Capstone Project –Identifying and Finding Exoplanets in Report

1. Summary of problem statement, data and findings

Domain and Context:

Space Exploration / Space Research is our domain focus on this capstone project; Space research organizations su source datasets; Take a simple example Nasa Hubble telescope, Kepler telescope are used to observe the deep spa neutron stars, asteroids and others; Presently, Nasa observed black hole and taken the full picture of it, that was rem 21st century; Even more NASA planned to have better than Hubble telescope to observe or to visualize beyond the c James Webb Telescope (JWTS) by 2021 (new telescope).

Problem Statement

There is an anticipation thought process, in our whole universe from the beginning of the big bang there is several st before our earth came into better shape for habitual human civilization. As per this calculation, Space researchers c alone in this universe? Is there any other earth like planets out there in our universe? In search for answers, space re universe), based on light intensity (one of the parameter), radiation, spectrum of wavelengths, movements of planets collected data to solve this problem; That's why our capstone project gives ample of concept called "Finding earth li Finding Exoplanets in deep space"; Solution outcome: In future, imagine scenarios like Spaceships are available fast colonizing Exoplanets which was distant away from several light years also become prominent success factors for l such an ambitious capstone project on this problem statement

Dataset used in this Capstone Project

This dataset collected from Kaggle competitions: "Exoplanet Hunting in deep space (Kepler labelled time series data several thousand stars was observed. Each of the star given as binary values 2 and 1; 2 indicated that the star has a hand 1 indicates non-exoplanet in an orbit. The data presented here are cleaned and are derived from observations r

Descriptions of the dataset :

Train dataset (exoTrain.csv):

- 5087 rows or observations.
- 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.

Test dataset (exoTest.csv):

- 570 rows or observations.
- 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.

Dependent features can be used as integer as it is or it can be converted into categorical column.
Independent features are in float data type, and there is no need for data type transformation

2. Overview of the final process

Briefly describe your problem solving methodology. Include information about the salient features of your data, data and how you combined techniques.

1. We have used preprocessing steps, like separating Exoplanets and Non-Exoplanets train & test datasets. Find
2. We have used Machine Learning Alogrithms like, SGD Classifier for Linear classification, Decision Tree and Ra like XGBoost, AdaBoost, PCA
3. We have used CNN 1D deep learning and in that we have used different optimization techniques such as SGD

3. Step-by-step walk through of the solution

Describe the steps you took to solve the problem. What did you find at each stage, and how did it inform the next ste

1. At initial steps We have the used preprocessing steps for train and test datasets.
2. We have different EDA and Visualizations like Fast fourier trasform, normalized data, Short time fourier transf
3. We have applied different Machine learning models and find it's accuracy, F1 score, created confusion matrix

From the data sets, **99.3%** in train set and **99.1%** in test set are Non-Exoplanets
This is highly imbalanced dataset which requires data sampling techniques

The **Light Intensity** plot provides the following insights:

1. Normal light intensity meterages with frequently distributed over with resolution of 1.0
2. Mean light intensity values shows that everything within range for each instances in the light intensity metera
3. Standard deviation light intensity mostly above zero value, shows that +/- square root values on each instance

The **Decomposed Seasonal** plot provides the following insights:

1. Original graph shows how light intensity distributed on train dataset for frequency of 900.
2. Seasonality graph shows seasonal trending pattern over the original graph and smoothening the signals in de
3. Residual graph shows the inverse instance values of original graph and smoothened seasonal decomposed g
4. Fourier decomposed graph shows at the beginning and at the end, distorted frequency wavelengths are availa

The **Tight Layout FFT** graph provides the following insights:

1. Absolute original graph shows normal one
2. Normalized graph shows, smoothening signals
3. Filtered graph shows, normalized graphs touch points
4. Scaled graph provides seasonal trending type of pattern; the pattern of significant instances exponencially de points it stays in the same level (horizontal view);

The **Light Intensity frequency distribution** graph provides the following insights:

1. Blue graph shows light intensity value of exoplanets, which has less data and hence frequency distribution de
2. Red graph shows light intensity value of non-exoplanets, which has the slightly huge data and hence the frequ

Since this is a highly imbalance dataset:

1. we are using batch data generator to synthesize data in deep neural network.
2. accuracy will be high even for pool model performance. Hence we would like to use other metrics like f1, reca

Model one - base model: Uses CNN Conv1D with max pooling, batch norm, fixed kernel and filter size and drop out, d layers and Sigmoid in the final dense layer, Adam optimizer with fixed learning rate using accuracy, f1, precision, rec

Model Two - is similar to Model one except for optimizer as SGD.

Model Three - is similar to Model two except for using learning rate and momentum in SGD.

Among these 3 models with preset parameters - we see model two performance is good. Beyond these preset mode the data set and Keras Classifier with Randomized Search CV for model and hyper parameter tuning.

Parameter Grid for Random Search involves searching:

1. Initializers: Uniform, Lecun_uniform, Normal, Zero, Glorot_normal, Glorot_uniform, He_normal, He_uniform
2. Activation funtions: Softmax, Softplus, Softsign, Relu, Tanh, Sigmoid, Hard_sigmoid, Linear
3. Drop rates: 0.2, 0.4, 0.6
4. Number of neurons: 32, 64, 128
5. Batch size : 32, 64
6. Optimizers: SGD, RMSProp, Adagrad, Adadelta, Adam, Adamax, Nadam
7. Filter sizes: 8, 16

AUTOML Platforms democratizes machine learning by making it accessible for everyone. AUTOML platforms involv engineering, algorithm selection, hyperparameter optimization, model tuning, etc. TPOT specifically is based on Gen representation of solution domain and a fitness function to evaluate the solution domain. TPOT will consider a popu evolution laws to have them optimize the objective function called fitness. For each generation it will select the best fitness function and use genetic operations to reproduce next generation. Recombination (exploitation) combines pa Mutations introduce random (exploration) perturbations. This way, the average population's fitness is supposed to in at fittest individual. However the model two was still performing better than the model from TPOT.

4. Model evaluation

Describe the final model (or ensemble) in detail. What was the objective, what parameters were prominent, and how models(s)? A convincing explanation of the robustness of your solution will go a long way to supporting your solutic

- When compared to Machine Learning model vs Deep learning CNN 1D model, CNN 1D model performed well with ADAM optimizer gives prominent solution for this Exoplanet datasets
- So Final model, Mostly works well on Deep Learning CNN 1D with Adam optimizer, In Machine learning model Gaussian Naive Bayes Macine Learning models provide high accuracy values

Model Name	Accuray Score (in %
Linear Model with SGD Classifier	99.47
Random Forest	99.15
Decision Tree	99.15
Naive Bayes	99.52
SG Boosting	99.12
ADA Boosting	99.12
CNN with Adam Optimizer	98.24
CNN with SGD Optimizer	99.82
CNN with SGD Optimizer, Learning Rate and Momentum	99.84

Model deployment:

Flask is a micro web framework written in Python. In order to productionize the best performing model, we have dep

Flask. We have exposes a Rest API call exoplanet_predict to get the prediction based on the input FLUX values. The and returns the predicted Label. Since the number of independent features are very high, we used a custom written l will read the test csv rows and convert each row into a JSON object and invoke the REST API. This REST Web applic deployed locally as well as on Google cloud.

 flask.png

Usage info:

1. SSH into the server
2. Start the web application nohup docker-compose up &
3. python3 ./test.py

5. Comparison to benchmark

How does your final solution compare to the benchmark you laid out at the outset? Did you improve on the benchma

- We have used several machine learning / deep learning algorithms with different parameters optimization tec which results enhancements on the benchmark for Exoplanet datasets
- There were some solutions which was performing well on training dataset but not generalizing well for test da hence it is performing well on test dataset. The final model is able to detect all 5 exoplanets correctly from te

6. Visualization(s)

In addition to quantifying your model and the solution, please include all relevant visualizations that support the idea

- For Exoplanets and Non-Exoplanets we have created visualizations based on light intensity, Mean and Standa
- Guassian histogram plots
- Original, Normalized, Filtered, Scaled graphs for Flux intensity based values on the Exoplanets and Non-Exopla

7. Implications

How does your solution affect the problem in the domain or business? What recommendations would you make, an

- Based on our solution, as we have different approaches for finding Exoplanets in Deep Space; When consider scienttist observed and collected the Exoplanets datas; We can take the Raw datasets and aggregate it to lab findings; We can re-use these ML / Deep learning models which provides high accuraccy values

8. Limitations

What are the limitations of your solution? Where does your model fall short in the real world? What can you do to en We have only kepler observed Exoplanet Datasets which also aggregated it; This AI/ML model solution is useful, wh datas on Exoplanets, example : Hubble Telescope, in Future JWST (James Webb Satellite Telescope)

We have tried to data preprocessing using data standardization, uniform 1d filters; experimented multiple algorithms initializers, architectures, etc. However we can further study the effect of feature engineering using feature crosses,

Our model deployment was done using a simple web application. In a production setup, the solution can be impleme to segregate data flow between batch processing and real time processing. We can use document database like Mo Kafka for data store and ingestion. We can setup independent batch training jobs using Spark ML and setup deployr

9. Closing Reflections

What have you learned from the process? What would you do differently next time?

We have learned about the based on FLUX range light intensity values, the dataset has been used for finding Exoplar
Learning algorithms; Next time, We have planned to collect image datasets on the exoplanets; Here, we have used o

```
from google.colab import drive
drive.mount('/content/drive/')
```

```
#Install packages that are required in colab
#!pip install imblearn
###!conda install numpy scipy scikit-learn pandas joblib
#!pip install deap update_checker tqdm stopit
#!pip install dask[delayed] dask-ml
#!pip install scikit-mdr skrebate
#!pip install tpot
#!conda install -c conda-forge ipywidgets
#pip install imblearn deap update_checker tqdm stopit dask[delayed] dask-ml scikit-mdr skrel
```

```
import pandas as pd
import numpy as np
import tensorflow as tf
import pickle
import matplotlib.pyplot as plt
import matplotlib.lines as mlines
import seaborn as sns; sns.set()
import statsmodels.api as sm
import scipy
import scipy.signal as signal
from scipy.ndimage.filters import uniform_filter1d
from scipy import stats, ndimage, fft
from imblearn.over_sampling import SMOTE
from statsmodels.tsa.seasonal import seasonal_decompose, DecomposeResult
from keras.models import Sequential, Model, model_from_json, model_from_yaml
from keras.layers import Conv1D, MaxPool1D, Dense, Dropout, Flatten
from keras.layers import BatchNormalization, Input, concatenate, Activation
from keras.optimizers import Adam, SGD
from keras.wrappers.scikit_learn import KerasClassifier
from keras.utils import plot_model
from keras import backend as K
from sklearn import metrics
from sklearn.metrics import precision_score, recall_score, confusion_matrix, classification_
from sklearn.metrics import accuracy_score, f1_score, cohen_kappa_score, roc_auc_score, roc_
from sklearn import model_selection
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV, ParameterGrid
from sklearn.model_selection import train_test_split, cross_val_score, cross_val_predict, KF
from sklearn.preprocessing import normalize, StandardScaler
from sklearn.decomposition import PCA
from sklearn import linear_model
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import AdaBoostClassifier
from xgboost import XGBClassifier
from tpot import TPOTClassifier
```

```
%load_ext autoreload
%autoreload 2
%matplotlib inline
```

Exoplanets in Deep Space, Class object creation with

1. Preprocessing steps
2. EDA and Visualization
3. Classical Machine Learning models
4. Deep Learning with CNN 1D different Optimizers

```
class Exoplanets_DeepSpace:

  # Preprocessing
  # reading csv file
  def readCsv_File(self, path):
      return pd.read_csv(path)

  # finding shape
  @staticmethod
  def shapeOfData(dataFrameExoPlanets):
      return dataFrameExoPlanets.shape

  # Head values
  @staticmethod
  def headValues(dfExoPlanets):
      return dfExoPlanets.head()

  # sample random values
  @staticmethod
  def sampleRandomValues(dfExoPlanets):
      return dfExoPlanets.sample(5)

  # describe stats
  @staticmethod
  def describeStats(dfExoPlanets):
      return dfExoPlanets.describe()

  # Features info
  @staticmethod
  def featuresInfo(dfExoPlanets):
      return dfExoPlanets.info()

  #only exoplanets
  @staticmethod
  def onlyExoplanets(dfExoPlanets):
      return dfExoPlanets[dfExoPlanets.LABEL==2]


  #only exoplanets
  @staticmethod
  def onlyNonExoplanets(dfExoNonPlanets):
      return dfExoNonPlanets[dfExoNonPlanets.LABEL==1]

  # EDA and Visualization
  # distribution parameter values for mean and standard deviation based on light intensity
  @staticmethod
  def get_distribution_params(intensity_vals, window_size=10):
      return intensity_vals.rolling(window_size).mean(), intensity_vals.rolling(window_size

  # plot the light intensity time series graph with resolution value upto 1.0
  @staticmethod
  def plot_light_intensity(X, figsize=(16,8), resolution=1.0):
    """Plots given light intensity time series"""
    if resolution < 1.0:
        resolution = int(1.0//resolution)
        X = X[::resolution]
```

```
        intensity_vals = pd.DataFrame(X)
        measurements = [i for i in range(1, len(X) + 1, 1)]

        rolling_mean_variety, rolling_std_variety = Exoplanets_DeepSpace.get_distribution_param

        plt.figure(figsize=(figsize[0],figsize[1]))
        plt.title = "Start light intensity variation"

        plt.plot(measurements, intensity_vals.values,color='b')
        plt.plot(measurements, rolling_mean_variety.values,color='r')
        plt.plot(measurements, rolling_std_variety.values,color='g')

        blue_line = mlines.Line2D([], [], color='blue', label='Light intensity meterages')
        red_line = mlines.Line2D([], [], color='red', label='Mean light intensity')
        green_line = mlines.Line2D([], [], color='green', label='Standard deviation of light in
        plt.legend(handles=[blue_line, red_line, green_line])

        plt.xlabel('Meterages', fontsize=18)

        plt.xticks(rotation=90)
        plt.ylabel('Light intensity', fontsize=18)

        plt.show()

    #Returns decomposed time series into seasonal, trend, residual, observed
    # and fourier transform components
    @staticmethod
    def seasonal_decompose_fft(X, freq):

        decomposition = seasonal_decompose(X, freq=freq)

        return DecomposeResult(seasonal=decomposition.seasonal,
                               trend=decomposition.trend,
                               resid=decomposition.resid,
                               observed=decomposition.observed,
                               fft=np.fft.fft(decomposition.seasonal))

    # Plots decomposition of seasonal_decompose_fft function
    @staticmethod
    def plot_decomposed_seasonal(decomposition):

        plt.figure(figsize=(16,8))
        plt.subplot(511)
        plt.plot(decomposition.observed, label='Original')
        plt.legend(loc='best')
        plt.subplot(512)
        plt.plot(decomposition.trend, label='Trend')
        plt.legend(loc='best')
        plt.subplot(513)
        plt.plot(decomposition.seasonal,label='Seasonality')
        plt.legend(loc='best')
        plt.subplot(514)
        plt.plot(decomposition.resid, label='Residuals')
        plt.legend(loc='best')
        if hasattr(decomposition, 'fft'):
            plt.subplot(515)
            plt.plot(decomposition.fft, label='Fourier decomposition')
            plt.legend(loc='best')
        plt.tight_layout()

    # absolute, normalized, filtered and scaled values plot layout for Flux
    @staticmethod
    def plot_tight_layout_fft(absolute,normalized, filtered, scaled,
                              series_number ):
        plt.figure(figsize=(16,8))
        plt.subplot(221)
        plt.plot(absolute[series_number], label='Original')
        plt.legend(loc='best')
        plt.subplot(222)
        plt.plot(normalized[series_number], label='Normalized')
```

```
        plt.legend(loc='best')
        plt.subplot(223)
        plt.plot(filtered[series_number],label='Filtered')
        plt.legend(loc='best')
        plt.subplot(224)
        plt.plot(scaled[series_number],label='Scaled')
        plt.legend(loc='best')
        plt.tight_layout()

    #Normalizing the data
    @staticmethod
    def normal(X):
        Y= (X-np.mean(X))/(np.max(X)-np.min(X))
        return Y

    #Sampleling the signal over a period of time (or space)
    #and divides it into its frequency components.
    @staticmethod
    def fast_fourier_transf(X):
      Y = scipy.fft(X, n=X.size)
      return np.abs(Y)

    #Sampleling the signal over a period of time (or space)
    #and divides it into its frequency components.
    @staticmethod
    def shorttime_fourier_transf(X):
      Y = signal.stft(X)
      return np.abs(Y)

    # Frequency of each light intensity of 7 Stars
    # 7 confirmed exoplanet-stars and 563 non-exoplanet-stars.
    @staticmethod
    def fluxFreq_ExoplanetTestPlot(X_train_fft):
      for i in [0,1,2,3,4,6]:
        Y = X_train_fft.iloc[i]
        X = np.arange(len(Y))*(1/(36.0*60.0))
        plt.figure(figsize=(15,5))
        plt.ylabel('Flux')
        plt.xlabel('Frequency')
        plt.plot(X, Y)
        plt.show()

    #Frequency of each light intensity of 7 Non-exoplanet Stars
    @staticmethod
    def fluxFreq_NonExoplanets(X_train_fft):
      for i in [j for j in range (37,44)]:
          Y = X_train_fft.iloc[i]
          X = np.arange(len(Y))*(1/(36.0*60.0))
          plt.figure(figsize=(15,5))
          plt.ylabel('Flux')
          plt.xlabel('Frequency')
          plt.plot(X, Y)
          plt.show()

    # plot gaussian histogram for 37 Exoplanets
    @staticmethod
    def gaussianHistExoplanet(X_train,labels_2):
      print("plotting the Gaussian Histogram",
            "for first 37 Exoplanets in training dataset")
      for i in labels_2:
        plt.hist(X_train.iloc[i,:], bins=200)
        plt.xlabel("Flux values")
        plt.show()

    # plot gaussian histogram for Non-Exoplanets
    @staticmethod
    def gaussianHistExoplanet(X_test,labels_1):
      for i in labels_1:
        plt.hist(X_test.iloc[i,:], bins=200)
        plt.xlabel("Flux values")
```

```
        plt.show()

    # plot the complete range of test data grapth both
    # exoplanets and non-exoplanets
    @staticmethod
    def ploCompleteTestDataGraph(exoTest):
      colors = {'1.0':'red', '2.0':'blue'}
      plt.figure(figsize=(20,10))
      for x in range(exoTest.shape[0]):
          if(exoTest.values[x,0]==1):
              plt.plot(exoTest.values[x,1:],color=colors[str(exoTest.values[x,0])],alpha=0.4)
      plt.show()

      plt.figure(figsize=(20,10))
      for x in range(exoTest.shape[0]):
          if(exoTest.values[x,0]==2):


            plt.plot(exoTest.values[x,1:],color=colors[str(exoTest.values[x,0])],alpha=0.4)
      plt.show()

    # Classical Machine Learning models
    # - Linear model SGD Classifier
    # - Random forest
    # - Decision Tree
    # - Boosting, Adaboosting
    # - Naive Bayes
    # - PCA ( Principle Component Analysis)

    #Applying Linear Classification : SGD Classifier
    @staticmethod
    def linearML_SGDClassifier(X_fft,y_train,y_test, X_test_fft):
      sm = SMOTE(ratio = 1.0)
      X_fft_sm, y_train_sm = sm.fit_sample(X_fft, y_train)
      print(len(X_fft_sm))
      model = linear_model.SGDClassifier(max_iter=1000, loss="perceptron", penalty="l2", alpha
      model.fit(X_fft_sm, y_train_sm)

      Y_train_predicted = model.predict(X_fft_sm)
      Y_test_predicted =  model.predict(X_test_fft)

      print("Train accuracy = %.4f" % accuracy_score(y_train_sm, Y_train_predicted))
      print("Test accuracy = %.4f" % accuracy_score(y_test, Y_test_predicted))

      confusion_matrix_train = confusion_matrix(y_train_sm, Y_train_predicted)
      confusion_matrix_test = confusion_matrix(y_test, Y_test_predicted)
      classification_report_train = classification_report(y_train_sm, Y_train_predicted)
      classification_report_test = classification_report(y_test, Y_test_predicted)
      print("Confusion Matrix (train sample):\n", confusion_matrix_train)
      print("Confusion Matrix (test sample):\n", confusion_matrix_test)
      print("\n")
      print("Classification_report (train sample):\n", classification_report_train)
      print("Classification_report (test sample):\n", classification_report_test)

    #Applying Decision Tree Classifier
    @staticmethod
    def decisionTreeML(X_train,y_train,X_test,y_test, max_depth_val=5, max_leaf_nodes_val=2):
      dtc = DecisionTreeClassifier(criterion='entropy',max_depth=max_depth_val,max_leaf_nodes=
      dtc.fit(X_train,y_train)
      y_predict = dtc.predict(X_test)
      print("Test accuracy = %.4f" % accuracy_score(y_test, y_predict))
      confusion_matrix_test = confusion_matrix(y_test, y_predict)
      classification_report_test = classification_report(y_test, y_predict)
      print("Confusion Matrix (test sample):\n", confusion_matrix_test)
      print("\n")
      print("Classification_report (test sample):\n", classification_report_test)

      return dtc

    #Applying Random Forest Classifier
```

```
    #@staticmethod
    # def randomForestClassifierML(X_train,y_train,X_test,y_test):
     # random_forest = RandomForestClassifier()
      #random_forest.fit(X_train, y_train)
     # y_predict = random_forest.predict(X_test)
      #print("Test accuracy = %.4f" % accuracy_score(y_test, y_predict))
      #confusion_matrix_test = confusion_matrix(y_test, y_predict)
      #classification_report_test = classification_report(y_test, y_predict)
      #print("Confusion Matrix (test sample):\n", confusion_matrix_test)
      #print("\n")
      #print("Classification_report (test sample):\n", classification_report_test)

      #return random_forest

    # Evaluate models for DecisionTree and RandomForest classifier
    @staticmethod
    def evaluateModelsByBoxPlot(dtc,random_forest,model_selection):
        models = []
        models.append(('DecisionTree', dtc))
        models.append(('RandomForest', random_forest))
        # evaluate each model in turn
        results = []
        names = []
        scoring = 'accuracy'
        for name, model in models:
            kfold = model_selection.KFold(n_splits=5,random_state=2)
            cv_results = model_selection.cross_val_score(model, X, Y, cv=kfold, scoring=scori
            results.append(cv_results)
            names.append(name)
            msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
            print(msg)
        # boxplot algorithm comparison
        fig = plt.figure()
        fig.suptitle('Algorithm Comparison')
        ax = fig.add_subplot(111)
        plt.boxplot(results)
        ax.set_xticklabels(names)
        plt.show()

    # Naive bayes model
    @staticmethod
    def gaussianNaiveBayesML(train_set, test_set,train_labels, test_labels, metrics, cv_val=1
      model = GaussianNB()
      model.fit(train_set, train_labels)
      print("model score :: ", model.score(train_set , train_labels))
      test_pred = model.predict(test_set)
      print(metrics.classification_report(test_labels, test_pred))
      print(metrics.confusion_matrix(test_labels, test_pred))
      scores = cross_val_score(model, train_set, train_labels, cv=cv_val)
      print("Cross-validated scores:", scores , scores)
      print("Average score:" , np.average(scores))
      return model

    #ada boost classifier model
    @staticmethod
    def adaBoostclassifierML(model,X_train,y_train,X_test,y_test, estimators_total=51):
      ada_model = AdaBoostClassifier(base_estimator=model,n_estimators= estimators_total)
      ada_model.fit(X_train, Y_train)
      y_pred_boost = ada_model.predict(X_test)
      ada_acc=metrics.accuracy_score(Y_test,y_pred_boost)
      print("ADABoost Ensemble Model Accuracy: ", ada_acc)
      ada_cm=metrics.confusion_matrix(Y_test,y_pred_boost)
      print(ada_cm)
      ada_cr=metrics.classification_report(Y_test,y_pred_boost)
      print(ada_cr)
      return ada_model

    #xgBoost classifierML model
    @staticmethod
    def xgBoostClassifier(X_train,y_train,X_test,y_test):
```

```
    model = XGBClassifier()
    # fit the model with the training data
    model.fit(X_train,Y_train)
    # predict the target on the train dataset
    predict_train = model.predict(X_train)
    print('\nTarget on train data',predict_train)
    # Accuray Score on train dataset
    accuracy_train = metrics.accuracy_score(Y_train,predict_train)
    print('\naccuracy_score on train dataset : ', accuracy_train)
    # predict the target on the test dataset
    predict_test = model.predict(X_test)
    print('\nTarget on test data',predict_test)
    # Accuracy Score on test dataset
    accuracy_score_xgb = metrics.accuracy_score(Y_test,predict_test)
    print('\naccuracy_score on test dataset : ', accuracy_score_xgb)
    return model
```

1. Preprocessing Steps

1.1 Read Exoplanets_DeepSpace class object into exoplanets variable

```
exoplanets =  Exoplanets_DeepSpace()
```

1.2 Read Train Data by using readCsv_File method in Exoplanets_DeepSpace()

```
#exoTrain = exoplanets.readCsv_File('/content/drive/My Drive/capstone_datasets/exoTrain.csv
exoTrain = exoplanets.readCsv_File('../webapp/web/exoTrain.csv')
```

1.3 Read Test Data by using readCsv_File method in Exoplanets_DeepSpace()

```
#exoTest = exoplanets.readCsv_File('/content/drive/My Drive/capstone_datasets/exoTest.csv')
exoTest = exoplanets.readCsv_File('../webapp/web/exoTest.csv')
```

1.4 Print the number of train and test data by using shapeOfData method in Exoplanets_DeepSpace()

```
exoTrainShape = exoplanets.shapeOfData(exoTrain)
exoTestShape = exoplanets.shapeOfData(exoTest)
print("Shape of train and test data :: ", exoTrainShape, exoTestShape)
```

```
Shape of train and test data ::  (5087, 3198) (570, 3198)
```

In training set data, we have 5087 rows/observations and 3198 columns/features. Column 1 is the label vector and values over time.

In test dataset, we have 570 rows/observations and 3198 columns/features. Column 1 is the label vector and colum over time.

1.5 Print the number of Exoplanets and Non-Exoplanets by using onlyExoplanets and onlyNonExoplanets method respectively in Exoplane

```
onlyExoplanetsTrain = exoplanets.onlyExoplanets(exoTrain)
onlyNonExoplanetsTrain = exoplanets.onlyNonExoplanets(exoTrain)
onlyExoplanetsTest = exoplanets.onlyExoplanets(exoTest)
onlyNonExoplanetsTest = exoplanets.onlyNonExoplanets(exoTest)
```

```
print("Shape of explonets from train and test data :: ", exoplanets.shapeOfData(onlyExoplan
print("Shape of non-explonets from train and test data :: ", exoplanets.shapeOfData(onlyNonE
```

```
Shape of explonets from train and test data ::  (37, 3198) (5, 3198)
Shape of non-explonets from train and test data ::  (5050, 3198) (565, 3198)
```

In training set, **37** confirmed exoplanet-stars and 5050 non-exoplanet-stars.

In test set, **5** confirmed exoplanet-stars and 565 non-exoplanet-stars.

1.6 Print the FeatureInfo of Exoplanets and Non-Exoplanets by using featuresInfo method in Exoplanets_DeepSpace()

```
print("Shape of explonets from train and test data's info :: ",exoplanets.featuresInfo(onlyE
print("Shape of non-explonets from train and test data's info :: ", exoplanets.featuresInfo
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 37 entries, 0 to 36
Columns: 3198 entries, LABEL to FLUX.3197
dtypes: float64(3197), int64(1)
memory usage: 924.7 KB
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5 entries, 0 to 4
Columns: 3198 entries, LABEL to FLUX.3197
dtypes: float64(3197), int64(1)
memory usage: 125.0 KB
Shape of explonets from train and test data's info ::  None None
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5050 entries, 37 to 5086
Columns: 3198 entries, LABEL to FLUX.3197
dtypes: float64(3197), int64(1)
memory usage: 123.3 MB
<class 'pandas.core.frame.DataFrame'>
Int64Index: 565 entries, 5 to 569
Columns: 3198 entries, LABEL to FLUX.3197
dtypes: float64(3197), int64(1)
memory usage: 13.8 MB
Shape of non-explonets from train and test data's info ::  None None
```

1.7 Describe Exoplanets and Non-Exoplanets data by using describeStats method in Exoplanets_DeepSpace()

```
print("Stats of explonets from train and test data's stats description :: ", exoplanets.desc
print("Stats of non-explonets from train and test data's stats description :: ", exoplanets
```

- Dependent features can be used as integer as it is or it can be converted into categorical column
- Independent features are in float data type, and there is no need for data type transformation

1.8 Print first 5 columns of train and test data by using headValues method in Exoplanets_DeepSpace()

```
print("explonets from train and test", exoplanets.headValues(onlyExoplanetsTrain), exoplane
print("non-explonets from train and test", exoplanets.headValues(onlyNonExoplanetsTrain), e
```

```
print(exoplanets.sampleRandomValues(onlyExoplanetsTrain), exoplanets.sampleRandomValues(onl
print(exoplanets.sampleRandomValues(onlyNonExoplanetsTrain), exoplanets.sampleRandomValues(
```

2. EDA and different Visualizations

2.1 Value count of Exo and Non-Exo planets in Train set

```
exoTrain.LABEL.value_counts(normalize=True)
```

```
1    0.992727
2    0.007273
Name: LABEL, dtype: float64
```

2.2 Value count of Exo and Non-Exo planets in Test set

```
exoTest.LABEL.value_counts(normalize=True)
```

```
1    0.991228
2    0.008772
Name: LABEL, dtype: float64
```

From 2.1 and 2.2, 99.3% in train set and 99.1% in test set are Non-Exoplanets
This is highly imbalanced dataset which requires data sampling techniques

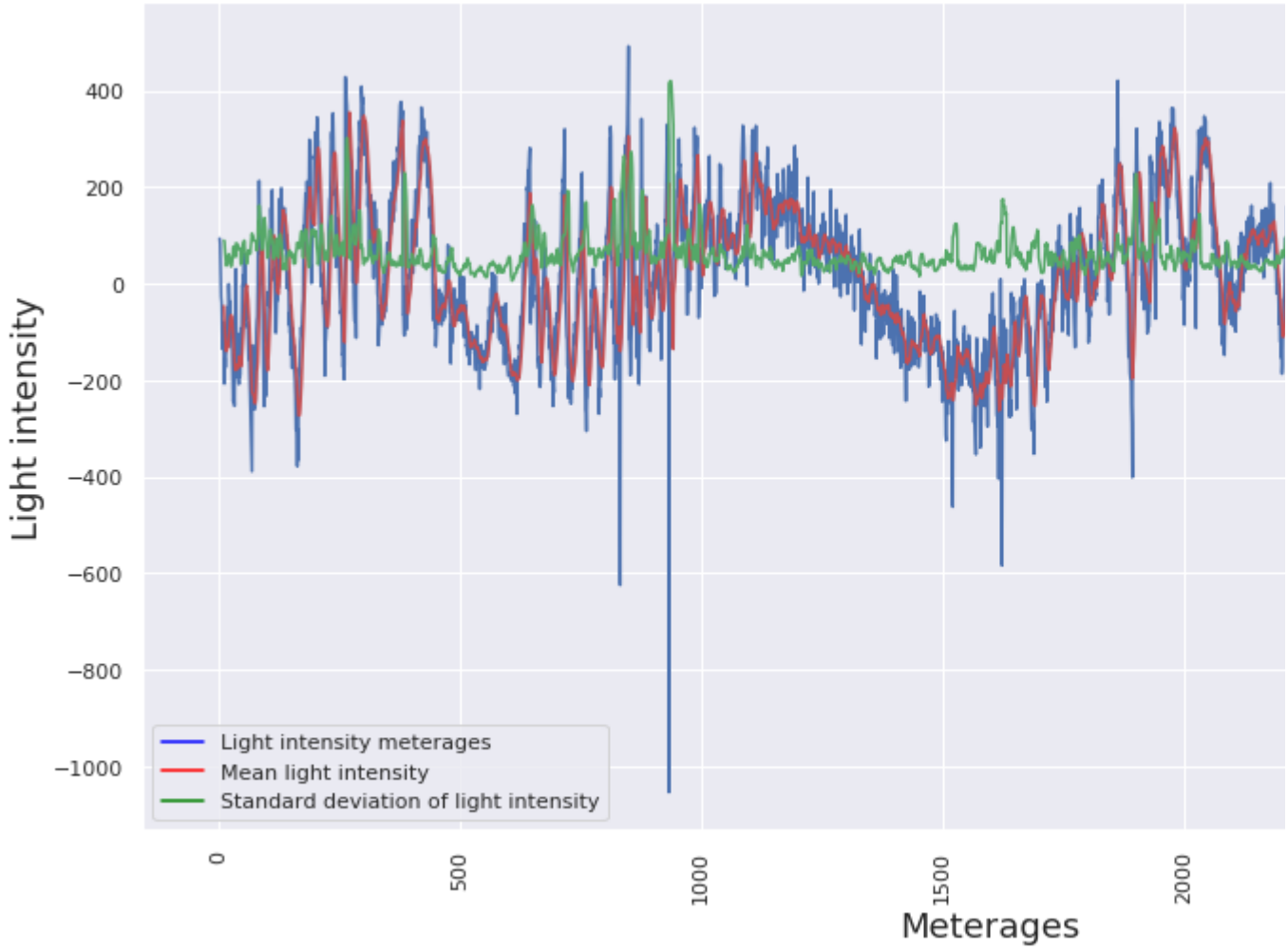
2.3 Split the Train and Test data into X,Y variables

```
X_train = exoTrain.loc[:, exoTrain.columns != 'LABEL'].values
y_train = exoTrain.LABEL.values

X_test = exoTest.loc[:, exoTest.columns != 'LABEL'].values
y_test = exoTest.LABEL.values
```

2.4 Light Intensity Plot

```
series_number = list(y_train).index(2)
print("Number of plotted series: ", series_number)

exoplanets.plot_light_intensity(X_train[series_number], resolution=1.0)
```

```
Number of plotted series:  0
```



The above graph provides insights on

1. Normal light intensity meterages with frequently distributed over with resolution of 1.0
2. Mean light intensity values shows that everything within range for each instances in the light intensity metera
3. Standard deviation light intensity mostly above zero value, shows that +/- square root values on each instanc

2.5 Planets Decomposed Season Plot

```
decomposition = exoplanets.seasonal_decompose_fft(X_train[series_number], freq=900)
exoplanets.plot_decomposed_seasonal(decomposition)
```

The above graph shows the below insights

1. Original graph shows how light intensity distributed on train dataset for frequency of 900.
2. Seasonality graph shows seasonal trending pattern over the original graph and smoothening the signals in de
3. Residual graph shows the inverse instance values of original graph and smoothened seasonal decomposed g
4. Fourier decomposed graph shows at the beginning and at the end, distorted frequency wavelengths are availa

```
X_dec = [exoplanets.seasonal_decompose_fft(X_train[i], freq=900) for i in range(0, len(X_tra
X_test_dec = [exoplanets.seasonal_decompose_fft(X_test[i], freq=900) for i in range(0, len(X
```

```
X_fft = absolute = [np.abs(X.fft[:(len(X.fft)//2)]) for X in X_dec]
X_test_fft = [np.abs(X.fft[:(len(X.fft)//2)]) for X in X_test_dec]
```

```
X_fft = normalized = normalize(X_fft)
X_test_fft = normalize(X_test_fft)
```

```
X_fft = filtered = ndimage.filters.gaussian_filter(X_fft, sigma=10)
X_test_fft = ndimage.filters.gaussian_filter(X_test_fft, sigma=10)
```

```
std_scaler = StandardScaler()
X_fft = scaled = std_scaler.fit_transform(X_fft)
X_test_fft = std_scaler.fit_transform(X_test_fft)
```

```
exoplanets.plot_tight_layout_fft(absolute,normalized,filtered,scaled,series_number)
```

The above graph insights are

1. Absolute original graph shows normal one
2. Normalized graph shows, smoothening signals
3. Filtered graph shows, normalized graphs touch points
4. Scaled graph provides seasonal trending type of pattern; the pattern of significant instances exponencially de points it stays in the same level (horizontal view);

```
X_train = exoTrain.drop('LABEL', axis=1)
Y_train = exoTrain.LABEL
x_test = exoTest.drop('LABEL', axis=1)
```

```
print(X_train.shape, Y_train.shape, x_test.shape)
```

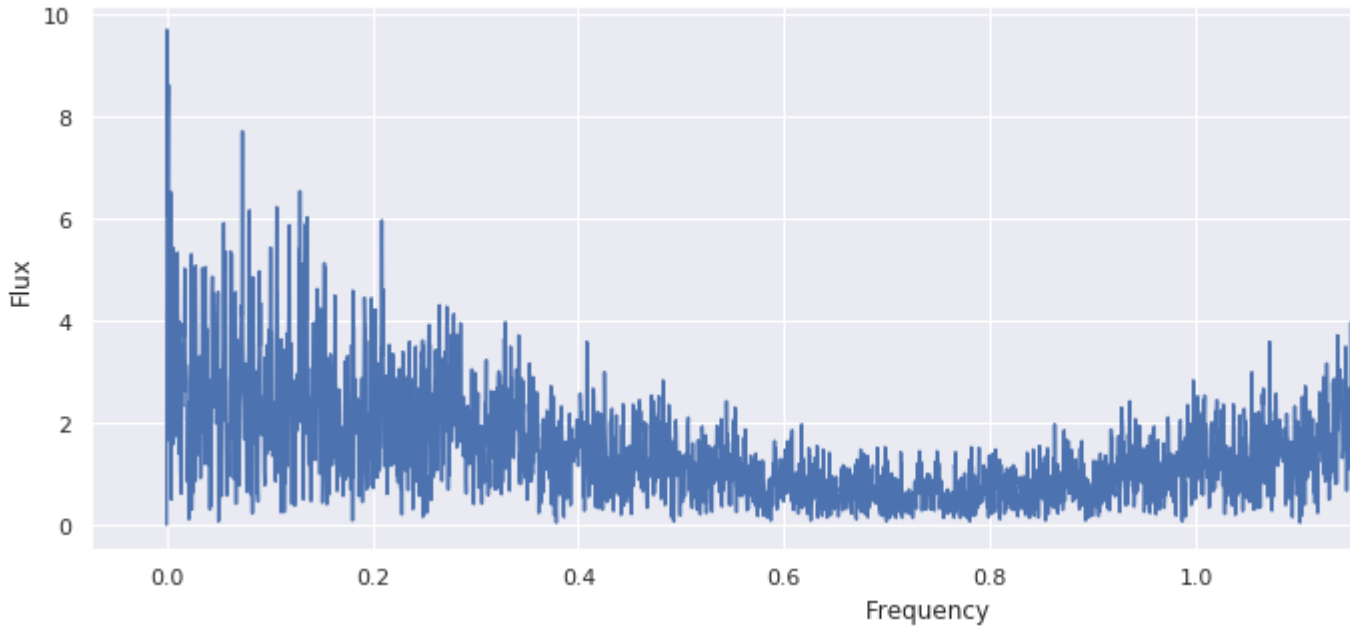
```
X_train= X_train.apply(exoplanets.normal,axis=1)

X_test = x_test.apply(exoplanets.normal, axis=1)

X_train_fft_1 = X_train.apply(exoplanets.fast_fourier_transf, axis=1)

X_test_fft_1 = X_test.apply(exoplanets.fast_fourier_transf,axis=1)

X_train_stft = X_train.apply(exoplanets.shorttime_fourier_transf, axis=1)

X_test_stft = X_test.apply(exoplanets.shorttime_fourier_transf,axis=1)

X_train_stft.shape
```

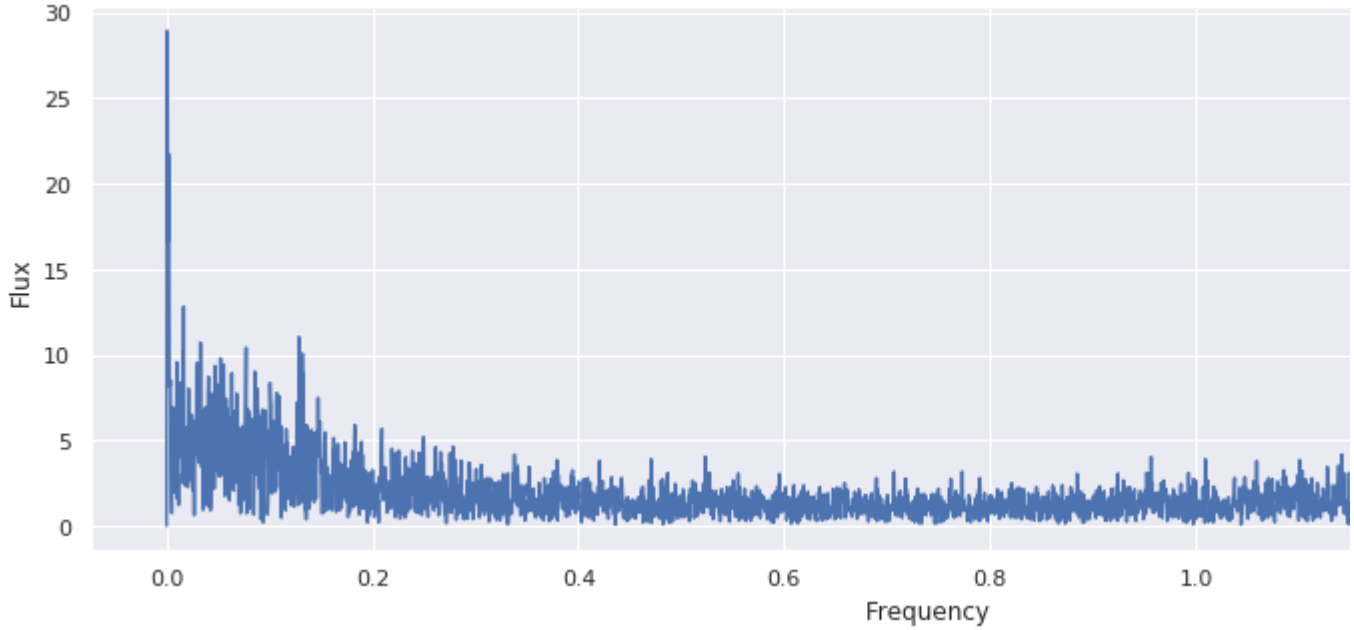
```
X_train_stft.head()
```

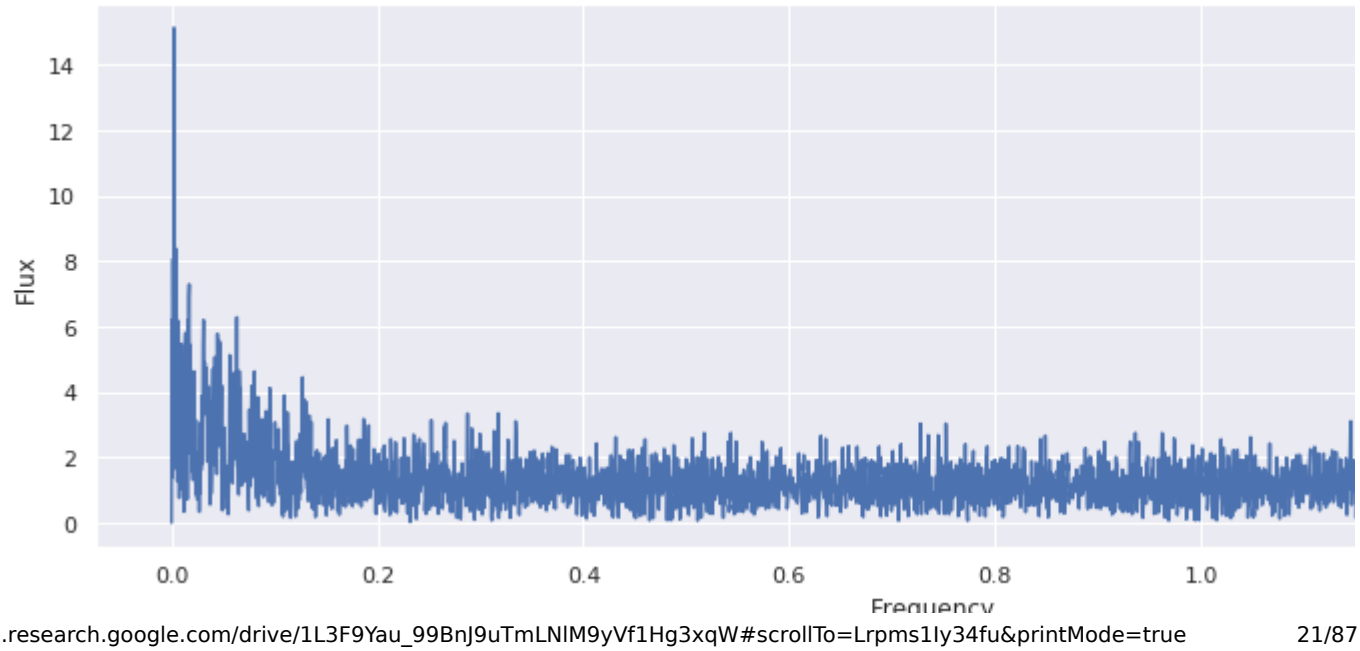
```
exoplanets.fluxFreq_ExoplanetTestPlot(X_train_fft_1)
```

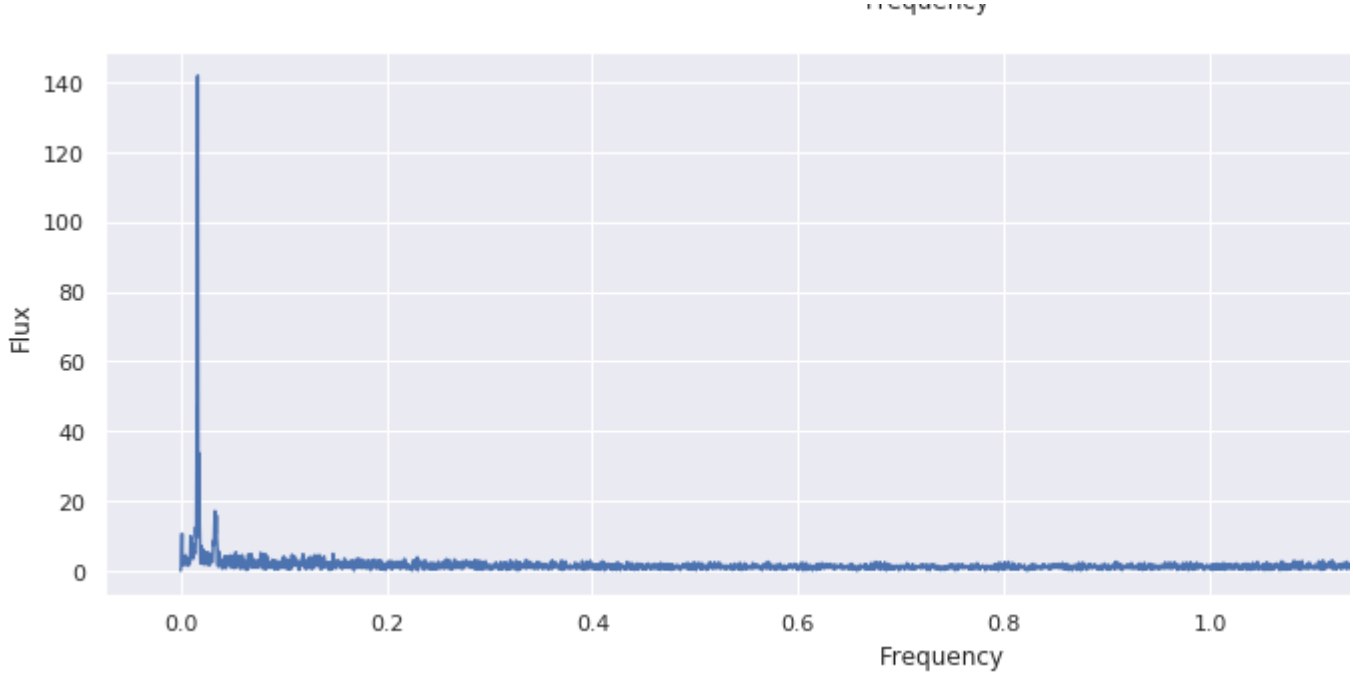
This above graph show Flux frequency range on exoplanets

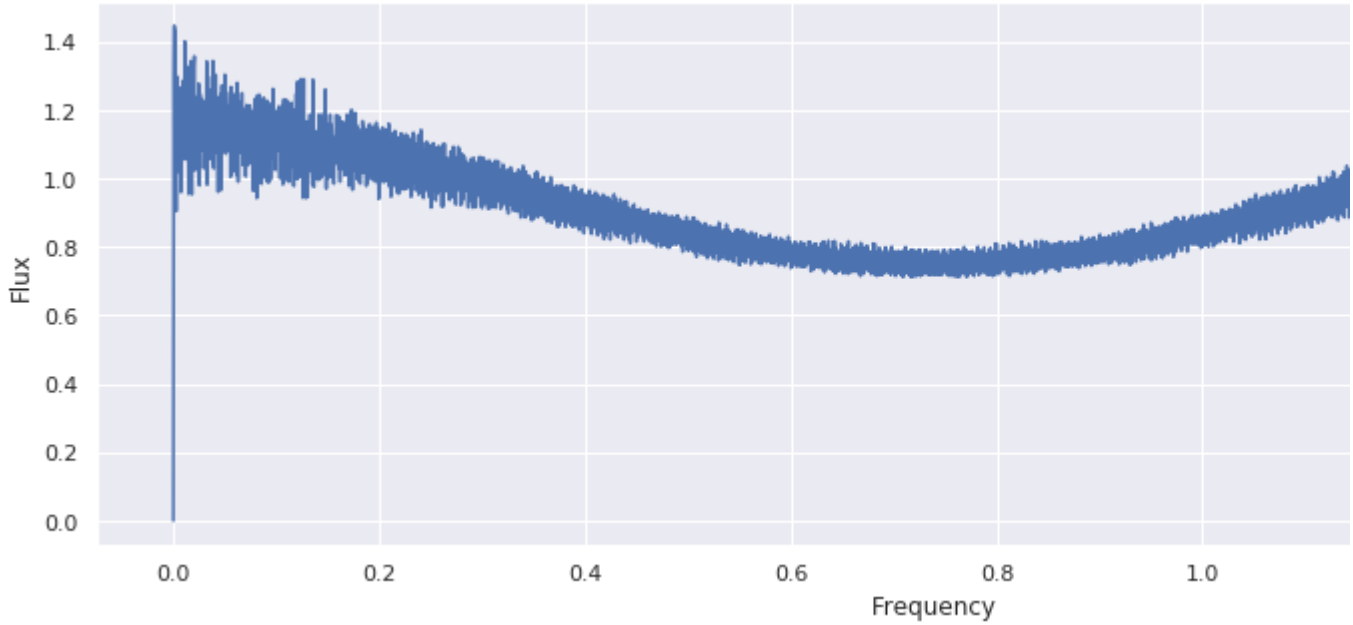
```
exoplanets.fluxFreq_NonExoplanets(X_train_fft_1)
```

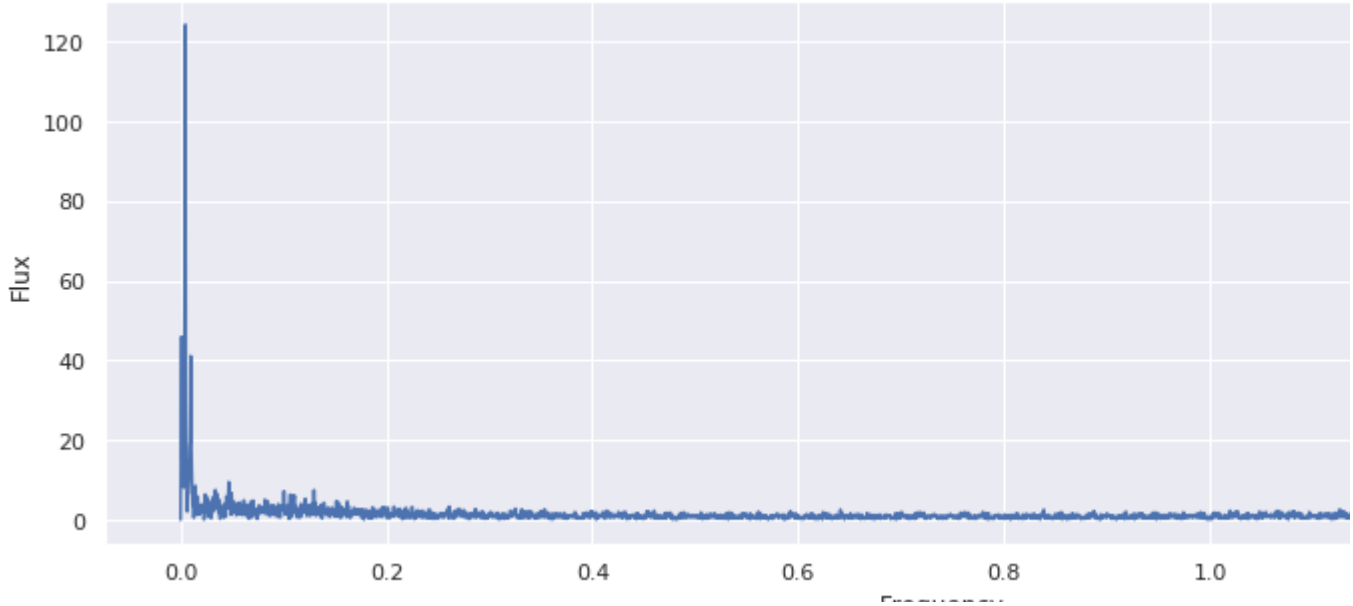

Flux
Frequency
0
2
4
6
8
10
0.0
0.2
0.4
0.6
0.8
1.0

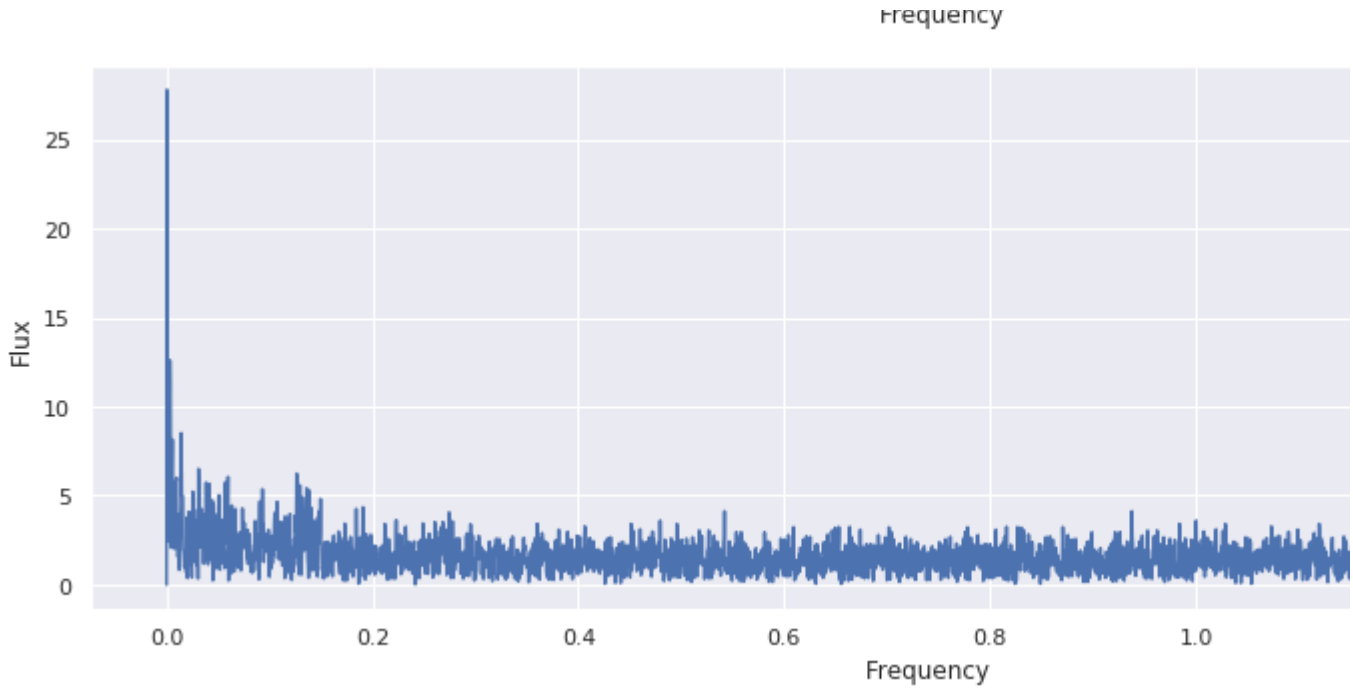

Flux
Frequency
0
5
10
15
20
25
30
0.0
0.2
0.4
0.6
0.8
1.0


Flux
Frequency
0
2
4
6
8
10
12
14
0.0
0.2
0.4
0.6
0.8
1.0


Flux
Frequency
0
20
40
60
80
100
120
140
0.0
0.2
0.4
0.6
0.8
1.0


Flux
Frequency
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4


Flux
0
20
40
60
80
100
120
0.0
0.2
0.4
0.6
0.8
1.0



This above graph show Flux frequency range on non-exoplanets

```
exoplanets.ploCompleteTestDataGraph(exoTest)
```

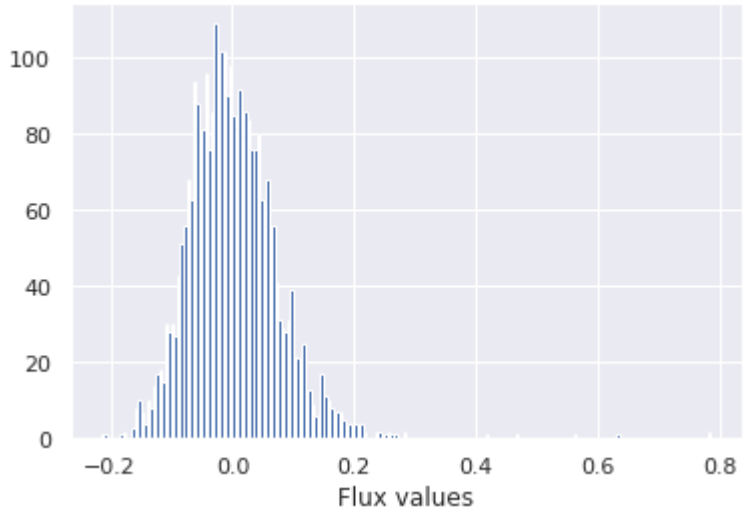
The above graph are test data values,

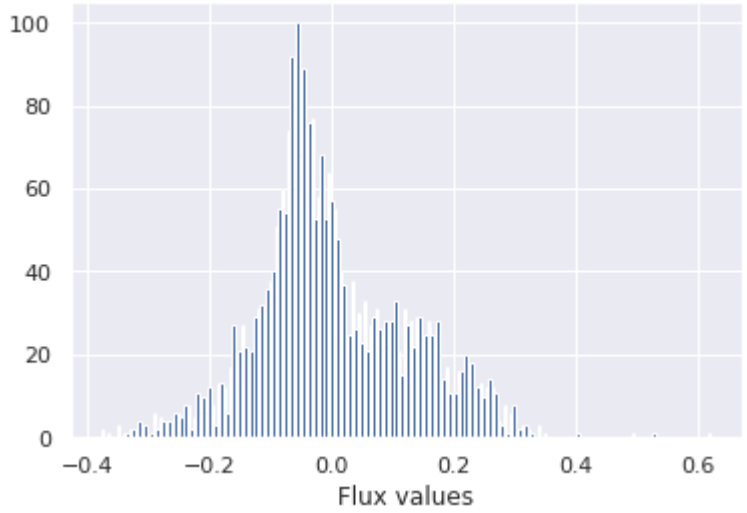
- Blue graph shows light intensity value of exoplanets, which has less data and hence frequency distribution de
- Red graph shows light intensity value of non-exoplanets, which has the slightly huge data and hence the frequ

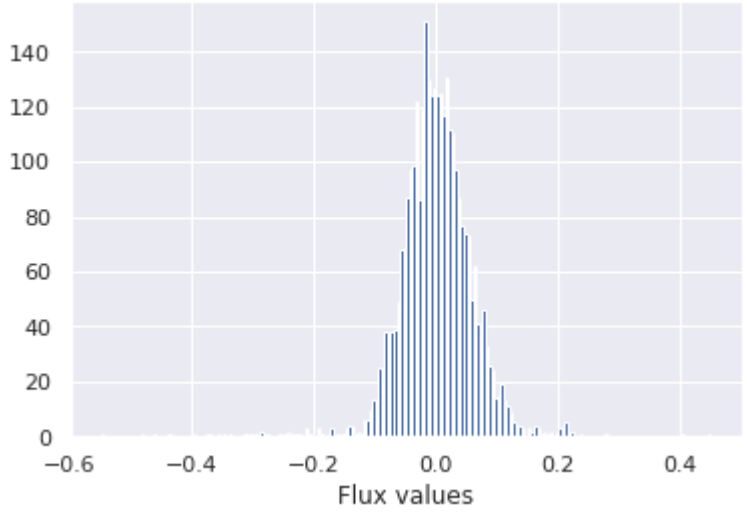
```
start=1
step=1
num=37

result=np.arange(0,num)*step+start
exoplanets.gaussianHistExoplanet(X_train,result)
```

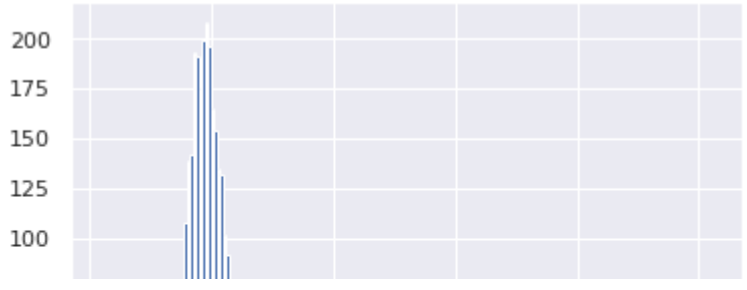
The above diagram shows gaussian histogram plots for first 37 exoplanets on train dataset

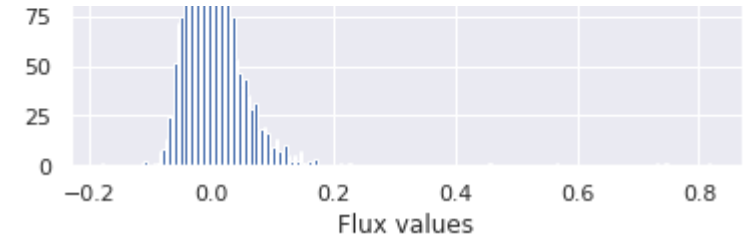
```
labels_1=[38,39,40,41,42,43,44,45]
exoplanets.gaussianHistExoplanet(X_test,labels_1)
```

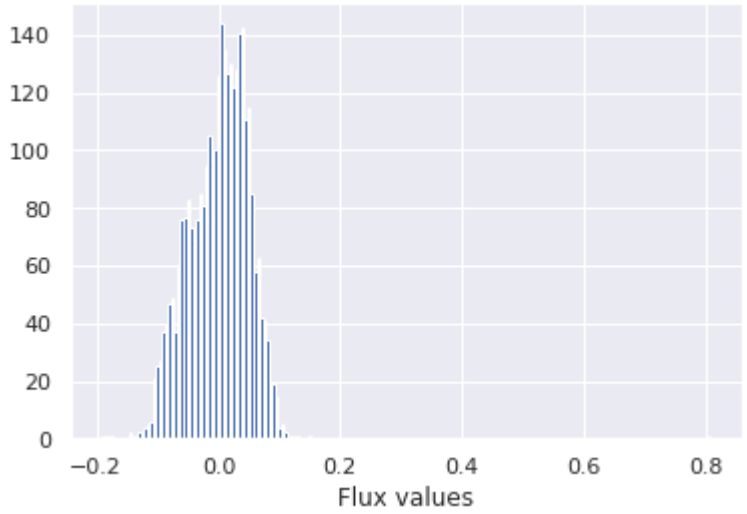

100
80
60
40
20
0
−0.2
0.0
0.2
0.4
0.6
0.8
Flux values

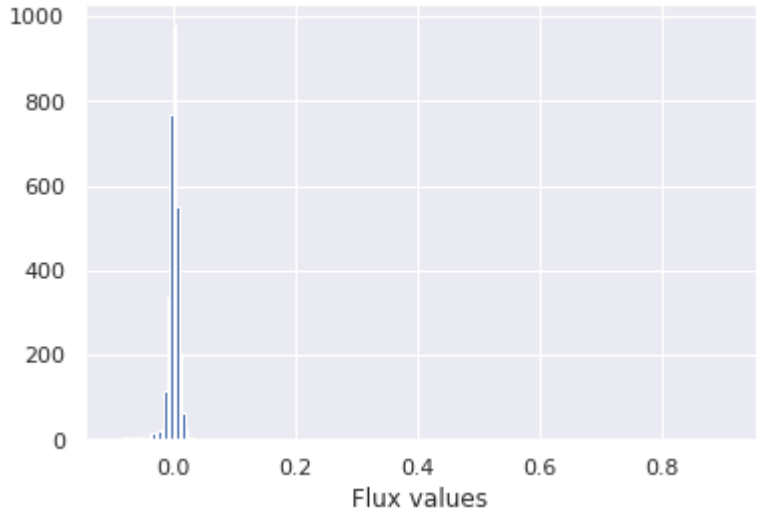

100
80
60
40
20
0
−0.4
−0.2
0.0
0.2
0.4
0.6
Flux values

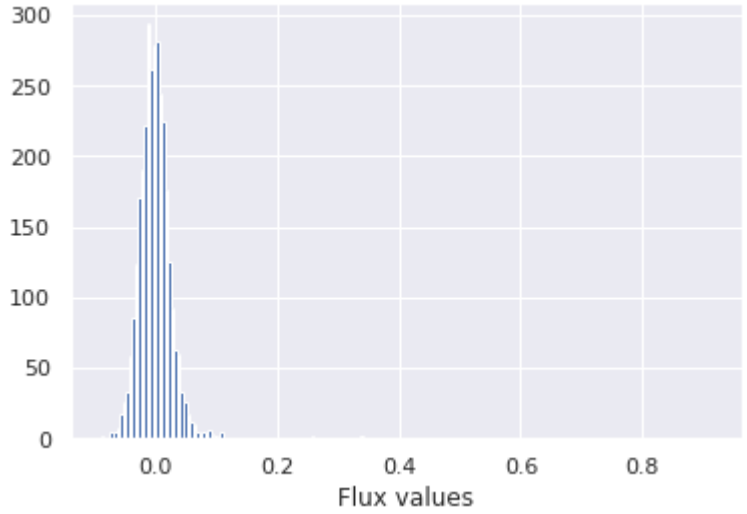

140
120
100
80
60
40
20
0
−0.6
−0.4
−0.2
0.0
0.2
0.4
Flux values


200
175
150
125
100

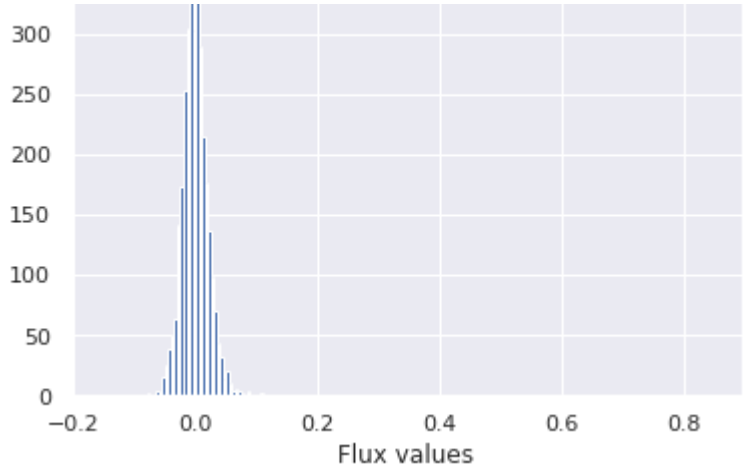

75
50
25
0
−0.2
0.0
0.2
0.4
0.6
0.8
Flux values


140
120
100
80
60
40
20
0
−0.2
0.0
0.2
0.4
0.6
0.8
Flux values


1000
800
600
400
200
0
0.0
0.2
0.4
0.6
0.8
Flux values


300
250
200
150
100
50
0
0.0
0.2
0.4
0.6
0.8
Flux values


350



The above diagram shows gaussian histogram plots for 7 non-exoplanets on test dataset

3. Classical Machine Learning models

3.1 Applying SGD Classifier for Linear model

```
exoplanets.linearML_SGDClassifier(X_fft,y_train,y_test, X_test_fft)
```

```
10100
Train accuracy = 1.0000
Test accuracy = 0.9947
Confusion Matrix (train sample):
 [[5050    0]
 [   0 5050]]
Confusion Matrix (test sample):
 [[565   0]
 [  3   2]]


Classification_report (train sample):
              precision    recall  f1-score   support

           1       1.00      1.00      1.00      5050
           2       1.00      1.00      1.00      5050

    accuracy                           1.00     10100
   macro avg       1.00      1.00      1.00     10100
weighted avg       1.00      1.00      1.00     10100

Classification_report (test sample):
              precision    recall  f1-score   support

           1       0.99      1.00      1.00       565
           2       1.00      0.40      0.57         5

    accuracy                           0.99       570
   macro avg       1.00      0.70      0.78       570
weighted avg       0.99      0.99      0.99       570
```

Accuracy score from Linear model with SGD Classifier is 99%

3.2 Decision tree

```
X = exoTrain.drop('LABEL', axis=1)
Y = exoTrain.pop('LABEL')
X_train, X_test, y_train, y_test = train_test_split(X, Y, train_size=0.7, random_state=1)
```

```
dtc1 = exoplanets.decisionTreeML(X_train,y_train,X_test,y_test, max_depth_val=5, max_leaf_n
```

Accuracy score from Decision Tree model is 99%

3.3 Random Forest

```
random_forest = RandomForestClassifier()
```

```
random_forest.fit(X_train, y_train)
```

```
y_predict = random_forest.predict(X_test)
accuracy_score(y_test, y_predict)
```

Accuracy score from Random Forest model is 99%

```
pd.DataFrame(
    confusion_matrix(y_test, y_predict),
    columns=['Predicted NonExoplanet', 'Predicted Exoplanet'],
    index=['True NonExoplanet', 'True Exoplanet']
)
```

```
exoplanets.evaluateModelsByBoxPlot(dtc1,random_forest,model_selection)
```

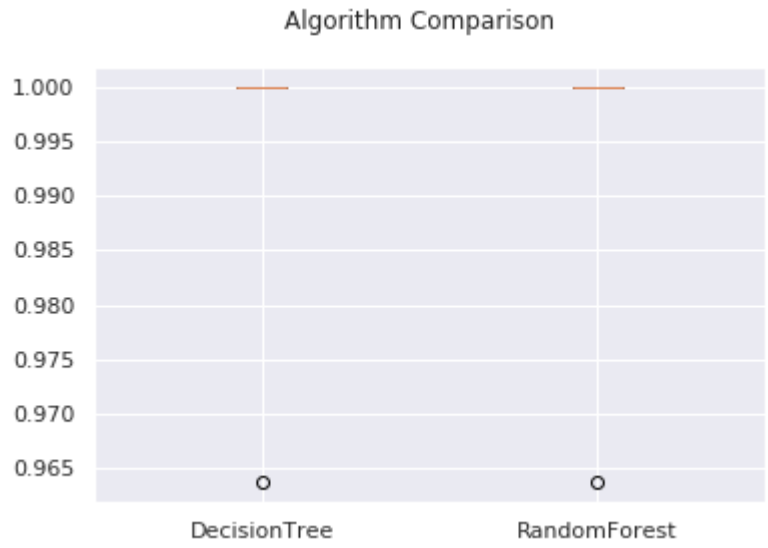
Accuracy score comparision between DecisionTree and Random Forest model are 99.27%

```
dtc2 = exoplanets.decisionTreeML(X_train,y_train,X_test,y_test, max_depth_val=5, max_leaf_n
```

```
Test accuracy = 0.9915
Confusion Matrix (test sample):
 [[1514    0]
 [  13    0]]


Classification_report (test sample):
              precision    recall  f1-score   support

           1       0.99      1.00      1.00      1514
           2       0.00      0.00      0.00        13

    accuracy                           0.99      1527
   macro avg       0.50      0.50      0.50      1527
weighted avg       0.98      0.99      0.99      1527

/conda/envs/rapids/lib/python3.6/site-packages/sklearn/metrics/classification.py
  'precision', 'predicted', average, warn_for)
/conda/envs/rapids/lib/python3.6/site-packages/sklearn/metrics/classification.py
  'precision', 'predicted', average, warn_for)
/conda/envs/rapids/lib/python3.6/site-packages/sklearn/metrics/classification.py
  'precision', 'predicted', average, warn_for)
```

```
exoplanets.evaluateModelsByBoxPlot(dtc2,random_forest,model_selection)
```

```
DecisionTree: 0.992731 (0.014538)
RandomForest: 0.992731 (0.014538)
```

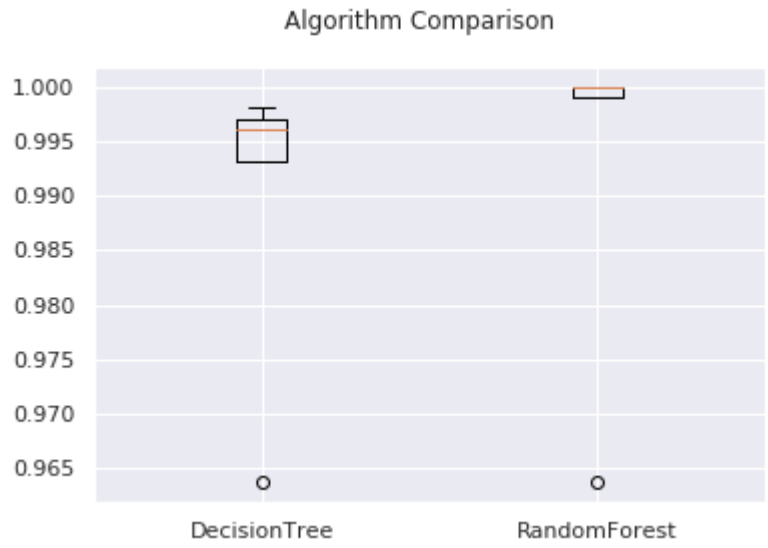



```
dtc3 = exoplanets.decisionTreeML(X_train,y_train,X_test,y_test, max_depth_val=10, max_leaf_
```

```
Test accuracy = 0.9856
Confusion Matrix (test sample):
 [[1504   10]
 [  12    1]]


Classification_report (test sample):
              precision    recall  f1-score   support

           1       0.99      0.99      0.99      1514
           2       0.09      0.08      0.08        13

    accuracy                           0.99      1527
   macro avg       0.54      0.54      0.54      1527
weighted avg       0.98      0.99      0.98      1527
```

```
exoplanets.evaluateModelsByBoxPlot(dtc3,random_forest,model_selection)
```

```
DecisionTree: 0.989585 (0.013069)
RandomForest: 0.992534 (0.014445)
```



3.4 Naive Bayes ML Model

```
X = exoTest.drop('LABEL', axis=1)
Y = exoTest.pop('LABEL')
train_set, test_set, train_labels, test_labels = train_test_split(X, Y, test_size=0.30, ran
```

```
model_gnb1 = exoplanets.gaussianNaiveBayesML(train_set, test_set, train_labels, test_labels
```

```
model_gnb2 = exoplanets.gaussianNaiveBayesML(train_set, test_set, train_labels, test_labels
```

Accuracy score from Naive Bayes model is 99.52%

```
model_gnb3 = exoplanets.gaussianNaiveBayesML(train_set, test_set, train_labels, test_labels
```

3.5 Boosting models

```
exoTrain = exoplanets.readCsv_File('../webapp/web/exoTrain.csv')
X_train = exoTrain.drop('LABEL', axis=1)
Y_train = exoTrain.pop('LABEL')
exoTest = exoplanets.readCsv_File('../webapp/web/exoTest.csv')
X_test = exoTest.drop('LABEL', axis=1)
```

```
Y_test = exoTest.pop('LABEL')


adaModel = exoplanets.adaBoostclassifierML(dtc1,X_train,Y_train,X_test,Y_test, 51 )
```

```
xgBoostModel = exoplanets.xgBoostClassifier(X_train,Y_train,X_test,Y_test)
```

```
Target on train data [2 2 2 ... 1 1 1]

accuracy_score on train dataset :  0.9998034204835856

Target on test data [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]

accuracy_score on test dataset :  0.9912280701754386
```

3.6 PCA (Principle Component Analysis)

```
sc = StandardScaler()
X_train_std =  sc.fit_transform(X_train)
train_cov_matrix = np.cov(X_train_std.T)
```

```
print('Covariance Matrix \n%s', train_cov_matrix)
```

```
Covariance Matrix
%s [[1.00019662 0.99648265 0.98994802 ... 0.33334625 0.33561797 0.33910376]
 [0.99648265 1.00019662 0.99576633 ... 0.33932407 0.34762696 0.35433145]
 [0.98994802 0.99576633 1.00019662 ... 0.33944074 0.34174613 0.34569843]
 ...
 [0.33334625 0.33932407 0.33944074 ... 1.00019662 0.98010694 0.92401911]
 [0.33561797 0.34762696 0.34174613 ... 0.98010694 1.00019662 0.9782549 ]
 [0.33910376 0.35433145 0.34569843 ... 0.92401911 0.9782549  1.00019662]]
```

```
eigenvalues, eigenvectors = np.linalg.eig(train_cov_matrix)
print('Eigen Vectors \n%s', eigenvectors)
print('\n Eigen Values \n%s', eigenvalues)
```

```
Eigen Vectors
%s [[ 2.10555965e-02 -2.47900939e-02 -7.95281724e-03 ...  2.36094576e-02
    1.47786583e-02  4.56383596e-03]
 [ 2.09636223e-02 -2.46964372e-02 -6.56299679e-03 ... -1.14957671e-02
    3.86093470e-03 -2.19082736e-03]
 [ 2.10079508e-02 -2.49033656e-02 -5.13363277e-03 ...  7.29935128e-03
    1.72382173e-03 -3.14793604e-02]
 ...
 [ 1.93854707e-03 -9.46307602e-05 -2.07973701e-02 ... -1.01179397e-02
    1.89120080e-03  1.68834207e-02]
 [ 2.07232905e-03  5.37875713e-04 -1.84585090e-02 ...  9.89091174e-03
   -2.34543649e-02  1.85550992e-02]
 [ 2.90384726e-03  8.16460510e-04 -1.77480552e-02 ... -6.45450410e-03
    3.01559162e-02 -1.97643349e-02]]

 Eigen Values
%s [8.75304099e+02 7.20624569e+02 4.77378032e+02 ... 8.99503196e-08
 9.01052390e-08 9.01369200e-08]
```

```
# Step 3 (continued): Sort eigenvalues in descending order

# Make a set of (eigenvalue, eigenvector) pairs
train_eig_pairs = [(eigenvalues[index], eigenvectors[:,index]) for index in range(len(eigen

# Sort the (eigenvalue, eigenvector) pairs from highest to lowest with respect to eigenvalu
train_eig_pairs.sort()

train_eig_pairs.reverse()
print(train_eig_pairs)

# Extract the descending ordered eigenvalues and eigenvectors
train_eigvalues_sorted = [train_eig_pairs[index][0] for index in range(len(eigenvalues))]
train_eigvectors_sorted = [train_eig_pairs[index][1] for index in range(len(eigenvalues))]

# Let's confirm our sorting worked, print out eigenvalues
print('Eigenvalues in descending order: \n%s' %train_eigvalues_sorted)
```

```
tot = sum(eigenvalues)
var_explained = [(i / tot) for i in sorted(train_eigvalues_sorted, reverse=True)]  # an arra
# eigen vector... there will be 8 entries as there are 8 eigen vectors)
cum_var_exp = np.cumsum(var_explained)  # an array of cumulative variance. There will be 8 e
# cumulative reaching almost 100%
```

```
#plt.bar(range(1,8), var_explained, alpha=0.5, align='center', label='individual explained v
#plt.step(range(1,8),cum_var_exp, where= 'mid', label='cumulative explained variance')
#plt.ylabel('Explained variance ratio')
#plt.xlabel('Principal components')
#plt.legend(loc = 'best')
#plt.show()
```

PCA with FLUX 1

```
train = exoTrain.rename(index=str, columns={"FLUX.1":"FLUX_1"})
```

```
X = train
scaler = StandardScaler(copy=True, with_mean=True, with_std=True)
#Fit the scaler to train.data
scaler.fit(X)
# call scaler.transform() on train.data and store the result in scaled_X
scaled_X = scaler.transform(X)
```

```
# Store the labels contained in train.targets below
# X['FLUX_1'].astype(int)
labels = train.FLUX_1

# Create a PCA() object
pca = PCA()

#Fit the pca object to scaled_X
pca.fit(scaled_X)

# Call pca.transform() on scaled_X and store the results below
X_with_pca = pca.transform(scaled_X)

variance = pca.explained_variance_ratio_
```

```
variance
```

4. Deep Leanring Model : CNN 1D with different optimizers, learning rate, decay rate and GridSearchCV

We are standardizing the data row wise

```
#INPUT_LIB = '/content/drive/My Drive/capstone_datasets/'
INPUT_LIB = '../webapp/web/'
raw_data = np.loadtxt(INPUT_LIB + 'exoTrain.csv', skiprows=1, delimiter=',')
x_train = raw_data[:, 1:]
y_train = raw_data[:, 0, np.newaxis] - 1.
raw_data = np.loadtxt(INPUT_LIB + 'exoTest.csv', skiprows=1, delimiter=',')
x_test = raw_data[:, 1:]
y_test = raw_data[:, 0, np.newaxis] - 1.
del raw_data
```

```
print(x_train.shape, x_test.shape)
x_train = ((x_train - np.mean(x_train, axis=1).reshape(-1,1)) / np.std(x_train, axis=1).resl
x_test = ((x_test - np.mean(x_test, axis=1).reshape(-1,1)) / np.std(x_test, axis=1).reshape
print(x_train.shape, x_test.shape)
x_train = np.stack([x_train, uniform_filter1d(x_train, axis=1, size=200)], axis=2)
x_test = np.stack([x_test, uniform_filter1d(x_test, axis=1, size=200)], axis=2)
print(x_train.shape, x_test.shape)
```

Since this is a highly imbalance dataset,

- we are using batch data generator to synthesize data in deep neural network.
- accuracy will be high even for pool model performance. Hence we would like to use other metrics like f1, reca

```
def batch_generator(x_train, y_train, batch_size=32):
    """
    Gives equal number of positive and negative samples, and rotates them randomly in time
    """
    half_batch = batch_size // 2
    x_batch = np.empty((batch_size, x_train.shape[1], x_train.shape[2]), dtype='float32')
    y_batch = np.empty((batch_size, y_train.shape[1]), dtype='float32')

    yes_idx = np.where(y_train[:,0] == 1.)[0]
    non_idx = np.where(y_train[:,0] == 0.)[0]

    while True:
        np.random.shuffle(yes_idx)
        np.random.shuffle(non_idx)

        x_batch[:half_batch] = x_train[yes_idx[:half_batch]]
        x_batch[half_batch:] = x_train[non_idx[half_batch:batch_size]]
        y_batch[:half_batch] = y_train[yes_idx[:half_batch]]
        y_batch[half_batch:] = y_train[non_idx[half_batch:batch_size]]

        for i in range(batch_size):
            sz = np.random.randint(x_batch.shape[1])
            x_batch[i] = np.roll(x_batch[i], sz, axis = 0)

        yield x_batch, y_batch

        from keras import backend as K

def recall_m(y_true, y_pred):
        true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
        possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
        recall = true_positives / (possible_positives + K.epsilon())
        return recall

def precision_m(y_true, y_pred):
        true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
        predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
        precision = true_positives / (predicted_positives + K.epsilon())
        return precision

def f1_m(y_true, y_pred):
    precision = precision_m(y_true, y_pred)
    recall = recall_m(y_true, y_pred)
    return 2*((precision*recall)/(precision+recall+K.epsilon()))

# metrics=['accuracy', f1_m,precision_m, recall_m])
# metrics=[f1_m,precision_m, recall_m])
```

Model one - base model: Uses CNN Conv1D with max pooling, batch norm, fixed kernel and filter size and drop out, c
layers and Sigmoid in the final dense layer, Adam optimizer with fixed learning rate using accuracy, f1, precision, rec

```
model_one = Sequential()
model_one.add(Conv1D(filters=8, kernel_size=11, activation='relu', input_shape=x_train.shape
model_one.add(MaxPool1D(strides=4))
model_one.add(BatchNormalization())
model_one.add(Conv1D(filters=16, kernel_size=11, activation='relu'))
model_one.add(MaxPool1D(strides=4))
model_one.add(BatchNormalization())
model_one.add(Conv1D(filters=32, kernel_size=11, activation='relu'))
model_one.add(MaxPool1D(strides=4))
model_one.add(BatchNormalization())
model_one.add(Conv1D(filters=64, kernel_size=11, activation='relu'))
model_one.add(MaxPool1D(strides=4))
model_one.add(Flatten())
model_one.add(Dropout(0.5))
model_one.add(Dense(64, activation='relu'))
model_one.add(Dropout(0.25))
model_one.add(Dense(64, activation='relu'))
model_one.add(Dense(1, activation='sigmoid'))
```

```
WARNING:tensorflow:From /conda/envs/rapids/lib/python3.6/site-packages/tensorflo
Instructions for updating:
Colocations handled automatically by placer.
WARNING:From /conda/envs/rapids/lib/python3.6/site-packages/tensorflow/python/fra
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From /conda/envs/rapids/lib/python3.6/site-packages/keras/bacl
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_
WARNING:From /conda/envs/rapids/lib/python3.6/site-packages/keras/backend/tensor
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_
```

```
#Start with a slightly lower learning rate, to ensure convergence
model_one.compile(optimizer=Adam(1e-5), loss = 'binary_crossentropy', metrics=['accuracy',
hist = model_one.fit_generator(batch_generator(x_train, y_train, 32),
                           validation_data=(x_test, y_test),
                           verbose=2, epochs=5,
                           steps_per_epoch=x_train.shape[1]//32)
```

```
WARNING:tensorflow:From /conda/envs/rapids/lib/python3.6/site-packages/tensorflo
Instructions for updating:
Use tf.cast instead.
WARNING:From /conda/envs/rapids/lib/python3.6/site-packages/tensorflow/python/op
Instructions for updating:
Use tf.cast instead.
Epoch 1/5
 - 17s - loss: 0.7551 - acc: 0.4937 - f1_m: 0.4541 - precision_m: 0.4938 - recal
Epoch 2/5
 - 1s - loss: 0.7381 - acc: 0.5114 - f1_m: 0.4574 - precision_m: 0.5160 - recall_
Epoch 3/5
 - 1s - loss: 0.7296 - acc: 0.5240 - f1_m: 0.4839 - precision_m: 0.5273 - recall_
Epoch 4/5
 - 1s - loss: 0.7280 - acc: 0.5331 - f1_m: 0.4921 - precision_m: 0.5375 - recall_
Epoch 5/5
 - 1s - loss: 0.7290 - acc: 0.5183 - f1_m: 0.4773 - precision_m: 0.5226 - recall_
```

```
#speed things up a little
model_one.compile(optimizer=Adam(4e-5), loss = 'binary_crossentropy', metrics=['accuracy',
hist = model_one.fit_generator(batch_generator(x_train, y_train, 32),
                               validation_data=(x_test, y_test),
                               verbose=2, epochs=40,
                               steps_per_epoch=x_train.shape[1]//32)
```

```
model_one.summary()
```

```
#!pip install pydot
#import pydot
#plot_model(model_one, to_file='model.png', show_shapes=True, show_layer_names=True)
```

```
print(model_one.evaluate(x_train, y_train, batch_size = 100))
print('\nModel Performance: Loss and Accuracy on validation data')
print(model_one.evaluate(x_test, y_test, batch_size = 100))
```

```
plt.plot(hist.history['loss'], color='b')
plt.plot(hist.history['val_loss'], color='r')
plt.show()
plt.plot(hist.history['acc'], color='b')
plt.plot(hist.history['val_acc'], color='r')
plt.show()
```

```
non_idx = np.where(y_test[:,0] == 0.)[0]
yes_idx = np.where(y_test[:,0] == 1.)[0]
y_hat = model_one.predict(x_test)[:,0]
plt.plot([y_hat[i] for i in yes_idx], 'bo')
plt.show()
plt.plot([y_hat[i] for i in non_idx], 'ro')
plt.show()
```

```
y_true = (y_test[:, 0] + 0.5).astype("int")
fpr, tpr, thresholds = roc_curve(y_true, y_hat)
plt.plot(thresholds, 1.-fpr)
plt.plot(thresholds, tpr)
plt.show()
crossover_index = np.min(np.where(1.-fpr <= tpr))
crossover_cutoff = thresholds[crossover_index]
```

```
crossover_specificity = 1.-fpr[crossover_index]
print("Crossover at {0:.2f} with specificity {1:.2f}".format(crossover_cutoff, crossover_sp
plt.plot(fpr, tpr)
plt.show()
print("ROC area under curve is {0:.2f}".format(roc_auc_score(y_true, y_hat)))
```

```
#Using threshold as 0.9
y_hat = model_one.predict(x_test)[:,0]
y_pred = np.where(y_hat > 0.9,1.,0.)

# accuracy: (tp + tn) / (p + n)
print('Accuracy:', accuracy_score(y_test, y_pred))
# f1: 2 tp / (2 tp + fp + fn)
print('F1 score:', f1_score(y_test, y_pred))
# recall: tp / (tp + fn)
print('Recall:', recall_score(y_test, y_pred))
# precision tp / (tp + fp)
print('Precision:', precision_score(y_test, y_pred))
#cohen's kappa
print('Cohens Kappa :', cohen_kappa_score(y_test, y_pred))
#AUC score
print('ROC AUC Score:', roc_auc_score(y_test, y_pred))
#Classification report
print('\n clasification report:\n', classification_report(y_test,y_pred))
#Confusion matrix
print('\n confussion matrix:\n',confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.980701754385965
F1 score: 0.35294117647058826
Recall: 0.6
Precision: 0.25
Cohens Kappa : 0.3448275862068966
ROC AUC Score: 0.7920353982300885

 clasification report:
              precision    recall  f1-score   support

         0.0       1.00      0.98      0.99       565
         1.0       0.25      0.60      0.35         5

    accuracy                           0.98       570
   macro avg       0.62      0.79      0.67       570
weighted avg       0.99      0.98      0.98       570


 confussion matrix:
 [[556   9]
 [  2   3]]
```

```
#y_train_pred = model_two.predict_classes(x_train)[:,0]
y_train_hat = model_one.predict(x_train)[:,0]
y_train_pred = np.where(y_train_hat > 0.9,1.,0.)
print('Accuracy:', accuracy_score(y_train, y_train_pred))
print('F1 score:', f1_score(y_train, y_train_pred))
print('Recall:', recall_score(y_train, y_train_pred))
print('Precision:', precision_score(y_train, y_train_pred))
print('Cohens Kappa :', cohen_kappa_score(y_train, y_train_pred))
print('ROC AUC Score:', roc_auc_score(y_train, y_train_pred))
print('\n clasification report:\n', classification_report(y_train, y_train_pred))
print('\n confussion matrix:\n',confusion_matrix(y_train, y_train_pred))
```

```
Accuracy: 0.9703164930214272
F1 score: 0.16574585635359118
Recall: 0.40540540540540543
Precision: 0.10416666666666667
Cohens Kappa : 0.1559778088125252
ROC AUC Score: 0.68993042547498

 clasification report:
              precision    recall  f1-score   support

         0.0       1.00      0.97      0.98      5050
         1.0       0.10      0.41      0.17        37

    accuracy                           0.97      5087
   macro avg       0.55      0.69      0.58      5087
weighted avg       0.99      0.97      0.98      5087


 confussion matrix:
 [[4921  129]
 [  22   15]]
```

Model Two : is similar to Model one except for optimizer as SGD.

```
model_two = Sequential()
model_two.add(Conv1D(filters=8, kernel_size=11, activation='relu', input_shape=x_train.shape
model_two.add(MaxPool1D(strides=4))
model_two.add(BatchNormalization())
model_two.add(Conv1D(filters=16, kernel_size=11, activation='relu'))
model_two.add(MaxPool1D(strides=4))
model_two.add(BatchNormalization())
model_two.add(Conv1D(filters=32, kernel_size=11, activation='relu'))
model_two.add(MaxPool1D(strides=4))
model_two.add(BatchNormalization())
model_two.add(Conv1D(filters=64, kernel_size=11, activation='relu'))
model_two.add(MaxPool1D(strides=4))
model_two.add(Flatten())
model_two.add(Dropout(0.5))
model_two.add(Dense(64, activation='relu'))
model_two.add(Dropout(0.25))
model_two.add(Dense(64, activation='relu'))
model_two.add(Dense(1, activation='sigmoid'))
```

```
model_two.compile(optimizer="sgd", loss = 'binary_crossentropy', metrics=['accuracy', f1_m,
hist = model_two.fit_generator(batch_generator(x_train, y_train, 32),
                        validation_data=(x_test, y_test),
                        verbose=2, epochs=5,
                        steps_per_epoch=x_train.shape[1]//32)
```

```
model_two.compile(optimizer="sgd", loss = 'binary_crossentropy', metrics=['accuracy', f1_m,
hist = model_two.fit_generator(batch_generator(x_train, y_train, 32),
                               validation_data=(x_test, y_test),
                               verbose=2, epochs=80,
                               steps_per_epoch=x_train.shape[1]//32)
```

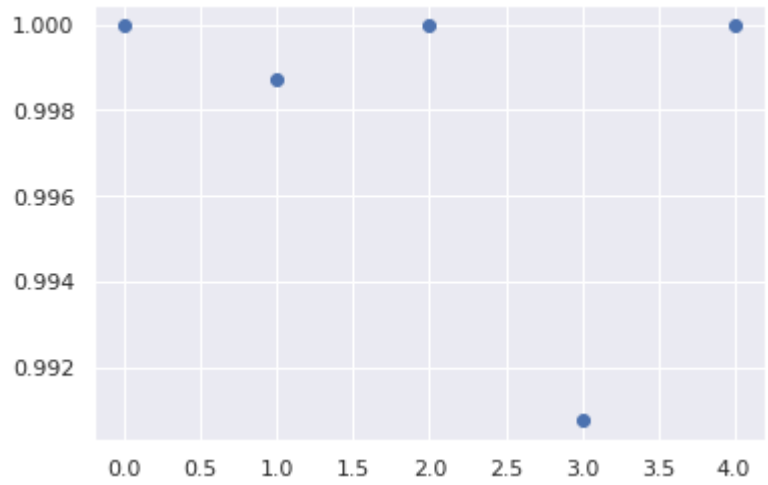
```
model_two.summary()
```

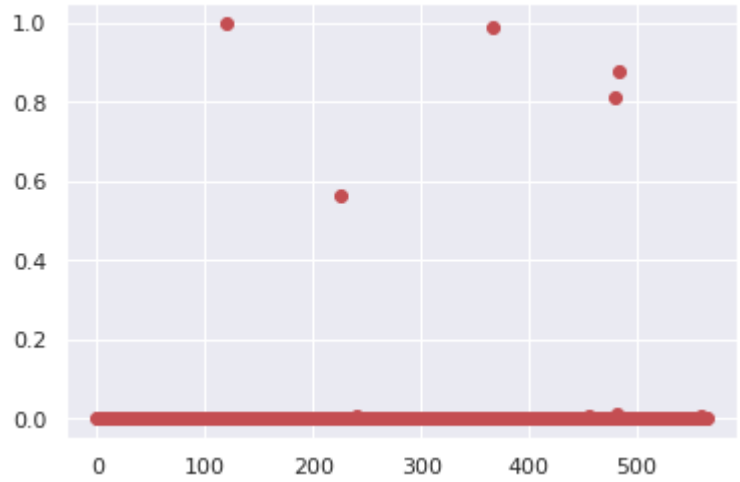
```
#plot_model(model_two, to_file='model.png', show_shapes=True, show_layer_names=True)


print(model_two.evaluate(x_train, y_train, batch_size = 100))
print('\nModel Performance: Loss and Accuracy on validation data')
print(model_two.evaluate(x_test, y_test, batch_size = 100))
```

```
plt.plot(hist.history['loss'], color='b')
plt.plot(hist.history['val_loss'], color='r')
plt.show()
plt.plot(hist.history['acc'], color='b')
plt.plot(hist.history['val_acc'], color='r')
plt.show()
```

```
non_idx = np.where(y_test[:,0] == 0.)[0]
yes_idx = np.where(y_test[:,0] == 1.)[0]
y_hat = model_two.predict(x_test)[:,0]
plt.plot([y_hat[i] for i in yes_idx], 'bo')
plt.show()
plt.plot([y_hat[i] for i in non_idx], 'ro')
```

```
plt.show()
```






```
y_true = (y_test[:, 0] + 0.5).astype("int")
fpr, tpr, thresholds = roc_curve(y_true, y_hat)
plt.plot(thresholds, 1.-fpr)
plt.plot(thresholds, tpr)
plt.show()
crossover_index = np.min(np.where(1.-fpr <= tpr))
crossover_cutoff = thresholds[crossover_index]
crossover_specificity = 1.-fpr[crossover_index]
print("Crossover at {0:.2f} with specificity {1:.2f}".format(crossover_cutoff, crossover_sp
plt.plot(fpr, tpr)
plt.show()
print("ROC area under curve is {0:.2f}".format(roc_auc_score(y_true, y_hat)))
```

```
#Using threshold as 0.9
y_hat = model_two.predict(x_test)[:,0]
y_pred = np.where(y_hat > 0.9,1.,0.)

# accuracy: (tp + tn) / (p + n)
print('Accuracy:', accuracy_score(y_test, y_pred))
# f1: 2 tp / (2 tp + fp + fn)
print('F1 score:', f1_score(y_test, y_pred))
# recall: tp / (tp + fn)
print('Recall:', recall_score(y_test, y_pred))
# precision tp / (tp + fp)
print('Precision:', precision_score(y_test, y_pred))
#cohen's kappa
print('Cohens Kappa :', cohen_kappa_score(y_test, y_pred))
#AUC score
print('ROC AUC Score:', roc_auc_score(y_test, y_pred))
#Classification report
print('\n clasification report:\n', classification_report(y_test,y_pred))
#Confusion matrix
print('\n confussion matrix:\n',confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.9964912280701754
F1 score: 0.8333333333333333
Recall: 1.0
Precision: 0.7142857142857143
Cohens Kappa : 0.8316100443131462
ROC AUC Score: 0.9982300884955752

 clasification report:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00       565
         1.0       0.71      1.00      0.83         5

    accuracy                           1.00       570
   macro avg       0.86      1.00      0.92       570
weighted avg       1.00      1.00      1.00       570


 confussion matrix:
 [[563   2]
 [  0   5]]
```

```
#y_train_pred = model_two.predict_classes(x_train)[:,0]
y_train_hat = model_two.predict(x_train)[:,0]
y_train_pred = np.where(y_train_hat > 0.9,1.,0.)
print('Accuracy:', accuracy_score(y_train, y_train_pred))
print('F1 score:', f1_score(y_train, y_train_pred))
print('Recall:', recall_score(y_train, y_train_pred))
print('Precision:', precision_score(y_train, y_train_pred))
print('Cohens Kappa :', cohen_kappa_score(y_train, y_train_pred))
print('ROC AUC Score:', roc_auc_score(y_train, y_train_pred))
print('\n clasification report:\n', classification_report(y_train, y_train_pred))
print('\n confussion matrix:\n',confusion_matrix(y_train, y_train_pred))
```

```
Accuracy: 0.9990171024179281
```

```
from keras.models import model_from_json
from keras.models import model_from_yaml

# Save model 1 - serialize model to JSON
final_model_json = model_two.to_json()
with open("final_model.json", "w") as json_file:
    json_file.write(final_model_json)


# Save model 1 - serialize model to YAML
model_yaml = model_two.to_yaml()
with open("final_model.yaml", "w") as yaml_file:
    yaml_file.write(model_yaml)

# Serialize weights to HDF5
model_two.save_weights("final_model_weights.h5")
print("Saved model to disk")
```

```
import pickle
# Save model using pickle
pickle.dump(model_two, open('final_model.pkl', 'wb'))


# Save model using keras
model_two.save("final_model.h5")
```

```
from keras.optimizers import SGD
epochs=60

learning_rate = 0.1
decay_rate = learning_rate / epochs
momentum = 0.8
sgd = SGD(lr=learning_rate, momentum=momentum, decay=decay_rate, nesterov=False)
```

Model Three : is similar to Model two except for using learning rate and momentum in SGD.

```
model_three = Sequential()
model_three.add(Conv1D(filters=8, kernel_size=11, activation='relu', input_shape=x_train.sha
model_three.add(MaxPool1D(strides=4))
model_three.add(BatchNormalization())
model_three.add(Conv1D(filters=16, kernel_size=11, activation='relu'))
model_three.add(MaxPool1D(strides=4))
model_three.add(BatchNormalization())
model_three.add(Conv1D(filters=32, kernel_size=11, activation='relu'))
model_three.add(MaxPool1D(strides=4))
model_three.add(BatchNormalization())
model_three.add(Conv1D(filters=64, kernel_size=11, activation='relu'))
model_three.add(MaxPool1D(strides=4))
model_three.add(Flatten())
model_three.add(Dropout(0.5))
model_three.add(Dense(64, activation='relu'))
model_three.add(Dropout(0.25))
model_three.add(Dense(64, activation='relu'))
model_three.add(Dense(1, activation='sigmoid'))
```

```
model_three.compile(optimizer=sgd, loss = 'binary_crossentropy', metrics=['accuracy', f1_m,
hist = model_three.fit_generator(batch_generator(x_train, y_train, 32),
```

```
                        validation_data=(x_test, y_test),
                        verbose=2, epochs=5,
                        steps_per_epoch=x_train.shape[1]//32)
```

```
Epoch 1/5
 - 2s - loss: 0.5882 - acc: 0.6985 - f1_m: 0.6899 - precision_m: 0.7125 - recall_
Epoch 2/5
 - 1s - loss: 0.4263 - acc: 0.8084 - f1_m: 0.8173 - precision_m: 0.7864 - recall_
Epoch 3/5
 - 1s - loss: 0.3513 - acc: 0.8460 - f1_m: 0.8548 - precision_m: 0.8098 - recall_
Epoch 4/5
 - 1s - loss: 0.3083 - acc: 0.8763 - f1_m: 0.8835 - precision_m: 0.8442 - recall_
Epoch 5/5
 - 1s - loss: 0.3028 - acc: 0.8753 - f1_m: 0.8817 - precision_m: 0.8508 - recall_
```

```
model_three.compile(optimizer=sgd, loss = 'binary_crossentropy', metrics=['accuracy', f1_m,
hist = model_three.fit_generator(batch_generator(x_train, y_train, 32),
                        validation_data=(x_test, y_test),
                        verbose=2, epochs=epochs,
                        steps_per_epoch=x_train.shape[1]//32)
```

```
Epoch 1/60
 - 2s - loss: 0.2393 - acc: 0.9107 - f1_m: 0.9149 - precision_m: 0.8840 - recall_
Epoch 2/60
 - 1s - loss: 0.2911 - acc: 0.8835 - f1_m: 0.8871 - precision_m: 0.8674 - recall_
Epoch 3/60
 - 1s - loss: 0.1886 - acc: 0.9366 - f1_m: 0.9383 - precision_m: 0.9206 - recall_
Epoch 4/60
 - 1s - loss: 0.1195 - acc: 0.9653 - f1_m: 0.9659 - precision_m: 0.9576 - recall_
Epoch 5/60
 - 1s - loss: 0.1424 - acc: 0.9571 - f1_m: 0.9583 - precision_m: 0.9416 - recall_
Epoch 6/60
 - 1s - loss: 0.0900 - acc: 0.9710 - f1_m: 0.9714 - precision_m: 0.9631 - recall_
Epoch 7/60
 - 1s - loss: 0.0831 - acc: 0.9722 - f1_m: 0.9726 - precision_m: 0.9639 - recall_
Epoch 8/60
 - 1s - loss: 0.0621 - acc: 0.9804 - f1_m: 0.9804 - precision_m: 0.9789 - recall_
Epoch 9/60
 - 1s - loss: 0.0645 - acc: 0.9807 - f1_m: 0.9811 - precision_m: 0.9728 - recall_
Epoch 10/60
 - 1s - loss: 0.0740 - acc: 0.9738 - f1_m: 0.9743 - precision_m: 0.9663 - recall_
Epoch 11/60
 - 1s - loss: 0.0629 - acc: 0.9830 - f1_m: 0.9833 - precision_m: 0.9771 - recall_
Epoch 12/60
 - 1s - loss: 0.0465 - acc: 0.9855 - f1_m: 0.9856 - precision_m: 0.9811 - recall_
Epoch 13/60
 - 1s - loss: 0.0408 - acc: 0.9871 - f1_m: 0.9872 - precision_m: 0.9852 - recall_
Epoch 14/60
 - 1s - loss: 0.0422 - acc: 0.9842 - f1_m: 0.9843 - precision_m: 0.9809 - recall_
Epoch 15/60
 - 1s - loss: 0.0380 - acc: 0.9886 - f1_m: 0.9888 - precision_m: 0.9858 - recall_
Epoch 16/60
 - 1s - loss: 0.0258 - acc: 0.9934 - f1_m: 0.9935 - precision_m: 0.9917 - recall_
Epoch 17/60
 - 1s - loss: 0.0422 - acc: 0.9880 - f1_m: 0.9883 - precision_m: 0.9817 - recall_
Epoch 18/60
 - 1s - loss: 0.0349 - acc: 0.9890 - f1_m: 0.9891 - precision_m: 0.9850 - recall_
Epoch 19/60
 - 1s - loss: 0.0322 - acc: 0.9880 - f1_m: 0.9883 - precision_m: 0.9837 - recall_
Epoch 20/60
 - 1s - loss: 0.0406 - acc: 0.9883 - f1_m: 0.9885 - precision_m: 0.9846 - recall_
Epoch 21/60
 - 1s - loss: 0.0282 - acc: 0.9924 - f1_m: 0.9926 - precision_m: 0.9869 - recall_
Epoch 22/60
 - 1s - loss: 0.0269 - acc: 0.9918 - f1_m: 0.9919 - precision_m: 0.9888 - recall_
Epoch 23/60
 - 1s - loss: 0.0245 - acc: 0.9921 - f1_m: 0.9922 - precision_m: 0.9900 - recall_
Epoch 24/60
 - 1s - loss: 0.0181 - acc: 0.9940 - f1_m: 0.9941 - precision_m: 0.9910 - recall_
Epoch 25/60
 - 1s - loss: 0.0243 - acc: 0.9915 - f1_m: 0.9915 - precision_m: 0.9893 - recall_
Epoch 26/60
 - 1s - loss: 0.0213 - acc: 0.9931 - f1_m: 0.9932 - precision_m: 0.9905 - recall_
Epoch 27/60
 - 1s - loss: 0.0156 - acc: 0.9953 - f1_m: 0.9954 - precision_m: 0.9924 - recall_
```

```
Epoch 28/60
 - 1s - loss: 0.0187 - acc: 0.9949 - f1_m: 0.9950 - precision_m: 0.9935 - recall_
Epoch 29/60
 - 1s - loss: 0.0270 - acc: 0.9893 - f1_m: 0.9894 - precision_m: 0.9851 - recall_
Epoch 30/60
 - 1s - loss: 0.0197 - acc: 0.9934 - f1_m: 0.9934 - precision_m: 0.9916 - recall_
Epoch 31/60
 - 1s - loss: 0.0202 - acc: 0.9940 - f1_m: 0.9941 - precision_m: 0.9911 - recall_
Epoch 32/60
 - 1s - loss: 0.0171 - acc: 0.9937 - f1_m: 0.9937 - precision_m: 0.9923 - recall_
Epoch 33/60
 - 1s - loss: 0.0134 - acc: 0.9946 - f1_m: 0.9947 - precision_m: 0.9928 - recall_
Epoch 34/60
 - 1s - loss: 0.0253 - acc: 0.9918 - f1_m: 0.9919 - precision_m: 0.9880 - recall_
Epoch 35/60
 - 1s - loss: 0.0173 - acc: 0.9949 - f1_m: 0.9950 - precision_m: 0.9933 - recall_
Epoch 36/60
 - 1s - loss: 0.0174 - acc: 0.9959 - f1_m: 0.9960 - precision_m: 0.9941 - recall_
Epoch 37/60
 - 1s - loss: 0.0166 - acc: 0.9943 - f1_m: 0.9943 - precision_m: 0.9934 - recall_
Epoch 38/60
 - 1s - loss: 0.0147 - acc: 0.9956 - f1_m: 0.9957 - precision_m: 0.9929 - recall_
Epoch 39/60
 - 1s - loss: 0.0110 - acc: 0.9959 - f1_m: 0.9960 - precision_m: 0.9941 - recall_
Epoch 40/60
 - 1s - loss: 0.0117 - acc: 0.9956 - f1_m: 0.9956 - precision_m: 0.9941 - recall_
Epoch 41/60
 - 1s - loss: 0.0154 - acc: 0.9943 - f1_m: 0.9944 - precision_m: 0.9929 - recall_
Epoch 42/60
 - 1s - loss: 0.0167 - acc: 0.9934 - f1_m: 0.9935 - precision_m: 0.9899 - recall_
Epoch 43/60
 - 1s - loss: 0.0117 - acc: 0.9959 - f1_m: 0.9960 - precision_m: 0.9941 - recall_
Epoch 44/60
 - 1s - loss: 0.0097 - acc: 0.9975 - f1_m: 0.9975 - precision_m: 0.9970 - recall_
Epoch 45/60
 - 1s - loss: 0.0140 - acc: 0.9962 - f1_m: 0.9963 - precision_m: 0.9935 - recall_
Epoch 46/60
 - 1s - loss: 0.0199 - acc: 0.9940 - f1_m: 0.9940 - precision_m: 0.9922 - recall_
Epoch 47/60
 - 1s - loss: 0.0111 - acc: 0.9962 - f1_m: 0.9963 - precision_m: 0.9946 - recall_
Epoch 48/60
 - 1s - loss: 0.0153 - acc: 0.9962 - f1_m: 0.9963 - precision_m: 0.9947 - recall_
Epoch 49/60
 - 1s - loss: 0.0130 - acc: 0.9968 - f1_m: 0.9969 - precision_m: 0.9958 - recall_
Epoch 50/60
 - 1s - loss: 0.0121 - acc: 0.9959 - f1_m: 0.9959 - precision_m: 0.9952 - recall_
Epoch 51/60
 - 1s - loss: 0.0109 - acc: 0.9972 - f1_m: 0.9972 - precision_m: 0.9964 - recall_
Epoch 52/60
 - 1s - loss: 0.0091 - acc: 0.9975 - f1_m: 0.9975 - precision_m: 0.9964 - recall_
Epoch 53/60
 - 1s - loss: 0.0174 - acc: 0.9946 - f1_m: 0.9946 - precision_m: 0.9940 - recall_
Epoch 54/60
 - 1s - loss: 0.0093 - acc: 0.9972 - f1_m: 0.9972 - precision_m: 0.9965 - recall_
Epoch 55/60
```

```
Epoch 55/60
 - 1s - loss: 0.0104 - acc: 0.9959 - f1_m: 0.9959 - precision_m: 0.9958 - recall_
Epoch 56/60
 - 1s - loss: 0.0073 - acc: 0.9978 - f1_m: 0.9978 - precision_m: 0.9970 - recall_
Epoch 57/60
 - 1s - loss: 0.0080 - acc: 0.9975 - f1_m: 0.9975 - precision_m: 0.9970 - recall_
Epoch 58/60
 - 1s - loss: 0.0061 - acc: 0.9975 - f1_m: 0.9975 - precision_m: 0.9976 - recall_
Epoch 59/60
 - 1s - loss: 0.0115 - acc: 0.9956 - f1_m: 0.9956 - precision_m: 0.9958 - recall_
Epoch 60/60
 - 1s - loss: 0.0084 - acc: 0.9968 - f1_m: 0.9969 - precision_m: 0.9958 - recall_
```

```
model_three.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv1d_9 (Conv1D)            (None, 3187, 8)           184       
_________________________________________________________________
max_pooling1d_9 (MaxPooling1 (None, 797, 8)            0         
_________________________________________________________________
batch_normalization_7 (Batch (None, 797, 8)            32        
_________________________________________________________________
conv1d_10 (Conv1D)           (None, 787, 16)           1424      
_________________________________________________________________
max_pooling1d_10 (MaxPooling (None, 197, 16)           0         
_________________________________________________________________
batch_normalization_8 (Batch (None, 197, 16)           64        
_________________________________________________________________
conv1d_11 (Conv1D)           (None, 187, 32)           5664      
_________________________________________________________________
max_pooling1d_11 (MaxPooling (None, 47, 32)            0         
_________________________________________________________________
batch_normalization_9 (Batch (None, 47, 32)            128       
_________________________________________________________________
conv1d_12 (Conv1D)           (None, 37, 64)            22592     
_________________________________________________________________
max_pooling1d_12 (MaxPooling (None, 9, 64)             0         
_________________________________________________________________
flatten_3 (Flatten)          (None, 576)               0         
_________________________________________________________________
dropout_5 (Dropout)          (None, 576)               0         
_________________________________________________________________
dense_7 (Dense)              (None, 64)                36928     
_________________________________________________________________
dropout_6 (Dropout)          (None, 64)                0         
_________________________________________________________________
dense_8 (Dense)              (None, 64)                4160      
_________________________________________________________________
dense_9 (Dense)              (None, 1)                 65        
=================================================================
Total params: 71,241
Trainable params: 71,129
Non-trainable params: 112
_________________________________________________________________
```

```
#plot_model(model_three, to_file='model.png', show_shapes=True, show_layer_names=True)
```

```
print(model_three.evaluate(x_train, y_train, batch_size = 100))
print('\nModel Performance: Loss and Accuracy on validation data')
print(model_three.evaluate(x_test, y_test, batch_size = 100))
```

```
5087/5087 [==============================] - 0s 98us/step
[0.00945733666575404, 0.9976410480527014, 0.01965795164143896, 0.019657951641438

Model Performance: Loss and Accuracy on validation data
570/570 [==============================] - 0s 90us/step
[0.00879225914822232, 0.9982456157082006, 0.17543859649122806, 0.175438596491228
```

```
y_train_hat = model_three.predict(x_train)[:,0]
y_train_pred = np.where(y_train_hat > 0.9,1.,0.)
print('Accuracy:', accuracy_score(y_train, y_train_pred))
print('F1 score:', f1_score(y_train, y_train_pred))
print('Recall:', recall_score(y_train, y_train_pred))
print('Precision:', precision_score(y_train, y_train_pred))
print('Cohens Kappa :', cohen_kappa_score(y_train, y_train_pred))
print('ROC AUC Score:', roc_auc_score(y_train, y_train_pred))
print('\n clasification report:\n', classification_report(y_train, y_train_pred))
print('\n confussion matrix:\n',confusion_matrix(y_train, y_train_pred))
```

```
#Using threshold as 0.9
y_hat = model_three.predict(x_test)[:,0]
y_pred = np.where(y_hat > 0.9,1.,0.)

# accuracy: (tp + tn) / (p + n)
print('Accuracy:', accuracy_score(y_test, y_pred))
# f1: 2 tp / (2 tp + fp + fn)
print('F1 score:', f1_score(y_test, y_pred))
# recall: tp / (tp + fn)
print('Recall:', recall_score(y_test, y_pred))
# precision tp / (tp + fp)
print('Precision:', precision_score(y_test, y_pred))
#cohen's kappa
print('Cohens Kappa :', cohen_kappa_score(y_test, y_pred))
#AUC score
print('ROC AUC Score:', roc_auc_score(y_test, y_pred))
#Classification report
print('\n clasification report:\n', classification_report(y_test,y_pred))
```

```
#Confusion matrix
print('\n confussion matrix:\n',confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.9964912280701754
F1 score: 0.8000000000000002
Recall: 0.8
Precision: 0.8
Cohens Kappa : 0.7982300884955752
ROC AUC Score: 0.8991150442477877

 clasification report:
               precision    recall  f1-score   support

          0.0       1.00      1.00      1.00       565
          1.0       0.80      0.80      0.80         5

     accuracy                           1.00       570
    macro avg       0.90      0.90      0.90       570
 weighted avg       1.00      1.00      1.00       570


 confussion matrix:
 [[564   1]
 [  1   4]]
```

```
K.clear_session()
```

```
#def create_model(init_mode='glorot_uniform', activation='relu', dropout_rate=0.5, neurons =
#                  optimizer='sgd', filters=8):
def create_model(init_mode='glorot_uniform', activation='relu', dropout_rate=0.5, neurons =
                 optimizer='sgd', filters=8):

    print(init_mode, activation, dropout_rate, neurons, optimizer, filters )
    model = Sequential()
    model.add(Conv1D(filters=filters, kernel_size=11, kernel_initializer=init_mode,
                     activation=activation, input_shape=x_train_sm.shape[1:]))
    model.add(MaxPool1D(strides=4))
    model.add(BatchNormalization())
    model.add(Conv1D(filters=(2 * filters), kernel_size=11, kernel_initializer=init_mode, a
    model.add(MaxPool1D(strides=4))
    model.add(BatchNormalization())
    model.add(Conv1D(filters=(4 * filters), kernel_size=11, kernel_initializer=init_mode, a
    model.add(MaxPool1D(strides=4))
    model.add(BatchNormalization())
    model.add(Conv1D(filters=(8 * filters), kernel_size=11, kernel_initializer=init_mode, a
    model.add(MaxPool1D(strides=4))
    model.add(Flatten())
    model.add(Dropout(dropout_rate))
    model.add(Dense(units=neurons, activation=activation, kernel_initializer=init_mode))
    model.add(Dropout(dropout_rate/2))
    model.add(Dense(units=neurons, activation=activation, kernel_initializer=init_mode))
    model.add(Dense(1, activation='sigmoid', kernel_initializer=init_mode))
    #model.compile(optimizer="sgd", loss = 'binary_crossentropy', metrics=['accuracy'])
    #model.compile(optimizer=optimizer, loss = 'binary_crossentropy', metrics=['accuracy',
    model.compile(optimizer=optimizer, loss = 'binary_crossentropy')
    return model
```

```
%%time
#INPUT_LIB = '/content/drive/My Drive/capstone_datasets/'
INPUT_LIB = '../webapp/web/'
```

```
raw_data = np.loadtxt(INPUT_LIB + 'exoTrain.csv', skiprows=1, delimiter=',')
x_train = raw_data[:, 1:]
y_train = raw_data[:, 0, np.newaxis] - 1.
raw_data = np.loadtxt(INPUT_LIB + 'exoTest.csv', skiprows=1, delimiter=',')
x_test = raw_data[:, 1:]
y_test = raw_data[:, 0, np.newaxis] - 1.
del raw_data

print(x_train.shape, x_test.shape)
x_train = ((x_train - np.mean(x_train, axis=1).reshape(-1,1)) / np.std(x_train, axis=1).resh
x_test = ((x_test - np.mean(x_test, axis=1).reshape(-1,1)) / np.std(x_test, axis=1).reshape
print(x_train.shape, x_test.shape)
#x_train = np.stack([x_train, uniform_filter1d(x_train, axis=1, size=200)], axis=2)
#x_test = np.stack([x_test, uniform_filter1d(x_test, axis=1, size=200)], axis=2)
#print(x_train.shape, x_test.shape)
```

```
(5087, 3197) (570, 3197)
(5087, 3197) (570, 3197)
CPU times: user 11.5 s, sys: 275 ms, total: 11.8 s
Wall time: 11.8 s
```

```
%%time
sm = SMOTE(ratio = 1.0)
print(x_train.shape, y_train.shape)
x_train_sm, y_train_sm = sm.fit_sample(x_train, y_train)


print(len(x_train_sm))
print(x_train_sm.shape, y_train_sm.shape)
x_train_sm = np.stack([x_train_sm, uniform_filter1d(x_train_sm, axis=1, size=200)], axis=2)
x_test = np.stack([x_test, uniform_filter1d(x_test, axis=1, size=200)], axis=2)
```

```
(5087, 3197) (5087, 1)
/conda/envs/rapids/lib/python3.6/site-packages/sklearn/utils/validation.py:724: D
  y = column_or_1d(y, warn=True)
10100
(10100, 3197) (10100,)
CPU times: user 288 ms, sys: 243 ms, total: 531 ms
Wall time: 583 ms
```

Parameter Grid for Random Search involves searching:

1. Initializers: Uniform, Lecun_uniform, Normal, Zero, Glorot_normal, Glorot_uniform, He_normal, He_uniform
2. Activation funtions: Softmax, Softplus, Softsign, Relu, Tanh, Sigmoid, Hard_sigmoid, Linear
3. Drop rates: 0.2, 0.4, 0.6
4. Number of neurons: 32, 64, 128
5. Batch size : 32, 64
6. Optimizers: SGD, RMSProp, Adagrad, Adadelta, Adam, Adamax, Nadam
7. Filter sizes: 8, 16

```
seed = 7
np.random.seed(seed)
#batch_size = 128
epochs = 3

K.clear_session()
#nb_epoch=3
```

```
#with tf.device('/cpu:0'):
if True:
    model_CV = KerasClassifier(build_fn=create_model, epochs = epochs, verbose=1)

    '''
    # define the grid search parameters
    #init_mode = ['glorot_normal', 'glorot_uniform']
    #activation = ['relu', 'sigmoid']
    #weight_constraint = [1, 2, 3, 4, 5]
    #optimizer = ['Adam', 'RMSprop'] #, 'sgd', 'Nadam']
    #epochs = [10, 20]

    init_mode = ['glorot_normal']
    activation = ['sigmoid']
    dropout_rate = [0.6]
    neurons = [32]
    batch_size = [32]
    optimizer = ['Adam']
    filters = [16]
    '''

    init_mode = ['uniform', 'lecun_uniform', 'normal', 'zero', 'glorot_normal', 'glorot_uni
    activation = ['softmax', 'softplus', 'softsign', 'relu', 'tanh', 'sigmoid', 'hard_sigmo
    dropout_rate = [0.2, 0.4, 0.6]
    neurons = [32, 64, 128]
    batch_size = [32, 64]
    optimizer = ['SGD', 'RMSprop', 'Adagrad', 'Adadelta', 'Adam', 'Adamax', 'Nadam']
    filters = [8, 16]

    #f1 = {'f1' : f1_m}
    f1 = make_scorer(f1_score)
    f1_scorer = make_scorer(f1_m)
    scoring = {'f1' : f1_m}

    param_grid = dict( init_mode = init_mode, activation = activation, dropout_rate = dropo
                          neurons = neurons, batch_size = batch_size, optimizer = optimizer

    #grid = GridSearchCV(estimator=model_CV, param_grid=param_grid, scoring = scoring, cv=3
    grid = RandomizedSearchCV(estimator=model_CV, param_distributions=param_grid, scoring =
                              n_iter=20) #, pre_dispatch=3, n_jobs=3)
    grid_result = grid.fit(x_train_sm, y_train_sm)
    print(f'Best Accuracy for {grid_result.best_score_} using {grid_result.best_params_}')
    result_df = pd.DataFrame(grid_result.cv_results_)
    print(result_df)
    result_df.to_csv('RandomizedSearchCV_result_df.csv', index=False, encoding='utf-8')
    means = grid_result.cv_results_['mean_test_score']
    stds = grid_result.cv_results_['std_test_score']
    #means_train = grid_result.cv_results_['mean_train_score']
    #stds_train = grid_result.cv_results_['std_train_score']
    params = grid_result.cv_results_['params']
    for mean, stdev, param in zip(means, stds, params):
        print(f' mean={mean:.4}, std={stdev:.4} using {param}')
```

```
lecun_uniform hard_sigmoid 0.4 64 Adadelta 8
Epoch 1/3
6733/6733 [==============================] - 4s 557us/step - loss: 0.3018
Epoch 2/3
6733/6733 [==============================] - 3s 400us/step - loss: 0.0320
Epoch 3/3
6733/6733 [==============================] - 3s 405us/step - loss: 0.0073
3367/3367 [==============================] - 1s 193us/step
lecun_uniform hard_sigmoid 0.4 64 Adadelta 8
Epoch 1/3
6733/6733 [==============================] - 4s 563us/step - loss: 0.3736
Epoch 2/3
6733/6733 [==============================] - 3s 413us/step - loss: 0.0320
Epoch 3/3
6733/6733 [==============================] - 3s 403us/step - loss: 0.0105
3367/3367 [==============================] - 1s 181us/step
lecun_uniform hard_sigmoid 0.4 64 Adadelta 8
Epoch 1/3
6734/6734 [==============================] - 4s 586us/step - loss: 0.3638
Epoch 2/3
6734/6734 [==============================] - 3s 395us/step - loss: 0.0403
Epoch 3/3
6734/6734 [==============================] - 3s 402us/step - loss: 0.0174
3366/3366 [==============================] - 1s 229us/step
uniform tanh 0.4 32 Adam 8
Epoch 1/3
6733/6733 [==============================] - 3s 455us/step - loss: 0.2319
Epoch 2/3
6733/6733 [==============================] - 2s 225us/step - loss: 0.0319
Epoch 3/3
6733/6733 [==============================] - 2s 230us/step - loss: 0.0208
3367/3367 [==============================] - 1s 197us/step
uniform tanh 0.4 32 Adam 8
Epoch 1/3
6733/6733 [==============================] - 3s 444us/step - loss: 0.2343
Epoch 2/3
6733/6733 [==============================] - 2s 229us/step - loss: 0.0407
Epoch 3/3
6733/6733 [==============================] - 2s 228us/step - loss: 0.0194
3367/3367 [==============================] - 1s 212us/step
uniform tanh 0.4 32 Adam 8
Epoch 1/3
6734/6734 [==============================] - 3s 470us/step - loss: 0.2195
Epoch 2/3
6734/6734 [==============================] - 2s 230us/step - loss: 0.0344
Epoch 3/3
6734/6734 [==============================] - 2s 224us/step - loss: 0.0150
3366/3366 [==============================] - 1s 281us/step
lecun_uniform softmax 0.4 64 Adamax 16
WARNING:tensorflow:From /conda/envs/rapids/lib/python3.6/site-packages/tensorflo
Instructions for updating:
Deprecated in favor of operator or tf.math.divide.
WARNING:From /conda/envs/rapids/lib/python3.6/site-packages/tensorflow/python/op
Instructions for updating:
```

```
Deprecated in favor of operator or tf.math.divide.
Epoch 1/3
6733/6733 [==============================] - 6s 921us/step - loss: 0.6353
Epoch 2/3
6733/6733 [==============================] - 4s 573us/step - loss: 0.5703
Epoch 3/3
6733/6733 [==============================] - 4s 573us/step - loss: 0.5068
3367/3367 [==============================] - 1s 377us/step
lecun_uniform softmax 0.4 64 Adamax 16
Epoch 1/3
6733/6733 [==============================] - 6s 922us/step - loss: 0.6916
Epoch 2/3
6733/6733 [==============================] - 4s 579us/step - loss: 0.6627
Epoch 3/3
6733/6733 [==============================] - 4s 577us/step - loss: 0.5852
3367/3367 [==============================] - 1s 410us/step
lecun_uniform softmax 0.4 64 Adamax 16
Epoch 1/3
6734/6734 [==============================] - 16s 2ms/step - loss: 0.6416
Epoch 2/3
6734/6734 [==============================] - 4s 575us/step - loss: 0.5728
Epoch 3/3
6734/6734 [==============================] - 4s 584us/step - loss: 0.5121
3366/3366 [==============================] - 2s 494us/step
/conda/envs/rapids/lib/python3.6/site-packages/sklearn/metrics/classification.py
  'precision', 'predicted', average, warn_for)
he_uniform sigmoid 0.4 64 Adam 16
Epoch 1/3
6733/6733 [==============================] - 5s 750us/step - loss: 0.1507
Epoch 2/3
6733/6733 [==============================] - 3s 424us/step - loss: 0.0135
Epoch 3/3
6733/6733 [==============================] - 3s 423us/step - loss: 0.0071
3367/3367 [==============================] - 1s 444us/step
he_uniform sigmoid 0.4 64 Adam 16
Epoch 1/3
6733/6733 [==============================] - 5s 795us/step - loss: 0.1544
Epoch 2/3
6733/6733 [==============================] - 3s 432us/step - loss: 0.0182
Epoch 3/3
6733/6733 [==============================] - 3s 427us/step - loss: 0.0055
3367/3367 [==============================] - 1s 439us/step
he_uniform sigmoid 0.4 64 Adam 16
Epoch 1/3
6734/6734 [==============================] - 6s 850us/step - loss: 0.1445
Epoch 2/3
6734/6734 [==============================] - 3s 435us/step - loss: 0.0114
Epoch 3/3
6734/6734 [==============================] - 3s 429us/step - loss: 0.0074
3366/3366 [==============================] - 1s 433us/step
he_uniform softsign 0.4 64 RMSprop 16
Epoch 1/3
6733/6733 [==============================] - 5s 791us/step - loss: 0.0744
Epoch 2/3
6733/6733 [==============================] - 2s 323us/step - loss: 0.0080
```

```
6733/6733 [==============================] - 2s 325us/step - loss: 0.0080
Epoch 3/3
6733/6733 [==============================] - 2s 322us/step - loss: 0.0041
3367/3367 [==============================] - 2s 480us/step
he_uniform softsign 0.4 64 RMSprop 16
Epoch 1/3
6733/6733 [==============================] - 6s 874us/step - loss: 0.0841
Epoch 2/3
6733/6733 [==============================] - 2s 317us/step - loss: 0.0144
Epoch 3/3
6733/6733 [==============================] - 2s 317us/step - loss: 0.0063
3367/3367 [==============================] - 1s 440us/step
he_uniform softsign 0.4 64 RMSprop 16
Epoch 1/3
6734/6734 [==============================] - 5s 750us/step - loss: 0.0883
Epoch 2/3
6734/6734 [==============================] - 2s 319us/step - loss: 0.0089
Epoch 3/3
6734/6734 [==============================] - 2s 320us/step - loss: 0.0074
3366/3366 [==============================] - 2s 627us/step
glorot_uniform softsign 0.2 32 Adagrad 16
Epoch 1/3
6733/6733 [==============================] - 6s 883us/step - loss: 0.0553
Epoch 2/3
6733/6733 [==============================] - 3s 408us/step - loss: 0.0051
Epoch 3/3
6733/6733 [==============================] - 3s 413us/step - loss: 0.0019
3367/3367 [==============================] - 2s 518us/step
glorot_uniform softsign 0.2 32 Adagrad 16
Epoch 1/3
6733/6733 [==============================] - 6s 893us/step - loss: 0.0702
Epoch 2/3
6733/6733 [==============================] - 3s 416us/step - loss: 0.0080
Epoch 3/3
6733/6733 [==============================] - 3s 420us/step - loss: 0.0039
3367/3367 [==============================] - 2s 539us/step
glorot_uniform softsign 0.2 32 Adagrad 16
Epoch 1/3
6734/6734 [==============================] - 7s 1ms/step - loss: 0.0640
Epoch 2/3
6734/6734 [==============================] - 3s 425us/step - loss: 0.0063
Epoch 3/3
6734/6734 [==============================] - 3s 429us/step - loss: 0.0033
3366/3366 [==============================] - 2s 609us/step
zero softmax 0.6 128 Adagrad 16
Epoch 1/3
6733/6733 [==============================] - 7s 1ms/step - loss: 0.6297
Epoch 2/3
6733/6733 [==============================] - 4s 583us/step - loss: 0.5961
Epoch 3/3
6733/6733 [==============================] - 4s 597us/step - loss: 0.5842
3367/3367 [==============================] - 3s 780us/step
zero softmax 0.6 128 Adagrad 16
Epoch 1/3
6733/6733 [==============================] - 8s 1ms/step - loss: 0.6933
```

```
Epoch 2/3
6733/6733 [==============================] - 4s 596us/step - loss: 0.6931
Epoch 3/3
6733/6733 [==============================] - 4s 598us/step - loss: 0.6931
3367/3367 [==============================] - 3s 827us/step
zero softmax 0.6 128 Adagrad 16
Epoch 1/3
6734/6734 [==============================] - 8s 1ms/step - loss: 0.6291
Epoch 2/3
6734/6734 [==============================] - 4s 584us/step - loss: 0.5938
Epoch 3/3
6734/6734 [==============================] - 4s 593us/step - loss: 0.5808
3366/3366 [==============================] - 3s 768us/step
he_uniform hard_sigmoid 0.6 32 Adam 8
/conda/envs/rapids/lib/python3.6/site-packages/sklearn/metrics/classification.py
  'precision', 'predicted', average, warn_for)
Epoch 1/3
6733/6733 [==============================] - 6s 904us/step - loss: 0.3627
Epoch 2/3
6733/6733 [==============================] - 2s 279us/step - loss: 0.0691
Epoch 3/3
6733/6733 [==============================] - 2s 280us/step - loss: 0.0347
3367/3367 [==============================] - 2s 673us/step
he_uniform hard_sigmoid 0.6 32 Adam 8
Epoch 1/3
6733/6733 [==============================] - 6s 936us/step - loss: 0.4165
Epoch 2/3
6733/6733 [==============================] - 2s 279us/step - loss: 0.0839
Epoch 3/3
6733/6733 [==============================] - 2s 282us/step - loss: 0.0371
3367/3367 [==============================] - 2s 609us/step
he_uniform hard_sigmoid 0.6 32 Adam 8
Epoch 1/3
6734/6734 [==============================] - 7s 965us/step - loss: 0.5848
Epoch 2/3
6734/6734 [==============================] - 2s 285us/step - loss: 0.2254
Epoch 3/3
6734/6734 [==============================] - 2s 289us/step - loss: 0.0596
3366/3366 [==============================] - 2s 647us/step
lecun_uniform softmax 0.4 64 SGD 8
Epoch 1/3
6733/6733 [==============================] - 7s 1ms/step - loss: 0.6705
Epoch 2/3
6733/6733 [==============================] - 2s 329us/step - loss: 0.6300
Epoch 3/3
6733/6733 [==============================] - 2s 330us/step - loss: 0.6061
3367/3367 [==============================] - 2s 675us/step
lecun_uniform softmax 0.4 64 SGD 8
Epoch 1/3
6733/6733 [==============================] - 7s 1ms/step - loss: 0.6932
Epoch 2/3
6733/6733 [==============================] - 2s 333us/step - loss: 0.6932
Epoch 3/3
6733/6733 [==============================] - 2s 333us/step - loss: 0.6931
3367/3367 [==============================] - 2s 745us/step
```

```
3367/3367 [==============================] - 3s 745us/step
lecun_uniform softmax 0.4 64 SGD 8
Epoch 1/3
6734/6734 [==============================] - 7s 1ms/step - loss: 0.6687
Epoch 2/3
6734/6734 [==============================] - 2s 341us/step - loss: 0.6268
Epoch 3/3
6734/6734 [==============================] - 2s 344us/step - loss: 0.6018
3366/3366 [==============================] - 3s 874us/step
/conda/envs/rapids/lib/python3.6/site-packages/sklearn/metrics/classification.py
  'precision', 'predicted', average, warn_for)
lecun_uniform sigmoid 0.4 32 Adam 8
Epoch 1/3
6733/6733 [==============================] - 7s 1ms/step - loss: 0.4430
Epoch 2/3
6733/6733 [==============================] - 2s 256us/step - loss: 0.0949
Epoch 3/3
6733/6733 [==============================] - 2s 257us/step - loss: 0.0321
3367/3367 [==============================] - 2s 727us/step
lecun_uniform sigmoid 0.4 32 Adam 8
Epoch 1/3
6733/6733 [==============================] - 7s 1ms/step - loss: 0.3979
Epoch 2/3
6733/6733 [==============================] - 2s 255us/step - loss: 0.0782
Epoch 3/3
6733/6733 [==============================] - 2s 255us/step - loss: 0.0348
3367/3367 [==============================] - 3s 950us/step
lecun_uniform sigmoid 0.4 32 Adam 8
Epoch 1/3
6734/6734 [==============================] - 7s 1ms/step - loss: 0.4677
Epoch 2/3
6734/6734 [==============================] - 2s 262us/step - loss: 0.1099
Epoch 3/3
6734/6734 [==============================] - 2s 255us/step - loss: 0.0423
3366/3366 [==============================] - 3s 791us/step
uniform relu 0.4 32 Nadam 16
Epoch 1/3
6733/6733 [==============================] - 17s 2ms/step - loss: 0.1224
Epoch 2/3
6733/6733 [==============================] - 2s 365us/step - loss: 0.0059
Epoch 3/3
6733/6733 [==============================] - 2s 358us/step - loss: 0.0018
3367/3367 [==============================] - 3s 839us/step
uniform relu 0.4 32 Nadam 16
Epoch 1/3
6733/6733 [==============================] - 8s 1ms/step - loss: 0.1336
Epoch 2/3
6733/6733 [==============================] - 2s 352us/step - loss: 0.0126
Epoch 3/3
6733/6733 [==============================] - 2s 358us/step - loss: 0.0131
3367/3367 [==============================] - 4s 1ms/step
uniform relu 0.4 32 Nadam 16
Epoch 1/3
6734/6734 [==============================] - 9s 1ms/step - loss: 0.1666
Epoch 2/3
```

```
6734/6734 [==============================] - 2s 355us/step - loss: 0.0200
Epoch 3/3
6734/6734 [==============================] - 2s 355us/step - loss: 0.0092
3366/3366 [==============================] - 3s 888us/step
zero softplus 0.4 32 SGD 16
Epoch 1/3
6733/6733 [==============================] - 10s 1ms/step - loss: 0.5764
Epoch 2/3
6733/6733 [==============================] - 3s 450us/step - loss: 0.5685
Epoch 3/3
6733/6733 [==============================] - 3s 449us/step - loss: 0.5686
3367/3367 [==============================] - 48s 14ms/step
zero softplus 0.4 32 SGD 16
Epoch 1/3
6733/6733 [==============================] - 11s 2ms/step - loss: 0.6934
Epoch 2/3
6733/6733 [==============================] - 3s 457us/step - loss: 0.6936
Epoch 3/3
6733/6733 [==============================] - 3s 455us/step - loss: 0.6936
3367/3367 [==============================] - 4s 1ms/step
zero softplus 0.4 32 SGD 16
Epoch 1/3
6734/6734 [==============================] - 10s 1ms/step - loss: 0.5706
Epoch 2/3
6734/6734 [==============================] - 3s 447us/step - loss: 0.5627
Epoch 3/3
6734/6734 [==============================] - 3s 452us/step - loss: 0.5628
3366/3366 [==============================] - 4s 1ms/step
glorot_normal sigmoid 0.6 32 Adam 8
/conda/envs/rapids/lib/python3.6/site-packages/sklearn/metrics/classification.py
  'precision', 'predicted', average, warn_for)
Epoch 1/3
6733/6733 [==============================] - 10s 1ms/step - loss: 0.4238
Epoch 2/3
6733/6733 [==============================] - 2s 266us/step - loss: 0.0844
Epoch 3/3
6733/6733 [==============================] - 2s 268us/step - loss: 0.0506
3367/3367 [==============================] - 3s 948us/step
glorot_normal sigmoid 0.6 32 Adam 8
Epoch 1/3
6733/6733 [==============================] - 9s 1ms/step - loss: 0.4222
Epoch 2/3
6733/6733 [==============================] - 2s 263us/step - loss: 0.0898
Epoch 3/3
6733/6733 [==============================] - 2s 268us/step - loss: 0.0330
3367/3367 [==============================] - 3s 986us/step
glorot_normal sigmoid 0.6 32 Adam 8
Epoch 1/3
6734/6734 [==============================] - 9s 1ms/step - loss: 0.3554
Epoch 2/3
6734/6734 [==============================] - 2s 261us/step - loss: 0.0747
Epoch 3/3
6734/6734 [==============================] - 2s 265us/step - loss: 0.0409
3366/3366 [==============================] - 3s 990us/step
```

```
zero softsign 0.4 64 Adadelta 8
Epoch 1/3
6733/6733 [==============================] - 11s 2ms/step - loss: 0.6638
Epoch 2/3
6733/6733 [==============================] - 3s 457us/step - loss: 0.6188
Epoch 3/3
6733/6733 [==============================] - 3s 466us/step - loss: 0.5923
3367/3367 [==============================] - 4s 1ms/step
zero softsign 0.4 64 Adadelta 8
Epoch 1/3
6733/6733 [==============================] - 12s 2ms/step - loss: 0.6932
Epoch 2/3
6733/6733 [==============================] - 3s 486us/step - loss: 0.6931
Epoch 3/3
6733/6733 [==============================] - 3s 488us/step - loss: 0.6931
3367/3367 [==============================] - 5s 1ms/step
zero softsign 0.4 64 Adadelta 8
Epoch 1/3
6734/6734 [==============================] - 11s 2ms/step - loss: 0.6615
Epoch 2/3
6734/6734 [==============================] - 3s 480us/step - loss: 0.6143
Epoch 3/3
6734/6734 [==============================] - 3s 477us/step - loss: 0.5874
3366/3366 [==============================] - 4s 1ms/step
normal hard_sigmoid 0.4 64 Adamax 16
/conda/envs/rapids/lib/python3.6/site-packages/sklearn/metrics/classification.py
  'precision', 'predicted', average, warn_for)
Epoch 1/3
6733/6733 [==============================] - 15s 2ms/step - loss: 0.2877
Epoch 2/3
6733/6733 [==============================] - 4s 563us/step - loss: 0.0258
Epoch 3/3
6733/6733 [==============================] - 4s 566us/step - loss: 0.0202
3367/3367 [==============================] - 5s 1ms/step
normal hard_sigmoid 0.4 64 Adamax 16
Epoch 1/3
6733/6733 [==============================] - 16s 2ms/step - loss: 0.2042
Epoch 2/3
6733/6733 [==============================] - 4s 571us/step - loss: 0.0249
Epoch 3/3
6733/6733 [==============================] - 4s 573us/step - loss: 0.0134
3367/3367 [==============================] - 5s 2ms/step
normal hard_sigmoid 0.4 64 Adamax 16
Epoch 1/3
6734/6734 [==============================] - 12s 2ms/step - loss: 0.2791
Epoch 2/3
6734/6734 [==============================] - 4s 558us/step - loss: 0.0358
Epoch 3/3
6734/6734 [==============================] - 4s 566us/step - loss: 0.0131
3366/3366 [==============================] - 4s 1ms/step
glorot_uniform linear 0.6 32 RMSprop 8
Epoch 1/3
6733/6733 [==============================] - 11s 2ms/step - loss: 0.0837
Epoch 2/3
6733/6733 [==============================] - 3s 438us/step - loss: 0.0146
```

```
Epoch 3/3
6733/6733 [==============================] - 3s 440us/step - loss: 0.0102
3367/3367 [==============================] - 6s 2ms/step
glorot_uniform linear 0.6 32 RMSprop 8
Epoch 1/3
6733/6733 [==============================] - 25s 4ms/step - loss: 0.1080
Epoch 2/3
6733/6733 [==============================] - 3s 452us/step - loss: 0.0174
Epoch 3/3
6733/6733 [==============================] - 3s 451us/step - loss: 0.0094
3367/3367 [==============================] - 26s 8ms/step
glorot_uniform linear 0.6 32 RMSprop 8
Epoch 1/3
6734/6734 [==============================] - 13s 2ms/step - loss: 0.1021
Epoch 2/3
6734/6734 [==============================] - 3s 458us/step - loss: 0.0196
Epoch 3/3
6734/6734 [==============================] - 3s 462us/step - loss: 0.0135
3366/3366 [==============================] - 6s 2ms/step
normal sigmoid 0.6 64 RMSprop 16
Epoch 1/3
6733/6733 [==============================] - 13s 2ms/step - loss: 0.2054
Epoch 2/3
6733/6733 [==============================] - 3s 505us/step - loss: 0.0345
Epoch 3/3
6733/6733 [==============================] - 3s 503us/step - loss: 0.0179
3367/3367 [==============================] - 5s 1ms/step
normal sigmoid 0.6 64 RMSprop 16
Epoch 1/3
6733/6733 [==============================] - 13s 2ms/step - loss: 0.2217
Epoch 2/3
6733/6733 [==============================] - 3s 516us/step - loss: 0.0254
Epoch 3/3
6733/6733 [==============================] - 4s 520us/step - loss: 0.0159
3367/3367 [==============================] - 6s 2ms/step
normal sigmoid 0.6 64 RMSprop 16
Epoch 1/3
6734/6734 [==============================] - 13s 2ms/step - loss: 0.2174
Epoch 2/3
6734/6734 [==============================] - 3s 516us/step - loss: 0.0272
Epoch 3/3
6734/6734 [==============================] - 3s 510us/step - loss: 0.0161
3366/3366 [==============================] - 5s 2ms/step
glorot_normal hard_sigmoid 0.2 32 Adam 8
Epoch 1/3
6733/6733 [==============================] - 13s 2ms/step - loss: 0.4456
Epoch 2/3
6733/6733 [==============================] - 2s 315us/step - loss: 0.0993
Epoch 3/3
6733/6733 [==============================] - 2s 326us/step - loss: 0.0327
3367/3367 [==============================] - 6s 2ms/step
glorot_normal hard_sigmoid 0.2 32 Adam 8
Epoch 1/3
6733/6733 [==============================] - 12s 2ms/step - loss: 0.3765
```

```
Epoch 2/3
6733/6733 [==============================] - 2s 326us/step - loss: 0.0488
Epoch 3/3
6733/6733 [==============================] - 2s 317us/step - loss: 0.0246
3367/3367 [==============================] - 5s 1ms/step
glorot_normal hard_sigmoid 0.2 32 Adam 8
Epoch 1/3
6734/6734 [==============================] - 13s 2ms/step - loss: 0.5420
Epoch 2/3
6734/6734 [==============================] - 2s 321us/step - loss: 0.1427
Epoch 3/3
6734/6734 [==============================] - 2s 322us/step - loss: 0.0518
3366/3366 [==============================] - 43s 13ms/step
normal softplus 0.2 32 Adamax 16
Epoch 1/3
6733/6733 [==============================] - 14s 2ms/step - loss: 0.1264
Epoch 2/3
6733/6733 [==============================] - 3s 378us/step - loss: 0.0058
Epoch 3/3
6733/6733 [==============================] - 3s 375us/step - loss: 0.0027
3367/3367 [==============================] - 6s 2ms/step
normal softplus 0.2 32 Adamax 16
Epoch 1/3
6733/6733 [==============================] - 20s 3ms/step - loss: 0.1261
Epoch 2/3
6733/6733 [==============================] - 3s 381us/step - loss: 0.0138
Epoch 3/3
6733/6733 [==============================] - 3s 390us/step - loss: 0.0088
3367/3367 [==============================] - 5s 2ms/step
normal softplus 0.2 32 Adamax 16
Epoch 1/3
6734/6734 [==============================] - 13s 2ms/step - loss: 0.1861
Epoch 2/3
6734/6734 [==============================] - 3s 381us/step - loss: 0.0244
Epoch 3/3
6734/6734 [==============================] - 3s 384us/step - loss: 0.0123
3366/3366 [==============================] - 6s 2ms/step
zero relu 0.2 64 SGD 8
Epoch 1/3
6733/6733 [==============================] - 16s 2ms/step - loss: 0.6475
Epoch 2/3
6733/6733 [==============================] - 3s 449us/step - loss: 0.5984
Epoch 3/3
6733/6733 [==============================] - 3s 481us/step - loss: 0.5804
3367/3367 [==============================] - 6s 2ms/step
zero relu 0.2 64 SGD 8
Epoch 1/3
6733/6733 [==============================] - 15s 2ms/step - loss: 0.6932
Epoch 2/3
6733/6733 [==============================] - 3s 480us/step - loss: 0.6931
Epoch 3/3
6733/6733 [==============================] - 3s 466us/step - loss: 0.6931
3367/3367 [==============================] - 7s 2ms/step
zero relu 0.2 64 SGD 8
Epoch 1/3
```

```
Epoch 1/3
6734/6734 [==============================] - 17s 3ms/step - loss: 0.6458
Epoch 2/3
6734/6734 [==============================] - 3s 478us/step - loss: 0.5946
Epoch 3/3
6734/6734 [==============================] - 3s 473us/step - loss: 0.5756
3366/3366 [==============================] - 29s 9ms/step
normal softplus 0.2 32 Adamax 16
/conda/envs/rapids/lib/python3.6/site-packages/sklearn/metrics/classification.py
  'precision', 'predicted', average, warn_for)
Epoch 1/3
10100/10100 [==============================] - 45s 4ms/step - loss: 0.0839
Epoch 2/3
10100/10100 [==============================] - 4s 378us/step - loss: 0.0059
Epoch 3/3
10100/10100 [==============================] - 4s 383us/step - loss: 0.0060
Best Accuracy for 0.9587020928678971 using {'optimizer': 'Adamax', 'neurons': 32
    mean_fit_time  std_fit_time  mean_score_time  std_score_time  \
0       10.897780      0.218733         0.690995        0.072834
1        7.590216      0.216886         0.782781        0.127035
2       18.643212      4.382407         1.450576        0.167937
3       13.039920      0.966130         1.491174        0.017065
4       10.966660      0.293137         1.745372        0.267001
5       13.483623      0.591179         1.878141        0.125859
6       17.003496      0.141466         2.675069        0.082975
7       12.021318      0.545806         2.171143        0.088243
8       12.927036      0.338144         2.580195        0.277893
9       12.516599      0.824696         2.774848        0.314708
10      17.664095      3.874935         3.182120        0.373091
11      17.350679      0.437707        18.634409       20.778184
12      14.770856      0.689301         3.286990        0.064497
13      19.384794      0.713799         4.247409        0.559529
14      23.598712      1.699786         4.794610        0.367646
15      23.999544      5.867081        12.480271        9.438167
16      21.881039      1.184168         5.237001        0.378992
17      18.523455      0.078021        17.759391       17.677416
18      22.478237      3.019329         5.772267        0.262168
19      23.715337      1.188894        13.724329       10.718296

   param_optimizer param_neurons param_init_mode param_filters  \
0         Adadelta            64   lecun_uniform             8
1             Adam            32         uniform             8
2           Adamax            64   lecun_uniform            16
3             Adam            64      he_uniform            16
4          RMSprop            64      he_uniform            16
5          Adagrad            32  glorot_uniform            16
6          Adagrad           128            zero            16
7             Adam            32      he_uniform             8
8              SGD            64   lecun_uniform             8
9             Adam            32   lecun_uniform             8
10           Nadam            32         uniform            16
11             SGD            32            zero            16
12            Adam            32   glorot_normal             8
13        Adadelta            64            zero             8
14          Adamax            64          normal            16
```

```
15         RMSprop            32  glorot_uniform             8
16         RMSprop            64          normal            16
17            Adam            32   glorot_normal             8
18          Adamax            32          normal            16
19             SGD            64            zero             8

   param_dropout_rate param_batch_size param_activation  \
0                 0.4               32     hard_sigmoid
1                 0.4               64             tanh
2                 0.4               32          softmax
3                 0.4               32          sigmoid
4                 0.4               64         softsign
5                 0.2               32         softsign
6                 0.6               32          softmax
7                 0.6               64     hard_sigmoid
8                 0.4               64          softmax
9                 0.4               64          sigmoid
10                0.4               64             relu
11                0.4               32         softplus
12                0.6               64          sigmoid
13                0.4               32         softsign
14                0.4               32     hard_sigmoid
15                0.6               32           linear
16                0.6               32          sigmoid
17                0.2               64     hard_sigmoid
18                0.2               64         softplus
19                0.2               32             relu

                                               params  split0_test_score  \
0   {'optimizer': 'Adadelta', 'neurons': 64, 'init...           0.678899
1   {'optimizer': 'Adam', 'neurons': 32, 'init_mod...           0.347418
2   {'optimizer': 'Adamax', 'neurons': 64, 'init_m...           0.021739
3   {'optimizer': 'Adam', 'neurons': 64, 'init_mod...           0.649123
4   {'optimizer': 'RMSprop', 'neurons': 64, 'init_...           0.457516
5   {'optimizer': 'Adagrad', 'neurons': 32, 'init_...           0.725490
6   {'optimizer': 'Adagrad', 'neurons': 128, 'init...           0.021739
7   {'optimizer': 'Adam', 'neurons': 32, 'init_mod...           0.166667
8   {'optimizer': 'SGD', 'neurons': 64, 'init_mode...           0.021739
9   {'optimizer': 'Adam', 'neurons': 32, 'init_mod...           0.164080
10  {'optimizer': 'Nadam', 'neurons': 32, 'init_mo...           0.320000
11  {'optimizer': 'SGD', 'neurons': 32, 'init_mode...           0.021739
12  {'optimizer': 'Adam', 'neurons': 32, 'init_mod...           0.104520
13  {'optimizer': 'Adadelta', 'neurons': 64, 'init...           0.021739
14  {'optimizer': 'Adamax', 'neurons': 64, 'init_m...           0.678899
15  {'optimizer': 'RMSprop', 'neurons': 32, 'init_...           0.732673
16  {'optimizer': 'RMSprop', 'neurons': 64, 'init_...           0.462500
17  {'optimizer': 'Adam', 'neurons': 32, 'init_mod...           0.138060
18  {'optimizer': 'Adamax', 'neurons': 32, 'init_m...           0.880952
19  {'optimizer': 'SGD', 'neurons': 64, 'init_mode...           0.021739

    split1_test_score  split2_test_score  mean_test_score  std_test_score  \
0            0.999393           1.000000         0.892753        0.151229
1            0.997577           1.000000         0.781643        0.307068
2            0.996973           0.000000         0.339604        0.464949
3            0.999696           1.000000         0.882928        0.165338
```

3	0.999090	1.000000	0.882928	0.165338
4	0.998787	1.000000	0.818750	0.255450
5	0.998787	1.000000	0.908083	0.129123
6	0.656961	0.000000	0.226256	0.304706
7	0.972255	1.000000	0.712945	0.386472
8	0.656961	0.000000	0.226256	0.304706
9	0.998182	1.000000	0.720726	0.393638
10	0.925033	0.991007	0.745322	0.301974
11	0.656961	0.000000	0.226256	0.304706
12	0.978610	1.000000	0.694346	0.417193
13	0.656961	0.000000	0.226256	0.304706
14	0.998484	1.000000	0.892451	0.151016
15	0.998484	1.000000	0.910377	0.125666
16	0.995166	1.000000	0.819204	0.252254
17	0.961471	1.000000	0.699814	0.397561
18	0.995166	1.000000	0.958702	0.055017
19	0.656961	0.000000	0.226256	0.304706

	rank_test_score
0	4
1	9
2	15
3	6
4	8
5	3
6	16
7	12
8	16
9	11
10	10
11	16
12	14
13	16
14	5
15	2
16	7
17	13
18	1
19	16

```
mean=0.8928, std=0.1512 using {'optimizer': 'Adadelta', 'neurons': 64, 'init_mod
mean=0.7816, std=0.3071 using {'optimizer': 'Adam', 'neurons': 32, 'init_mode':
mean=0.3396, std=0.4649 using {'optimizer': 'Adamax', 'neurons': 64, 'init_mode
mean=0.8829, std=0.1653 using {'optimizer': 'Adam', 'neurons': 64, 'init_mode':
mean=0.8187, std=0.2555 using {'optimizer': 'RMSprop', 'neurons': 64, 'init_mode
mean=0.9081, std=0.1291 using {'optimizer': 'Adagrad', 'neurons': 32, 'init_mode
mean=0.2263, std=0.3047 using {'optimizer': 'Adagrad', 'neurons': 128, 'init_mod
mean=0.7129, std=0.3865 using {'optimizer': 'Adam', 'neurons': 32, 'init_mode':
mean=0.2263, std=0.3047 using {'optimizer': 'SGD', 'neurons': 64, 'init_mode':
mean=0.7207, std=0.3936 using {'optimizer': 'Adam', 'neurons': 32, 'init_mode':
mean=0.7453, std=0.302 using {'optimizer': 'Nadam', 'neurons': 32, 'init_mode':
mean=0.2263, std=0.3047 using {'optimizer': 'SGD', 'neurons': 32, 'init_mode':
mean=0.6943, std=0.4172 using {'optimizer': 'Adam', 'neurons': 32, 'init_mode':
mean=0.2263, std=0.3047 using {'optimizer': 'Adadelta', 'neurons': 64, 'init_mod
mean=0.8925, std=0.151 using {'optimizer': 'Adamax', 'neurons': 64, 'init_mode'
mean=0.9104, std=0.1257 using {'optimizer': 'RMSprop', 'neurons': 32, 'init_mode
```

```
mean=0.8192, std=0.2523 using {'optimizer': 'RMSprop', 'neurons': 64, 'init_mode
mean=0.6998, std=0.3976 using {'optimizer': 'Adam', 'neurons': 32, 'init_mode':
mean=0.9587, std=0.05502 using {'optimizer': 'Adamax', 'neurons': 32, 'init_mode
mean=0.2263, std=0.3047 using {'optimizer': 'SGD', 'neurons': 64, 'init_mode':
```

```
randon_search_model = grid_result.best_estimator_
```

```
x_train_input = np.stack([x_train, uniform_filter1d(x_train, axis=1, size=200)], axis=2)

y_train_hat = randon_search_model.predict(x_train_input)[:,0]
y_train_pred = np.where(y_train_hat > 0.9,1.,0.)
print('Accuracy:', accuracy_score(y_train, y_train_pred))
print('F1 score:', f1_score(y_train, y_train_pred))
print('Recall:', recall_score(y_train, y_train_pred))
print('Precision:', precision_score(y_train, y_train_pred))
print('Cohens Kappa :', cohen_kappa_score(y_train, y_train_pred))
print('ROC AUC Score:', roc_auc_score(y_train, y_train_pred))
print('\n clasification report:\n', classification_report(y_train, y_train_pred))
print('\n confussion matrix:\n',confusion_matrix(y_train, y_train_pred))
```

```
5087/5087 [==============================] - 7s 1ms/step
Accuracy: 0.9986239433850993
F1 score: 0.9135802469135803
Recall: 1.0
Precision: 0.8409090909090909
Cohens Kappa : 0.9128919178749043
ROC AUC Score: 0.9993069306930693

 clasification report:
               precision    recall  f1-score   support

          0.0       1.00      1.00      1.00      5050
          1.0       0.84      1.00      0.91        37

     accuracy                           1.00      5087
    macro avg       0.92      1.00      0.96      5087
 weighted avg       1.00      1.00      1.00      5087


 confussion matrix:
 [[5043    7]
 [   0   37]]
```

```
y_hat = randon_search_model.predict(x_test)[:,0]
y_pred = np.where(y_hat > 0.9,1.,0.)
print('Accuracy:', accuracy_score(y_test, y_pred))
print('F1 score:', f1_score(y_test, y_pred))
print('Recall:', recall_score(y_test, y_pred))
print('Precision:', precision_score(y_test, y_pred))
print('Cohens Kappa :', cohen_kappa_score(y_test, y_pred))
print('ROC AUC Score:', roc_auc_score(y_test, y_pred))
print('\n clasification report:\n', classification_report(y_test,y_pred))
print('\n confussion matrix:\n',confusion_matrix(y_test, y_pred))
```

```
570/570 [==============================] - 0s 494us/step
Accuracy: 0.9964912280701754
F1 score: 0.7499999999999999
Recall: 0.6
Precision: 1.0
Cohens Kappa : 0.7483443708609272
ROC AUC Score: 0.8

 clasification report:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00       565
         1.0       1.00      0.60      0.75         5

    accuracy                           1.00       570
   macro avg       1.00      0.80      0.87       570
weighted avg       1.00      1.00      1.00       570


 confussion matrix:
 [[565   0]
 [  2   3]]
```

We got the best result for model with these parameters in random search:

1. Initializers: He_uniform
2. Activation funtions: Sigmoid
3. Drop rates: 0.4
4. Number of neurons: 64
5. Batch size : 32
6. Optimizers: Adam
7. Filter sizes: 16 However the performance of model two was still better than this model.

```
randon_search_conv_model = randon_search_model
final_model_json = randon_search_conv_model.model.to_json()
with open("final_model_rs_conv.json", "w") as json_file:
    json_file.write(final_model_json)
print("json written")

# Save model 1 - serialize model to YAML
model_yaml = randon_search_conv_model.model.to_yaml()
with open("final_model_rs_conv.yaml", "w") as yaml_file:
    yaml_file.write(model_yaml)
print("yaml written")

# Serialize weights to HDF5
randon_search_conv_model.model.save_weights("final_model_rs_conv_weights.h5")
print("weights written")

pickle.dump(randon_search_conv_model.model, open('final_model_rs_conv.pkl', 'wb'))
print("pickle written")

randon_search_conv_model.model.save("final_model_rs_conv.h5")

print("Saved model to disk")
```

```
json written
yaml written
weights written
pickle written
Saved model to disk
```

```
model = tf.keras.models.load_model('final_model_rs_conv.h5',
        custom_objects={
            #'recall_m' : recall_m,
            #'precision_m' : precision_m,
            #'f1_m' : f1_m
            'f1' : f1
            })
```

```
# print results
print(f'Best Accuracy for {grid_result.best_score_} using {grid_result.best_params_}')
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
#means_train = grid_result.cv_results_['mean_train_score']
#stds_train = grid_result.cv_results_['std_train_score']
params = grid_result.cv_results_['params']
print("\nTop results and params:")
for mean, stdev, param in zip(means, stds, params):
    print(f' mean={mean:.4}, std={stdev:.4} using {param}')
```

```
Best Accuracy for 0.9587020928678971 using {'optimizer': 'Adamax', 'neurons': 32

Top results and params:
 mean=0.8928, std=0.1512 using {'optimizer': 'Adadelta', 'neurons': 64, 'init_mod
 mean=0.7816, std=0.3071 using {'optimizer': 'Adam', 'neurons': 32, 'init_mode':
 mean=0.3396, std=0.4649 using {'optimizer': 'Adamax', 'neurons': 64, 'init_mode
 mean=0.8829, std=0.1653 using {'optimizer': 'Adam', 'neurons': 64, 'init_mode':
 mean=0.8187, std=0.2555 using {'optimizer': 'RMSprop', 'neurons': 64, 'init_mode
 mean=0.9081, std=0.1291 using {'optimizer': 'Adagrad', 'neurons': 32, 'init_mode
 mean=0.2263, std=0.3047 using {'optimizer': 'Adagrad', 'neurons': 128, 'init_mod
 mean=0.7129, std=0.3865 using {'optimizer': 'Adam', 'neurons': 32, 'init_mode':
 mean=0.2263, std=0.3047 using {'optimizer': 'SGD', 'neurons': 64, 'init_mode':
 mean=0.7207, std=0.3936 using {'optimizer': 'Adam', 'neurons': 32, 'init_mode':
 mean=0.7453, std=0.302 using {'optimizer': 'Nadam', 'neurons': 32, 'init_mode':
 mean=0.2263, std=0.3047 using {'optimizer': 'SGD', 'neurons': 32, 'init_mode':
 mean=0.6943, std=0.4172 using {'optimizer': 'Adam', 'neurons': 32, 'init_mode':
 mean=0.2263, std=0.3047 using {'optimizer': 'Adadelta', 'neurons': 64, 'init_mod
 mean=0.8925, std=0.151 using {'optimizer': 'Adamax', 'neurons': 64, 'init_mode'
 mean=0.9104, std=0.1257 using {'optimizer': 'RMSprop', 'neurons': 32, 'init_mode
 mean=0.8192, std=0.2523 using {'optimizer': 'RMSprop', 'neurons': 64, 'init_mode
 mean=0.6998, std=0.3976 using {'optimizer': 'Adam', 'neurons': 32, 'init_mode':
 mean=0.9587, std=0.05502 using {'optimizer': 'Adamax', 'neurons': 32, 'init_mode
 mean=0.2263, std=0.3047 using {'optimizer': 'SGD', 'neurons': 64, 'init_mode':
```

Prepare data for TPOT

```
train = pd.read_csv('../webapp/web/exoTrain.csv')
test = pd.read_csv('../webapp/web/exoTest.csv')
print(train.shape, test.shape)
```

```
train.rename(columns={'LABEL': 'class'}, inplace=True)
test.rename(columns={'LABEL': 'class'}, inplace=True)
print(train.isnull().values.any(), test.isnull().values.any())
train=train.dropna()
test=test.dropna()
print(train.shape, test.shape)


print(train.isnull().values.any(), test.isnull().values.any())



for col_name in train.columns:
    if(train[col_name].dtype == 'object'):
        print("train:", col_name)
        train[col_name]= train[col_name].astype('category')
        train[col_name] = train[col_name].cat.codes

for col_name in test.columns:
    if(test[col_name].dtype == 'object'):
        print("tests:", col_name)
        test[col_name]= test[col_name].astype('category')
        test[col_name] = test[col_name].cat.codes



X_train = train.drop('class', axis=1)
#X = X.drop('Loan_ID',axis=1)
y_train = train['class']

X_test = test.drop('class', axis=1)
#X = X.drop('Loan_ID',axis=1)
y_test = test['class']

print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
X_train_array=X_train.values
X_test_array=X_test.values
print(type(X_train_array),type(X_test_array), type(y_train), np.shape(X_train_array), np.sha


#X_train, X_test, y_train, y_test = train_test_split(X, y.values,train_size=0.90, test_size=
#X_train_array=X_train.values
#X_test_array=X_test.values
#print(type(X_train_array),type(X_test_array), type(y_train), np.shape(X_train_array), np.sl
```

```
(5087, 3198) (570, 3198)
False False
(5087, 3198) (570, 3198)
False False
(5087, 3197) (570, 3197) (5087,) (570,)
<class 'numpy.ndarray'> <class 'numpy.ndarray'> <class 'pandas.core.series.Series
```

AUTOML Platforms democratizes machine learning by making it accessible for everyone. AUTOML platforms involv engineering, algorithm selection, hyperparameter optimization, model tuning, etc. TPOT specifically is based on Gen representation of solution domain and a fitness function to evaluate the solution domain. TPOT will consider a popu evolution laws to have them optimize the objective function called fitness. For each generation it will select the best fitness function and use genetic operations to reproduce next generation. Recombination (exploitation) combines pa Mutations introduce random (exploration) perturbations. This way, the average population's fitness is supposed to i at fittest individual. However the model two was still performing better than the model from TPOT.

```
!pwd
```

```
!mkdir tpot_checkpoint
```

```
/rapids/notebooks/utils/hostdir/capstone/predict_exoplanet/util
```

```
exoplanet_model_tpot = TPOTClassifier(
                         generations = 5,    #100: Number of iterations to the run pipelin
                         population_size=10, #100: Number of individuals to retain in the 
                         offspring_size = 2, #None: Number of offspring to produce in each 
                         mutation_rate=0.8,  #0.9: Mutation rate for the genetic programmi
                         crossover_rate=0.2, #0.1: Crossover rate for the genetic programm
                         scoring='accuracy', #accuracy: One of the available scoring funct
                         cv=5,               #5: Cross-validation strategy used when evalu
                         subsample = 1.0,    #1.0: Fraction of training samples that are u
                         n_jobs = 1,         #1: Number of processes to use in parallel fo
                         max_time_mins=30,  #None: How many minutes TPOT has to optimize t
                         max_eval_time_mins=5,    #5: How many minutes TPOT has to evaluat
                         random_state=3,     #None: Integer. Use this parameter to make su
                         config_dict = None, #None: 'TPOT light', 'TPOT Spare', 'TPOT MDR'
                      #   template = None,    #None: Template of predefined pipeline struc
                         warm_start = True,  #False: Flag indicating whether the TPOT inst
                         memory = 'auto',    #None: 'auto' or a caching dir  path or Memor
                         use_dask = False,   #False: Whether to use Dask-ML's pipeline opt
                         periodic_checkpoint_folder = './tpot_checkpoint', #None: If suppl
                         early_stop=None,    #None: Ends the optimization process if there
                         verbosity=3,        #0: 0 to 4.  How much information TPOT commun
                        )
```

```
exoplanet_model_tpot.fit(X_train_array, y_train)
```

```
30 operators have been imported by TPOT.
HBox(children=(IntProgress(value=0, description='Optimization Progress', max=10,
Skipped pipeline #1 due to time out. Continuing to the next pipeline.
Skipped pipeline #6 due to time out. Continuing to the next pipeline.
Skipped pipeline #9 due to time out. Continuing to the next pipeline.
Skipped pipeline #12 due to time out. Continuing to the next pipeline.
Saving periodic pipeline from pareto front to ./tpot_checkpoint/pipeline_gen_1_ic
Skipped pipeline #16 due to time out. Continuing to the next pipeline.
Generation 1 - Current Pareto front scores:
-1      0.992726788022092       ExtraTreesClassifier(input_matrix, ExtraTreesCla

Periodic pipeline was not saved, probably saved before...
Skipped pipeline #18 due to time out. Continuing to the next pipeline.

34.80017466666666 minutes have elapsed. TPOT will close down.
TPOT closed during evaluation in one generation.
WARNING: TPOT may not provide a good pipeline if TPOT is stopped/interrupted in a


TPOT closed prematurely. Will use the current best pipeline.
TPOTClassifier(config_dict=None, crossover_rate=0.2, cv=5,
               disable_update_check=False, early_stop=None, generations=1000000,
               max_eval_time_mins=5, max_time_mins=30, memory='auto',
               mutation_rate=0.8, n_jobs=1, offspring_size=2,
               periodic_checkpoint_folder='./tpot_checkpoint',
               population_size=10, random_state=3, scoring='accuracy',
               subsample=1.0, template='RandomTree', use_dask=False,
               verbosity=3, warm_start=True)
```

```
y_pred=exoplanet_model_tpot.predict(X_test_array)
y_train_pred=exoplanet_model_tpot.predict(X_train_array)
```

```
from sklearn.metrics import precision_score, recall_score, confusion_matrix, classification_

print('Accuracy:', accuracy_score(y_test, y_pred))
print('F1 score:', f1_score(y_test, y_pred))
print('Recall:', recall_score(y_test, y_pred))
print('Precision:', precision_score(y_test, y_pred))
print('\n clasification report:\n', classification_report(y_test,y_pred))
print('\n confussion matrix:\n',confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.9912280701754386
F1 score: 0.9955947136563876
Recall: 1.0
Precision: 0.9912280701754386

 clasification report:
               precision    recall  f1-score   support

           1       0.99      1.00      1.00       565
           2       0.00      0.00      0.00         5

    accuracy                           0.99       570
   macro avg       0.50      0.50      0.50       570
weighted avg       0.98      0.99      0.99       570


 confussion matrix:
 [[565   0]
 [  5   0]]
/conda/envs/rapids/lib/python3.6/site-packages/sklearn/metrics/classification.py
  'precision', 'predicted', average, warn_for)
/conda/envs/rapids/lib/python3.6/site-packages/sklearn/metrics/classification.py
  'precision', 'predicted', average, warn_for)
/conda/envs/rapids/lib/python3.6/site-packages/sklearn/metrics/classification.py
  'precision', 'predicted', average, warn_for)
```

```
print(exoplanet_model_tpot.score(X_test_array, y_test))
```

```
0.9912280701754386
```

```
print('Accuracy:', accuracy_score(y_train, y_train_pred))
print('F1 score:', f1_score(y_train, y_train_pred))
print('Recall:', recall_score(y_train, y_train_pred))
print('Precision:', precision_score(y_train, y_train_pred))
print('\n clasification report:\n', classification_report(y_train, y_train_pred))
print('\n confussion matrix:\n',confusion_matrix(y_train, y_train_pred))
```

```
Accuracy: 0.9927265578926676
F1 score: 0.9963500049324258
Recall: 1.0
Precision: 0.9927265578926676
/conda/envs/rapids/lib/python3.6/site-packages/sklearn/metrics/classification.py
  'precision', 'predicted', average, warn_for)
/conda/envs/rapids/lib/python3.6/site-packages/sklearn/metrics/classification.py
  'precision', 'predicted', average, warn_for)
/conda/envs/rapids/lib/python3.6/site-packages/sklearn/metrics/classification.py
  'precision', 'predicted', average, warn_for)

 clasification report:
               precision    recall  f1-score   support

            1       0.99      1.00      1.00      5050
            2       0.00      0.00      0.00        37

    accuracy                           0.99      5087
   macro avg       0.50      0.50      0.50      5087
weighted avg       0.99      0.99      0.99      5087


 confussion matrix:
 [[5050    0]
 [  37    0]]
```

```
exoplanet_model_tpot.export('exoplanet_model_tpot.py')

#tpot_model2.export('tpot_loanpred_model2_30mins.py')

!cat exoplanet_model_tpot.py
```

```
import numpy as np
import pandas as pd
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.model_selection import train_test_split

# NOTE: Make sure that the class is labeled 'target' in the data file
tpot_data = pd.read_csv('PATH/TO/DATA/FILE', sep='COLUMN_SEPARATOR', dtype=np.flo
features = tpot_data.drop('target', axis=1).values
training_features, testing_features, training_target, testing_target = \
            train_test_split(features, tpot_data['target'].values, random_state=

# Average CV score on the training set was:0.992726788022092
exported_pipeline = ExtraTreesClassifier(bootstrap=False, criterion="gini", max_

exported_pipeline.fit(training_features, training_target)
results = exported_pipeline.predict(testing_features)
```

```
import pickle
#exoplanet_model_tpot_pickle = pickle.dumps(exoplanet_model_tpot.fitted_pipeline_)
#exoplanet_model_tpot1 = pickle.loads(tpot_exoplanet_model_pickle)
```

```
# save the model to disk
pickle.dump(exoplanet_model_tpot.fitted_pipeline_, open('exoplanet_model_tpot_pickle.pkl',

# load the model from disk
exoplanet_model_tpot1 = pickle.load(open('exoplanet_model_tpot_pickle.pkl', 'rb'))

!ls -lrt exoplanet_model_tpot_pickle.pkl
print(exoplanet_model_tpot.score(X_test_array, y_test))
```

```
-rw-r--r-- 1 root root 738950 Oct 13 19:38 exoplanet_model_tpot_pickle.pkl
0.9912280701754386
```

```
from joblib import dump, load
dump(exoplanet_model_tpot.fitted_pipeline_, 'exoplanet_model_tpot.joblib')
exoplanet_model_tpot2 = load('exoplanet_model_tpot.joblib')
print(exoplanet_model_tpot2.score(X_test_array, y_test))
!ls -lrt exoplanet_model_tpot.joblib
```

```
0.9912280701754386
-rw-r--r-- 1 root root 734039 Oct 13 19:38 exoplanet_model_tpot.joblib
```

```
!mkdir /rapids/notebooks/utils/hostdir/capstone/tpot_checkpoint2
```

```
mkdir: cannot create directory '/rapids/notebooks/utils/hostdir/capstone/tpot_che
```

```
exoplanet_model_tpot_f1 = TPOTClassifier(
                          generations = 5,    #100: Number of iterations to the run pipeline
                          population_size=10, #100: Number of individuals to retain in the g
                          offspring_size = 2, #None: Number of offspring to produce in each
                          mutation_rate=0.8,  #0.9: Mutation rate for the genetic programmi
                          crossover_rate=0.2, #0.1: Crossover rate for the genetic programm
                          scoring='f1_weighted', #accuracy: One of the available scoring fu
                          cv=5,               #5: Cross-validation strategy used when evalua
                          subsample = 1.0,    #1.0: Fraction of training samples that are us
                          n_jobs = 1,         #1: Number of processes to use in parallel fo
                          max_time_mins=30,  #None: How many minutes TPOT has to optimize th
                          max_eval_time_mins=5,    #5: How many minutes TPOT has to evaluate
                          random_state=3,     #None: Integer. Use this parameter to make su
                          config_dict = 'TPOT light', #None: 'TPOT light', 'TPOT Spare', 'TP
                      #   template = None,    #None: Template of predefined pipeline struct
                          warm_start = False,  #False: Flag indicating whether the TPOT inst
                          memory = 'auto',    #None: 'auto' or a caching dir  path or Memory
                          use_dask = False,   #False: Whether to use Dask-ML's pipeline opti
                          periodic_checkpoint_folder = '/rapids/notebooks/utils/hostdir/caps
                          early_stop=None,    #None: Ends the optimization process if there
                          verbosity=3,        #0: 0 to 4.  How much information TPOT commun
                         )
```

```
exoplanet_model_tpot_f1.fit(X_train_array, y_train)
```

```
19 operators have been imported by TPOT.
HBox(children=(IntProgress(value=0, description='Optimization Progress', max=10,
Saving periodic pipeline from pareto front to /rapids/notebooks/utils/hostdir/ca
Pipeline encountered that has previously been evaluated during the optimization
Generation 1 - Current Pareto front scores:
-1      0.9891035129414358      KNeighborsClassifier(input_matrix, KNeighborsCla

Periodic pipeline was not saved, probably saved before...
_pre_test decorator: _random_mutation_operator: num_test=0 This solver needs sam
_pre_test decorator: _random_mutation_operator: num_test=1 index 1 is out of bou
_pre_test decorator: _random_mutation_operator: num_test=2 This solver needs sam
Generation 2 - Current Pareto front scores:
-1      0.9891035129414358      KNeighborsClassifier(input_matrix, KNeighborsCla

Periodic pipeline was not saved, probably saved before...
_pre_test decorator: _random_mutation_operator: num_test=0 This solver needs sam
Generation 3 - Current Pareto front scores:
-1      0.9891035129414358      KNeighborsClassifier(input_matrix, KNeighborsCla

Periodic pipeline was not saved, probably saved before...
Generation 4 - Current Pareto front scores:
-1      0.9891035129414358      KNeighborsClassifier(input_matrix, KNeighborsCla

Periodic pipeline was not saved, probably saved before...
Generation 5 - Current Pareto front scores:
-1      0.9891035129414358      KNeighborsClassifier(input_matrix, KNeighborsCla

Periodic pipeline was not saved, probably saved before...
Generation 6 - Current Pareto front scores:
-1      0.9891035129414358      KNeighborsClassifier(input_matrix, KNeighborsCla

Periodic pipeline was not saved, probably saved before...
Generation 7 - Current Pareto front scores:
-1      0.9891035129414358      KNeighborsClassifier(input_matrix, KNeighborsCla

Periodic pipeline was not saved, probably saved before...
_pre_test decorator: _random_mutation_operator: num_test=0 Expected n_neighbors <
Generation 8 - Current Pareto front scores:
-1      0.9891035129414358      KNeighborsClassifier(input_matrix, KNeighborsCla

Periodic pipeline was not saved, probably saved before...
Skipped pipeline #27 due to time out. Continuing to the next pipeline.
Generation 9 - Current Pareto front scores:
-1      0.9891035129414358      KNeighborsClassifier(input_matrix, KNeighborsCla

Periodic pipeline was not saved, probably saved before...

31.4415732 minutes have elapsed. TPOT will close down.
TPOT closed during evaluation in one generation.
WARNING: TPOT may not provide a good pipeline if TPOT is stopped/interrupted in a


TPOT closed prematurely. Will use the current best pipeline.
TPOTClassifier(config_dict='TPOT light', crossover_rate=0.2, cv=5,
```

```
                disable_update_check=False, early_stop=None, generations=1000000,
                max_eval_time_mins=5, max_time_mins=30, memory='auto',
                mutation_rate=0.8, n_jobs=1, offspring_size=2,
                periodic_checkpoint_folder='/rapids/notebooks/utils/hostdir/capsto
                population_size=10, random_state=3, scoring='f1_weighted',
                subsample=1.0, template='RandomTree', use_dask=False,
                verbosity=3, warm_start=False)
```

```
import pickle
#exoplanet_model_tpot_pickle = pickle.dumps(exoplanet_model_tpot.fitted_pipeline_)
#exoplanet_model_tpot1 = pickle.loads(tpot_exoplanet_model_pickle)
# save the model to disk
pickle.dump(exoplanet_model_tpot_f1.fitted_pipeline_, open('exoplanet_model_tpot_pickle_f1.p

# load the model from disk
exoplanet_model_tpot1_f1_load = pickle.load(open('exoplanet_model_tpot_pickle_f1.pkl', 'rb'

!ls -lrt exoplanet_model_tpot_pickle.pkl
print(exoplanet_model_tpot1_f1_load.score(X_test_array, y_test))
```

```
-rw-r--r-- 1 root root 738950 Oct 13 19:38 exoplanet_model_tpot_pickle.pkl
0.9912280701754386
```

```
y_pred2=exoplanet_model_tpot1_f1_load.predict(X_test_array)
y_train_pred2=exoplanet_model_tpot1_f1_load.predict(X_train_array)
print('Accuracy:', accuracy_score(y_train, y_train_pred2))
print('F1 score:', f1_score(y_train, y_train_pred2))
print('Recall:', recall_score(y_train, y_train_pred2))
print('Precision:', precision_score(y_train, y_train_pred2))
print('\n clasification report:\n', classification_report(y_train, y_train_pred2))
print('\n confussion matrix:\n',confusion_matrix(y_train, y_train_pred2))

print('Accuracy:', accuracy_score(y_test, y_pred2))
print('F1 score:', f1_score(y_test, y_pred2))
print('Recall:', recall_score(y_test, y_pred2))
print('Precision:', precision_score(y_test, y_pred2))
print('\n clasification report:\n', classification_report(y_test,y_pred2))
print('\n confussion matrix:\n',confusion_matrix(y_test, y_pred2))
```

```
Accuracy: 0.9927265578926676
F1 score: 0.9963500049324258
Recall: 1.0
Precision: 0.9927265578926676

 clasification report:
               precision    recall  f1-score   support

           1       0.99      1.00      1.00      5050
           2       0.00      0.00      0.00        37

    accuracy                           0.99      5087
   macro avg       0.50      0.50      0.50      5087
weighted avg       0.99      0.99      0.99      5087


 confussion matrix:
 [[5050    0]
 [  37    0]]
Accuracy: 0.9912280701754386
F1 score: 0.9955947136563876
Recall: 1.0
Precision: 0.9912280701754386

 clasification report:
               precision    recall  f1-score   support

           1       0.99      1.00      1.00       565
           2       0.00      0.00      0.00         5

    accuracy                           0.99       570
   macro avg       0.50      0.50      0.50       570
weighted avg       0.98      0.99      0.99       570


 confussion matrix:
 [[565   0]
 [  5   0]]
/conda/envs/rapids/lib/python3.6/site-packages/sklearn/metrics/classification.py
  'precision', 'predicted', average, warn_for)
/conda/envs/rapids/lib/python3.6/site-packages/sklearn/metrics/classification.py
  'precision', 'predicted', average, warn_for)
/conda/envs/rapids/lib/python3.6/site-packages/sklearn/metrics/classification.py
  'precision', 'predicted', average, warn_for)
/conda/envs/rapids/lib/python3.6/site-packages/sklearn/metrics/classification.py
  'precision', 'predicted', average, warn_for)
/conda/envs/rapids/lib/python3.6/site-packages/sklearn/metrics/classification.py
  'precision', 'predicted', average, warn_for)
/conda/envs/rapids/lib/python3.6/site-packages/sklearn/metrics/classification.py
  'precision', 'predicted', average, warn_for)
```