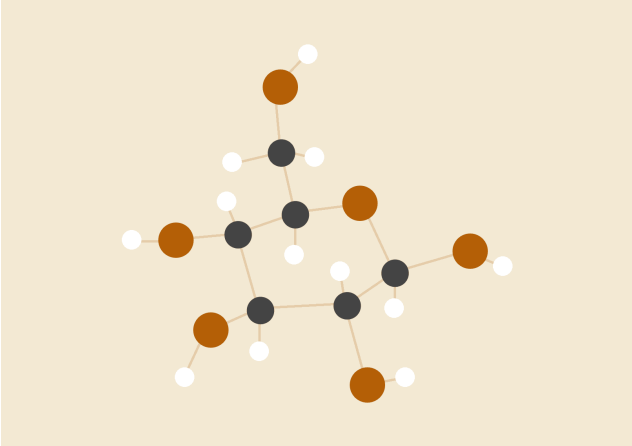
PROJECT REPORT

Detecting Exoplanets using Machine Learning



Harsh Mishra
Roll number- 19049
3rd year BS-MS

CONTENTS

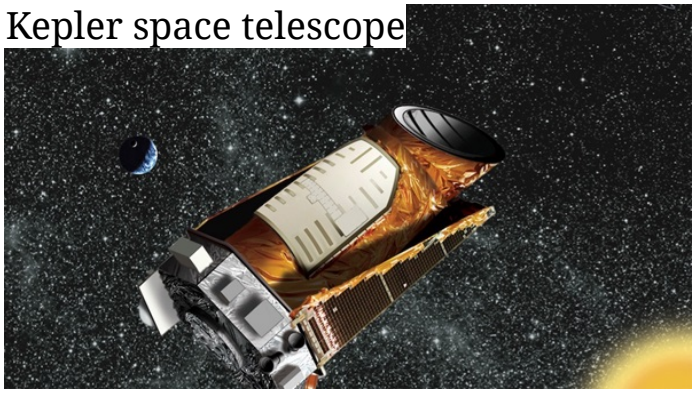
ABSTRACT

The Kepler space telescope is a NASA-launched space telescope that was designed to discover Earth-size planets orbiting other stars. It uses a photometer that tracks the brightness of about 150,000 main sequence stars in a fixed field of view. These data are sent to Earth and evaluated to see if exoplanets that pass in front of their host star cause periodic dimming.It has been collecting light from thousands of stars for many years to detect the presence of exoplanets. In fact, it examined 530,506 stars and discovered 2,662 planets throughout its nine-and-a-half-year mission.

In this project,I have used the dataset which comes from the Mikulski Archive, a large archive of astronomical data for classifying the light curve of the stars to check the presence of the exoplanets. After performing data exploration, I did model comparison using feature engineering and normalising data techniques. Then, I compared it with advanced techniques like Neural networks, CNN and SVC models.

As brightness is a standard in various applications, these prediction models might be beneficial in future research with greater datasets.

Kepler space telescope



PROBLEM STATEMENT

Our problem statement is to identify Exoplanets based on flux values with high accuracy. The data shows how the flux (light intensity) of thousands of stars changed over time as seen by the Kepler satellite telescope. A binary label of 1 or 2 is assigned to each star. The marking 2 indicates that at least one exoplanet has been verified in orbit around the star; some observations are multi-planet systems. Our task is to apply machine learning techniques to this dataset and analyse if its helpful on detection of exoplanets or not.

BACKGROUND

Our solar system's planets all revolve around the Sun. *Exoplanets* are planets that orbit around other stars. They are extremely difficult to observe directly with telescopes. The intense glare of the stars they orbit obscures them. Planets do not emit light, but the stars around which they orbit do. If a star is observed over several months or years, there may be a regular 'dimming' of the flux (light intensity). This is known as 'transiting,' and the absolute percentage of light dimmed is astronomically small - on the order of measuring the decrease in flux from a lighthouse when a fly transits, observed from many miles away. This flux decrease indicates that the star may have an orbiting body; such a star might be classified as a 'candidate' system. Further investigation of our candidate system, may strengthen the view to the status of 'confirmed' as possessing exoplanents.

ACTION PLAN FOLLOWED-

1. Data Visualization- Visualising classes and checking data imbalance.
2. If Data imbalance- Go to preprocessing steps
3. Preprocessing-
    - Scaling
    - Normalising data
    - Applying filter like gaussian, fourier to make the data smooth
    - Oversampling/undersampling as per the requirement
4. Applying machine learning and neural network to preprocessed data.
5. Evaluating the models with the help of metrics like precision, recall, F1 etc.
6. Interpreting the results and possible future works.

DATASET DESCRIPTION

The dataset from this project has been taken from kaggle website: https://www.kaggle.com/keplersmachines/kepler-labelled-time-series-data.

The dataset comprises 2 csv files namely: exoTest.csv(testset) and exoTrain.csv(trainset). Every row represents a star and contains a time series data about its brightness. Thus each column represents the brightness of a star at a specific time. All the general information about the CSV can be obtained using general code df.info() df.describe(), where df is the stored data set.

Trainset:

	LABEL	FLUX.1	FLUX.2	FLUX.3	FLUX.4	FLUX.5	FLUX.6	FLUX.7	FLUX.8	FLUX.9	...	FLUX.3188	FLUX.3189	FLUX.3190	FLUX.3191
0	2	93.85	83.81	20.10	-26.98	-39.56	-124.71	-135.18	-96.27	-79.89	...	-78.07	-102.15	-102.15	25.13
1	2	-38.88	-33.83	-58.54	-40.09	-79.31	-72.81	-86.55	-85.33	-83.97	...	-3.28	-32.21	-32.21	-24.89
2	2	532.64	535.92	513.73	496.92	456.45	466.00	464.50	486.39	436.56	...	-71.69	13.31	13.31	-29.89
3	2	326.52	347.39	302.35	298.13	317.74	312.70	322.33	311.31	312.42	...	5.71	-3.73	-3.73	30.05
4	2	-1107.21	-1112.59	-1118.95	-1095.10	-1057.55	-1034.48	-998.34	-1022.71	-989.57	...	-594.37	-401.66	-401.66	-357.24

5 rows × 3198 columns

- 5087 rows or observations and 3198 columns or features.
- Column 1 is the label vector. Columns 2 - 3198 are the flux values over time.
- 37 confirmed exoplanet-stars and 5050 non-exoplanet-stars.

Testset:

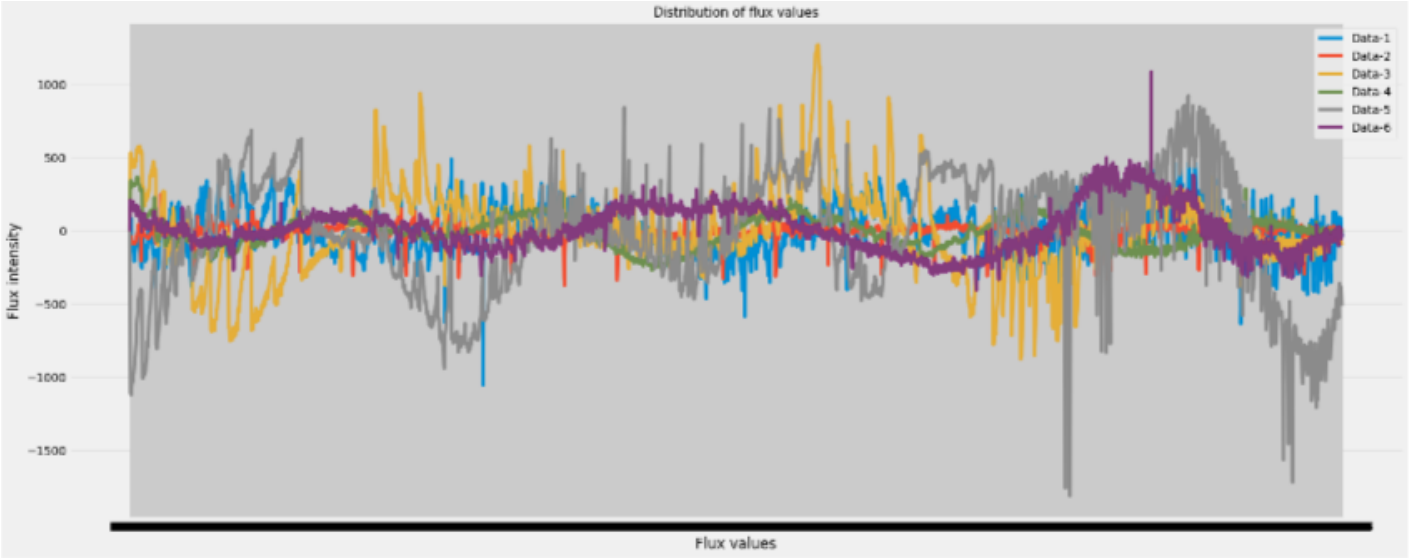
	LABEL	FLUX.1	FLUX.2	FLUX.3	FLUX.4	FLUX.5	FLUX.6	FLUX.7	FLUX.8	FLUX.9	...	FLUX.3188	FLUX.3189	FLUX.3190	FLUX.3191
0	2	119.88	100.21	86.46	48.68	46.12	39.39	18.57	6.98	6.63	...	14.52	19.29	14.44	-1.62
1	2	5736.59	5699.98	5717.16	5692.73	5663.83	5631.16	5626.39	5569.47	5550.44	...	-581.91	-984.09	-1230.89	-1600.45
2	2	844.48	817.49	770.07	675.01	605.52	499.45	440.77	362.95	207.27	...	17.82	-51.66	-48.29	-59.99
3	2	-826.00	-827.31	-846.12	-836.03	-745.50	-784.69	-791.22	-746.50	-709.53	...	122.34	93.03	93.03	68.81
4	2	-39.57	-15.88	-9.16	-6.37	-16.13	-24.05	-0.90	-45.20	-5.04	...	-37.87	-61.85	-27.15	-21.18

5 rows × 3198 columns

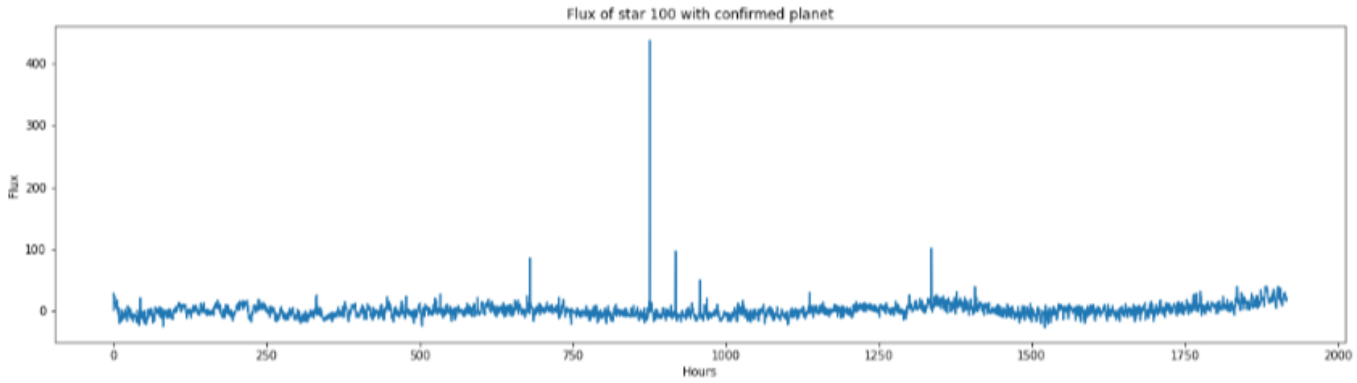
- 570 rows or observations and 3198 columns or features.
- Column 1 is the label vector. Columns 2 - 3198 are the flux values over time.
- 5 confirmed exoplanet-stars and 565 non-exoplanet-stars.

DATA VISUALIZATION

Before going further, let's perform some data exploration techniques to visualise the dataset with the help of matplotlib and python. Below, I have considered the first 6 records of the train set.



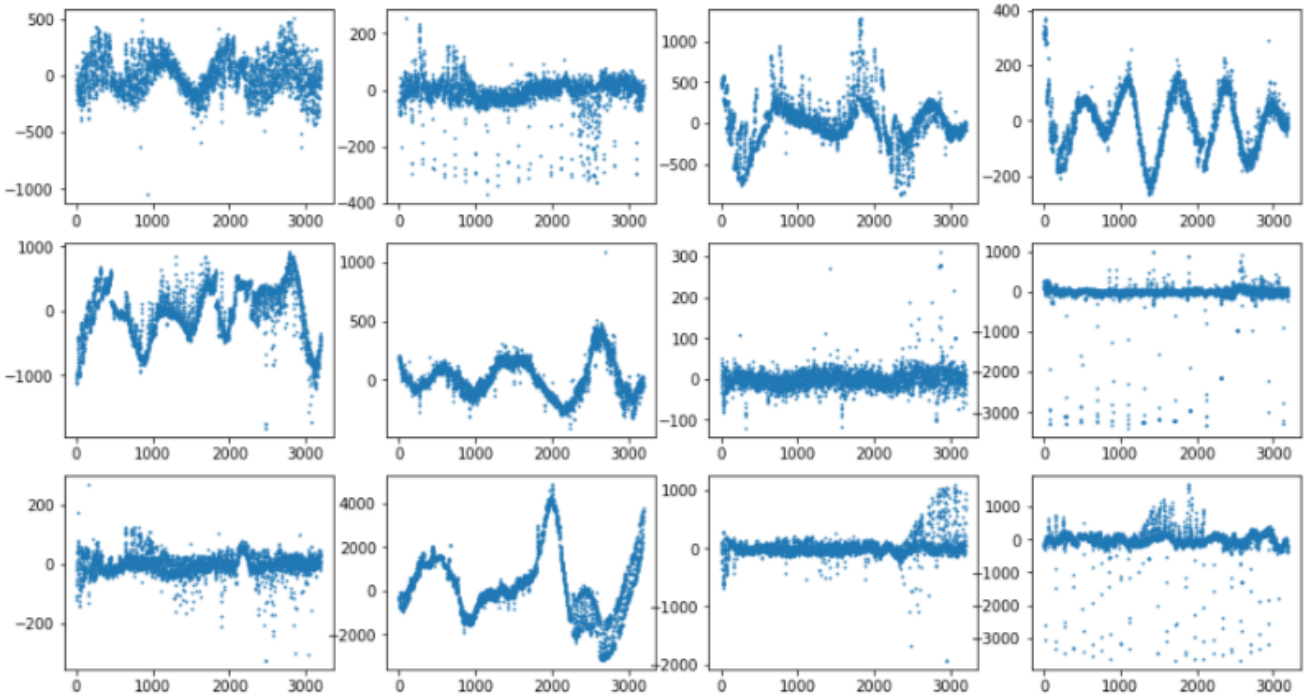
We can see the distribution of flux values here in different colours with the legends telling us about which colour corresponds to which data record. This helps us to interpret the different variation in flux values in our time series data. Now, let us plot for a specific star(star 100) with a confirmed planet. A spike can be seen in flux at a certain point.



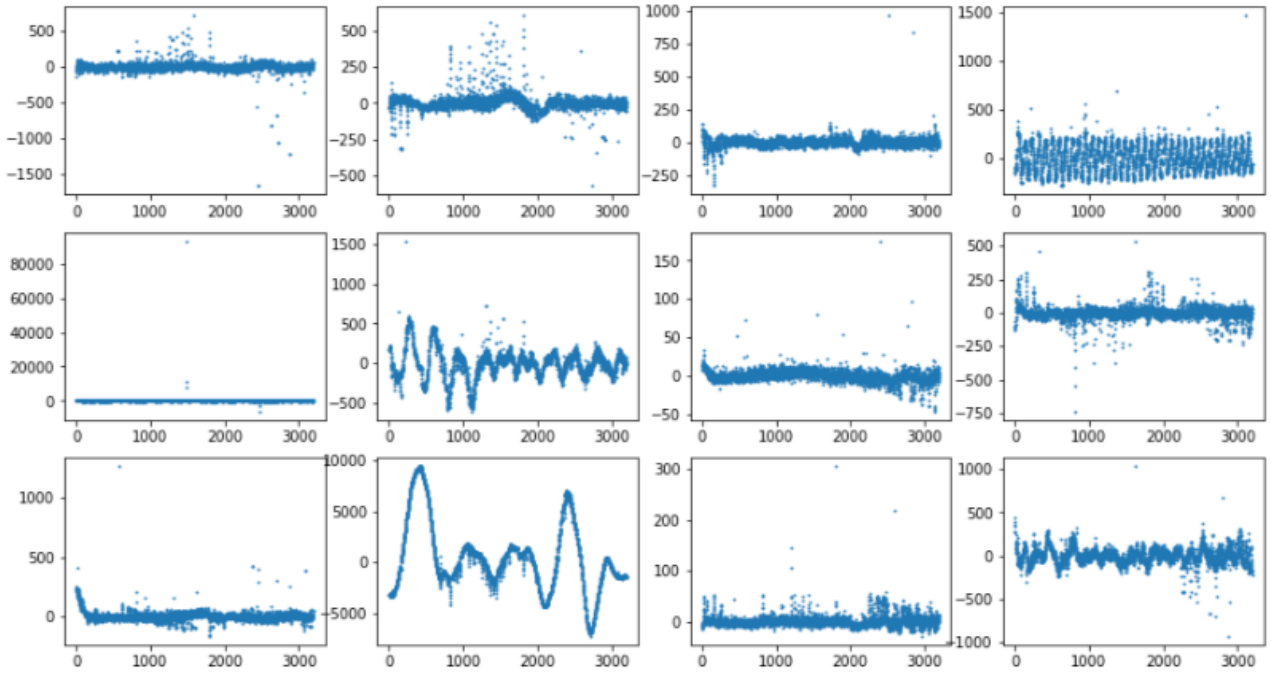
In the subsequent section, I will show the scatter plot and histograms for stars with and without exoplanets, so that we get to know what machine learning techniques should be used.

Scatter plots are used to visualise the relationship between variables, with dots representing the association. They are widely used to represent relation among variables and how change in one affects the other.I used the matplotlib library's scatter() function to create scatter plots.

STARS WITH EXOPLANETS- By setting train['LABEL']==2

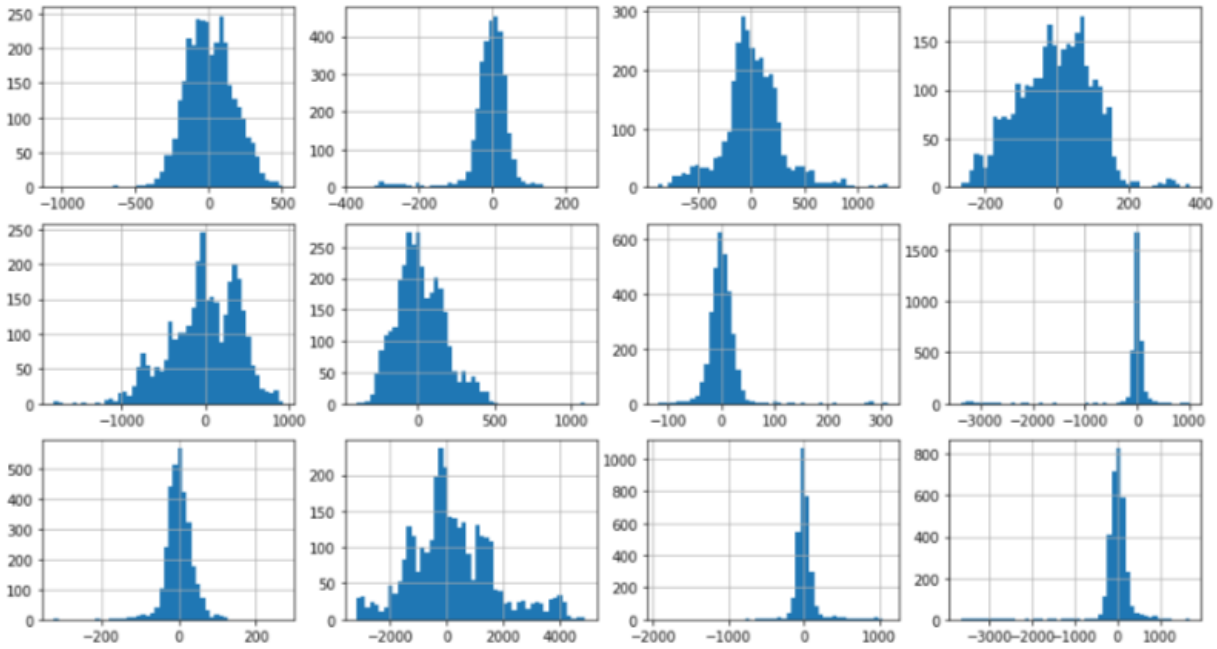


STARS WITHOUT EXOPLANETS- By setting train['LABEL']==1

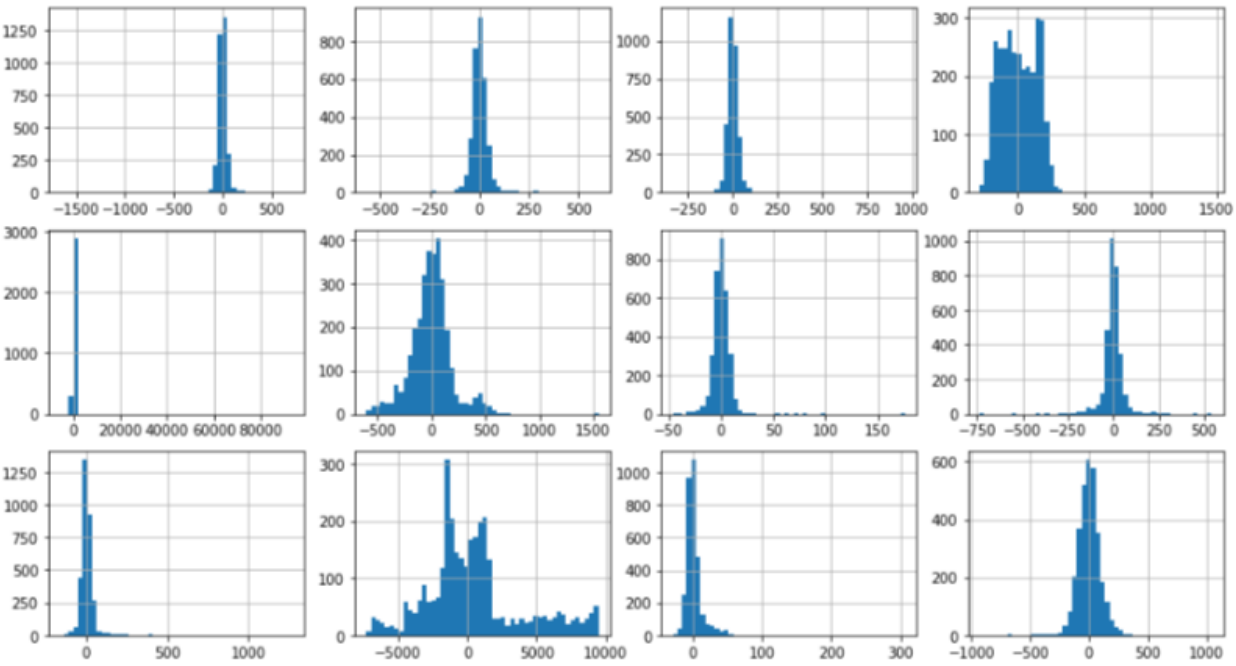


A *histogram* is a visual representation of data presented in the form of groupings. It is a precise approach for displaying numerical data distribution graphically. It's a type of bar plot in which the X-axis shows bin ranges and the Y-axis represents frequency. The capacity of a histogram to detect local regions of greater occurrence is influenced by the bin width (and consequently the number of categories or ranges). I used the hist() function and set the number of bins to 50.

STARS WITH EXOPLANETS- By setting train['LABEL']==2

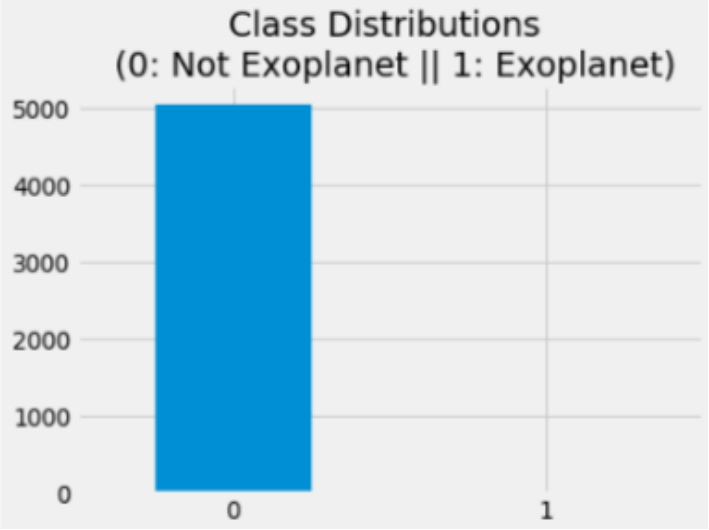


STARS WITHOUT EXOPLANETS- By setting train['LABEL']==1



DATA PRE-PROCESSING

During data visualization, we were looking for U-shaped curves in the signal. The dataset doesn't seems to have any null or missing values.As we can see, this is a large data set to work with or put straight into a model, therefore we'll pre-process the data using various approaches like feature engineering, scaling, normalisation, fourier transform, over sampling, and so on, which we'll go through one by one. Let's check if there is imbalance in data or not -



This plot shows that the data is highly imbalanced. Running any algorithm will result in an accuracy measure that is highly influenced by dominating class. The algorithm won't have ample datasets to learn the patterns and we would be having a bad model showing every data is for exoplanet.

UNDERSAMPLING AND OVERSAMPLING-

Undersampling refers to deleting the samples from the majority classes while oversampling refers to deletion from minority classes. In our code we have used oversampling techniques because we know we have very few minority class data and so to solve this problem of data imbalance, we used the technique of oversampling known as SMOTE. SMOTE stands for Synthetic Minority Oversampling Technique. Its goal is to achieve a more balanced distribution of classes by recreating minority class examples at random.

SMOTE creates new minority instances by combining existing minority classes. For the minority class, it uses linear interpolation to create virtual training records. For each example in the minority class, these synthetic training records are constructed by randomly picking one or more of the k-nearest neighbours. The data is rebuilt after the oversampling procedure, and many classification models can be applied to the processed data.

In my code, I am oversampling my training dataset using SMOTE. Then 30% of the data is extracted and merged with testing data. Now the resulting sample contains both artificial samples as well as the original sample. After splitting the train and test into 4 and applying smote we have,

```
Size of train dataset =  (5087, 3198)
Size of test dataset =  (570, 3198)
After oversampling
Size of X-train dataset =  (7070, 3197)
Size of X-test dataset =  (3600, 3197)
Size of y-train dataset =  (7070,)
Size of y-test dataset =  (3600,)
```

Hence, using this technique of SMOTE we can see out data sets are now oversampled and minority and majority class and instances are comparable now. The synthetic points are added between the chosen point and its neighbours.

For the implementation of neural networks at a later stage I tried oversampling with another method. I used the *RandomOverSampler* class of imblearn to perform random over-sampling. Here also I set the argument sampling_strategy to 0.3 which would ensure that the minority class was oversampled to have 30% the number of examples as the majority class, for binary classification problems.

```
After oversampling, counts of label '1': 1515
After oversampling, counts of label '0': 5050
```

I have implemented feature engineering for better interpretation of the dataset. The following are the filters I used-

SPLITTING AND NORMALIZATION-

As seen earlier label 0 and label 1 had very high difference. So, I splitted the data into train data sets and test data sets (X_train, X_test, y_train, y_test). This is because our algorithm or model recognises the train data and derives a pattern from it, which is then cross-validated with the test data for improved efficiency, and knowing the pattern allows it to predict the new batch of data.

Firstly I rescaled the data by converting floating-point feature values from their natural range (for example, 100 to 900) into a standard range—usually 0 and 1 (or sometimes -1 to +1).I used the below formula for rescaling-

$$x' = (x - x_{min})/(x_{max} - x_{min})$$

Then after the data split, I normalized the data. Normalization is the process of transforming characteristics into a scale that is comparable. This enhances the model's performance and training stability. For normalisation, I used normalize() from sklearn to scale input vectors individually to unit norm (vector length).

GAUSSIAN-

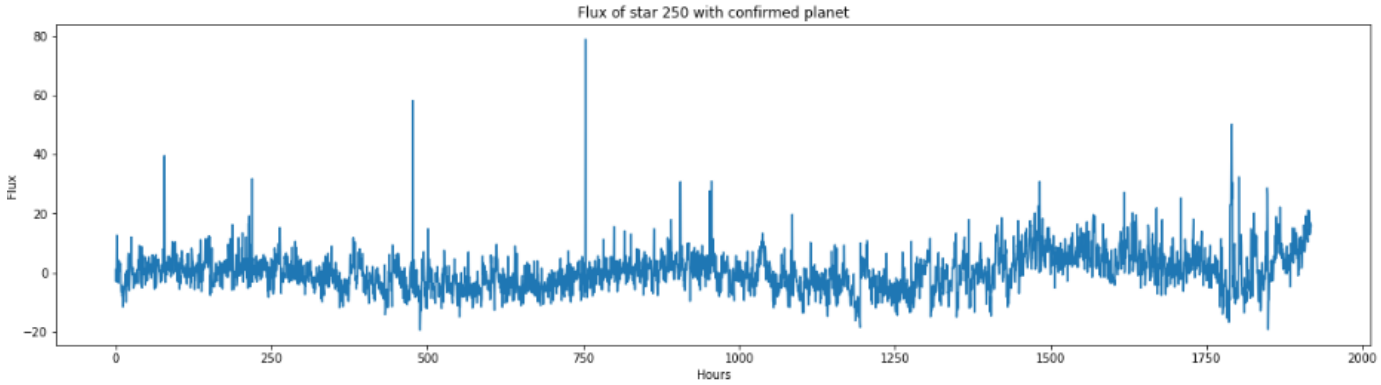
The "LABEL" column has 2 or 1 as values. 2 indicates that there is at least one exoplanet in orbit around the star. Instead of 1 and 2, having values 0 and 1 is preferable. I have used python MatplotLib to plot the light curves of a few stars. As it is typical in the field of Digital Signal Processing to apply to signals (continuous or discrete) filters that enable to clean, eliminate noise, or amplify signals, gaussian filter is applied to the data. I used a gaussian filter for pre-processing step of neural networks and not usual ML models as I already was considering normalised,scaled and fourier there and the results were contradictory for some models. The gaussian filter is a very valuable tool for smoothing out signal sources and reducing noise. It's fair to believe that in this case, the filter will help to make the light curve more even and less irregular. However, because this filter has the potential to exclude vital information from the signal, it is critical that it be well-designed.

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}$$

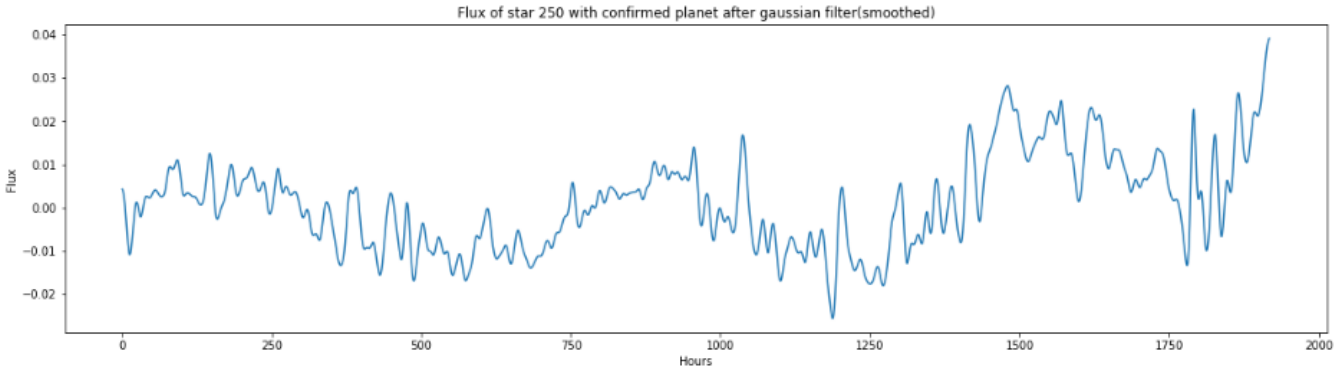
The values of the parameters are chosen such that the data is smoothed. I used the gaussian_filter function from scipy to implement this.

FOURIER-

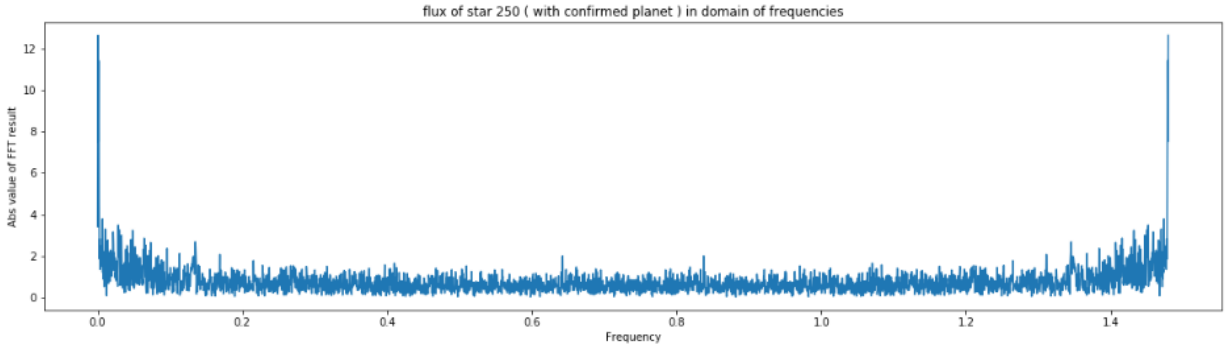
In Digital Signal Processing it is common to switch to the frequency domain of a signal to see the components of the signal. This is possible in our case because it is reasonable to believe that a transiting planet alters the stellar flow frequency; another benefit is that we go from a time-dependent signal to a time-independent sequence of numbers. So, after a gaussian filter, I performed a Fourier transform which helps to switch to frequency domain. This was implemented using fft from scipy.fft . Let’s try to understand these preprocessing steps with the help of plots.



At random, I have chosen star 250 for the flux distribution. If we apply the gaussian filter to the previous plot we observe that the curve has smoothened and we get the plot as below-



As discussed before, now we apply fourier transform to switch to domain of frequencies and we get the plot as below-



APPLYING MACHINE LEARNING MODELS

Now we are ready to implement machine learning on our pre-processed data but before going to that let's discuss the various evaluation metrics we will be using to compare different models-

1. **Precision**- It is the number of correct positive results divided by the number of positive results predicted by the classifier. Less value indicates high false positive.
2. **Recall or Sensitivity**- It is calculated by dividing the number of valid positive results by the total number of relevant samples (all samples that should have been identified as positive).
3. **Area Under the ROC curve (AUC – ROC)** - I have used ROC(Receiver Operator Characteristic) in this project. Measuring the area under the ROC curve is also a very useful method for evaluating a model. By plotting the true positive rate (sensitivity) versus the false-positive rate (1 — specificity), we get the ROC curve. This curve allows us to visualize the trade-off between the true positive rate and the false positive rate.
4. **F1 Score-** It is used to measure a test's accuracy and is calculated as the harmonic mean of the precision and recall and is calculated as-

$$F_1 = 2 \times \frac{precision \times recall}{precision + recall}$$

5. **Confusion Matrix** - It is not an actual metric but is helpful in describing the performance of a classification model as it is a visual representation of the Actual VS Predicted values consisting of positive , negative , false positive and false negative results.

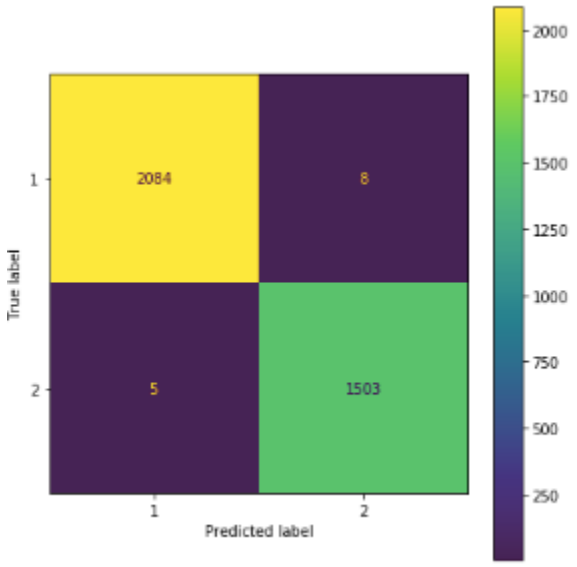
So we will use the above metrics for model comparison.

LOGISTIC REGRESSION

The first model which I used is logistic regression. Logistic Regression is a Machine Learning technique for resolving classification issues. It's a type of predictive analytic approach based on the probability concept. It is used to predict the likelihood of a categorical dependent variable. The dependent variable in logistic regression is a binary variable with data coded as 1 or 0. Logistic regression is a special type of linear regression where it uses Sigmoid function instead of any linear function where the range vary from [0,1]. This function maps any real value into another variable having value between 0 & 1.

I used the LogisticRegression() function of sklearn.linear_model package for implementation of this algorithm which would return a set of outputs or classes based on probability when we pass the inputs through a prediction function and return a probability score in the range 0 to 1.

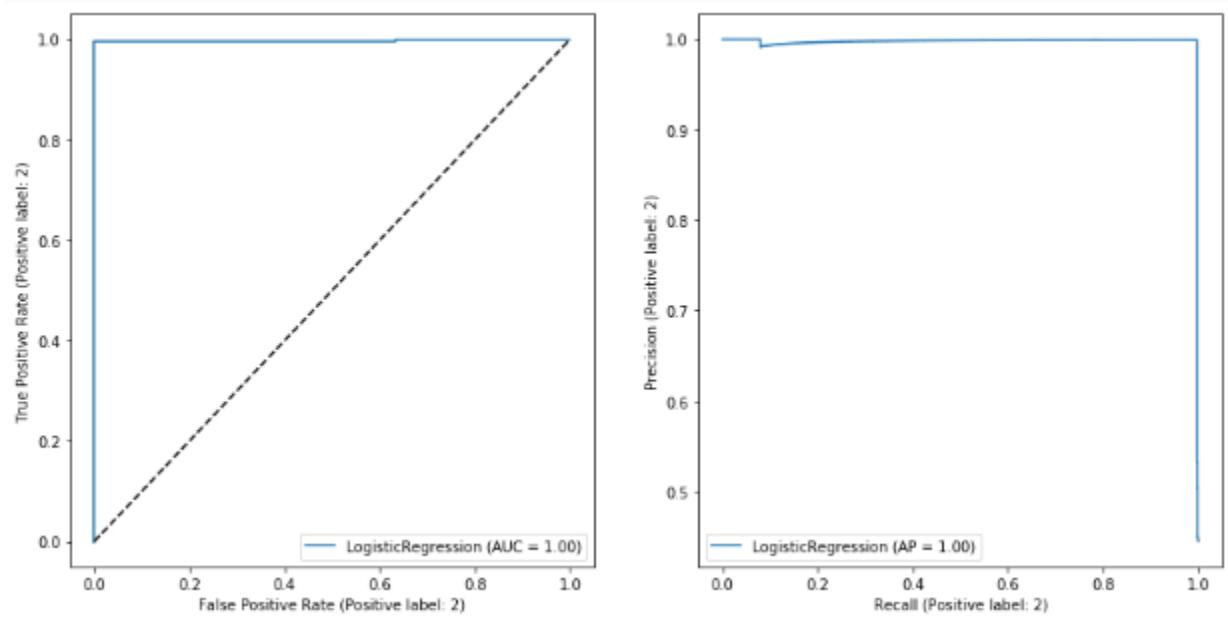
In this section, I have presented the confusion matrix, Precision-Recall and ROC curves. The classification report and details of precision, recall is in the Results section of the report.



This is the confusion matrix for logistic regression which I got. Here, the value of False Positive(FP)= 8 which means 8 negative class data points were incorrectly classified as belonging to the positive class by the model. Thus, this can be considered a good model for classification. The value of False Negative (FN)=5 which is also a very small number.

Now, I will show the ROC and Precision-Recall curves. ROC curves in logistic regression are used for determining the best cutoff value for predicting whether a new observation is a "failure" (0) or a "success" (1). The higher the AUC, the better the performance of the model at distinguishing between the positive and

negative classes. In our case we got a perfect AUC score of 1 which means that the classifier is able to perfectly distinguish between all the Positive and the Negative class points correctly.

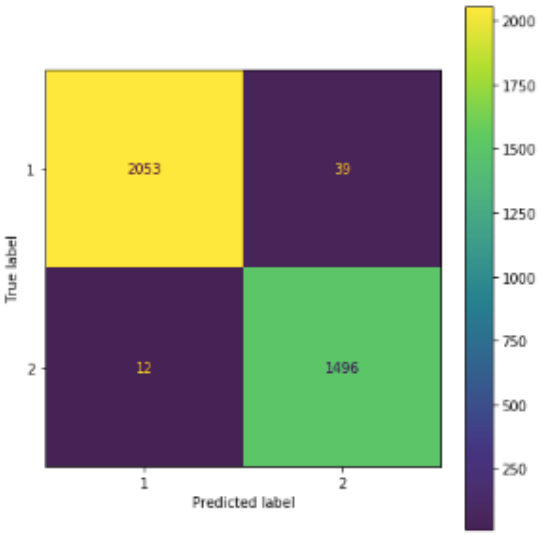


Precision- Recall curve shows the tradeoff between precision and recall for different thresholds. For Precision- Recall we calculated the AP, or average precision and got 1 and is similar to the area under the precision-recall curve.

DECISION TREE CLASSIFIER

For classification and regression, Decision Trees are a non-parametric supervised learning approach. The objective is to learn basic decision rules from data attributes to develop a model that predicts the value of a target variable. A tree is an approximation to a piecewise constant.

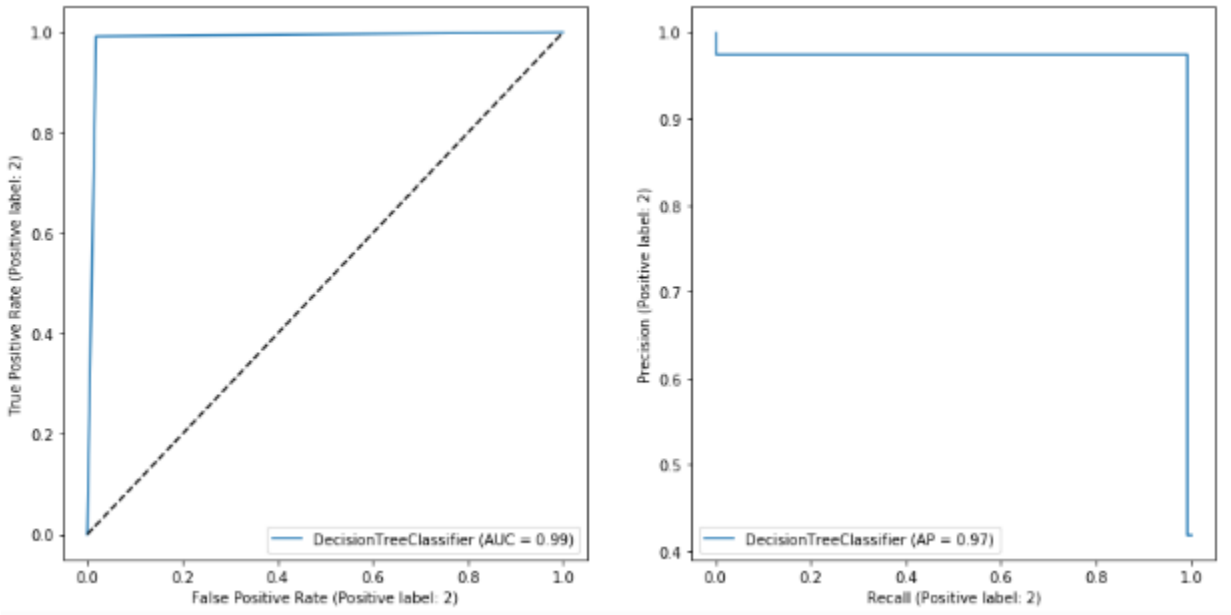
I used the DecisionTreeClassifier() function of sklearn.tree package for implementation of this algorithm which would return an output array, for example the apply() method returns the index of the leaf of the sample's prediction.



This is the confusion matrix for the DecisionTree classifier. Here, we have False Positive (FP)=39 and False Negative (FN)=12. So, these many data points were incorrectly classified. The model has performed badly but when compared to

logistic regression.

ROC and Precision-Recall curves for DecisionTree classifier are as below-

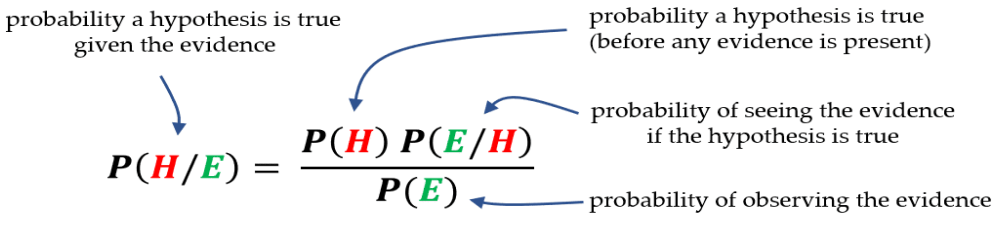


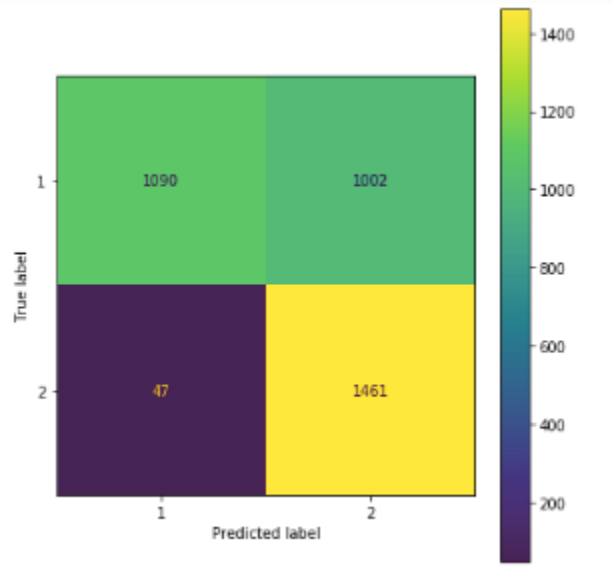
The AUC is a measure of the ability to rank instances based on their likelihood of belonging to a class. Here too we have a good AUC score of 0.99 but still some misclassification happened. Unlike logistic regression, here we have an average precision of 0.97.

GAUSSIAN NAIVEBAYES CLASSIFIER

The Bayes theorem provides the basis for a collection of supervised machine learning classification algorithms known as Naive Bayes. It is a simple classification technique, but has high functionality. They're useful when the inputs' dimensionality is high. Gaussian Naive Bayes is a Naive Bayes variation that allows continuous data and follows the Gaussian normal distribution.

I implemented it in code by using GaussianNB() from sklearn.naive_bayes.The formula for Bayes theorem is as below-

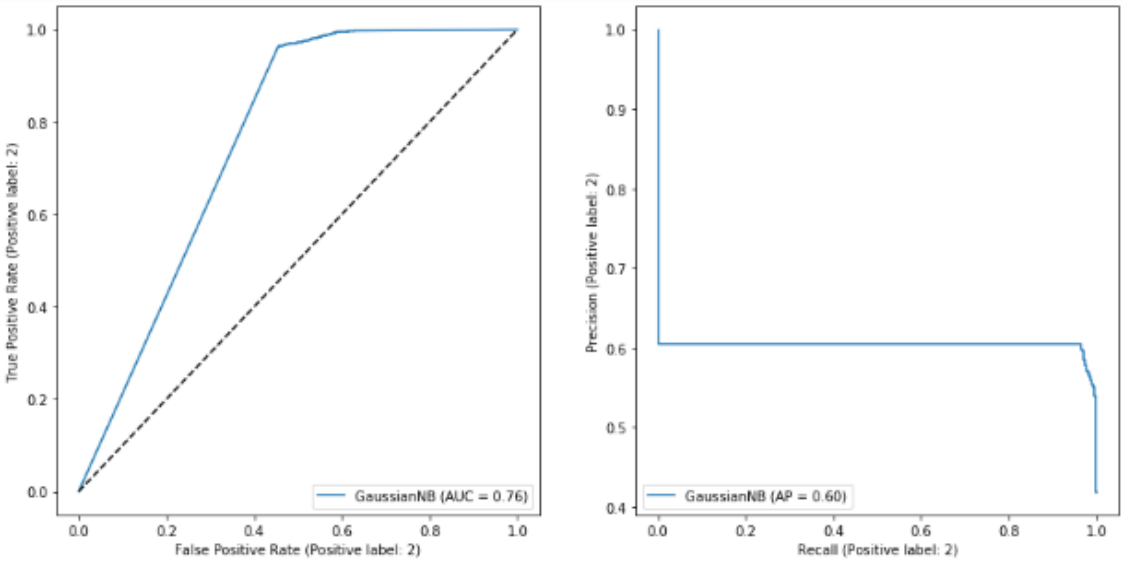




This is the confusion matrix for the Gaussian Naive Bayes classifier. Here, the values False Positive and False Negative are very high, 1002 and 47 respectively. These many classes were misclassified which clearly means that the performance of this classifier is extremely poor.

It can also be seen from the colour variation of the heatmap where the first row is almost green.

Now, let us look at the ROC and precision curves to check if the model really performed bad or not.

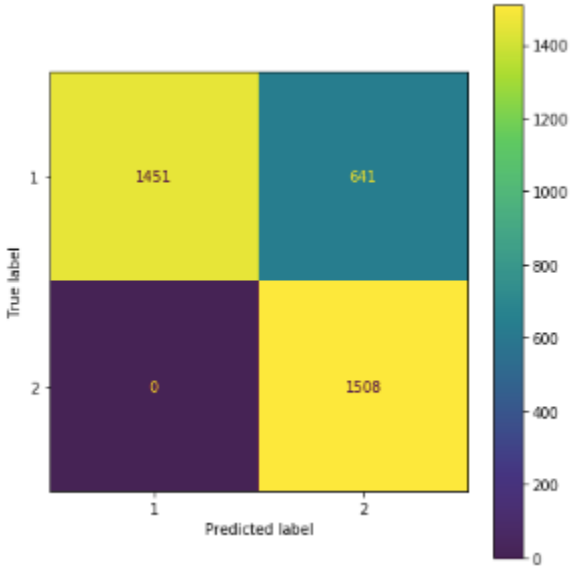


We know that ROC AUC is a single number summary of classifier performance and the higher the value, the better the classifier.But as expected, we get an AUC score of 0.76 which is very low and also an average precision of 0.60. For a purely random classifier, we will have a ROC AUC equal to 0.5 so we can say it's close to random but not exactly random.The precision-recall curve is in the form of steps and so the under this curve is less implying low accuracy. We will look at the F1 score and other details in the results section but from the above analysis it is clear that Gaussian Naive Bayes is not a good classifier in this case.

K-MEANS

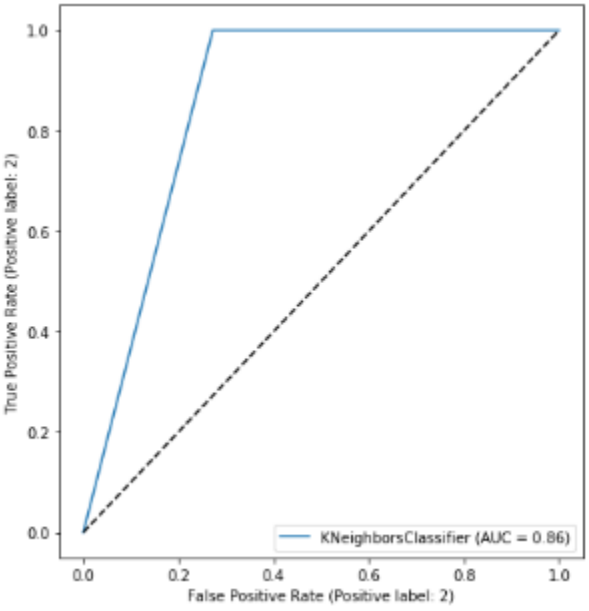
CLASSIFICATION

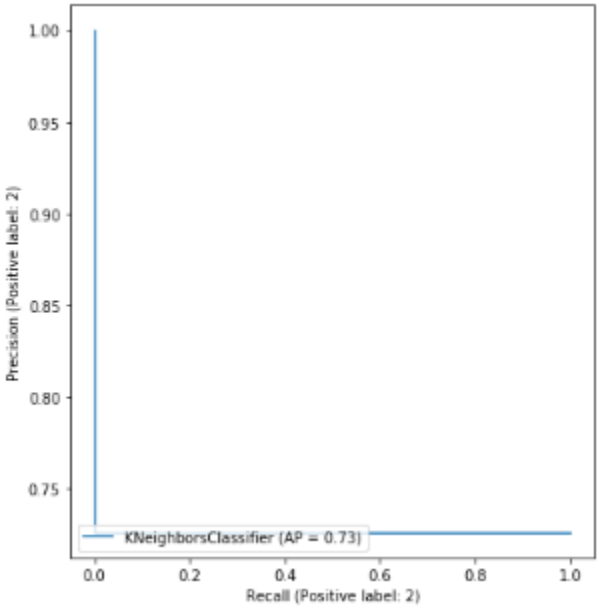
Here, I have used KNeighborsClassifier() from the package sklearn.neighbors , where I inserted trains and test values into the function and calculated the score and classification report which you can see in the result section and code in the implementation section.



This is the confusion matrix for KMeans Classifier. The number of misclassified data points in this case is very high- 641. But the model did classified negative samples correctly as False negative=0 Overall, this model has shown poor performance and is less accurate than previously discussed models.

From the below curves, we can see that the AUC of the ROC curve is 0.86 and the average precision(AP) from precision-recall curve is 0.73. Such low scores tell that this classifier is not capable of distinguishing between classes perfectly.

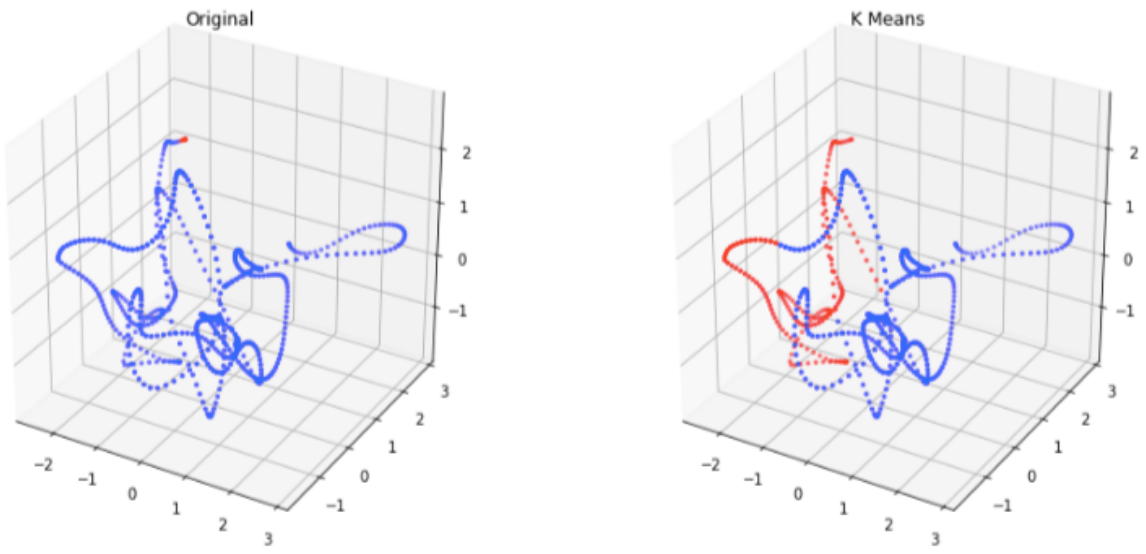




K-MEANS CLUSTERING

It is a type of clustering approach in which it forms a region by calculating the closest distance between two spots and forming clusters with the nearest neighbour. Here, we need to find a value for K that minimises the number of errors while retaining the algorithm's capacity to produce correct predictions when it is given fresh data.

I tried to link this classification problem to clustering. Here we are giving input as 2 clusters and checking whether those 2 clusters resemble our class labels or not. K-means clustering was implemented by using KMeans() from the package sklearn.cluster. The below is the 3D plot which we get-



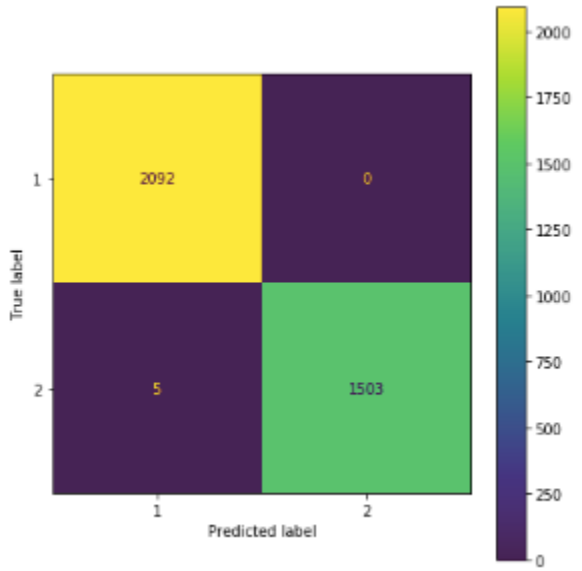
From the above figure we can observe that both the clusters are different and we can differentiate them although all of our stars with exoplanets are in one cluster (Red cluster in the right figure).

KNN offers various benefits, such as no training period because it keeps the training information and only learns from it while generating real-time predictions. Furthermore, we may add new data at any moment without having to create new code. However, because it calculates distance all the time, it does not perform well with huge datasets, and large data sets will use a lot of GPU resources and render time. It also requires normalised data; it cannot operate with unstructured data. The data sets must be scaled in terms of features.

RANDOM FOREST CLASSIFIER

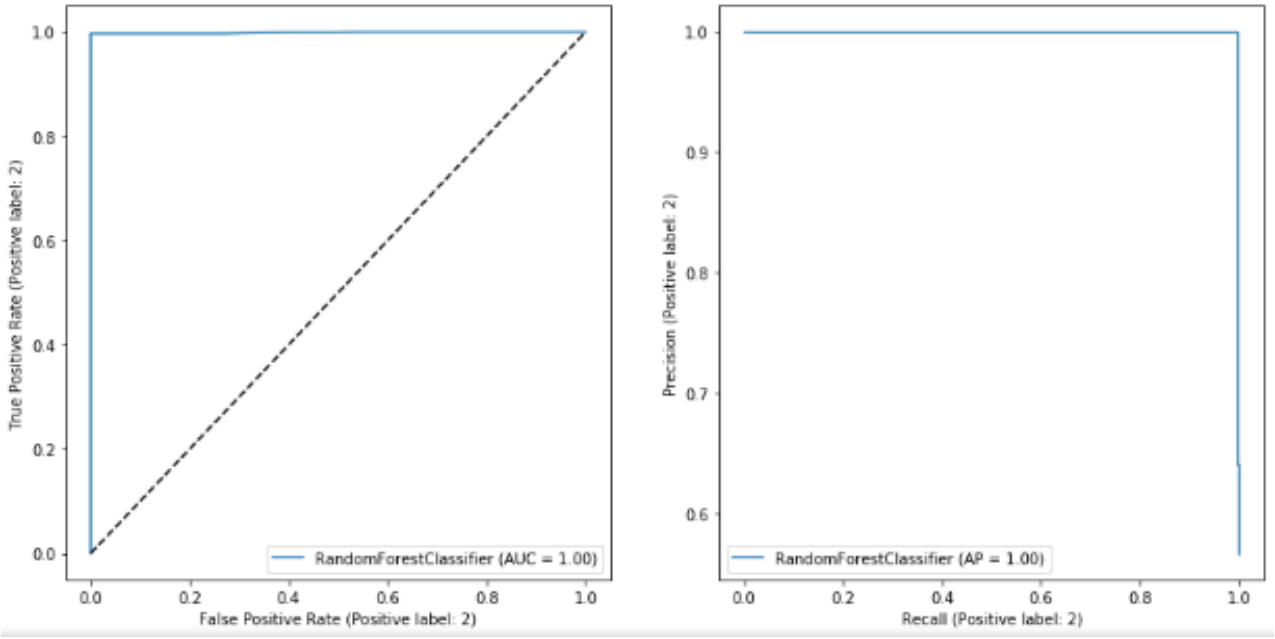
Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset.The greater the number of trees in the forest, the more accurate it is and the problem of overfitting is avoided.

For code implementation, I used RandomForestClassifier() from sklearn.ensemble.



The RandomForest classifier gives us the confusion matrix on the left. The value of False Positives is 0 and that of False Negatives is 5. This tells us that this classifier has shown excellent performance. 0 negative class data points were incorrectly classified as belonging to the positive class by the model. From the Results section, you will see that it gives the highest F1 score for minority class.

It also gives the best ROC-AUC score and precision recall curves-



APPLYING NEURAL NETWORKS

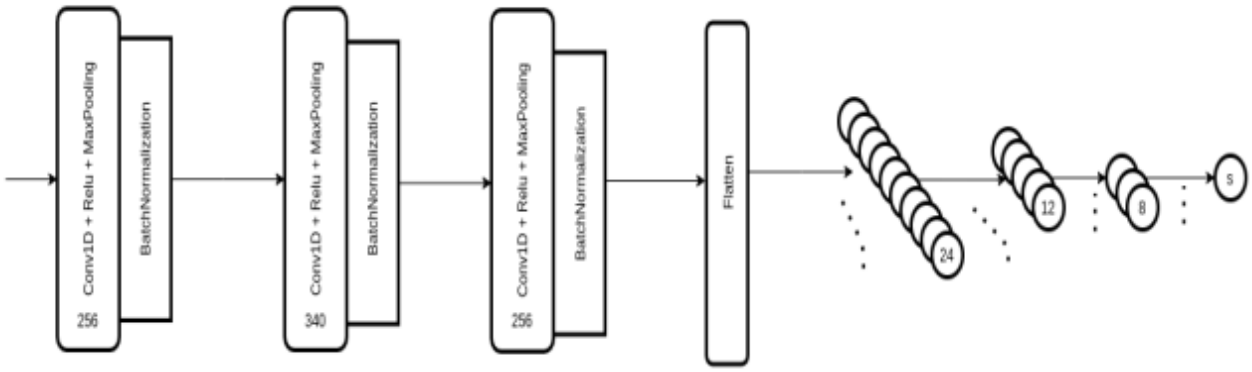
After applying basic machine learning models, let's try to implement neural networks for better accuracy. A neural network is a set of algorithms that attempts to detect underlying relationships in a set of data using a method that mimics how the human brain works. Neural networks, in this context, refer to systems of neurons that can be biological or artificial in nature.

In this project, I have implemented two models- CNN and SVC. Let's discuss both of them one by one-

CNN MODEL

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning model that can take an input picture, assign relevance to various aspects/objects in the image, and distinguish between them. When compared to other classification methods, the amount of pre-processing required by a ConvNet is significantly less. While basic approaches need hand-engineering of filters, ConvNets can learn these filters/characteristics with enough training.

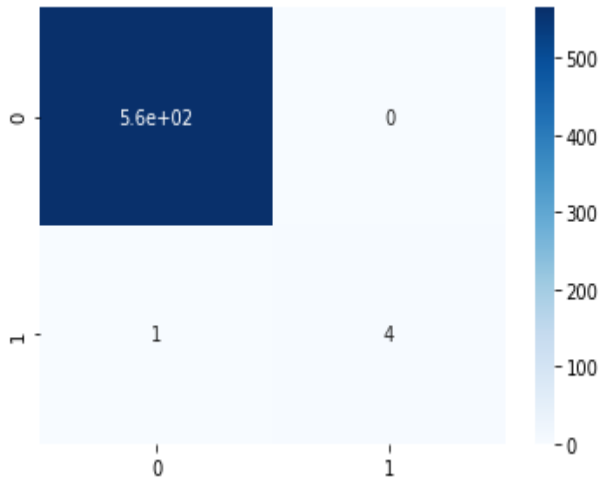
The architecture of CNN used by me is based on the idea of Wang, Zhiguang, Weizhong Yan, and Tim Oates (paper in References section). An addition of 3 max pooling layers after each conv1d layer has been done to reduce the number of parameters.As this is a binary classification issue, the binary cross entropy is the optimal loss function to use so I used the dropout technique in two layers.



Below is the snapshot which was taken while running the code in jupyter notebook and list out all the details about the network.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d (Conv1D)              (None, 3190, 256)         2304
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 638, 256)          0
_________________________________________________________________
batch_normalization (BatchNo (None, 638, 256)          1024
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 633, 340)          522580
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 127, 340)          0
_________________________________________________________________
batch_normalization_1 (Batch (None, 127, 340)          1360
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 124, 256)          348416
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 25, 256)           0
_________________________________________________________________
batch_normalization_2 (Batch (None, 25, 256)           1024
_________________________________________________________________
flatten (Flatten)            (None, 6400)              0
_________________________________________________________________
dropout (Dropout)            (None, 6400)              0
_________________________________________________________________
dense (Dense)                (None, 24)                153624
_________________________________________________________________
dropout_1 (Dropout)          (None, 24)                0
_________________________________________________________________
dense_1 (Dense)              (None, 12)                300
_________________________________________________________________
dense_2 (Dense)              (None, 8)                 104
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 9
=================================================================
Total params: 1,030,745
Trainable params: 1,029,041
Non-trainable params: 1,704
```

The amount of parameters generated by using the max pool layers is reduced by a factor of 20. In reality, we go from roughly 20 million to about 1 million characteristics. The computational complexity of the model has lowered as a result of this improvement, and the training time has definitely dropped.



This is the confusion matrix for CNN model. As you can see the value of False positives is 0 and False negatives is 1. So, very few data points were misclassified and the model shows excellent performance(close to 100% accuracy).These data tell us that the neural network can predict 5 out of 5 exoplanets in the test set.

SVC MODEL

Support Vector Machines are typically thought of as a classification method, however they may be used to solve both classification and regression issues. It can handle both continuous and categorical data with ease. To differentiate various classes, SVM creates a hyperplane in multidimensional space. SVM iteratively creates the best hyperplane, which is then utilised to minimise an error. The goal of SVM is to identify a maximum marginal hyperplane (MMH) that splits a dataset into classes as evenly as possible.

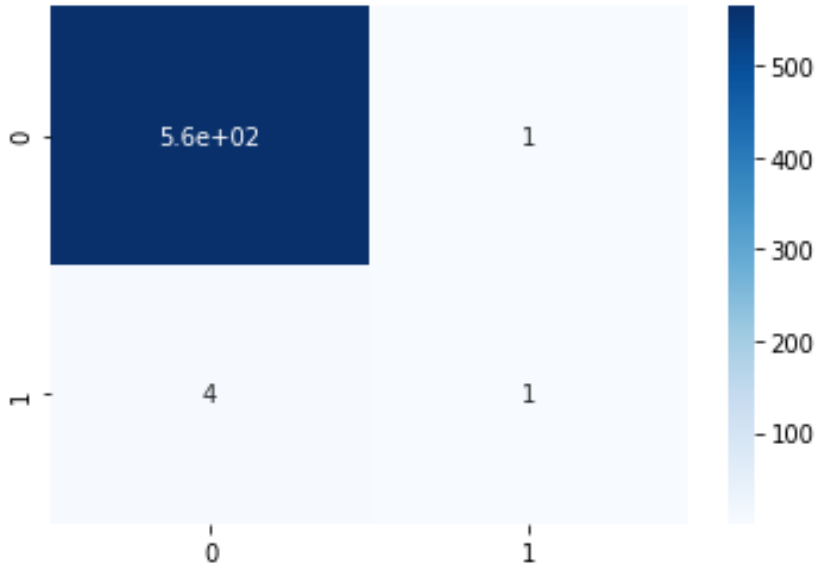
The following are the main benefits of SVC- it efficiently operates in high-dimensional space; it is versatile in that it allows multiple kernel functions to be used. As I'm working with frequency domain data, the classifier takes as input a vector matching the amplitude of an FFT processed signal, then uses a Radial Basis Function(RBF) to map the input vector in a high-dimensional feature space-

$$k(\mathbf{x}, \mathbf{y}) = e^{-\gamma \|\mathbf{x}-\mathbf{y}\|^2}, \gamma > 0$$

C and gamma are two kernel parameters. A high C tries to accurately categorise

all training instances; a low C encourages a broader margin. The greater gamma is, the closer other instances must be to be influenced; the smaller gamma is, the less influence a single training example has. So, once I've decided on the parameters, the SVC model is ready to run.

The below one is the confusion matrix for the SVC model. As you can see the value of False positives is 1 and False negatives is 4. Only one exoplanet is found using the SVC model; the other four positive cases are missed. Deep neural networks has outperformed the SVM strategy.



RESULTS

In this section, we will look at the classification reports of the different models which were used and try to compare which one showed best performance.

```
Logistic Regression

              precision    recall  f1-score   support

           1       1.00      1.00      1.00      2092
           2       0.99      1.00      1.00      1508

    accuracy                           1.00      3600
   macro avg       1.00      1.00      1.00      3600
weighted avg       1.00      1.00      1.00      3600

F1 score of minority class: 0.9956939383901955


DecisionTree Classifier

              precision    recall  f1-score   support

           1       0.99      0.98      0.99      2092
           2       0.97      0.99      0.98      1508

    accuracy                           0.99      3600
   macro avg       0.98      0.99      0.99      3600
weighted avg       0.99      0.99      0.99      3600

F1 score of minority class: 0.9832402234636872


Gaussian NaiveBayes Classifier

              precision    recall  f1-score   support

           1       0.96      0.52      0.68      2092
           2       0.59      0.97      0.74      1508

    accuracy                           0.71      3600
   macro avg       0.78      0.74      0.71      3600
weighted avg       0.81      0.71      0.70      3600

F1 score of minority class: 0.7358348023167968
```

```
k-Nearest Neighbour Classifier

              precision    recall  f1-score   support

           1       1.00      0.69      0.82      2092
           2       0.70      1.00      0.82      1508

    accuracy                           0.82      3600
   macro avg       0.85      0.85      0.82      3600
weighted avg       0.88      0.82      0.82      3600


F1 score of minority class: 0.8247197156138912

Random Forest Classifier

              precision    recall  f1-score   support

           1       1.00      1.00      1.00      2092
           2       1.00      1.00      1.00      1508

    accuracy                           1.00      3600
   macro avg       1.00      1.00      1.00      3600
weighted avg       1.00      1.00      1.00      3600


F1 score of minority class: 0.9983394221188974
```

Now, let's look at the **classification reports of our neural networks**-

CNN-

```
accuracy :  0.9982456140350877
                         precision    recall  f1-score   support

 NO exoplanet confirmed       1.00      1.00      1.00       565
YES exoplanet confirmed       1.00      0.80      0.89         5

               accuracy                           1.00       570
              macro avg       1.00      0.90      0.94       570
           weighted avg       1.00      1.00      1.00       570
```

SVC-

```
                         precision    recall  f1-score   support

 NO exoplanet confirmed       0.99      1.00      1.00       565
YES exoplanet confirmed       0.50      0.20      0.29         5

               accuracy                           0.99       570
              macro avg       0.75      0.60      0.64       570
           weighted avg       0.99      0.99      0.99       570
```

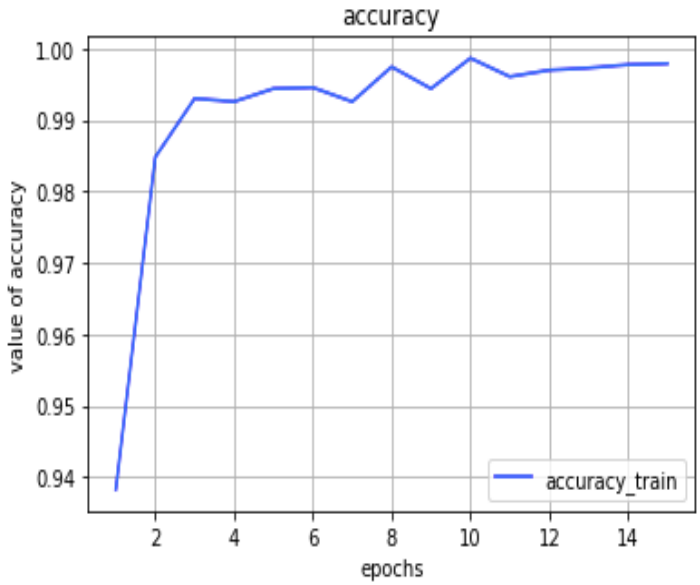
For code implementation, the following parameters for the CNN (designed in Keras) were chosen:

- optimizer = Adam
- learning rate = 0.001
- padding (for each conv1d layer) = same
- kernel size (for each conv1d layer respectively) = 8, 6, 4
- dropout = 0.3
- epochs = 15 or more
- batch_size = 10

In SVC model I setted the following hyper-parameters :

- C = 100
- gamma = 0.001
- kernel = rbf

The Performance of the CNN during training can be seen from the below plots. The loss function tends to 0 and the accuracy tends to 1. This is consistent with our theoretical assumptions.



DISCUSSION

Now, let's understand the results and the classification reports which we got.A classification report primarily examines the accuracy of the predictions made by the applied algorithm or model, which we have linked to our data sets. This report includes some basic concepts and terminology, such as precision, which is the ratio of true positives to false positives in our data sets, recall, which is the ratio of true positives to false negatives in our data sets, F1-score, which is the harmonic mean of precision and recall, with a best value of 1 and a worst score of 0, and support, which is the number of times each class appeared.We also have three additional parameters in the list: accuracy, which is the percentage of correct predictions made by our model, or the number of correct predictions divided by the total number of predictions in our data sets. Weighted average (doing the average of the unweighted mean per label), macro average (doing the average of the unweighted mean per label) (doing the average of the support-weighted mean per label). Moreover the data we are working on is highly imbalanced so by default our model would predict every scenario to be valid (majority class) and still have a good accuracy, which ultimately leads to overfitting and wrong correlations. Thus, oversampling is very important.

All of the models scored poorly without feature engineering and with the original testing dataset but the performance of the models increased drastically with the help of SMOTE and applying other filters. For logistic regression, the precision of label 1 is more than the precision of label 2 because of the ability of a classifier an instance Recall for both the classes is high i.e 1 as it only shows the ratio of true positives to the sum of true positives and false negatives in both the cases.positive that is actually positive), so class has a lot of positive values. Decision tree model also performed quite well but its F1 score of minority class is less than that of logistic regression.

Gaussian Naive Bayes did not perform well during classification. It had a very low precision for Label 2 (0.59) and low recall for Label 1 (0.52). Apart from this it gave the lowest F1 score of minority class (0.735) and thus when compared to others, this model was the worst. We then implemented a K-means classifier which gave an F1 score of around 0.824 and so didn't perform that well, the clustering showed us 2 different clusters in the test data. Random forest gave the highest F1 score for minority class and was the most accurate one.

From the above results, we can conclude that **Random Forest** with SMOTE gave the best F1-score and overall scores followed by Logistic Regression.

The results of neural networks are amazing. As I only had 37 good instances in the training set and the other positive examples were provided by the oversampling approach, each development of a validation set (with various dimensions) for the neural network resulted in a performance loss.We tuned the hyper-parameters during training using, the speed of convergence, and the loss function value.

CNN gave an accuracy of around 0.998 which is a very good number.Predictive models of this sort would be extremely beneficial to researchers even if the findings were not 100 percent correct. The SVC model also performed quite well though it found only an exoplanet; it failed to recognize the other 4 positive examples. So, we can conclude that though running a neural network took time because of 15 epochs, it was worth it because of the high accuracy. We can further increase this accuracy by increasing the number of epochs.

FUTURE ROAD AHEAD

We may also use a dimension reduction strategy to our data sets, which will allow us to convert a high-dimensional space into a low-dimensional space while keeping all of the key attributes. We might utilize TSNE or PCA techniques.

And, in addition to these models, we may employ other models such as the local outlier factor algorithm and others that are outside of our syllabus for anomaly identification.

The transit identification of exoplanets is an essential approach for the finding of new planets in the field of astrophysics. The light curves of hundreds of additional stars acquired by the Kepler telescope may be found in the MAST database (Mikulski Archive for Space Telescopes). We can try this strategy on many other datasets and then try to find a correlation between them.

CODE IMPLEMENTATION/SNIPPETS

There are 2 main code file for machine learning models and neural networks.

One accompanying file called models.py is imported separately for CNN and SVC.

1. Importing several packages and libraries to call the model and many critical functions for classification.

```
'''@author: harshmishra
   Detecting exoplanets using Machine Learning
'''
#Imports
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler, StandardScaler, RobustScaler, normalize
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,plot_confusion_matrix,precision_score,recall_score,f1_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from scipy import ndimage
from sklearn import  metrics
from sklearn.metrics import roc_curve,roc_auc_score,plot_roc_curve
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from imblearn.over_sampling import SMOTE
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier
from scipy.signal import savgol_filter
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
%matplotlib inline
```

2. Initialising test and train data frames using pandas library and visualising classes to check data imbalance-

```
train = pd.read_csv('exoTrain.csv')
test = pd.read_csv('exoTest.csv')
print(train.shape)
print(test.shape)
```

```
(5087, 3198)
(570, 3198)
```

```
#To check data imbalance
train['LABEL'].value_counts().plot(kind = 'bar', title = 'Class Distributions \n (0: Not Exoplanet || 1: Exoplar
# PROBLEM OF IMBALANCED CLASS -RUNNING ANY ALGORITHM WILL RESULT IN A ACCURACY MEASURE THAT IS HIGHLY INFLUENCEI
```

3. Data visualization- For the scatter plot, I used .scatter() method:

```
#STARS WITH EXOPLANETS

fig = plt.figure(figsize=(15,40))
for i in range(12):
    ax = fig.add_subplot(14,4,i+1)
    ax.scatter(np.arange(3197),train[train['LABEL'] == 2].iloc[i,1:],s=1)

#STARS WITHOUT EXOPLANETS
fig = plt.figure(figsize=(15,40))
for i in range(12):
    ax = fig.add_subplot(14,4,i+1)
    ax.scatter(np.arange(3197),train[train['LABEL']==1].iloc[i,1:],s=1)
```

And for histograms, I used .hist() method:

```
#Plotting histograms
#STARS WITH EXOPLANETS
fig = plt.figure(figsize=(15,40))
for i in range(12):
    ax = fig.add_subplot(14,4,i+1)
    train[train['LABEL']==2].iloc[i,1:].hist(bins=50)

#STARS WITHOUT EXOPLANETS
fig = plt.figure(figsize=(15,40))
for i in range(12):
    ax = fig.add_subplot(14,4,i+1)
    train[train['LABEL']==1].iloc[i,1:].hist(bins=50)
```

Code which I used to plot 6 time series in order to look at the variation of the distribution-

```
#Visualizing the 6 rercords
plt.figure(figsize=(25,10))
plt.title('Distribution of flux values', fontsize=15)
plt.xlabel('Flux values')
plt.ylabel('Flux intensity')
plt.plot(train.iloc[0,])
plt.plot(train.iloc[1,])
plt.plot(train.iloc[2,])
plt.plot(train.iloc[3,])
plt.plot(train.iloc[4,])
plt.plot(train.iloc[5,])

plt.legend(('Data-1', 'Data-2', 'Data-3', 'Data-4', 'Data-5', 'Data-6' ))
plt.show()
```

4. Splitting and Standardisation using SMOTE technique. Here I have defined all the filters whose combinations I tried in the models. These were talked about in detail in the pre-processing part.

```
def reset(train,test):
    train_X = train.drop('LABEL', axis=1)
    train_y = train['LABEL'].values
    test_X = test.drop('LABEL', axis=1)
    test_y = test['LABEL'].values
    return train_X,train_y,test_X,test_y

train_X,train_y,test_X,test_y = reset(train,test)

def std_scaler(df1,df2):
    std_scaler = StandardScaler()
    train_X = std_scaler.fit_transform(df1)
    test_X = std_scaler.fit_transform(df2)
    return train_X,test_X

def norm(df1,df2):
    train_X = normalize(df1)
    test_X = normalize(df2)
    return train_X,test_X

def gaussian(df1,df2):
    train_X = ndimage.filters.gaussian_filter(df1, sigma=10)
    test_X = ndimage.filters.gaussian_filter(df2, sigma=10)
    return train_X,test_X

def fourier(df1,df2):
    train_X = np.abs(np.fft.fft(df1, axis=1))
    test_X = np.abs(np.fft.fft(df2, axis=1))
    return train_X,test_X
```

First we reset the data and then apply the filters. For oversampling, SMOTE was used which added 30% to the test data to reduce imbalance. So now the minority cases have a significant no. as compared to the majority cases.The function is

```
#For oversampling
def smote(a,b):
    model = SMOTE()
    X,y = model.fit_sample(a, b)
    return X,y
```

The similar approach was considered while applying neural network but here we used RandomOversampler() and not SMOTE but the percentage was same i.e 30%. The code for normalization, gaussian filter, fast fourier transform and oversampling is as below-

```
#normalized data
data_train_norm = normalize(data_train)
data_test_norm = normalize(data_test)


# function to apply gaussian filter to all data
def gauss_filter(dataset,sigma):

    dts = []

    for x in range(dataset.shape[0]):
        dts.append(gaussian_filter(dataset[x], sigma))

    return np.asarray(dts)

# apply the gaussian filter to all rows data
data_train_gaussian = gauss_filter(data_train_norm,7.0)
data_test_gaussian = gauss_filter(data_test_norm,7.0)

#print the light curves smoothed
plt.figure(figsize=(20,5))
plt.title('Flux of star 250 with confirmed planet after gaussian filter(smoothed)')
plt.ylabel('Flux')
plt.xlabel('Hours')
plt.plot( time , data_train_gaussian[250])
```

```
# apply Fast Fourier Transform to the data smoothed
frequency = np.arange(len(data_train[0])) * (1/(36.0*60.0))

data_train_fft1 = scipy.fft.fft2(data_train_norm, axes=1)
data_test_fft1 = scipy.fft.fft2(data_test_norm, axes=1)

data_train_fft = np.abs(data_train_fft1)   #calculate the abs value
data_test_fft = np.abs(data_test_fft1)
```

We then plot the FFT signal to see if our data has become smooth and cleaned or not and then perform oversampling.

```
#get the length of the FFT data, make something here below in order to make the sequences of the same size
# only if they have differet dimensions

len_seq = len(data_train_fft[0])

#plot the FFT of the signals
plt.figure(figsize=(20,5))
plt.title('flux of star 250 ( with confirmed planet ) in domain of frequencies')
plt.ylabel('Abs value of FFT result')
plt.xlabel('Frequency')
plt.plot( frequency, data_train_fft[250] )

#oversampling technique to the data
rm = RandomOverSampler(sampling_strategy=0.3)
data_train_ovs, y_train_ovs = rm.fit_sample( data_train_fft, y_train)

#Print the dataset count after oversampling
print("After oversampling, counts of label '1': {}".format(sum(y_train_ovs==1)))
print("After oversampling, counts of label '0': {}".format(sum(y_train_ovs==0)))
```

```
After oversampling, counts of label '1': 1515
After oversampling, counts of label '0': 5050
```

5. Now, I will show the code for the different machine learning models implemented. All have been written as a function so that they can be called with ease. Apart from using the essential functions from sklearn. We are also using plot_confusion_matrix() function under the package of sklearn.metrics, .plot_roc_curve() for ROC curve and similarly other functions for evaluating the matrics-

```
#Machine learning models-

def logistic(train_X,train_y,test_X,test_y):
    lgr = LogisticRegression(max_iter=1000)
    lgr.fit(train_X,train_y)
    prediction_lgr=lgr.predict(test_X)
    print("-------------------------------------------")
    print("Logistic Regression")
    print("")
    print(classification_report(test_y,prediction_lgr))
    fig = plt.figure(figsize=(22,7))
    ax = fig.add_subplot(1,3,1)
    plot_confusion_matrix(lgr,test_X,test_y,ax=ax)
    ax = fig.add_subplot(1,3,2)
    metrics.plot_roc_curve(lgr,test_X, test_y,ax=ax)
    plt.plot([0, 1], [0, 1], 'k--')
    ax = fig.add_subplot(1,3,3)
    metrics.plot_precision_recall_curve(lgr, test_X, test_y,ax=ax)
    f1=metrics.f1_score(test_y, prediction_lgr,pos_label=2)
    print("F1 score of minority class:",f1)
    plt.show()
    return f1
```

```
def decisionTree(train_X,train_y,test_X,test_y):
    clf = tree.DecisionTreeClassifier()
    clf = clf.fit(train_X, train_y)
    y_pred_clf = clf.predict(test_X)
    print("-------------------------------------------")
    print("DecisionTree Classifier")
    print("")
    print(classification_report(test_y,y_pred_clf))
    fig = plt.figure(figsize=(22,7))
    ax = fig.add_subplot(1,3,1)
    plot_confusion_matrix(clf,test_X,test_y,ax=ax)
    ax = fig.add_subplot(1,3,2)
    metrics.plot_roc_curve(clf,test_X, test_y,ax=ax)
    plt.plot([0, 1], [0, 1], 'k--')
    ax = fig.add_subplot(1,3,3)
    metrics.plot_precision_recall_curve(clf, test_X, test_y,ax=ax)
    f1=metrics.f1_score(test_y, y_pred_clf,pos_label=2)
    print("F1 score of minority class:",f1)
    plt.show()
    return f1
```

```
def naiveBayes(train_X,train_y,test_X,test_y):
    gnb = GaussianNB()
    gnb.fit(train_X, train_y)
    y_pred=gnb.predict(test_X)
    print("-------------------------------------------")
    print("Gaussian NaiveBayes Classifier")
    print("")
    print(classification_report(test_y,y_pred))
    fig = plt.figure(figsize=(22,7))
    ax = fig.add_subplot(1,3,1)
    plot_confusion_matrix(gnb,test_X,test_y,ax=ax)
    ax = fig.add_subplot(1,3,2)
    metrics.plot_roc_curve(gnb,test_X, test_y,ax=ax)
    plt.plot([0, 1], [0, 1], 'k--')
    ax = fig.add_subplot(1,3,3)
    metrics.plot_precision_recall_curve(gnb, test_X, test_y,ax=ax)
    f1 = metrics.f1_score(test_y, y_pred,pos_label=2)
    print("F1 score of minority class:",f1)
    plt.show()
    return f1
```

```
def randomForest(train_X,train_y,test_X,test_y):
    rnd = RandomForestClassifier()
    rnd.fit(train_X, train_y)
    y_pred_rnd = rnd.predict(test_X)
    print("-------------------------------------------")
    print("Random Forest Classifier")
    print("")
    print(classification_report(test_y,y_pred_rnd))
    fig = plt.figure(figsize=(22,7))
    ax = fig.add_subplot(1,3,1)
    plot_confusion_matrix(rnd,test_X,test_y,ax=ax)
    ax = fig.add_subplot(1,3,2)
    metrics.plot_roc_curve(rnd,test_X,test_y,ax=ax)
    plt.plot([0, 1], [0, 1], 'k--')
    ax = fig.add_subplot(1,3,3)
    metrics.plot_precision_recall_curve(rnd, test_X, test_y,ax=ax)
    plt.show()
    f1 = metrics.f1_score(test_y, y_pred_rnd,pos_label=2)
    print("F1 score of minority class:",f1)
    return f1
```

```
def knn(train_X,train_y,test_X,test_y):
    neigh = KNeighborsClassifier(n_neighbors=3)
    neigh.fit(train_X, train_y)
    y_pred_neigh = neigh.predict(test_X)
    print("----------------------------------------")
    print("k-Nearest Neighbour Classifier")
    print("")
    print(classification_report(test_y,y_pred_neigh))
    fig = plt.figure(figsize=(22,7))
    ax = fig.add_subplot(1,3,1)
    plot_confusion_matrix(neigh,test_X,test_y,ax=ax)
    ax = fig.add_subplot(1,3,2)
    metrics.plot_roc_curve(neigh,test_X, test_y,ax=ax)
    plt.plot([0, 1], [0, 1], 'k--')
    ax = fig.add_subplot(1,3,3)
    metrics.plot_precision_recall_curve(neigh, test_X, test_y,ax=ax)
    plt.show()
    f1 = metrics.f1_score(test_y, y_pred_neigh,pos_label=2)
    print("F1 score of minority class:",f1)
    return f1
```

For CNN, I used keras implemented 1D convolution layer (e.g. temporal convolution).This layer creates a convolution kernel that is convolved with the layer input over a single spatial (or temporal) dimension to produce a tensor of outputs and used Max pooling operation for 1D temporal data whic downsamples the input representation by taking the maximum value over a spatial window of size

```
def FCN_model(len_seq):

    # len_seq = the size of the input sequences

    model = tf.keras.Sequential()

    #change the input shape if you have sequences less long
    model.add(layers.Conv1D(filters=256, kernel_size=8, activation='relu', input_shape=(len_seq,1)))
    model.add(layers.MaxPool1D(strides=5))
    model.add(layers.BatchNormalization())


    model.add(layers.Conv1D(filters=340, kernel_size=6, activation='relu'))
    model.add(layers.MaxPool1D(strides=5))
    model.add(layers.BatchNormalization())


    model.add(layers.Conv1D(filters=256, kernel_size=4, activation='relu'))
    model.add(layers.MaxPool1D(strides=5))
    model.add(layers.BatchNormalization())


    model.add(layers.Flatten())
    model.add(layers.Dropout(0.3))


    model.add(layers.Dense(24, activation='relu'))
    model.add(layers.Dropout(0.3))

    model.add(layers.Dense(12, activation='relu'))


    model.add(layers.Dense(8, activation = 'relu'))


    model.add(layers.Dense(1, activation='sigmoid'))

    return model
```

SVC model- Below is the code for SVC model. Initially, I tried to avoid using the gridsearchCV by altering the kernel and setting C to a "reasonable" number, but results were bad.

```
#create the SVC model
def SVC_model():

    tuned_parameters = [{'kernel': ['rbf'], 'gamma': [1e-3, 1e-4],
                         'C': [1, 10, 100, 1000]},
                        {'kernel': ['linear'], 'C': [1, 10, 100, 1000]}]


    clf = GridSearchCV( SVC(), param_grid = tuned_parameters,scoring = 'recall')


    return clf
```

The entire code files along with datasets can be found in my github account by clicking the link- https://github.com/harsh26apr/Exoplanet_detection-ML

REFERENCES

1. https://www.kaggle.com/keplersmachines/kepler-labelled-time-series-data
2. Wang, Zhiguang, Weizhong Yan, and Tim Oates. "Time series classification from scratch with deep neural networks: A strong baseline." 2017 International joint conference on neural networks (IJCNN). IEEE, 2017.
3. https://scikit-learn.org
4. https://towardsdatascience.com/using-machine-learning-to-find-exoplanets-with-nasas-dataset-bb818515e3b3