by creating an imblearn Pipeline object [22], which was then used as the *estimator* in GridSearchCV.

Fixing the boosting hyperparameters at these values, the next step was to optimise the decision tree related hyperparameters. The most important ones that were considered are the tree's *max_depth*, *min_samples_split*, *min_samples_leaf* and *max_features*. *max_depth* controls the number of nodes of each tree. The higher this number the more complex and thus prone to over-fitting the weak learners become. *min_samples_split* determines the number of samples that are required to be at a node for further splitting to take place. Very high values usually lead to high bias while low values can cause over-fitting. *min_samples_leaf* defines the minimum number of samples required at a leaf node and has a similar effect to *min_samples_split*. Finally, *max_features* defines the number of features considered when looking for the best split. A value of this parameter that is lower than the total number of features reduces over-fitting but increases bias.

Initially, a grid search was performed in order to tune *max_depth* and *min_samples_split*. Values from 3 to 19 in steps of 2 were tested for *max_depth* and from 2 to 200 with steps of 25 for *min_samples_split*. This resulted in an optimal *max_depth* of 3 and a *min_samples_split* of 2, which are the default values provided by Scikit-learn. Fixing these two parameters, a grid search was then performed to tune *min_samples_leaf*. Values from 1 to 81 in steps of 5 were tested and the optimal value was found to be 61 (it was decided to round this to 60). Finally, after fixing this parameter, a last grid search was done in order to tune the *max_features* parameter. The optimal value was found to be 10.

As mentioned above using lower learning rate values results in better robustness and generalisation of the model. Therefore, the Gradient Boosting model was trained and tested a few more times using the optimal decision tree hyperparameters found above and gradually reducing the learning rate. At the same time, the number of weak learners, *n_estimators*, was increased proportionally to the reduction of the learning rate. *learning_rate* $= 0.001$ and *n_estimators* $= 2000$ were found to produce the same results as the model with the higher learning rate and low number of estimators tested above. Despite the fact that these values lead to a longer training time, they were chosen because of the additional robustness that they provide to the model.

4 Results

4.1 Model Evaluation and Validation

The final model is a Gradient Boosting classifier with the following hyperparameters: *learning_rate* = 0.001, *n_estimators* = 2000, *max_depth* = 3, *min_samples_split* = 2, *min_samples_leaf* = 60 and *max_features* = 10. The default values are used for all the other hyperparameters.

The selection of the Gradient Boosting algorithm was discussed extensively in previous sections. It has been compared with a number of other supervised learning methods and was found to perform better on this classification problem. Subsequently, the model was refined using grid search, leading to an improvement of approximately 12% on the test F_2 score and approximately 35% on the test recall score, compared to the unoptimised Gradient Boosting algorithm (with the default hyperparameter settings provided in Scikit-learn).

The final scores of this optimised classifier, when used to predict the labels of the samples in the test set (25% of the dataset), are shown in the following table:

Metric	**Value**
F_2 score	44.54%
Recall	72.94%
Accuracy	68.30%

Table 3: Final model results.

The primary aim of this project was to develop a classifier that can aid in the discovery of exoplanets. Therefore, we would like the final model to be able to detect as many stars with exoplanets as possible. This ability is measured by the recall score, which was found to be equal to 72.94%. This is a particularly good result, as it means that the majority of exoplanets in the test set are detected. Of course it is not ideal, since there is about one quarter of potential exoplanet candidates that are not found by the model. This is definitely one area for improvement, as we would ideally want to find all the exoplanets or exoplanet candidates. On the other hand, a model could have a very high recall score by just classifying most or all of the samples as stars with exoplanets candidates. This is obviously undesirable, since we want the classifier to narrow down the list of stars that need to be manually checked by scientists. Considering the large imbalance in the dataset, the accuracy score of 68.3% shows that this is not the case. Most of the non-exoplanet stars in the test set have actually been correctly classified as negatives.

The F_2 score, which was the main evaluation metric chosen for this application, was found to have a quite average value of 44.54%. This is because while the Gradient Boosting model is successful in detecting most of the stars with exoplanet candidates (True Positives), it also classifies a number of non-exoplanet stars as positives (False Positives). As a result, precision goes down resulting in a mediocre F_2 score.

Before making a conclusion about the quality of the solution, it would be useful to try to get a better understanding of when the model performs well and when it doesn't. This can be achieved by visualising the time-series of a random selection of True Positives (TP), False Positives (FP), True Negatives (TN) and False Negatives (FN).

Firstly, the flux time-series of three randomly selected true positive samples are shown in Figure 5 below. These are stars which were correctly classified by the Gradient Boosting model as having exoplanets or exoplanet candidates. It can be seen that in all three cases there is a periodic reduction of flux, caused by the transiting exoplanet. This is the primary characteristic that we would expect to distinguish a star that has exoplanets from one that doesn't and the classifier seems to do a good job in detecting it, resulting in an above-average recall score, as explained above.

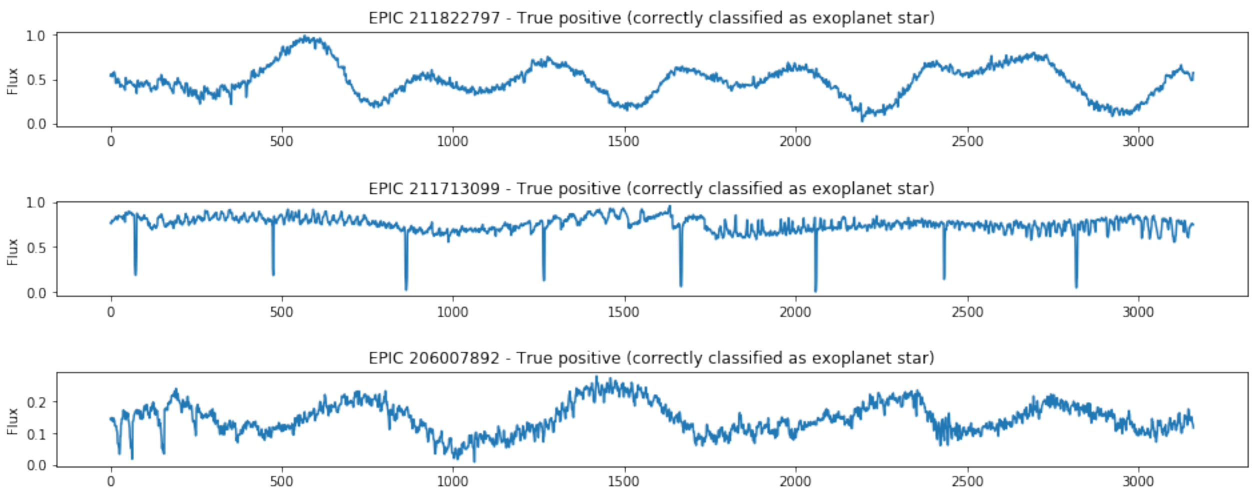


Figure 5: Examples of True Positive samples in the test dataset.

Next, let us visualise the fluxes of some of the stars that have been incorrectly classified as positives. It is important to try to understand why the algorithm thinks that these non-exoplanet stars may have an exoplanet. By looking at the plots of the three randomly selected false positive samples, it can be seen that some of them do actually exhibit a similar pattern to the one seen in the true positive samples discussed above. In particular, the first two time-series (primarily the first one) have a number of regular drops of flux. In contrast, such a pattern is not apparent in the third plot, which is however distorted by a large spike that was not effectively removed by the median filter.

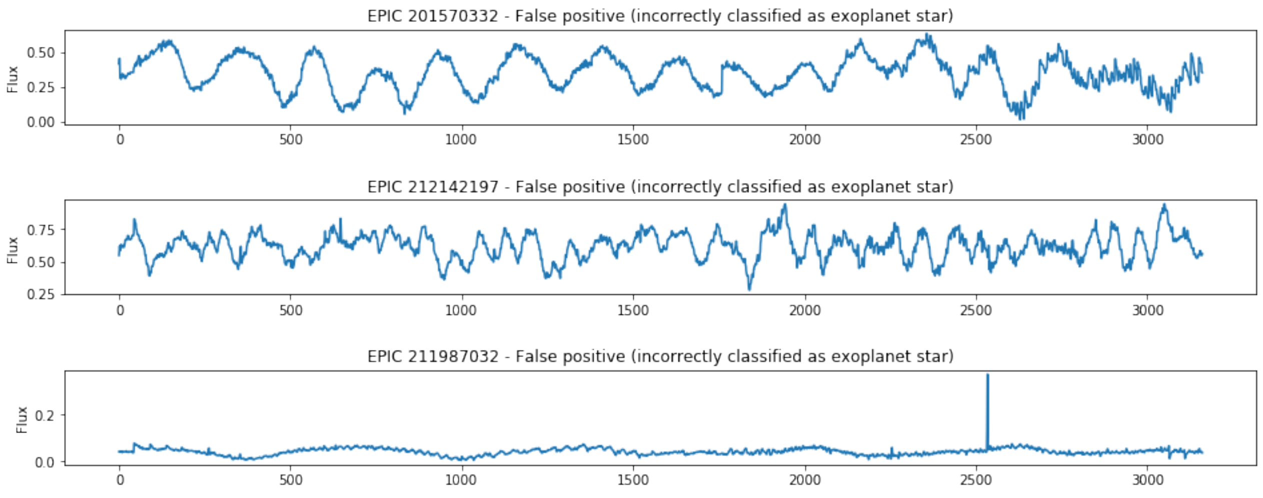


Figure 6: Examples of False Positive samples in the test dataset.

At this point, it should be reminded that the "true" labels for each of the time-series were obtained by checking whether the corresponding stars are included in NASA's "K2 candidates" table. This is a list of all the stars that have been checked by astronomers and were considered to be candidates for exoplanet stars. However, not all of the thousands of stars monitored by the Kepler telescope have been assessed yet. Hence, it is possible that the some stars in our dataset which may be orbited by an exoplanet are not included in the K2 table and have thus been labelled as negatives or 0. For this reason, it is believed that some of the false positives (like the ones displayed below) may actually have exoplanet candidates, but they just haven't been manually classified yet by scientists. This means that some of them may actually be true positives, which would then mean that the classifier may have discovered some new exoplanet candidates.

Figure 7 below displays the time-series of three randomly selected stars that have no exoplanets and were correctly classified by the supervised learning model (True Negatives). None of them seem to exhibit any periodic flux reduction pattern, that would be expected from a star with exoplanets. The Gradient Boosting model has been able to understand this and correctly classify the stars as negatives.

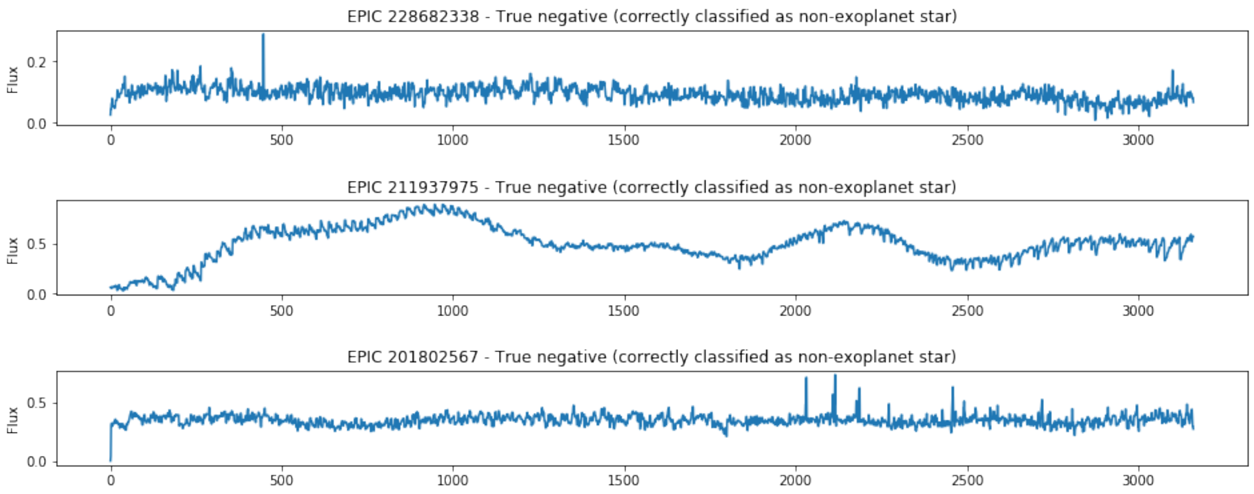


Figure 7: Examples of True Negative samples in the test dataset.

Finally, let us examine some of the cases where the classifier incorrectly predicted that a star does not have an exoplanet candidate, while its true label is positive. As it can be observed from the three plots below, these are cases where there are no clear periodic dimming patterns. It looks like this, in addition to the noise in the data, has not allowed the algorithm to discover the exoplanets that *may* be orbiting these stars. It is important to note that the "K2 candidates" table also includes a number of false positives, as shown in Figure 1. These are stars that were initially considered to be candidates for exoplanets, but were then labelled as 'false positive'. Therefore, some of the samples that the model classified as false negatives may actually be true negatives. In fact, the first star shown below (EPIC 206065006) is classified as a false positive in the K2 table, which means that the classification of our Gradient Boosting model is actually correct. One possible future improvement would be to only assign positive labels to the stars that are classified as 'confirmed' or 'candidate' in the "K2 candidates" table, rather than also including the 'false positive' ones.

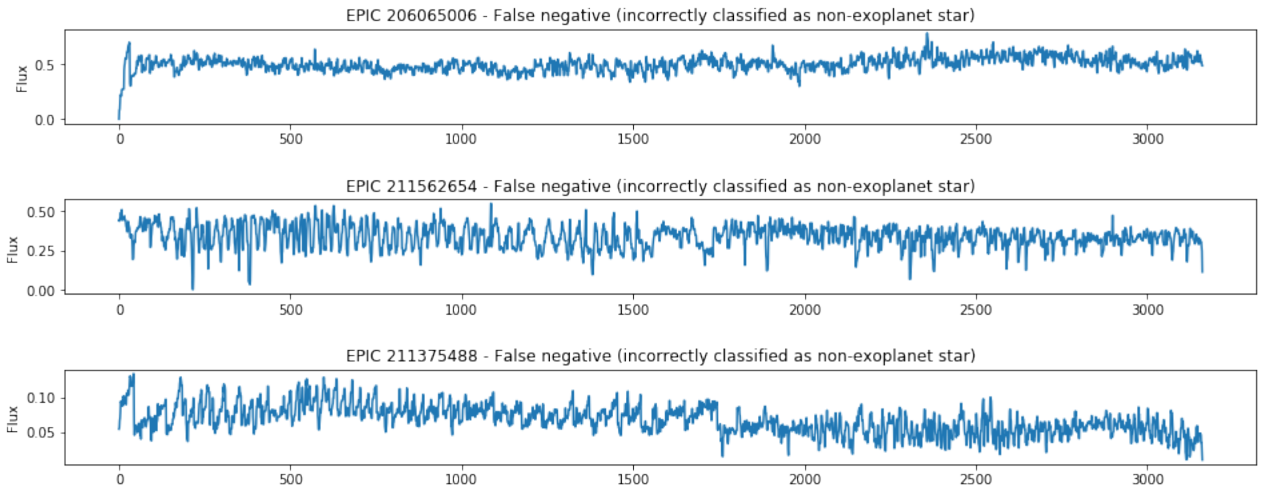


Figure 8: Examples of False Negative samples in the test dataset.

A sensitivity analysis was performed in order to test the robustness of the model. More specifically, the optimised Gradient Boosting model derived above was re-trained and tested using a different mix of training and test data. This was done by changing the random seed of Scikit-learn's *train_test_split* function, which splits the dataset into training and test subsets. Moreover, the random seed of the SMOTE algorithm used to over-sample the training set was also varied at the same time. This was done twice, using two different random seeds. The test results obtained using a random seed of 1 are: F_2 = 41.94%, recall = 74.29% and accuracy = 69.40%. The results obtained using a random seed of 200 are: F_2 = 44.58%, recall = 68.97% and accuracy = 70.80%. In both cases, the results are not very different to the ones shown in Table 3 (random seed = 42). Therefore, it is concluded that the model is robust.

4.2 Justification

The Gradient Boosting model that was created performed significantly better than the logistic regression model that was used as a benchmark. A comparison of the key performance metrics evaluated on the test data using the two models is shown in the following table:

Metric	**Benchmark**	**Gradient Boosting**
F_2 score	1.42%	44.54%
Recall	1.18%	72.94%
Accuracy	90.40%	68.30%

Table 4: Gradient Boosting model vs. benchmark model.

It can be seen that the final model developed performs better on all metrics except accuracy. However, as discussed above, the very high accuracy of the benchmark model is simply because it classifies almost all the samples as negative (non-exoplanet). The most important step that made the final solution so superior compared to the benchmark model is the extraction of features from the flux time-series. Training the classifier using these features rather than the 3,162 flux values of each star resulted in a significant improvement. Then, the rescaling and smoothing of the data as well as the selection and tuning of the Gradient Boosting algorithm resulted in a further refinement of the solution.

Given the limitations of the initial labelling of the dataset discussed above, the final solution is deemed to be satisfactory. The results are by no means perfect and further development and improvement is required. Nevertheless, the Gradient Boosting classifier that was developed managed to learn the main features that characterise the time-series of stars with exoplanets. It was able to detect a large proportion of the stars with exoplanet candidates in the test dataset, while also correctly classifying most of the non-exoplanet stars. As explained above, it is likely that the model detected a number of new exoplanet candidates, which astrophysicists may have not found yet. This would be considered a very fulfilling achievement, given the limited time in which this solution was developed. Some ideas for further improvement are discussed in Section 5.2 below.

5 Conclusion

A comprehensive visualisation of the results is included in Section 4 above, in order to assist the discussion of the performance of the Gradient Boosting classifier. For this reason, it is believed that there is no need of a separate free-form visualisation to be provided here.

5.1 Reflection

The development of the final Gradient Boosting classifier was a result of a long and complex process. Initially, a significant amount of time was spent in order to collect and understand the flux data used for training and testing the algorithm. A Kaggle dataset was first examined and was considered to be unsatisfactory due to its large class imbalance. Hence, a custom dataset was built by downloading individual astronomical files for thousands of stars, directly through NASA's public repository. Some initial editing was then required in order to obtain time-series of equal lengths for all the stars. This process, combined with some under-sampling, resulted in a slightly more balanced dataset of 4,000 samples.

At that point, a simplistic benchmark model was defined, as a reference for comparison of the final model's results. The F_2 score was chosen as the most suitable evaluation metric, taking into consideration the imbalanced nature of the dataset.

A median filter was then used to remove noise and outliers from each of the time-series. Then, a feature extraction package called FATS was used to extract 22 light-curve related features from the filtered flux time-series. Finally, the values of these features were scaled using min-max normalisation.

The next step involved testing a number of classification algorithms, in their basic, unoptimised configurations, in order to determine which is the most suitable for this application. Gradient Boosting was found to produce the best results on the test set, compared to all the other supervised learning algorithms. A grid search approach was subsequently used in order to refine the Gradient Boosting model by tuning its hyperparameters. In particular, grid search was performed in several steps, each time optimising a different set of hyperparameters. This resulted in a final, optimised classification model. A sensitivity test was performed, using small perturbations of the training and test datasets, in order to ensure that the model is robust. Finally, the results of the solution were evaluated.

In general, the approach used to solve this problem was quite comprehensive and all-inclusive, as it included tasks ranging from constructing a suitable dataset to processing the data and finally training and testing an appropriate machine learning algorithm. In fact, building the dataset from the individual NASA files was found to be one of the most difficult aspects of the project. However, the most interesting and challenging part was determining how to extract useful features from the long time-series data and then deciding which algorithm is the most appropriate for this particular application. Given the time constraints and complexity of the open problem of exoplanet detection, the final solution has met all expectations.

5.2 Improvement

While the final solution is satisfactory, there is definitely room for improvement. The approach of extracting features from the time-series and then using them to train a classifier is not the only possible solution and as explained above, depends largely on domain knowledge. Another approach that is worth investigating is developing a deep learning algorithm that is trained directly using the time-series data. A well designed neural network architecture may be able to automatically detect the features that are most relevant and then use these for the classification of the time-series. Convolutional neural networks (CNN), in particular, are likely to be suitable for this application, given their good performance with spatial and temporal data. Moreover, another category of neural networks that could be explored are long short-term memory (LSTM) networks, which have been found to produce good results in classification problems involving time-series data [23]. An attempt to test deep learning methods was made during the later stages of the project, but was abandoned due to time-constraints.

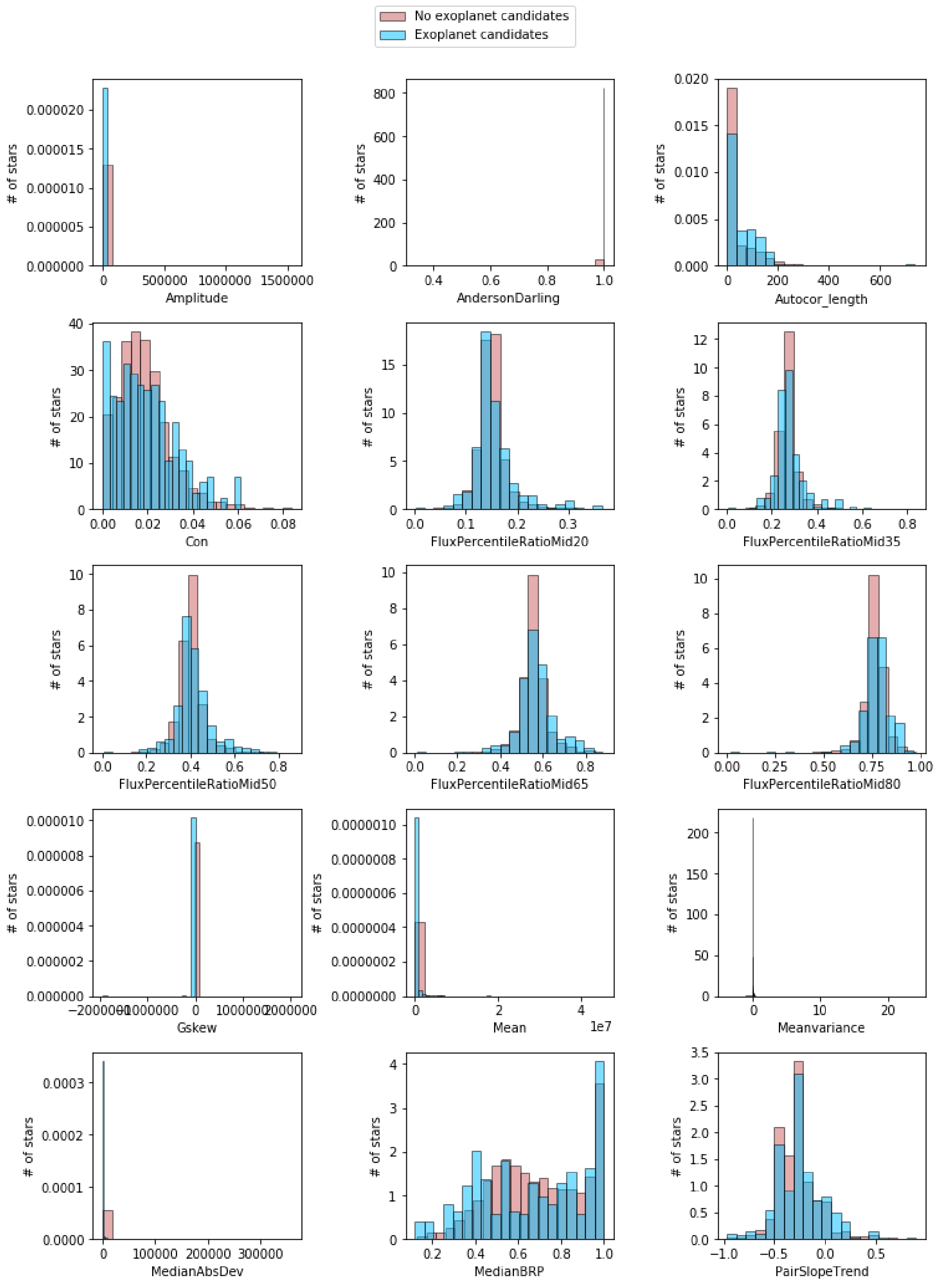
Appendix

The 22 features extracted from the flux time-series using the FATS package are listed in the following table. The description of each feature, as provided in the package's documentation [16], is also included. More information can be found on the FATS website.

Feature	**Feature name (FATS)**	**Description**
Amplitude	*'Amplitude'*	Half of the difference between the median of the maximum 5% and the median of the minimum 5% magnitudes.
Anderson-Darling statistic	*'AndersonDarling'*	A statistical test of whether a given sample of data is drawn from a given probability distribution.
Autocorrelation Length	*'Autocor_length'*	The length of the autocorrelation function, which is the cross-correlation of a signal with itself.
Con Index	*'Con'*	Index introduced for the selection of variable stars from the OGLE database (Wozniak 2000).
Flux percentile ratios (mid20, mid 35, mid 50, mid 65 and mid 80)	*'FluxPercentileRatio Mid20-80'*	Percentiles used to characterise the sorted magnitudes distribution.
GSkew	*'Gskew'*	Median-based measure of the skew.
Mean	*'Mean'*	Mean magnitude.
Mean variance	*'Meanvariance'*	A simple variability index defined as the ratio of the standard deviation, to the mean magnitude.
Median absolute deviation	*'MedianAbsDev'*	The median discrepancy of the data from the median data.
Median buffer range percentage	*'MedianBRP'*	Fraction of photometric points within amplitude/10 of the median magnitude.
Pair slope trend	*'PairSlopeTrend'*	Considering the last 30 (time-sorted) measurements of source magnitude, the fraction of increasing first differences minus the fraction of decreasing first differences.
Percent amplitude	*'PercentAmplitude'*	Largest percentage difference between either the max or min magnitude and the median.
Percent difference flux percentile	*'PercentDifference FluxPercentile'*	Ratio of the difference between 95% and 5% magnitude values over the median magnitude.
Q_{3-1}	*'Q31'*	The difference between the third quartile, Q3 , and the first quartile, Q1 , of a raw light curve.
Range of a cumulative sum	*'Rcs'*	The range of a cumulative sum (Ellaway 1978) of each light-curve.
Skewness	*'Skew'*	Skewness.
Small Kurtosis	*'SmallKurtosis'*	The small sample kurtosis of the magnitudes.
Standard deviation	*'Std'*	Standard deviation.

Table 5: FATS features.

The distribution of each of these features calculated for positive and negative samples in the dataset are shown in the following density histograms:



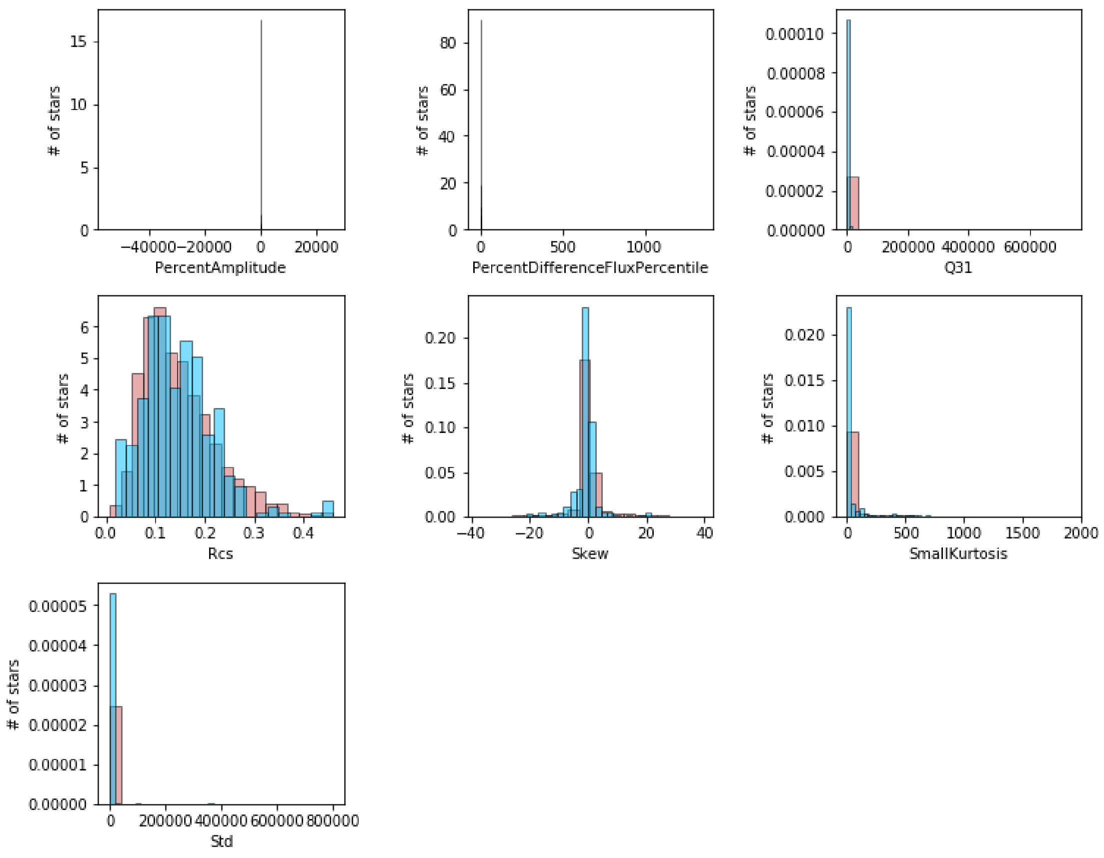


Figure 9: Distributions of flux features.

References

[1] European Space Agency. How many stars are there in the Universe?. [online] Available at: https://www.esa.int/Our_Activities/Space_Science/Herschel/How_many_stars_are_there_in_the_Universe [Accessed 1 Apr. 2019].

[2] What is the Radial Velocity Method? - Universe Today. [online] Available at: https://www.universetoday.com/138014/radial-velocity-method/ [Accessed 24 Feb. 2019].

[3] NASA. Kepler and K2 - Mission overview. [online] Available at: https://www.nasa.gov/mission_pages/kepler/overview/index.html [Accessed 24 Feb. 2019].

[4] Exoplanet Archive Planet Counts. [online] Available at: https://exoplanetarchive.ipac.caltech.edu/docs/counts_detail.html [Accessed 24 Feb. 2019].

[5] A Spacecraft's Second Life: NASA's K2 Mission. [online] Available at: https://www.nasa.gov/feature/ames/nasas-k2-mission-the-kepler-space-telescopes-second-chance-to-shine [Accessed 1 Apr. 2019].

[6] NASA Exoplanet Archive. [online] Available at: https://exoplanetarchive.ipac.caltech.edu/ [Accessed 1 Apr. 2019].

[7] Jenkins, J.M., Tenenbaum, P., Seader, S., Burke, C.J., McCauliff, S.D., Smith, J.C., Twicken, J.D. and Chandrasekaran, H., 2017. Kepler Data Processing Handbook: Transiting Planet Search. Kepler Science Document, KSCI-19081-002

[8] Kepler Objects of Interest. [online] Available at: https://exoplanetarchive.ipac.caltech.edu/cgi-bin/TblView/nph-tblView?app=ExoTbls&config=cumulative [Accessed 24 Feb. 2019].

[9] K2 Candidates. [online] Available at: https://exoplanetarchive.ipac.caltech.edu/cgi-bin/TblView/nph-tblView?app=ExoTbls&config=k2candidates [Accessed 1 Apr. 2019].

[10] Bryson, S. T., Abdul-Masih, M., Batalha, N., et al. 2015, The Kepler Certified False Positive Table, KSCI-19093-002

[11] Kaggle. Exoplanet Hunting in Deep Space. [online] Available at: https://www.kaggle.com/keplersmachines/kepler-labelled-time-series-data [Accessed 24 Feb. 2019].

[12] K2 Campaign fields. [online] Available at: https://keplerscience.arc.nasa.gov/k2-fields.html [Accessed 3 Apr. 2019].

[13] Two Ways to Get Kepler Light Curves. [online] Available at: https://www.nasa.gov/kepler/education/getlightcurves [Accessed 7 Apr. 2019].

[14] Astropy Documentation — Astropy v3.1.2. [online] Available at: http://docs.astropy.org/en/stable/index.html [Accessed 6 Apr. 2019].

[15] Gilliland, R. L., Chaplin, W. J., Dunham, E. W., et al. (2011). Kepler mission stellar and instrument noise properties. The Astrophysical Journal Supplement Series, 197(1), 6.

[16] Feature Analysis for Time Series. [online] Available at: http://isadoranun.github.io/tsfeat/FeaturesDocumentation.html [Accessed 4 Apr. 2019].

[17] Chawla, N.V., Bowyer, K.W., Hall, L.O. and Kegelmeyer, W.P., 2002. SMOTE: synthetic minority over-sampling technique. Journal of artificial intelligence research, 16, pp.321-357.

[18] Kaggle. XGBoost for exoplanet detection, F1 = 0.88. [online] Available at: `https://www.kaggle.com/jfcgon/xgboost-for-exoplanet-detection-f1-0-88` [Accessed 24 Feb. 2019].

[19] scikit-learn: machine learning in Python. [online] Available at: `https://scikit-learn.org/stable/` [Accessed 6 Apr. 2019].

[20] imblearn.over_sampling.SMOTE. [online] Available at: `https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.over_sampling.SMOTE.html` [Accessed 6 Apr. 2019].

[21] Friedman, J.H., 2002. Stochastic gradient boosting. Computational statistics & data analysis, 38(4), pp.367-378.

[22] imblearn.pipeline.Pipeline. [online] Available at: `https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.pipeline.Pipeline.html` [Accessed 7 Apr. 2019].

[23] Karim, F., Majumdar, S., Darabi, H. and Chen, S., 2018. LSTM fully convolutional networks for time series classification. IEEE Access, 6, pp.1662-1669.