Capstone Project –Identifying and Finding Exoplanets in deep space - Final 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 such as Nasa / ESA/ ISRO provides a lot of open source datasets; Take a simple example Nasa Hubble telescope, Kepler telescope are used to observe the deep space movements of stars, galaxies, black holes, neutron stars, asteroids and others; Presently, Nasa observed black hole and taken the full picture of it, that was remarkable achievement done by human era of this 21st century; Even more NASA planned to have better than Hubble telescope to observe or to visualize beyond the current limitations, they have planned to launch 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 stars and galaxies over the period of time, even before our earth came into better shape for habitual human civilization. As per this calculation, Space researchers come up with the problem statement, are we alone in this universe? Is there any other earth like planets out there in our universe? In search for answers, space researches started observing deep space (our universe), based on light intensity (one of the parameter), radiation, spectrum of wavelengths, movements of planets in an periodic time frames; they captured and collected data to solve this problem; That's why our capstone project gives ample of concept called "Finding earth like planets in deep space" or "Identifying and Finding Exoplanets in deep space"; Solution outcome: In future, imagine scenarios like Spaceships are available faster than speed of light is possible, then colonizing Exoplanets which was distant away from several light years also become prominent success factors for human race; So we are pretty much privilege to 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)" Flux is light intensity and change in flux of several thousand stars was observed. Each of the star given as binary values 2 and 1; 2 indicated that the star has at least one exoplanet in orbit and on the other hand 1 indicates non-exoplanet in an orbit. The data presented here are cleaned and are derived from observations made by the NASA Kepler space telescope.

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 pre-processing steps, the algorithms you used, and how you combined techniques.

1. We have used preprocessing steps, like separating Exoplanets and Non-Exoplanets train & test datasets. Finding each describe stats
2. We have used Machine Learning Alogrithms like, SGD Classifier for Linear classification, Decision Tree and Random Forest, Naive Bayes, Boosting algorithms like XGBoost, AdaBoost, PCA
3. We have used CNN 1D deep learning and in that we have used different optimization techniques such as SGD, ADAM with different learning rates

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 steps? Build up to the final solution.

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 transform
3. We have applied different Machine learning models and find it's accuracy, F1 score, created confusion matrix for each Ml / Deep learning models

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 meterages
3. Standard deviation light intensity mostly above zero value, shows that +/- square root values on each instances

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 decomposed seasonal patterns
3. Residual graph shows the inverse instance values of original graph and smoothened seasonal decomposed graph
4. Fourier decomposed graph shows at the beginning and at the end, distorted frequency wavelengths are available;

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 decaying from high to low after reaching 200 plus points it stays in the same level (horizontal view);

The **ploCompleteTestDataGraph** graph provides the following insights:

1. Blue graph shows light intensity value of exoplanets, which has less data and hence frequency distribution density is low;
2. Red graph shows light intensity value of non-exoplanets, which has the slightly huge data and hence the frequency distribution density is high

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, recall and precision.

Model one - base model: Uses CNN Conv1D with max pooling, batch norm, fixed kernel and filter size and drop out, dense layers with Relu for non-linearity in hidden layers and Sigmoid in the final dense layer, Adam optimizer with fixed learning rate using accuracy, f1, precision, recall as metrics functions.

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 models we use SMOTE for data sampling to balance 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 involves automation of feature preprocessing, feature engineering, algorithm selection, hyperparameter optimization, model tuning, etc. TPOT specifically is based on Genetic Alogrithms where it performs genetic representation of solution domain and a fitness function to evaluate the solution domain. TPOT will consider a population in solution domain and apply simplified evolution laws to have them optimize the objective function called fitness. For each generation it will select the best (meaning here the fittest) individuals using fitness function and use genetic operations to reproduce next generation. Recombination (exploitation) combines parent features to form children solutions. Mutations introduce random (exploration) perturbations. This way, the average population's fitness is supposed to improve from one generation to the next to arrive 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 did you evaluate the success of your models(s)? A convincing explanation of the robustness of your solution will go a long way to supporting your solution. .

- When compared to Machine Learning model vs Deep learning CNN 1D model, CNN 1D model performed well based on accuracy score and in specific CNN 1D 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 models also works fine such as SGD classifier, 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 deployed the model as REST Web Application using Flask. We have exposes a Rest API call exoplanet_predict to get the prediction based on the input FLUX values. The REST service takes FLUX values as JSON object and returns the predicted Label. Since the number of independent features are very high, we used a custom written REST client to invoke the API. The

custom client will read the test csv rows and convert each row into a JSON object and invoke the REST API. This REST Web application is containerized using Docker and is deployed locally as well as on Google cloud.

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 benchmark? Why or why not?

- We have used several machine learning / deep learning algorithms with different parameters optimization techniques, so we have applied several solutions 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 dataset. Our model has been regularized well and hence it is performing well on test dataset. The final model is able to detect all 5 exoplanets correctly from test dataset.

6. Visualization(s)

In addition to quantifying your model and the solution, please include all relevant visualizations that support the ideas/insights that you gleaned from the data.

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

7. Implications

How does your solution affect the problem in the domain or business? What recommendations would you make, and with what level of confidence?

- Based on our solution, as we have different approaches for finding Exoplanets in Deep Space; When considering Future days, In Space Exploration, Space scienttist observed and collected the Exoplanets datas; We can take the Raw datasets and aggregate it to label 1 and label 2 for labelling it out as Exoplanet 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 enhance the solution?
We have only kepler observed Exoplanet Datasets which also aggregated it; This AI/ML model solution is useful, when any telescope observed and aggragated 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; used deep learning with different optimizers, initializers, architectures, etc. However we can further study the effect of feature engineering using feature crosses, polynomial features.

Our model deployment was done using a simple web application. In a production setup, the solution can be implemented using approaches like lambda architecture to segregate data flow between batch processing and real time processing. We can use document database like Mongo DB (instead of csv), streaming platform like Kafka for data store and ingestion. We can setup independent batch training jobs using Spark ML and setup deployment pipeline using Kubernetes.

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 Exoplanets in deep space using various ML, Deep Learning algorithms; Next time, We have planned to collect image datasets on the exoplanets; Here, we have used only normal dataset values

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

In [0]:

```
#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
```

```
Requirement already satisfied: imblearn in /conda/envs/rapids/lib/python3.6/
site-packages (0.0)
Requirement already satisfied: imbalanced-learn in /conda/envs/rapids/lib/py
thon3.6/site-packages (from imblearn) (0.5.0)
Requirement already satisfied: scikit-learn>=0.21 in /conda/envs/rapids/lib/
python3.6/site-packages (from imbalanced-learn->imblearn) (0.21.1)
Requirement already satisfied: joblib>=0.11 in /conda/envs/rapids/lib/python
3.6/site-packages (from imbalanced-learn->imblearn) (0.13.2)
Requirement already satisfied: numpy>=1.11 in /conda/envs/rapids/lib/python
3.6/site-packages (from imbalanced-learn->imblearn) (1.16.2)
Requirement already satisfied: scipy>=0.17 in /conda/envs/rapids/lib/python
3.6/site-packages (from imbalanced-learn->imblearn) (1.2.1)
Requirement already satisfied: deap in /conda/envs/rapids/lib/python3.6/site
-packages (1.2.2)
Requirement already satisfied: update_checker in /conda/envs/rapids/lib/pyth
on3.6/site-packages (0.16)
Requirement already satisfied: tqdm in /conda/envs/rapids/lib/python3.6/site
-packages (4.32.1)
Requirement already satisfied: stopit in /conda/envs/rapids/lib/python3.6/si
```

In [0]:

```
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.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, class
from sklearn.metrics import accuracy_score, f1_score, cohen_kappa_score, roc_auc_s
from sklearn import model_selection
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV, ParameterGri
from sklearn.model_selection import train_test_split, cross_val_score, cross_val_p
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
```

```
Using TensorFlow backend.
/conda/envs/rapids/lib/python3.6/site-packages/sklearn/externals/joblib/__init
__.py:15: DeprecationWarning: sklearn.externals.joblib is deprecated in 0.21 a
nd will be removed in 0.23. Please import this functionality directly from job
lib, which can be installed with: pip install joblib. If this warning is raise
d when loading pickled models, you may need to re-serialize those models with
scikit-learn 0.21+.
  warnings.warn(msg, category=DeprecationWarning)
```

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

In [0]:

```
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 i
  @staticmethod
  def get_distribution_params(intensity_vals, window_size=10):
      return intensity_vals.rolling(window_size).mean(), intensity_vals.rolling(wi

  # 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_distribut

    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 meterag
    red_line = mlines.Line2D([], [], color='red', label='Mean light intensity')
    green_line = mlines.Line2D([], [], color='green', label='Standard deviation of
    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])],a
  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])],alp
  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="
  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_predic
  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_node
  dtc = DecisionTreeClassifier(criterion='entropy',max_depth=max_depth_val,max_l
  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, scor
        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,
  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=5
  ada_model = AdaBoostClassifier(base_estimator=model,n_estimators= estimators_t
  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

In [0]:
```
exoplanets =  Exoplanets_DeepSpace()
```

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

In [0]:
```
#exoTrain = exoplanets.readCsv_File('/content/drive/My Drive/capstone_datasets/exo
exoTrain = exoplanets.readCsv_File('exoTrain.csv')
```

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

In [0]:
```
#exoTest = exoplanets.readCsv_File('/content/drive/My Drive/capstone_datasets/exoT
exoTest = exoplanets.readCsv_File('exoTest.csv')
```

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

In [0]:
```
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 columns 2 contains (3198 columns) the flux values over time.

In test dataset, we have 570 rows/observations and 3198 columns/features. Column 1 is the label vector and columns 2 contains (3198 columns) the flux values over time.

1.5 Print the number of Exoplanets and Non-Exoplanets by using onlyExoplanets and

onlyNonExoplanets method respectively in Exoplanets_DeepSpace()

In [0]:
```
onlyExoplanetsTrain = exoplanets.onlyExoplanets(exoTrain)
onlyNonExoplanetsTrain = exoplanets.onlyNonExoplanets(exoTrain)
onlyExoplanetsTest = exoplanets.onlyExoplanets(exoTest)
onlyNonExoplanetsTest = exoplanets.onlyNonExoplanets(exoTest)
```

In [0]:
```
print("Shape of explonets from train and test data :: ", exoplanets.shapeOfData(on
print("Shape of non-explonets from train and test data :: ", exoplanets.shapeOfDat
```

```
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()

In [0]:
```
print("Shape of explonets from train and test data's info :: ",exoplanets.features
print("Shape of non-explonets from train and test data's info :: ", exoplanets.fea
```

```
<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()

In [0]:
```
print("Stats of explonets from train and test data's stats description :: ", exopl
print("Stats of non-explonets from train and test data's stats description :: ", e
```

```
Stats of explonets from train and test data's stats description ::          L
ABEL           FLUX.1          FLUX.2          FLUX.3         FLUX.4  \
count   37.0       37.000000       37.000000       37.000000      37.000000
mean     2.0     4096.965405     3524.308108     2771.165405    2223.029189
std      0.0    24781.136932    21305.507167    16806.475389   13533.909607
min      2.0    -1831.310000    -1781.440000    -1930.840000   -2016.720000
25%      2.0      -66.470000      -76.330000      -76.230000     -72.250000
50%      2.0       31.290000       46.370000       33.300000      16.630000
75%      2.0      207.370000      163.570000      150.450000     135.340000
max      2.0   150725.800000   129578.360000   102184.980000   82253.980000

             FLUX.5         FLUX.6         FLUX.7         FLUX.8        FLUX.9
\
count     37.000000      37.000000      37.000000      37.000000     37.000000
mean    1836.404324    1283.214595    1129.359730     486.174865     46.912703
std    11181.958157    7923.521630    7055.157534    3178.094634    767.865143
min    -1963.310000   -1956.120000   -2128.240000   -2188.200000  -2212.820000
25%      -79.310000    -115.640000    -135.180000     -96.270000    -83.970000
50%       17.010000      -3.760000      -4.090000       5.320000      1.480000
```

- 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()

In [0]:

```python
print("explonets from train and test", exoplanets.headValues(onlyExoplanetsTrain),
print("non-explonets from train and test", exoplanets.headValues(onlyNonExoplanets
```

```
explonets from train and test    LABEL   FLUX.1   FLUX.2   FLUX.3   FLUX.4   F
LUX.5   FLUX.6   FLUX.7  \
0       2    93.85    83.81    20.10   -26.98   -39.56  -124.71 -135.18
1       2   -38.88   -33.83   -58.54   -40.09   -79.31   -72.81  -86.55
2       2   532.64   535.92   513.73   496.92   456.45   466.00  464.50
3       2   326.52   347.39   302.35   298.13   317.74   312.70  322.33
4       2 -1107.21 -1112.59 -1118.95 -1095.10 -1057.55 -1034.48 -998.34

    FLUX.8   FLUX.9    ...      FLUX.3188  FLUX.3189  FLUX.3190  FLUX.3191  \
0   -96.27   -79.89    ...         -78.07    -102.15    -102.15      25.13
1   -85.33   -83.97    ...          -3.28     -32.21     -32.21     -24.89
2   486.39   436.56    ...         -71.69      13.31      13.31     -29.89
3   311.31   312.42    ...           5.71      -3.73      -3.73      30.05
4 -1022.71 -989.57    ...        -594.37    -401.66    -401.66    -357.24

   FLUX.3192  FLUX.3193  FLUX.3194  FLUX.3195  FLUX.3196  FLUX.3197
0      48.57      92.54      39.32      61.42       5.08     -39.54
1      -4.86       0.76     -11.70       6.46      16.00      19.93
2     -20.88       5.06     -11.80     -28.91     -70.02     -96.67
3      20.03     -12.67      -8.77     -17.31     -17.35      13.98
4    -443.76    -438.54    -399.71    -384.65    -411.79    -510.54

[5 rows x 3198 columns]    LABEL   FLUX.1   FLUX.2   FLUX.3   FLUX.4   FLUX.5
FLUX.6   FLUX.7  \
0       2   119.88   100.21    86.46    48.68    46.12    39.39    18.57
1       2  5736.59  5699.98  5717.16  5692.73  5663.83  5631.16  5626.39
2       2   844.48   817.49   770.07   675.01   605.52   499.45   440.77
3       2  -826.00  -827.31  -846.12  -836.03  -745.50  -784.69  -791.22
4       2   -39.57   -15.88    -9.16    -6.37   -16.13   -24.05    -0.90

    FLUX.8   FLUX.9    ...      FLUX.3188  FLUX.3189  FLUX.3190  FLUX.3191  \
0     6.98     6.63    ...          14.52      19.29      14.44      -1.62
1  5569.47  5550.44    ...        -581.91    -984.09   -1230.89   -1600.45
2   362.95   207.27    ...          17.82     -51.66     -48.29     -59.99
3  -746.50  -709.53    ...         122.34      93.03      93.03      68.81
4   -45.20    -5.04    ...         -37.87     -61.85     -27.15     -21.18

   FLUX.3192  FLUX.3193  FLUX.3194  FLUX.3195  FLUX.3196  FLUX.3197
0      13.33      45.50      31.93      35.78     269.43      57.72
1   -1824.53   -2061.17   -2265.98   -2366.19   -2294.86   -2034.72
2     -82.10    -174.54     -95.23    -162.68     -36.79      30.63
3       9.81      20.75      20.25    -120.81    -257.56    -215.41
4     -33.76     -85.34     -81.46     -61.98     -69.34     -17.84

[5 rows x 3198 columns]
non-explonets from train and test    LABEL  FLUX.1  FLUX.2  FLUX.3  FLUX.4  F
LUX.5  FLUX.6  FLUX.7  FLUX.8  \
37       1 -141.22  -81.79  -52.28  -32.45   -1.55  -35.61  -23.28   19.45
38       1  -35.62  -28.55  -27.29  -28.94  -15.13  -51.06    2.67   -5.21
39       1  142.40  137.03   93.65  105.64   98.22   99.06   86.40   60.78
40       1 -167.02 -137.65 -150.05 -136.85  -98.73 -103.14 -107.70 -123.19
41       1  207.74  223.60  246.15  224.06  210.77  189.56  172.68  170.31
```

```
    FLUX.9    ...      FLUX.3188  FLUX.3189  FLUX.3190  FLUX.3191  FLUX.3192
\
37   53.11    ...         -50.79     -22.34     -36.23      27.44      13.52
38    9.67    ...         -43.98     -38.22     -46.23     -54.40     -23.51
39   45.18    ...          -0.99      -3.03     -30.27     -24.22     -35.10
40 -125.65    ...         -97.43     -79.79     -80.62     -78.22    -105.06
41  148.79    ...         -53.06    -136.92    -174.97    -180.46    -164.01

    FLUX.3193  FLUX.3194  FLUX.3195  FLUX.3196  FLUX.3197
37      38.66     -17.53      31.49      31.38      50.03
38     -26.96      -3.95      -0.34      10.52      -7.69
39     -39.64      23.78      23.40      -0.50       0.97
40     -69.67     -90.45     -73.67     -66.71     -66.07
41    -126.58      84.05      63.81     108.36      78.10

[5 rows x 3198 columns]    LABEL  FLUX.1  FLUX.2  FLUX.3  FLUX.4  FLUX.5  FLU
X.6  FLUX.7  FLUX.8  \
5      1   14.28   10.63   14.56   12.42   12.07   12.92   12.27    3.19
6      1 -150.48 -141.72 -157.60 -184.60 -164.89 -173.87 -162.91 -167.04
7      1  -10.06  -12.78  -13.16   -9.81  -18.91  -20.33  -22.85  -19.17
8      1  454.66  440.60  382.29  361.63  298.63  253.29  155.86  110.38
9      1  187.40  209.60  199.91  179.62  171.21  161.84  163.02  171.61

   FLUX.9    ...      FLUX.3188  FLUX.3189  FLUX.3190  FLUX.3191  FLUX.3192  \
5    8.47    ...           3.86      -4.06      -3.56      -1.13      -7.18
6 -172.76    ...           7.15       5.16      -9.08     -39.11     -32.31
7  -17.97    ...          21.49      30.63      24.19      33.00      35.70
8   31.71    ...         -56.78     -61.64    -120.32     -65.39    -126.75
9  113.53    ...         -23.75     -35.72     -21.93     -16.47     -21.84

   FLUX.3193  FLUX.3194  FLUX.3195  FLUX.3196  FLUX.3197
5      -4.78      -4.34       7.67      -0.33      -7.53
6      -8.40     -16.80      -8.03     -12.73     -11.41
7      35.89     -33.44     -30.83     -33.00     -20.15
8     -78.18    -184.39    -142.31    -113.12    -111.78
9     -26.64     -13.90      17.03       4.36       2.91

[5 rows x 3198 columns]
```

In [0]:

```
print(exoplanets.sampleRandomValues(onlyExoplanetsTrain), exoplanets.sampleRandomV
print(exoplanets.sampleRandomValues(onlyNonExoplanetsTrain), exoplanets.sampleRand
```

```
    LABEL   FLUX.1   FLUX.2   FLUX.3   FLUX.4   FLUX.5   FLUX.6   FLUX.7  \
3       2   326.52   347.39   302.35   298.13   317.74   312.70   322.33
27      2   124.39    72.73    36.85    -4.68     6.96   -44.61   -89.79
17      2   -65.20   -76.33   -76.23   -72.58   -69.62   -74.51   -69.48
31      2   194.82   162.51   126.17   129.70    82.27    60.71    58.71
21      2  2053.62  2126.05  2146.33  2159.84  2237.59  2236.12  2244.47

     FLUX.8   FLUX.9    ...     FLUX.3188  FLUX.3189  FLUX.3190  FLUX.3191  \
3    311.31   312.42    ...          5.71      -3.73      -3.73      30.05
27  -121.71  -120.59    ...        -14.38     -21.65      -6.04      -7.15
17   -61.06   -49.29    ...         18.66     -11.72     -11.72       4.56
31    23.36    32.57    ...         29.21      47.66       0.48     -28.59
21  2279.61  2288.22    ...       1832.59    1935.53    1965.84    2094.19

    FLUX.3192  FLUX.3193  FLUX.3194  FLUX.3195  FLUX.3196  FLUX.3197
3       20.03     -12.67      -8.77     -17.31     -17.35      13.98
27      67.58      56.43      -1.95       7.09       1.63     -10.77
17      11.47      31.26      21.71      13.42      13.24       9.21
31     -33.15     -14.98      -1.56      22.25      21.55       3.49
21    2212.52    2292.64    2454.48    2568.16    2625.45    2578.80

[5 rows x 3198 columns]    LABEL   FLUX.1   FLUX.2   FLUX.3   FLUX.4   FLUX.5
FLUX.6   FLUX.7  \
3      2  -826.00  -827.31  -846.12  -836.03  -745.50  -784.69  -791.22
4      2   -39.57   -15.88    -9.16    -6.37   -16.13   -24.05    -0.90
0      2   119.88   100.21    86.46    48.68    46.12    39.39    18.57
1      2  5736.59  5699.98  5717.16  5692.73  5663.83  5631.16  5626.39
2      2   844.48   817.49   770.07   675.01   605.52   499.45   440.77

    FLUX.8   FLUX.9    ...     FLUX.3188  FLUX.3189  FLUX.3190  FLUX.3191  \
3  -746.50  -709.53    ...        122.34      93.03      93.03      68.81
4   -45.20    -5.04    ...        -37.87     -61.85     -27.15     -21.18
0     6.98     6.63    ...         14.52      19.29      14.44      -1.62
1  5569.47  5550.44    ...       -581.91    -984.09   -1230.89   -1600.45
2   362.95   207.27    ...         17.82     -51.66     -48.29     -59.99

   FLUX.3192  FLUX.3193  FLUX.3194  FLUX.3195  FLUX.3196  FLUX.3197
3       9.81      20.75      20.25    -120.81    -257.56    -215.41
4     -33.76     -85.34     -81.46     -61.98     -69.34     -17.84
0      13.33      45.50      31.93      35.78     269.43      57.72
1   -1824.53   -2061.17   -2265.98   -2366.19   -2294.86   -2034.72
2     -82.10    -174.54     -95.23    -162.68     -36.79      30.63

[5 rows x 3198 columns]
      LABEL  FLUX.1  FLUX.2  FLUX.3  FLUX.4  FLUX.5  FLUX.6  FLUX.7  FLUX.8  \
3509      1  -10.94   -8.73  -12.65   -3.44   -7.31   -2.60   -5.41  -10.03
323       1   -3.81  -11.60    6.06   -2.24   -6.93  201.22   -4.87  -14.13
2037      1  -21.66   23.51   46.90   69.92   67.98  102.20  104.47  116.97
3091      1    4.34   -4.01    3.88    2.96   -1.25    0.40   -2.32   -0.84
4226      1  -37.70  -35.98  -36.41  -31.02  -29.73  -23.17  -23.25  -25.98

      FLUX.9    ...     FLUX.3188  FLUX.3189  FLUX.3190  FLUX.3191  \
3509   -3.12    ...         -1.10      -1.60       5.23       1.81
323    -0.24    ...          6.35      10.85       5.78      11.11
```

```
2037  123.00    ...         141.18     127.62      92.59      65.54
3091   -8.19    ...          -3.25      -3.25      -3.47      -1.50
4226  -10.66    ...         -12.27     -14.94       1.28      -3.48

      FLUX.3192  FLUX.3193  FLUX.3194  FLUX.3195  FLUX.3196  FLUX.3197
3509       7.79       1.84      -2.33      -4.15      -2.92      -4.06
323        9.06      15.60       2.39       8.94       4.74      14.04
2037      -4.10      -8.13     -54.07     -67.94    -122.22    -133.21
3091      -0.09      -4.75       8.33      -2.13      -0.55      -4.86
4226     -15.23     -14.06     -15.71      -8.08      -7.65     -11.63

[5 rows x 3198 columns]      LABEL    FLUX.1    FLUX.2    FLUX.3    FLUX.4    FLUX.
5    FLUX.6    FLUX.7  \
518      1     82.37     67.15     86.93     16.01     36.42     45.19     22.23
161      1   3470.28   3183.19   2978.44   2452.22   2499.91   2015.81   1398.53
35       1    269.83    265.80    198.28    207.41    119.13     42.13    -53.14
313      1    101.48     84.16     83.06     68.70     37.34     28.11     82.01
515      1     -8.75     -3.38    -13.36     -8.79     -9.30    -14.66    -15.37

      FLUX.8   FLUX.9    ...      FLUX.3188  FLUX.3189  FLUX.3190  FLUX.3191  \
518    30.78    12.42    ...          -9.17       8.30       2.82     -10.53
161  1164.13   712.00    ...         936.78    1670.34    2526.31    3500.60
35    -77.26  -139.12    ...         168.88     140.80      94.60     104.45
313    12.17     0.04    ...          51.21      46.36      54.86      31.65
515   -10.16   -17.68    ...          -5.83      -4.11      -4.10      -1.77

     FLUX.3192  FLUX.3193  FLUX.3194  FLUX.3195  FLUX.3196  FLUX.3197
518     -14.53      -7.39      18.03      28.53      22.92       6.00
161    4646.81    4646.81    -240.62    -671.66    -287.00    -507.37
35       59.14      18.75     102.55     -26.17      59.86      91.81
313      18.67      24.32       1.32      27.75      14.98      50.32
515      -5.07       2.19      -0.94      -3.56      -0.77      -1.53

[5 rows x 3198 columns]
```

2. EDA and different Visualizations

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

In [0]:
```python
exoTrain.LABEL.value_counts(normalize=True)
```

Out[16]:
```
1    0.992727
2    0.007273
Name: LABEL, dtype: float64
```

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

In [0]:
```python
exoTest.LABEL.value_counts(normalize=True)
```

Out[17]:
```
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

In [0]:
```
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

In [0]:
```
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
```

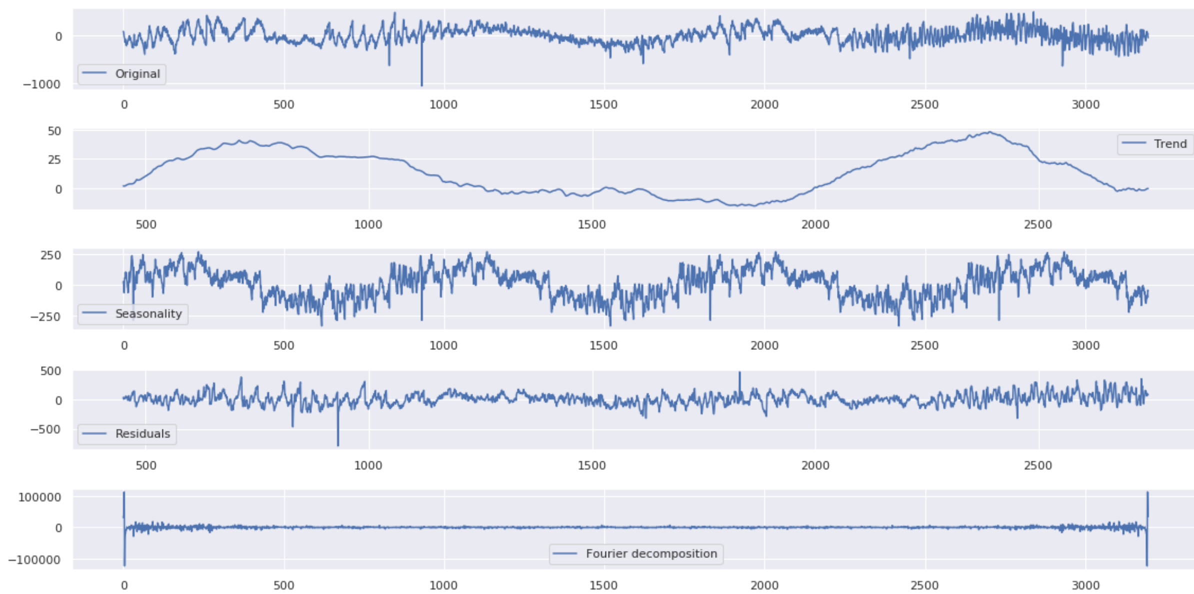
The above graph provides insights on

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 meterages
3. Standard deviation light intensity mostly above zero value, shows that +/- square root values on each instances

2.5 Planets Decomposed Season Plot

In [0]:
```
decomposition = exoplanets.seasonal_decompose_fft(X_train[series_number], freq=900
exoplanets.plot_decomposed_seasonal(decomposition)
```

```
/conda/envs/rapids/lib/python3.6/site-packages/numpy/core/numeric.py:538: Comp
lexWarning: Casting complex values to real discards the imaginary part
  return array(a, dtype, copy=False, order=order)
```



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 decomposed seasonal patterns
3. Residual graph shows the inverse instance values of original graph and smoothened seasonal decomposed graph
4. Fourier decomposed graph shows at the beginning and at the end, distorted frequency wavelengths are available;

In [0]:
```
X_dec = [exoplanets.seasonal_decompose_fft(X_train[i], freq=900) for i in range(0,
X_test_dec = [exoplanets.seasonal_decompose_fft(X_test[i], freq=900) for i in rang
```

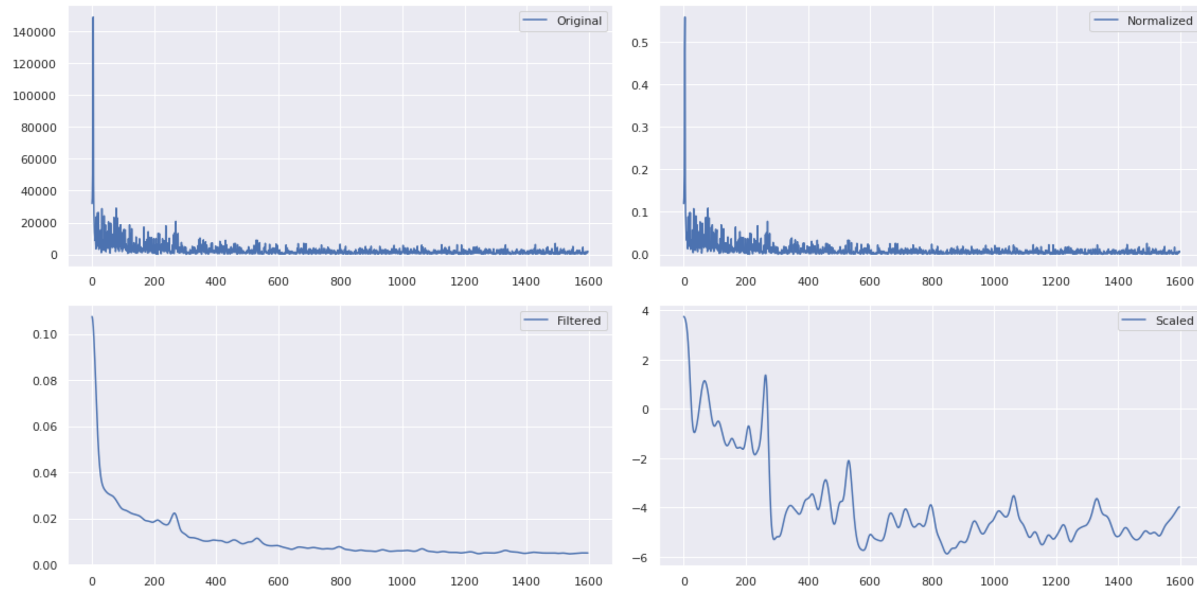
In [0]:
```
X_fft = absolute = [np.abs(X.fft[:(len(X.fft)//2)]) for X in X_dec]
X_test_fft = [np.abs(X.fft[:(len(X.fft)//2)]) for X in X_test_dec]
```

In [0]:
```
X_fft = normalized = normalize(X_fft)
X_test_fft = normalize(X_test_fft)
```

In [0]:
```
X_fft = filtered = ndimage.filters.gaussian_filter(X_fft, sigma=10)
X_test_fft = ndimage.filters.gaussian_filter(X_test_fft, sigma=10)
```

In [0]:
```
std_scaler = StandardScaler()
X_fft = scaled = std_scaler.fit_transform(X_fft)
X_test_fft = std_scaler.fit_transform(X_test_fft)
```

In [0]:
```
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 decaying from high to low after reaching 200 plus points it stays in the same level (horizontal view);

In [0]:
```
X_train = exoTrain.drop('LABEL', axis=1)
Y_train = exoTrain.LABEL
x_test = exoTest.drop('LABEL', axis=1)
```

In [0]:
```
print(X_train.shape, Y_train.shape, x_test.shape)
```

```
(5087, 3197) (5087,) (570, 3197)
```

In [0]:
```

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

In [0]:
```
X_test = x_test.apply(exoplanets.normal, axis=1)
```

In [0]:
```
import scipy
X_train_fft_1 = X_train.apply(exoplanets.fast_fourier_transf, axis=1)
```

In [0]:
```
X_test_fft_1 = X_test.apply(exoplanets.fast_fourier_transf,axis=1)
```

In [0]:
```
X_train_stft = X_train.apply(exoplanets.shorttime_fourier_transf, axis=1)
```

In [0]:
```
X_test_stft = X_test.apply(exoplanets.shorttime_fourier_transf,axis=1)
```

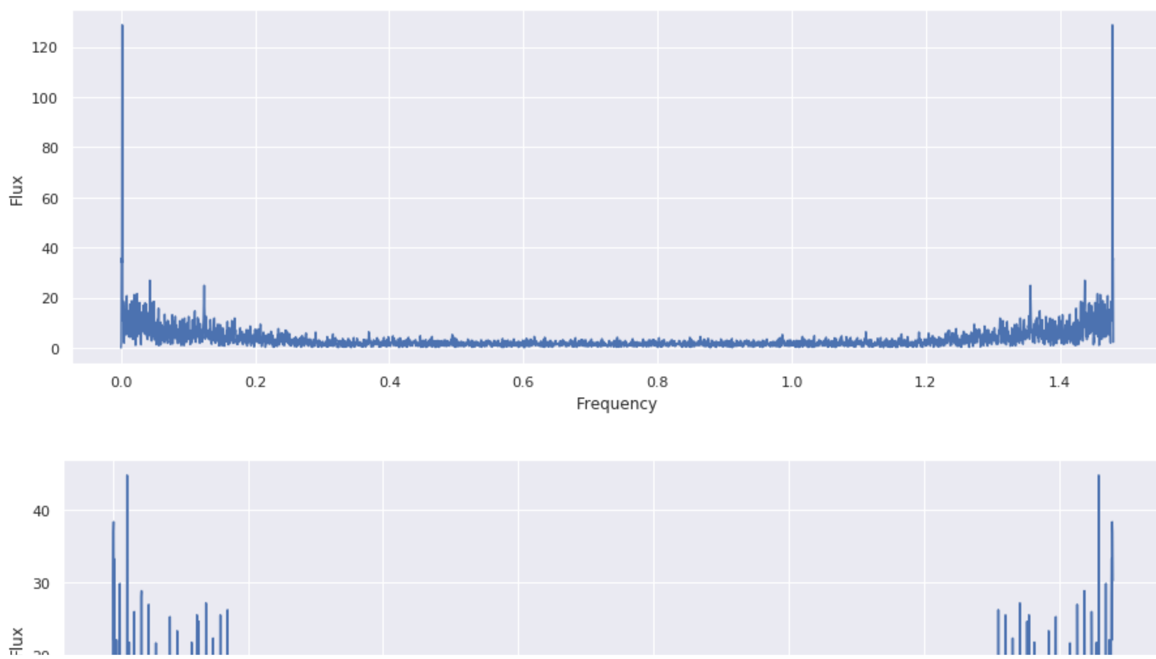
In [0]:
```
X_train_stft.shape
```

Out[35]:
```
(5087,)
```

In [0]:
```
X_train_stft.head()
```

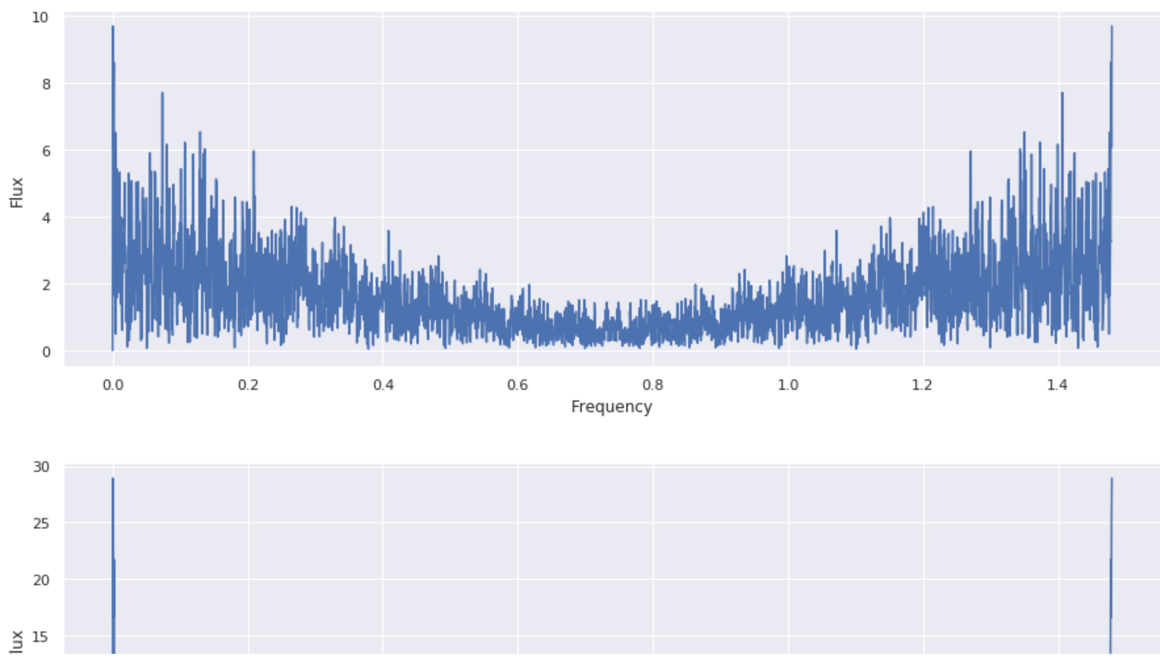
Out[36]:
```
0    [[0.0, 0.00390625, 0.0078125, 0.01171875, 0.01...
1    [[0.0, 0.00390625, 0.0078125, 0.01171875, 0.01...
2    [[0.0, 0.00390625, 0.0078125, 0.01171875, 0.01...
3    [[0.0, 0.00390625, 0.0078125, 0.01171875, 0.01...
4    [[0.0, 0.00390625, 0.0078125, 0.01171875, 0.01...
dtype: object
```

In [0]:
```
exoplanets.fluxFreq_ExoplanetTestPlot(X_train_fft_1)
```



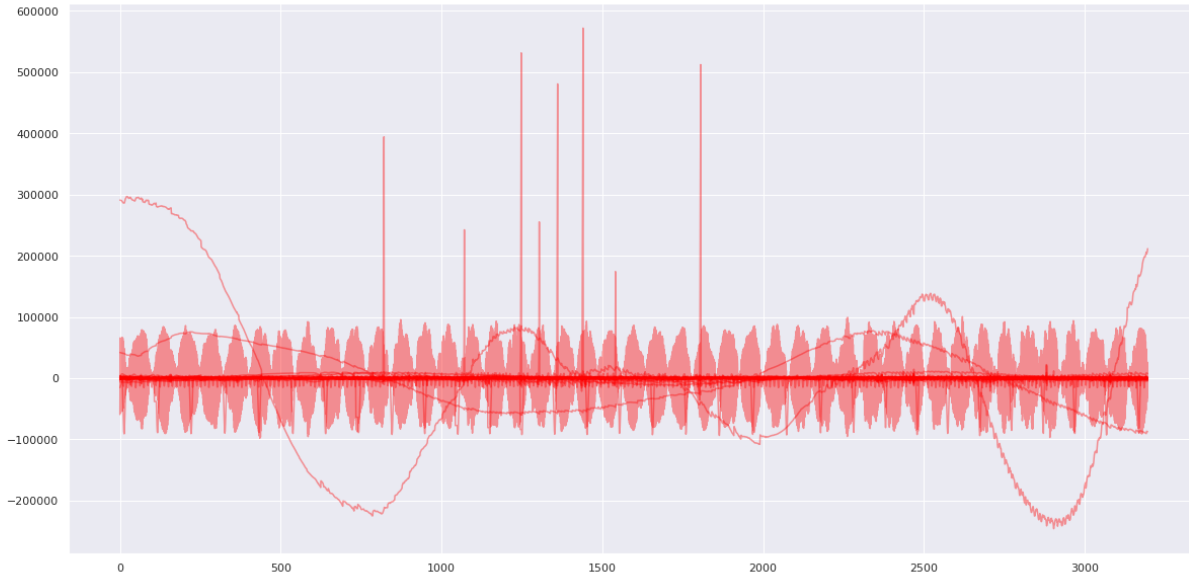
This above graph show Flux frequency range on exoplanets

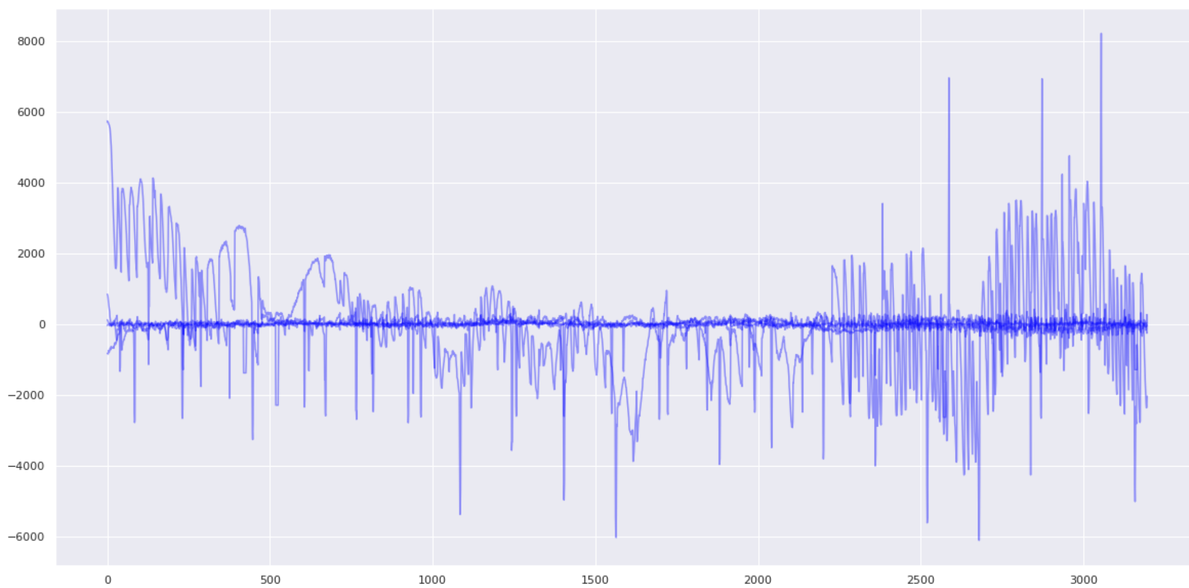
In [0]:
```
exoplanets.fluxFreq_NonExoplanets(X_train_fft_1)
```



This above graph show Flux frequency range on non-exoplanets

```
In [0]: exoplanets.ploCompleteTestDataGraph(exoTest)
```

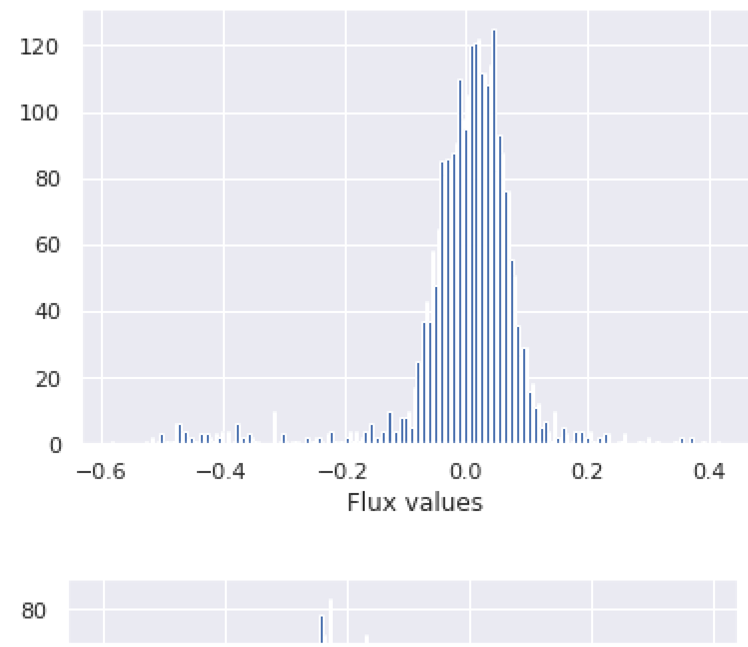




The above graph are test data values,

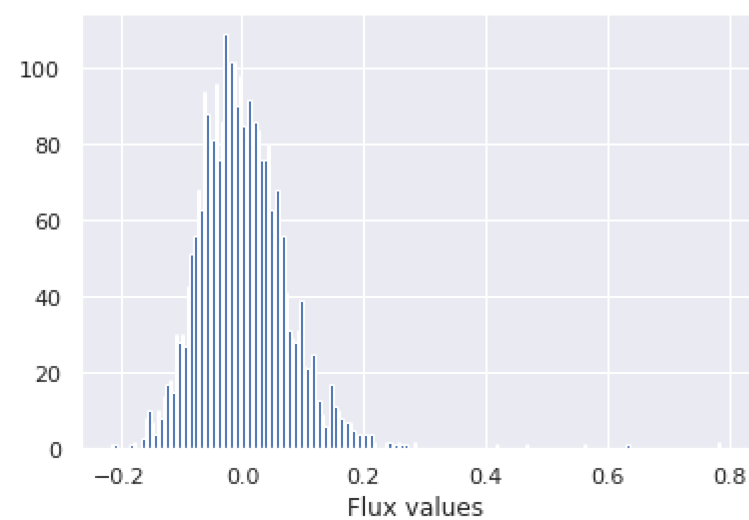
- Blue graph shows light intensity value of exoplanets, which has less data and hence frequency distribution density is low;
- Red graph shows light intensity value of non-exoplanets, which has the slightly huge data and hence the frequency distribution density is high

```python
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

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



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

In [0]:
```
exoplanets.linearML_SGDClassifier(X_fft,y_train,y_test, X_test_fft)
```

```
10100
Train accuracy = 0.9999
Test accuracy = 0.9947
Confusion Matrix (train sample):
 [[5049    1]
 [   0 5050]]
Confusion Matrix (test sample):
 [[562   3]
 [  0   5]]


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       1.00      0.99      1.00       565
           2       0.62      1.00      0.77         5

    accuracy                           0.99       570
   macro avg       0.81      1.00      0.88       570
weighted avg       1.00      0.99      1.00       570
```

Accuracy score from Linear model with SGD Classifier is 99%

3.2 Decision tree

In [0]:
```
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_s
```

In [0]:
```
dtc1 = exoplanets.decisionTreeML(X_train,y_train,X_test,y_test, max_depth_val=5, m
```

```
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:1437: UndefinedMetricWarning: Precision and F-score are ill-defined and bei
ng set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
/conda/envs/rapids/lib/python3.6/site-packages/sklearn/metrics/classification.
py:1437: UndefinedMetricWarning: Precision and F-score are ill-defined and bei
ng set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
/conda/envs/rapids/lib/python3.6/site-packages/sklearn/metrics/classification.
py:1437: UndefinedMetricWarning: Precision and F-score are ill-defined and bei
ng set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
```

Accuracy score from Decision Tree model is 99%

3.3 Random Forest

In [0]:
```
random_forest = RandomForestClassifier()
```

In [0]:
```
random_forest.fit(X_train, y_train)
```

```
/conda/envs/rapids/lib/python3.6/site-packages/sklearn/ensemble/forest.py:245:
FutureWarning: The default value of n_estimators will change from 10 in versio
n 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

Out[46]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=None, max_features='auto', max_leaf_nodes=Non
e,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=10,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warn_start=False)
```

In [0]:
```
y_predict = random_forest.predict(X_test)
accuracy_score(y_test, y_predict)
```

Out[47]: 0.991486574983628

Accuracy score from Random Forest model is 99%

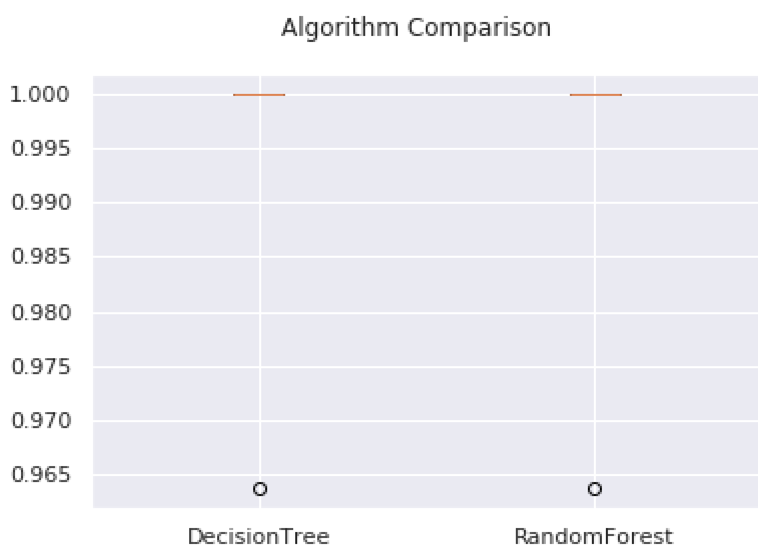
In [0]:
```
pd.DataFrame(
    confusion_matrix(y_test, y_predict),
    columns=['Predicted NonExoplanet', 'Predicted Exoplanet'],
    index=['True NonExoplanet', 'True Exoplanet']
)
```

Out[48]:

	Predicted NonExoplanet	Predicted Exoplanet
True NonExoplanet	1514	0
True Exoplanet	13	0

In [0]:
```
exoplanets.evaluateModelsByBoxPlot(dtc1,random_forest,model_selection)
```

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



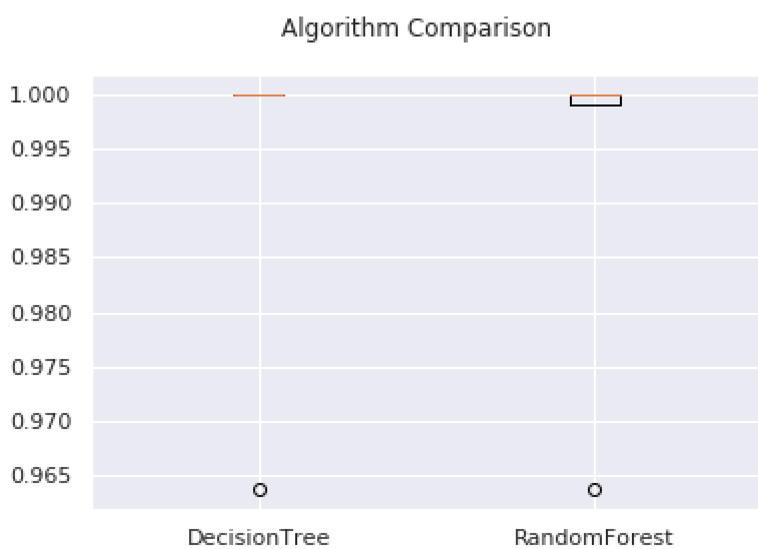
Accuracy score comparision between DecisionTree and Random Forest model are 99.27%

In [0]:
```
dtc2 = exoplanets.decisionTreeML(X_train,y_train,X_test,y_test, max_depth_val=5, m
```

```
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:1437: UndefinedMetricWarning: Precision and F-score are ill-defined and bei
ng set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
/conda/envs/rapids/lib/python3.6/site-packages/sklearn/metrics/classification.
py:1437: UndefinedMetricWarning: Precision and F-score are ill-defined and bei
ng set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
/conda/envs/rapids/lib/python3.6/site-packages/sklearn/metrics/classification.
py:1437: UndefinedMetricWarning: Precision and F-score are ill-defined and bei
ng set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
```

In [0]:
```
exoplanets.evaluateModelsByBoxPlot(dtc2,random_forest,model_selection)
```

```
DecisionTree: 0.992731 (0.014538)
RandomForest: 0.992534 (0.014445)
```

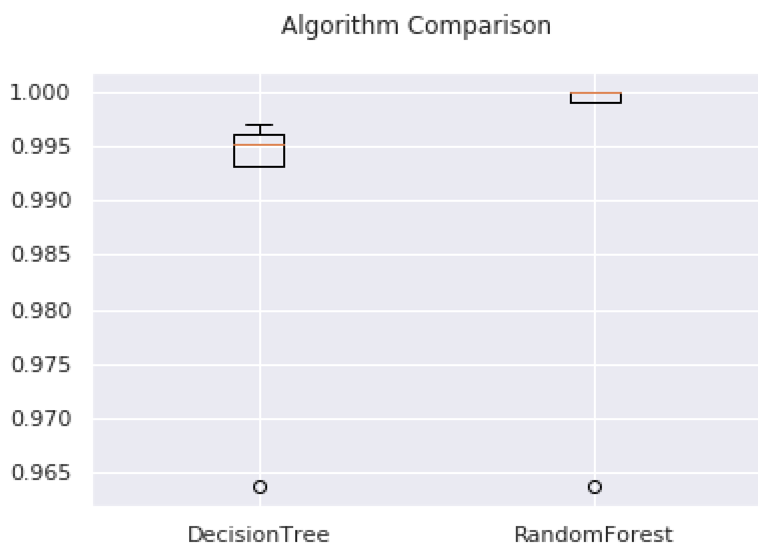


In [0]:
```
dtc3 = exoplanets.decisionTreeML(X_train,y_train,X_test,y_test, max_depth_val=10,
```

```
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
```

In [0]:
```
exoplanets.evaluateModelsByBoxPlot(dtc3,random_forest,model_selection)
```

```
DecisionTree: 0.988995 (0.012737)
RandomForest: 0.992534 (0.014445)
```



3.4 Naive Bayes ML Model

In [0]:
```
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=
```

In [0]:
```
model_gnb1 = exoplanets.gaussianNaiveBayesML(train_set, test_set, train_labels, te
```

```
model score ::  1.0
              precision    recall  f1-score   support

           1       0.98      1.00      0.99       168
           2       0.00      0.00      0.00         3

    accuracy                           0.98       171
   macro avg       0.49      0.50      0.50       171
weighted avg       0.97      0.98      0.97       171

[[168   0]
 [  3   0]]

/conda/envs/rapids/lib/python3.6/site-packages/sklearn/metrics/classification.
py:1437: UndefinedMetricWarning: Precision and F-score are ill-defined and bei
ng set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
/conda/envs/rapids/lib/python3.6/site-packages/sklearn/metrics/classification.
py:1437: UndefinedMetricWarning: Precision and F-score are ill-defined and bei
ng set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
/conda/envs/rapids/lib/python3.6/site-packages/sklearn/metrics/classification.
py:1437: UndefinedMetricWarning: Precision and F-score are ill-defined and bei
ng set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
/conda/envs/rapids/lib/python3.6/site-packages/sklearn/model_selection/_split.
py:657: Warning: The least populated class in y has only 2 members, which is t
oo few. The minimum number of members in any class cannot be less than n_split
s=10.
  % (min_groups, self.n_splits)), Warning)

Cross-validated scores: [0.97560976 0.97560976 1.         1.         1.
1.
 1.         1.         1.         1.        ] [0.97560976 0.97560976 1.
1.         1.         1.
 1.         1.         1.         1.        ]
Average score: 0.9951219512195122
```

```
In [0]: model_gnb2 = exoplanets.gaussianNaiveBayesML(train_set, test_set, train_labels, te
```

```
model score ::  1.0
              precision    recall  f1-score   support

           1       0.98      1.00      0.99       168
           2       0.00      0.00      0.00         3

    accuracy                           0.98       171
   macro avg       0.49      0.50      0.50       171
weighted avg       0.97      0.98      0.97       171

[[168   0]
 [  3   0]]

/conda/envs/rapids/lib/python3.6/site-packages/sklearn/metrics/classification.
py:1437: UndefinedMetricWarning: Precision and F-score are ill-defined and bei
ng set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
/conda/envs/rapids/lib/python3.6/site-packages/sklearn/metrics/classification.
py:1437: UndefinedMetricWarning: Precision and F-score are ill-defined and bei
ng set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
/conda/envs/rapids/lib/python3.6/site-packages/sklearn/metrics/classification.
py:1437: UndefinedMetricWarning: Precision and F-score are ill-defined and bei
ng set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
/conda/envs/rapids/lib/python3.6/site-packages/sklearn/model_selection/_split.
py:657: Warning: The least populated class in y has only 2 members, which is t
oo few. The minimum number of members in any class cannot be less than n_split
s=15.
  % (min_groups, self.n_splits)), Warning)


Cross-validated scores: [0.96428571 0.96428571 1.         1.         1.
1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.        ] [0.96428571 0.96428571 1.         1.
1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.        ]
Average score: 0.9952380952380953
```

Accuracy score from Naive Bayes model is 99.52%

```
In [0]: model_gnb3 = exoplanets.gaussianNaiveBayesML(train_set, test_set, train_labels, te
```

```
model score ::  1.0
              precision    recall  f1-score   support

           1       0.98      1.00      0.99       168
           2       0.00      0.00      0.00         3

    accuracy                           0.98       171
   macro avg       0.49      0.50      0.50       171
weighted avg       0.97      0.98      0.97       171

[[168   0]
 [  3   0]]


/conda/envs/rapids/lib/python3.6/site-packages/sklearn/metrics/classification.
py:1437: UndefinedMetricWarning: Precision and F-score are ill-defined and bei
ng set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
/conda/envs/rapids/lib/python3.6/site-packages/sklearn/metrics/classification.
py:1437: UndefinedMetricWarning: Precision and F-score are ill-defined and bei
ng set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
/conda/envs/rapids/lib/python3.6/site-packages/sklearn/metrics/classification.
py:1437: UndefinedMetricWarning: Precision and F-score are ill-defined and bei
ng set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
/conda/envs/rapids/lib/python3.6/site-packages/sklearn/model_selection/_split.
py:657: Warning: The least populated class in y has only 2 members, which is t
oo few. The minimum number of members in any class cannot be less than n_split
s=12.
  % (min_groups, self.n_splits)), Warning)

Cross-validated scores: [0.97142857 0.97058824 1.         1.         1.
1.
 1.         1.         1.         1.         1.         1.        ] [0.9714285
7 0.97058824 1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.        ]
Average score: 0.9951680672268908
```

3.5 Boosting models

In [0]:
```
X_train=exoTrain.drop('LABEL',axis=1)
Y_train=exoTrain[['LABEL']]
X_test=exoTest.drop('LABEL',axis=1)
Y_test=exoTest[['LABEL']]
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-58-28ee115671a7> in <module>
----> 1 X_train=exoTrain.drop('LABEL',axis=1)
      2 Y_train=exoTrain[['LABEL']]
      3 X_test=exoTest.drop('LABEL',axis=1)
      4 Y_test=exoTest[['LABEL']]

/conda/envs/rapids/lib/python3.6/site-packages/pandas/core/frame.py in drop(se
lf, labels, axis, index, columns, level, inplace, errors)
   3695                                            index=index, columns=column
s,
   3696                                            level=level, inplace=inplac
e,
-> 3697                                            errors=errors)
   3698 
   3699     @rewrite_axis_style_signature('mapper', [('copy', True),

/conda/envs/rapids/lib/python3.6/site-packages/pandas/core/generic.py in drop
(self, labels, axis, index, columns, level, inplace, errors)
   3109         for axis, labels in axes.items():
   3110             if labels is not None:
-> 3111                 obj = obj._drop_axis(labels, axis, level=level, errors
=errors)
   3112 
   3113         if inplace:

/conda/envs/rapids/lib/python3.6/site-packages/pandas/core/generic.py in _drop
_axis(self, labels, axis, level, errors)
   3141                 new_axis = axis.drop(labels, level=level, errors=error
s)
   3142             else:
-> 3143                 new_axis = axis.drop(labels, errors=errors)
   3144             result = self.reindex(**{axis_name: new_axis})
   3145 

/conda/envs/rapids/lib/python3.6/site-packages/pandas/core/indexes/base.py in
drop(self, labels, errors)
   4402             if errors != 'ignore':
   4403                 raise KeyError(
-> 4404                     '{} not found in axis'.format(labels[mask]))
   4405         indexer = indexer[~mask]
   4406         return self.delete(indexer)

KeyError: "['LABEL'] not found in axis"
```

In [0]:
```
adaModel = exoplanets.adaBoostclassifierML(dtc1,X_train,Y_train,X_test,Y_test, 51
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-60-e4ff432c8691> in <module>
----> 1 adaModel = exoplanets.adaBoostclassifierML(dtc1,X_train,Y_train,X_test
,Y_test, 51 )

NameError: name 'Y_test' is not defined
```

In [0]:
```
  xgBoostModel = exoplanets.xgBoostClassifier(X_train,Y_train,X_test,Y_test)
```

3.6 PCA (Principle Component Analysis)

In [0]:
```
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 [[1.00028098 0.99712261 0.99207029 ... 0.33409043 0.34124266 0.34739567]
 [0.99712261 1.00028098 0.99633951 ... 0.33970061 0.3503073  0.35881251]
 [0.99207029 0.99633951 1.00028098 ... 0.33932738 0.34128499 0.34625162]
 ...
 [0.33409043 0.33970061 0.33932738 ... 1.00028098 0.98422575 0.93095404]
 [0.34124266 0.3503073  0.34128499 ... 0.98422575 1.00028098 0.97870329]
 [0.34739567 0.35881251 0.34625162 ... 0.93095404 0.97870329 1.00028098]]
```

In [0]:
```
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 ... -7.57087559e-03
   3.89552406e-03 -1.71817780e-03]
 [ 2.09636223e-02 -2.46964372e-02 -6.56299679e-03 ... -2.84869233e-02
   2.80397120e-03  9.89338619e-03]
 [ 2.10079508e-02 -2.49033656e-02 -5.13363277e-03 ...  1.89219867e-03
   3.09559145e-03 -2.77519834e-02]
 ...
 [ 1.93854707e-03 -9.46307602e-05 -2.07973701e-02 ... -4.37649105e-03
   6.78764744e-04 -1.89506872e-02]
 [ 2.07232905e-03  5.37875713e-04 -1.84585090e-02 ...  1.72342376e-02
   1.03437646e-02 -2.04288571e-02]
 [ 2.90384726e-03  8.16460510e-04 -1.77480552e-02 ... -8.82038871e-05
   1.31250501e-02  6.57317102e-03]]

 Eigen Values
%s [8.75304099e+02 7.20624569e+02 4.77378032e+02 ... 9.26352126e-08
 9.15448008e-08 9.19194187e-08]
```

In [0]:
```
# 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(

# Sort the (eigenvalue, eigenvector) pairs from highest to lowest with respect to
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(eigenva
train_eigvectors_sorted = [train_eig_pairs[index][1] for index in range(len(eigenv

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

```
[(875.30409897194, array([0.0210556 , 0.02096362, 0.02100795, ..., 0.0019385
5, 0.00207233,
       0.00290385])), (720.6245694203241, array([-2.47900939e-02, -2.4696437
2e-02, -2.49033656e-02, ...,
       -9.46307602e-05,  5.37875713e-04,  8.16460510e-04])), (477.3780324510
3515, array([-0.00795282, -0.006563  , -0.00513363, ..., -0.02079737,
       -0.01845851, -0.01774806])), (313.4537965998446, array([0.01461582,
0.01555235, 0.01637114, ..., 0.03185703, 0.0343885 ,
       0.03509377])), (186.1011376585836, array([-0.0015188 , -0.00179225, -
0.00101699, ..., -0.02456015,
       -0.02915631, -0.03009064])), (130.2347950489151, array([0.01222345,
0.01657508, 0.01651252, ..., 0.03535319, 0.03559412,
       0.03171302])), (106.01254476788519, array([ 0.00351295,  0.00544481,
0.00102992, ..., -0.03575779,
       -0.02403619, -0.01402125])), (91.84439612425892, array([ 0.00469232,
0.00479449,  0.00260332, ..., -0.00315732,
        0.00849479,  0.0199473 ])), (66.99877178711522, array([ 0.01048009,
0.00796865,  0.0076927 , ...,  0.00318016,
       -0.001949  , -0.01162155])), (58.83077515222046, array([-0.00386042,
```

In [0]:
```
tot = sum(eigenvalues)
var_explained = [(i / tot) for i in sorted(train_eigvalues_sorted, reverse=True)]
# 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 w
# cumulative reaching almost 100%
```

In [0]:
```
#plt.bar(range(1,8), var_explained, alpha=0.5, align='center', label='individual e
#plt.step(range(1,8),cum_var_exp, where= 'mid', label='cumulative explained varian
#plt.ylabel('Explained variance ratio')
#plt.xlabel('Principal components')
#plt.legend(loc = 'best')
#plt.show()
```

PCA with FLUX 1

PCA with FLUX 1

In [0]:
```
train = exoTrain.rename(index=str, columns={"FLUX.1":"FLUX_1"})
```

In [0]:
```
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_
```

In [0]:
```
variance
```

Out[85]:
```
array([2.73649792e-01, 2.25291717e-01, 1.49244590e-01, ...,
       3.35925614e-12, 3.27092039e-12, 3.15447905e-12])
```

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

We are standardizing the data row wise

In [0]:

```
#INPUT_LIB = '/content/drive/My Drive/capstone_datasets/'
INPUT_LIB = './'
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, ax
x_test = ((x_test - np.mean(x_test, axis=1).reshape(-1,1)) / np.std(x_test, axis=1
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)
(5087, 3197, 2) (570, 3197, 2)
```

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, recall and precision.

In [0]:
```python
def batch_generator(x_train, y_train, batch_size=32):
    """
    Gives equal number of positive and negative samples, and rotates them randomly
    """
    half_batch = batch_size // 2
    x_batch = np.empty((batch_size, x_train.shape[1], x_train.shape[2]), dtype='fl
    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, dense layers with Relu for non-linearity in hidden layers and Sigmoid in the final dense layer, Adam optimizer with fixed learning rate using accuracy, f1, precision, recall as metrics functions.

In [0]:

```
model_one = Sequential()
model_one.add(Conv1D(filters=8, kernel_size=11, activation='relu', input_shape=x_t
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/tensorf
low/python/framework/op_def_library.py:263: colocate_with (from tensorflow.pyt
hon.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.

WARNING:From /conda/envs/rapids/lib/python3.6/site-packages/tensorflow/python/
framework/op_def_library.py:263: colocate_with (from tensorflow.python.framewo
rk.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.

WARNING:tensorflow:From /conda/envs/rapids/lib/python3.6/site-packages/keras/b
ackend/tensorflow_backend.py:3445: calling dropout (from tensorflow.python.op
s.nn_ops) with keep_prob is deprecated and will be removed in a future versio
n.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - ke
ep_prob`.

WARNING:From /conda/envs/rapids/lib/python3.6/site-packages/keras/backend/tens
orflow_backend.py:3445: calling dropout (from tensorflow.python.ops.nn_ops) wi
th keep_prob is deprecated and will be removed in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - ke
ep_prob`.
```

In [0]:

```python
#Start with a slightly lower learning rate, to ensure convergence
model_one.compile(optimizer=Adam(1e-5), loss = 'binary_crossentropy', metrics=['ac
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/tensorf
low/python/ops/math_ops.py:3066: to_int32 (from tensorflow.python.ops.math_op
s) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.

WARNING:From /conda/envs/rapids/lib/python3.6/site-packages/tensorflow/python/
ops/math_ops.py:3066: to_int32 (from tensorflow.python.ops.math_ops) is deprec
ated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.

Epoch 1/5
 - 17s - loss: 0.7432 - acc: 0.4924 - f1_m: 0.3844 - precision_m: 0.4854 - rec
all_m: 0.3270 - val_loss: 0.6253 - val_acc: 0.6965 - val_f1_m: 0.0112 - val_pr
ecision_m: 0.0112 - val_recall_m: 0.0112
Epoch 2/5
 - 1s - loss: 0.7315 - acc: 0.5082 - f1_m: 0.4190 - precision_m: 0.5081 - reca
ll_m: 0.3636 - val_loss: 0.6411 - val_acc: 0.6561 - val_f1_m: 0.0094 - val_pre
cision_m: 0.0080 - val_recall_m: 0.0112
Epoch 3/5
 - 1s - loss: 0.7326 - acc: 0.5060 - f1_m: 0.4331 - precision_m: 0.5077 - reca
ll_m: 0.3864 - val_loss: 0.6393 - val_acc: 0.6544 - val_f1_m: 0.0094 - val_pre
cision_m: 0.0080 - val_recall_m: 0.0112
Epoch 4/5
 - 1s - loss: 0.7159 - acc: 0.5294 - f1_m: 0.4651 - precision_m: 0.5440 - reca
ll_m: 0.4154 - val_loss: 0.6437 - val_acc: 0.6351 - val_f1_m: 0.0080 - val_pre
cision_m: 0.0062 - val_recall_m: 0.0112
Epoch 5/5
 - 1s - loss: 0.7109 - acc: 0.5246 - f1_m: 0.4708 - precision_m: 0.5333 - reca
ll_m: 0.4318 - val_loss: 0.6374 - val_acc: 0.6386 - val_f1_m: 0.0102 - val_pre
cision_m: 0.0094 - val_recall_m: 0.0112
```

In [0]:

```
#speed things up a little
model_one.compile(optimizer=Adam(4e-5), loss = 'binary_crossentropy', metrics=['ac
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)
```

```
Epoch 1/40
 - 2s - loss: 0.6985 - acc: 0.5477 - f1_m: 0.5096 - precision_m: 0.5564 - re
call_m: 0.4773 - val_loss: 0.6428 - val_acc: 0.6211 - val_f1_m: 0.0140 - val
_precision_m: 0.0102 - val_recall_m: 0.0225
Epoch 2/40
 - 1s - loss: 0.6922 - acc: 0.5634 - f1_m: 0.5347 - precision_m: 0.5722 - re
call_m: 0.5114 - val_loss: 0.6381 - val_acc: 0.6298 - val_f1_m: 0.0075 - val
_precision_m: 0.0056 - val_recall_m: 0.0112
Epoch 3/40
 - 1s - loss: 0.6873 - acc: 0.5811 - f1_m: 0.5501 - precision_m: 0.5956 - re
call_m: 0.5189 - val_loss: 0.6455 - val_acc: 0.6281 - val_f1_m: 0.0132 - val
_precision_m: 0.0094 - val_recall_m: 0.0225
Epoch 4/40
 - 1s - loss: 0.6552 - acc: 0.6102 - f1_m: 0.5861 - precision_m: 0.6219 - re
call_m: 0.5612 - val_loss: 0.6529 - val_acc: 0.6193 - val_f1_m: 0.0187 - val
_precision_m: 0.0130 - val_recall_m: 0.0337
Epoch 5/40
 - 1s - loss: 0.6542 - acc: 0.6181 - f1_m: 0.6041 - precision_m: 0.6267 - re
call_m: 0.5903 - val_loss: 0.6621 - val_acc: 0.6035 - val_f1_m: 0.0187 - val
```

In [0]:
```
model_one.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv1d_1 (Conv1D)            (None, 3187, 8)           184       
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 797, 8)            0         
_________________________________________________________________
batch_normalization_1 (Batch (None, 797, 8)            32        
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 787, 16)           1424      
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 197, 16)           0         
_________________________________________________________________
batch_normalization_2 (Batch (None, 197, 16)           64        
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 187, 32)           5664      
_________________________________________________________________
max_pooling1d_3 (MaxPooling1 (None, 47, 32)            0         
_________________________________________________________________
batch_normalization_3 (Batch (None, 47, 32)            128       
_________________________________________________________________
conv1d_4 (Conv1D)            (None, 37, 64)            22592     
_________________________________________________________________
max_pooling1d_4 (MaxPooling1 (None, 9, 64)             0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 576)               0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 576)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 64)                36928     
_________________________________________________________________
dropout_2 (Dropout)          (None, 64)                0         
_________________________________________________________________
dense_2 (Dense)              (None, 64)                4160      
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 65        
=================================================================
Total params: 71,241
Trainable params: 71,129
Non-trainable params: 112
_________________________________________________________________
```

In [0]:
```
!pip install pydot
```

```
Collecting pydot
  Downloading https://files.pythonhosted.org/packages/33/d1/b1479a770f66d962f545c2101630ce1d5592d90cb4f083d38862e93d16d2/pydot-1.4.1-py2.py3-none-any.whl (https://files.pythonhosted.org/packages/33/d1/b1479a770f66d962f545c2101630ce1d5592d90cb4f083d38862e93d16d2/pydot-1.4.1-py2.py3-none-any.whl)
Requirement already satisfied: pyparsing>=2.1.4 in /conda/envs/rapids/lib/python3.6/site-packages (from pydot) (2.4.0)
Installing collected packages: pydot
Successfully installed pydot-1.4.1
```

In [0]:

```python
import pydot
plot_model(model_one, to_file='model.png', show_shapes=True, show_layer_names=True
```

```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)
<ipython-input-70-9ff3f91b12b3> in <module>
      1 import pydot
----> 2 plot_model(model_one, to_file='model.png', show_shapes=True, show_laye
r_names=True)
      3 

/conda/envs/rapids/lib/python3.6/site-packages/keras/utils/vis_utils.py in plo
t_model(model, to_file, show_shapes, show_layer_names, rankdir)
    130             'LR' creates a horizontal plot.
    131     """
--> 132     dot = model_to_dot(model, show_shapes, show_layer_names, rankdir)
    133     _, extension = os.path.splitext(to_file)
    134     if not extension:

/conda/envs/rapids/lib/python3.6/site-packages/keras/utils/vis_utils.py in mod
el_to_dot(model, show_shapes, show_layer_names, rankdir)
     53     from ..models import Sequential
     54 
---> 55     _check_pydot()
     56     dot = pydot.Dot()
     57     dot.set('rankdir', rankdir)

/conda/envs/rapids/lib/python3.6/site-packages/keras/utils/vis_utils.py in _ch
eck_pydot()
     18     if pydot is None:
     19         raise ImportError(
---> 20             'Failed to import `pydot`. '
     21             'Please install `pydot`. '
     22             'For example with `pip install pydot`.')

ImportError: Failed to import `pydot`. Please install `pydot`. For example wit
h `pip install pydot`.
```

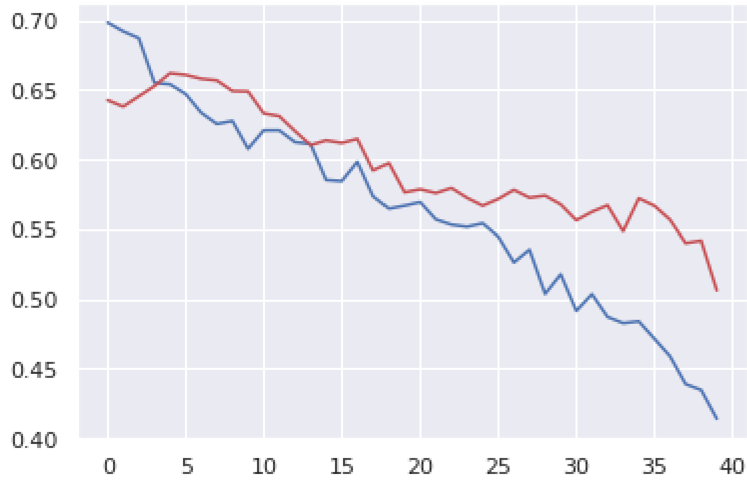
In [0]:

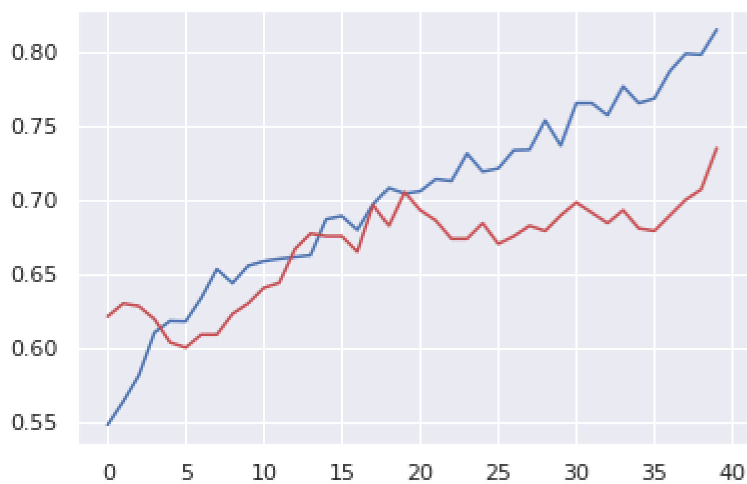
```python
print(model_one.evaluate(x_train, y_train, batch_size = 100))
print('\nModel Performance: Loss and Accuracy on validation data')
print(model_one.evaluate(x_test, y_test, batch_size = 100))
```

```
5087/5087 [==============================] - 1s 154us/step
[0.48375132783635183, 0.767643008352568, 0.016633650397467007, 0.0158222539030
7346, 0.017532767775205443]

Model Performance: Loss and Accuracy on validation data
570/570 [==============================] - 0s 269us/step
[0.5063526070954507, 0.7350877178342718, 0.05012531029550653, 0.02923976695328
4282, 0.17543859649122806]
```

In [0]:

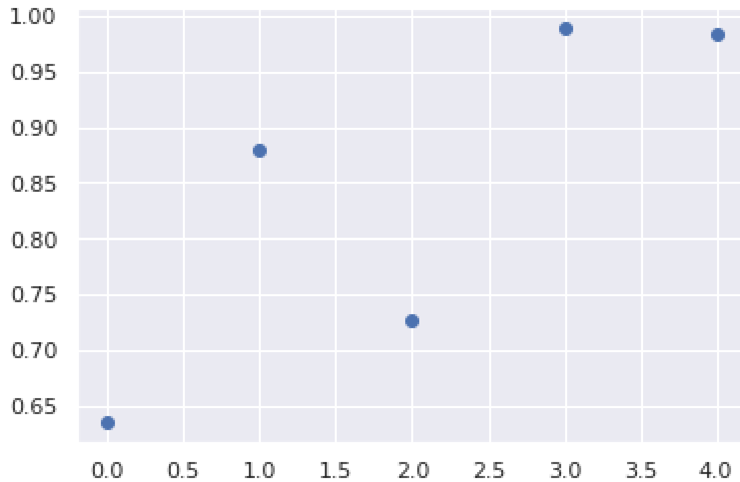
```
plt.plot(hist.history['loss'], color='b')
plt.plot(hist.history['val_loss'], color='r')
plt.show()
plt.plot(hist.history['acc'], color='b')
plt.plot(hist.history['val_acc'], color='r')
plt.show()
```

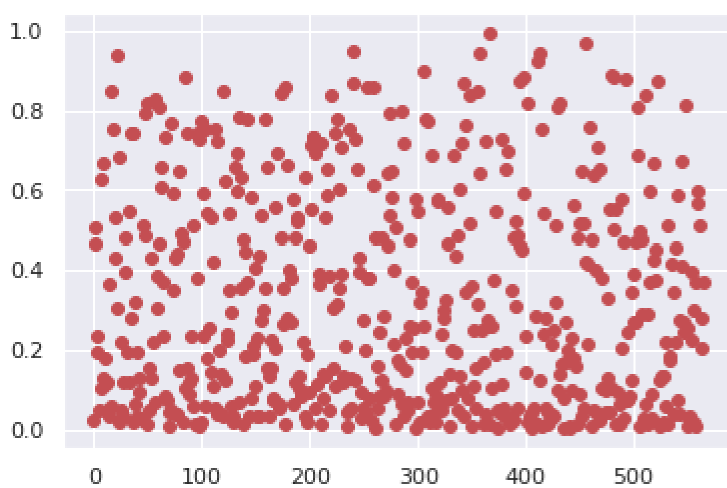




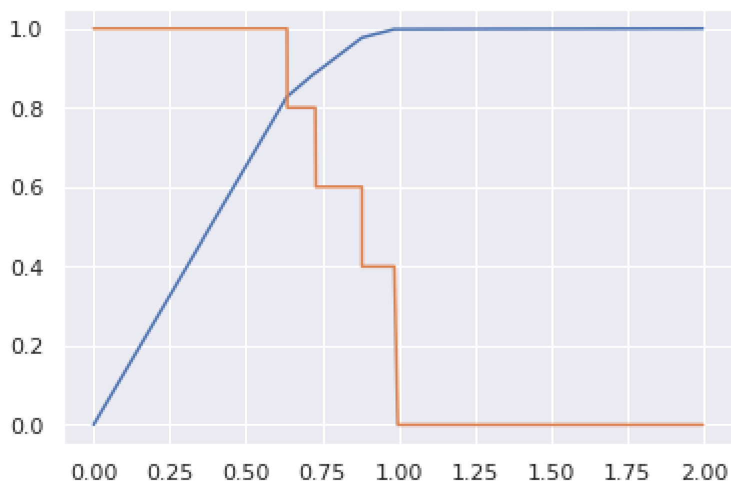
In [0]:

```
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()
```

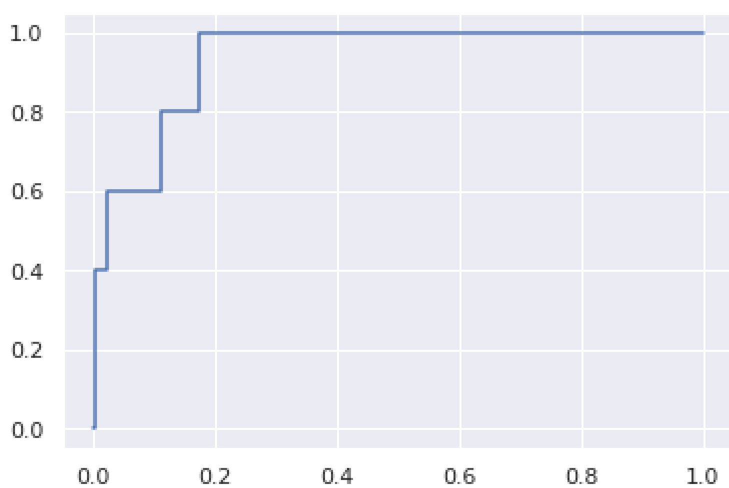




In [0]:
```
y_true = (y_test[:, 0] + 0.5).astype("int")
fpr, tpr, thresholds = roc_curve(y_true, y_hat)
plt.plot(thresholds, 1.-fpr)
plt.plot(thresholds, tpr)
plt.show()
crossover_index = np.min(np.where(1.-fpr <= tpr))
crossover_cutoff = thresholds[crossover_index]
crossover_specificity = 1.-fpr[crossover_index]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




```
Crossover at 0.63 with specificity 0.83
```




```
ROC area under curve is 0.94
```

In [0]:
```
#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 confusion matrix:\n',confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.9824561403508771
F1 score: 0.2857142857142857
Recall: 0.4
Precision: 0.2222222222222222
Cohens Kappa : 0.2775665399239543
ROC AUC Score: 0.6938053097345132

 clasification report:
               precision    recall  f1-score   support

         0.0       0.99      0.99      0.99       565
         1.0       0.22      0.40      0.29         5

    accuracy                           0.98       570
   macro avg       0.61      0.69      0.64       570
weighted avg       0.99      0.98      0.98       570


 confussion matrix:
 [[558   7]
 [  3   2]]
```

#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))

In [ ]:
```
Model Two : is similar to Model one except for optimizer as SGD.
```

In [0]:
```
model_two = Sequential()
model_two.add(Conv1D(filters=8, kernel_size=11, activation='relu', input_shape=x_t
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'))
```

In [0]:
```
model_two.compile(optimizer="sgd", loss = 'binary_crossentropy', metrics=['accurac
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)
```

```
Epoch 1/5
 - 2s - loss: 0.6956 - acc: 0.5732 - f1_m: 0.5715 - precision_m: 0.5773 - reca
ll_m: 0.5739 - val_loss: 0.5338 - val_acc: 0.8333 - val_f1_m: 0.0160 - val_pre
cision_m: 0.0125 - val_recall_m: 0.0225
Epoch 2/5
 - 1s - loss: 0.6345 - acc: 0.6443 - f1_m: 0.6336 - precision_m: 0.6514 - reca
ll_m: 0.6237 - val_loss: 0.5184 - val_acc: 0.8123 - val_f1_m: 0.0321 - val_pre
cision_m: 0.0250 - val_recall_m: 0.0449
Epoch 3/5
 - 1s - loss: 0.5915 - acc: 0.6799 - f1_m: 0.6798 - precision_m: 0.6801 - reca
ll_m: 0.6862 - val_loss: 0.4142 - val_acc: 0.8509 - val_f1_m: 0.0281 - val_pre
cision_m: 0.0241 - val_recall_m: 0.0337
Epoch 4/5
 - 1s - loss: 0.5558 - acc: 0.7156 - f1_m: 0.7191 - precision_m: 0.7098 - reca
ll_m: 0.7342 - val_loss: 0.3969 - val_acc: 0.8456 - val_f1_m: 0.0259 - val_pre
cision_m: 0.0211 - val_recall_m: 0.0337
Epoch 5/5
 - 1s - loss: 0.5299 - acc: 0.7320 - f1_m: 0.7343 - precision_m: 0.7276 - reca
ll_m: 0.7456 - val_loss: 0.4134 - val_acc: 0.8211 - val_f1_m: 0.0299 - val_pre
cision_m: 0.0225 - val_recall_m: 0.0449
```

In [0]:
```
model_two.compile(optimizer="sgd", loss = 'binary_crossentropy', metrics=['accurac
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)
```

```
Epoch 1/80
 - 2s - loss: 0.4693 - acc: 0.7737 - f1_m: 0.7765 - precision_m: 0.7705 - re
call_m: 0.7872 - val_loss: 0.4424 - val_acc: 0.8053 - val_f1_m: 0.0299 - val
_precision_m: 0.0225 - val_recall_m: 0.0449
Epoch 2/80
 - 1s - loss: 0.4432 - acc: 0.7901 - f1_m: 0.7927 - precision_m: 0.7824 - re
call_m: 0.8081 - val_loss: 0.3808 - val_acc: 0.8158 - val_f1_m: 0.0259 - val
_precision_m: 0.0211 - val_recall_m: 0.0337
Epoch 3/80
 - 1s - loss: 0.4151 - acc: 0.8056 - f1_m: 0.8093 - precision_m: 0.7948 - re
call_m: 0.8283 - val_loss: 0.3005 - val_acc: 0.8596 - val_f1_m: 0.0281 - val
_precision_m: 0.0241 - val_recall_m: 0.0337
Epoch 4/80
 - 1s - loss: 0.3721 - acc: 0.8336 - f1_m: 0.8377 - precision_m: 0.8204 - re
call_m: 0.8598 - val_loss: 0.4062 - val_acc: 0.8018 - val_f1_m: 0.0281 - val
_precision_m: 0.0241 - val_recall_m: 0.0337
Epoch 5/80
 - 1s - loss: 0.3650 - acc: 0.8384 - f1_m: 0.8421 - precision_m: 0.8218 - re
call_m: 0.8674 - val_loss: 0.2574 - val_acc: 0.8825 - val_f1_m: 0.0374 - val
```

In [0]:

```
model_two.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv1d_5 (Conv1D)            (None, 3187, 8)           184       
_________________________________________________________________
max_pooling1d_5 (MaxPooling1 (None, 797, 8)            0         
_________________________________________________________________
batch_normalization_4 (Batch (None, 797, 8)            32        
_________________________________________________________________
conv1d_6 (Conv1D)            (None, 787, 16)           1424      
_________________________________________________________________
max_pooling1d_6 (MaxPooling1 (None, 197, 16)           0         
_________________________________________________________________
batch_normalization_5 (Batch (None, 197, 16)           64        
_________________________________________________________________
conv1d_7 (Conv1D)            (None, 187, 32)           5664      
_________________________________________________________________
max_pooling1d_7 (MaxPooling1 (None, 47, 32)            0         
_________________________________________________________________
batch_normalization_6 (Batch (None, 47, 32)            128       
_________________________________________________________________
conv1d_8 (Conv1D)            (None, 37, 64)            22592     
_________________________________________________________________
max_pooling1d_8 (MaxPooling1 (None, 9, 64)             0         
_________________________________________________________________
flatten_2 (Flatten)          (None, 576)               0         
_________________________________________________________________
dropout_3 (Dropout)          (None, 576)               0         
_________________________________________________________________
dense_4 (Dense)              (None, 64)                36928     
_________________________________________________________________
dropout_4 (Dropout)          (None, 64)                0         
_________________________________________________________________
dense_5 (Dense)              (None, 64)                4160      
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 65        
=================================================================
Total params: 71,241
Trainable params: 71,129
Non-trainable params: 112
_________________________________________________________________
```

In [0]:
```
plot_model(model_two, to_file='model.png', show_shapes=True, show_layer_names=True
```

```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)
<ipython-input-84-592674ac8fb0> in <module>
----> 1 plot_model(model_two, to_file='model.png', show_shapes=True, show_laye
r_names=True)

/conda/envs/rapids/lib/python3.6/site-packages/keras/utils/vis_utils.py in plo
t_model(model, to_file, show_shapes, show_layer_names, rankdir)
    130             'LR' creates a horizontal plot.
    131     """
--> 132     dot = model_to_dot(model, show_shapes, show_layer_names, rankdir)
    133     _, extension = os.path.splitext(to_file)
    134     if not extension:

/conda/envs/rapids/lib/python3.6/site-packages/keras/utils/vis_utils.py in mod
el_to_dot(model, show_shapes, show_layer_names, rankdir)
     53     from ..models import Sequential
     54 
---> 55     _check_pydot()
     56     dot = pydot.Dot()
     57     dot.set('rankdir', rankdir)

/conda/envs/rapids/lib/python3.6/site-packages/keras/utils/vis_utils.py in _ch
eck_pydot()
     18     if pydot is None:
     19         raise ImportError(
---> 20             'Failed to import `pydot`. '
     21             'Please install `pydot`. '
     22             'For example with `pip install pydot`.')

ImportError: Failed to import `pydot`. Please install `pydot`. For example wit
h `pip install pydot`.
```

In [0]:
```
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))
```

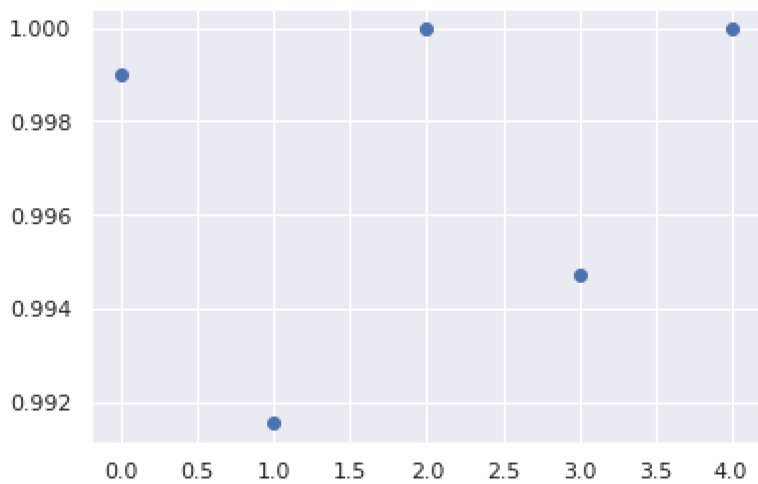
```
5087/5087 [==============================] - 0s 93us/step
[0.011488029176923306, 0.9976410480527014, 0.01965795164143896, 0.019657951641
43896, 0.01965795164143896]

Model Performance: Loss and Accuracy on validation data
570/570 [==============================] - 0s 88us/step
[0.03130376166786606, 0.9964912314164011, 0.17543859649122806, 0.1754385964912
2806, 0.17543859649122806]
```

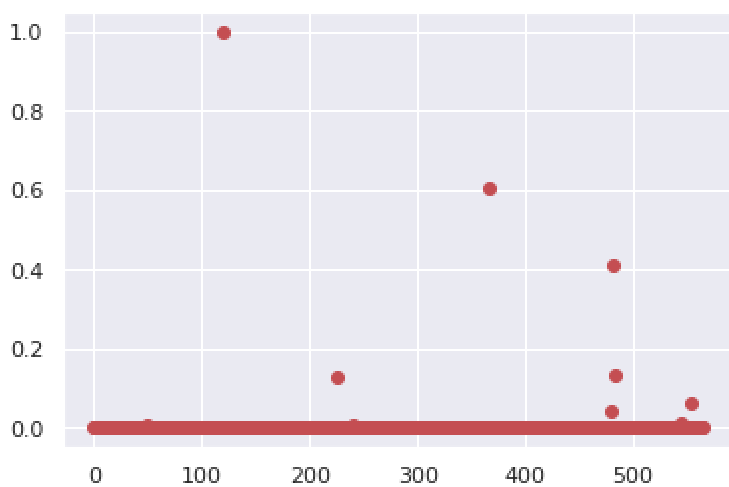
In [0]:

```
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()
```

In [0]:

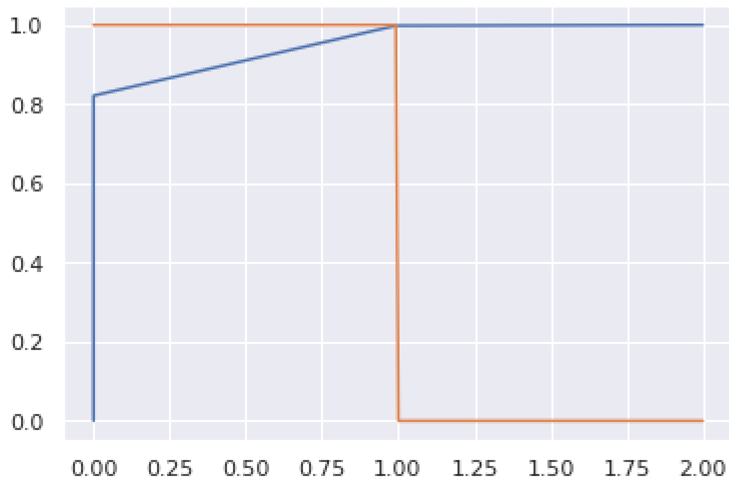
```
non_idx = np.where(y_test[:,0] == 0.)[0]
yes_idx = np.where(y_test[:,0] == 1.)[0]
y_hat = model_two.predict(x_test)[:,0]
plt.plot([y_hat[i] for i in yes_idx], 'bo')
plt.show()
plt.plot([y_hat[i] for i in non_idx], 'ro')
plt.show()
```



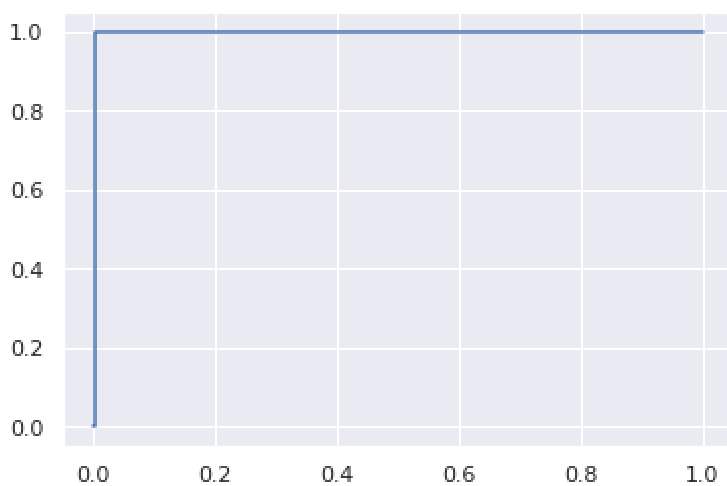


In [0]:

```
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, cro
plt.plot(fpr, tpr)
plt.show()
print("ROC area under curve is {0:.2f}".format(roc_auc_score(y_true, y_hat)))
```




```
Crossover at 0.99 with specificity 1.00
```



```
ROC area under curve is 1.00
```

In [0]:

```python
#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: 0.9982456140350877
F1 score: 0.9090909090909091
Recall: 1.0
Precision: 0.8333333333333334
Cohens Kappa : 0.9082125603864735
ROC AUC Score: 0.9991150442477876

 clasification report:
               precision    recall  f1-score   support

         0.0       1.00      1.00      1.00       565
         1.0       0.83      1.00      0.91         5

    accuracy                           1.00       570
   macro avg       0.92      1.00      0.95       570
weighted avg       1.00      1.00      1.00       570


 confussion matrix:
 [[564   1]
 [  0   5]]
```

In [0]:

```
#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: 0.9982307843522705
F1 score: 0.8860759493670887
Recall: 0.9459459459459459
Precision: 0.8333333333333334
Cohens Kappa : 0.8851880180055922
ROC AUC Score: 0.9722799036660423

 clasification report:
               precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      5050
         1.0       0.83      0.95      0.89        37

    accuracy                           1.00      5087
   macro avg       0.92      0.97      0.94      5087
weighted avg       1.00      1.00      1.00      5087


 confussion matrix:
 [[5043    7]
 [   2   35]]
```

In [0]:

```
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")
```

```
Saved model to disk
```

In [0]:
```
import pickle
# Save model using pickle
pickle.dump(model_two, open('final_model.pkl', 'wb'))
```

In [0]:
```
# Save model using keras
model_two.save("final_model.h5")
```

In [0]:
```
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.

In [0]:
```
model_three = Sequential()
model_three.add(Conv1D(filters=8, kernel_size=11, activation='relu', input_shape=x
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'))
```

In [0]:
```
model_three.compile(optimizer=sgd, loss = 'binary_crossentropy', metrics=['accurac
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
 - 3s - loss: 0.6354 - acc: 0.6581 - f1_m: 0.6609 - precision_m: 0.6545 - reca
ll_m: 0.6957 - val_loss: 0.8422 - val_acc: 0.6702 - val_f1_m: 0.0281 - val_pre
cision_m: 0.0187 - val_recall_m: 0.0561
Epoch 2/5
 - 1s - loss: 0.5053 - acc: 0.7519 - f1_m: 0.7605 - precision_m: 0.7319 - reca
ll_m: 0.8024 - val_loss: 0.6392 - val_acc: 0.5035 - val_f1_m: 0.0201 - val_pre
cision_m: 0.0122 - val_recall_m: 0.0561
Epoch 3/5
 - 1s - loss: 0.3249 - acc: 0.8624 - f1_m: 0.8687 - precision_m: 0.8359 - reca
ll_m: 0.9091 - val_loss: 0.1793 - val_acc: 0.9561 - val_f1_m: 0.0321 - val_pre
cision_m: 0.0561 - val_recall_m: 0.0225
Epoch 4/5
 - 1s - loss: 0.2475 - acc: 0.9116 - f1_m: 0.9154 - precision_m: 0.8895 - reca
ll_m: 0.9482 - val_loss: 0.0819 - val_acc: 0.9737 - val_f1_m: 0.0321 - val_pre
cision_m: 0.0561 - val_recall_m: 0.0225
Epoch 5/5
 - 1s - loss: 0.1893 - acc: 0.9309 - f1_m: 0.9328 - precision_m: 0.9153 - reca
ll_m: 0.9552 - val_loss: 0.0667 - val_acc: 0.9737 - val_f1_m: 0.0421 - val_pre
cision_m: 0.0561 - val_recall_m: 0.0337
```

In [0]:
```
model_three.compile(optimizer=sgd, loss = 'binary_crossentropy', metrics=['accurac
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)
```

```
call_m: 0.9848 - val_loss: 0.0581 - val_acc: 0.9895 - val_f1_m: 0.0421 - val
_precision_m: 0.0561 - val_recall_m: 0.0337
Epoch 6/60
 - 1s - loss: 0.0584 - acc: 0.9823 - f1_m: 0.9826 - precision_m: 0.9763 - re

call_m: 0.9905 - val_loss: 0.2819 - val_acc: 0.8825 - val_f1_m: 0.0468 - val
_precision_m: 0.0401 - val_recall_m: 0.0561
Epoch 7/60
 - 1s - loss: 0.0497 - acc: 0.9839 - f1_m: 0.9841 - precision_m: 0.9809 - re
call_m: 0.9886 - val_loss: 0.0686 - val_acc: 0.9860 - val_f1_m: 0.0499 - val
_precision_m: 0.0561 - val_recall_m: 0.0449
Epoch 8/60
 - 1s - loss: 0.0349 - acc: 0.9902 - f1_m: 0.9903 - precision_m: 0.9876 - re
call_m: 0.9937 - val_loss: 0.1116 - val_acc: 0.9702 - val_f1_m: 0.0510 - val
_precision_m: 0.0468 - val_recall_m: 0.0561
Epoch 9/60
 - 1s - loss: 0.0518 - acc: 0.9839 - f1_m: 0.9842 - precision_m: 0.9780 - re
call_m: 0.9912 - val_loss: 0.0628 - val_acc: 0.9912 - val_f1_m: 0.0561 - val
_precision_m: 0.0561 - val_recall_m: 0.0337
Epoch 10/60
```

In [0]:
```
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
_________________________________________________________________
```

In [0]:
```
plot_model(model_three, to_file='model.png', show_shapes=True, show_layer_names=Tr
```

```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)
<ipython-input-96-cb527f7556cd> in <module>
----> 1 plot_model(model_three, to_file='model.png', show_shapes=True, show_la
yer_names=True)

/conda/envs/rapids/lib/python3.6/site-packages/keras/utils/vis_utils.py in plo
t_model(model, to_file, show_shapes, show_layer_names, rankdir)
    130                 'LR' creates a horizontal plot.
    131         """
--> 132     dot = model_to_dot(model, show_shapes, show_layer_names, rankdir)
    133     _, extension = os.path.splitext(to_file)
    134     if not extension:

/conda/envs/rapids/lib/python3.6/site-packages/keras/utils/vis_utils.py in mod
el_to_dot(model, show_shapes, show_layer_names, rankdir)
     53     from ..models import Sequential
     54 
---> 55     _check_pydot()
     56     dot = pydot.Dot()
     57     dot.set('rankdir', rankdir)

/conda/envs/rapids/lib/python3.6/site-packages/keras/utils/vis_utils.py in _ch
eck_pydot()
     18     if pydot is None:
     19         raise ImportError(
---> 20             'Failed to import `pydot`. '
     21             'Please install `pydot`. '
     22             'For example with `pip install pydot`.')

ImportError: Failed to import `pydot`. Please install `pydot`. For example wit
h `pip install pydot`.
```

In [0]:
```
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 96us/step
[0.009895077891811122, 0.9960684134211689, 0.01965795164143896, 0.019657951641
43896, 0.01965795164143896]

Model Performance: Loss and Accuracy on validation data
570/570 [==============================] - 0s 94us/step
[0.04262540397778873, 0.9929824628328022, 0.15594540980824254, 0.1754385964912
2806, 0.1403508792843735]
```

In [0]:
```python
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))
```

```
Accuracy: 0.9984273638686849
F1 score: 0.9024390243902439
Recall: 1.0
Precision: 0.8222222222222222
Cohens Kappa : 0.901653923113358
ROC AUC Score: 0.9992079207920792

 clasification report:
               precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      5050
         1.0       0.82      1.00      0.90        37

    accuracy                           1.00      5087
   macro avg       0.91      1.00      0.95      5087
weighted avg       1.00      1.00      1.00      5087


 confussion matrix:
 [[5042    8]
 [   0   37]]
```

In [0]:

```
#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: 0.9894736842105263
F1 score: 0.4000000000000001
Recall: 0.4
Precision: 0.4
Cohens Kappa : 0.3946902654867256
ROC AUC Score: 0.6973451327433627

 clasification report:
               precision    recall  f1-score   support

         0.0       0.99      0.99      0.99       565
         1.0       0.40      0.40      0.40         5

    accuracy                           0.99       570
   macro avg       0.70      0.70      0.70       570
weighted avg       0.99      0.99      0.99       570


 confussion matrix:
 [[562   3]
 [  3   2]]
```

In [0]:

```
import numpy

from sklearn.model_selection import GridSearchCV

from keras.wrappers.scikit_learn import KerasClassifier
```

In [0]:
```
def create_model(init_mode='uniform'):
  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 = Sequential()
  model_three.add(Conv1D(filters=8, kernel_size=11, activation='relu', input_shape
  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, kernel_initializer=init_mode, activation='relu'))
  model_three.add(Dropout(0.25))
  model_three.add(Dense(64, kernel_initializer=init_mode,activation='relu'))
  model_three.add(Dense(1,kernel_initializer=init_mode, activation='sigmoid'))
  model_three.compile(optimizer= sgd, loss = 'binary_crossentropy', metrics=['accu
  return model_three
```

In [0]:

```python
seed = 7

numpy.random.seed(seed)

batch_size = 128

epochs = 10



model_CV = KerasClassifier(build_fn=create_model, epochs=epochs,

                           batch_size=batch_size, verbose=1)

# define the grid search parameters

init_mode = ['uniform', 'lecun_uniform', 'normal', 'zero',

             'glorot_normal', 'glorot_uniform', 'he_normal', 'he_uniform']



param_grid = dict(init_mode=init_mode)

#grid = GridSearchCV(estimator=model_CV, param_grid=param_grid, n_jobs=-1, cv=3)
grid = GridSearchCV(estimator=model_CV, param_grid=param_grid, n_jobs=3, cv=3)

grid_result = grid.fit(x_train, y_train)
```

```
ERROR:exception calling callback for <Future at 0x7fb5c8ca94e0 state=finishe
d raised TerminatedWorkerError>
Traceback (most recent call last):
  File "/conda/envs/rapids/lib/python3.6/site-packages/joblib/externals/lok
y/_base.py", line 625, in _invoke_callbacks
    callback(self)
  File "/conda/envs/rapids/lib/python3.6/site-packages/joblib/parallel.py",
line 309, in __call__
    self.parallel.dispatch_next()
  File "/conda/envs/rapids/lib/python3.6/site-packages/joblib/parallel.py",
line 731, in dispatch_next
    if not self.dispatch_one_batch(self._original_iterator):
  File "/conda/envs/rapids/lib/python3.6/site-packages/joblib/parallel.py",
line 759, in dispatch_one_batch
    self._dispatch(tasks)
  File "/conda/envs/rapids/lib/python3.6/site-packages/joblib/parallel.py",
line 716, in _dispatch
    job = self._backend.apply_async(batch, callback=cb)
  File "/conda/envs/rapids/lib/python3.6/site-packages/joblib/_parallel_back
```

In [0]:
```
# 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']

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}')
```

In [0]:
```
K.clear_session()
```

In [0]:
```
#def create_model(init_mode='glorot_uniform', activation='relu', dropout_rate=0.5,
#                  optimizer='sgd', filters=8):
def create_model(init_mode='glorot_uniform', activation='relu', dropout_rate=0.5,
                 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=ini
    model.add(MaxPool1D(strides=4))
    model.add(BatchNormalization())
    model.add(Conv1D(filters=(4 * filters), kernel_size=11, kernel_initializer=ini
    model.add(MaxPool1D(strides=4))
    model.add(BatchNormalization())
    model.add(Conv1D(filters=(8 * filters), kernel_size=11, kernel_initializer=ini
    model.add(MaxPool1D(strides=4))
    model.add(Flatten())
    model.add(Dropout(dropout_rate))
    model.add(Dense(units=neurons, activation=activation, kernel_initializer=init_
    model.add(Dropout(dropout_rate/2))
    model.add(Dense(units=neurons, activation=activation, kernel_initializer=init_
    model.add(Dense(1, activation='sigmoid', kernel_initializer=init_mode))
    #model.compile(optimizer="sgd", loss = 'binary_crossentropy', metrics=['accura
    #model.compile(optimizer=optimizer, loss = 'binary_crossentropy', metrics=['ac
    model.compile(optimizer=optimizer, loss = 'binary_crossentropy')
    return model
```

In [0]:

```
%%time
#INPUT_LIB = '/content/drive/My Drive/capstone_datasets/'
INPUT_LIB = './'
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, ax
x_test = ((x_test - np.mean(x_test, axis=1).reshape(-1,1)) / np.std(x_test, axis=1
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 10.8 s, sys: 1.57 s, total: 12.3 s
Wall time: 12.3 s
```

In [0]:

```
%%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)]
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:72
4: DataConversionWarning: A column-vector y was passed when a 1d array was exp
ected. Please change the shape of y to (n_samples, ), for example using ravel
().
  y = column_or_1d(y, warn=True)

10100
(10100, 3197) (10100,)
CPU times: user 258 ms, sys: 280 ms, total: 538 ms
Wall time: 578 ms
```

In [0]:

```

```

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

In [0]:

```
seed = 7
np.random.seed(seed)
#batch_size = 128
epochs = 2

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 = ['uniform', 'lecun_uniform', 'normal', 'zero', 'glorot_normal', 'g
    activation = ['softmax', 'softplus', 'softsign', 'relu', 'tanh', 'sigmoid', 'h
    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]

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

    #grid = GridSearchCV(estimator=model_CV, param_grid=param_grid, scoring = scor
    grid = RandomizedSearchCV(estimator=model_CV, param_distributions=param_grid,
                              n_iter=5) #, 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_pa
    result_df = pd.DataFrame(grid_result.cv_results_)
    print(result_df)
    result_df.to_csv('RandomizedSearchCV_result_df.csv', index=False, encoding='ut
    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']
```

```python
    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 softmax 0.4 64 Adamax 16
WARNING:tensorflow:From /conda/envs/rapids/lib/python3.6/site-packages/tenso
rflow/python/ops/math_grad.py:102: div (from tensorflow.python.ops.math_ops)
is deprecated and will be removed in a future version.
Instructions for updating:
Deprecated in favor of operator or tf.math.divide.

WARNING:From /conda/envs/rapids/lib/python3.6/site-packages/tensorflow/pytho
n/ops/math_grad.py:102: div (from tensorflow.python.ops.math_ops) is depreca
ted and will be removed in a future version.
Instructions for updating:
Deprecated in favor of operator or tf.math.divide.






Epoch 1/2
6733/6733 [==============================] - 6s 932us/step - loss: 0.6377
```

In [0]:

```python
randon_search_model = grid_result.best_estimator_
```

In [0]:
```
x_train_input = np.stack([x_train, uniform_filter1d(x_train, axis=1, size=200)], a

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 [==============================] - 2s 446us/step
Accuracy: 0.9998034204835856
F1 score: 0.9866666666666666
Recall: 1.0
Precision: 0.9736842105263158
Cohens Kappa : 0.9865676646959571
ROC AUC Score: 0.9999009900990099

 clasification report:
               precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      5050
         1.0       0.97      1.00      0.99        37

    accuracy                           1.00      5087
   macro avg       0.99      1.00      0.99      5087
weighted avg       1.00      1.00      1.00      5087


 confussion matrix:
 [[5049    1]
 [   0   37]]
```

In [0]:
```python
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 430us/step
Accuracy: 0.9912280701754386
F1 score: 0.0
Recall: 0.0
Precision: 0.0
Cohens Kappa : 0.0
ROC AUC Score: 0.5

 clasification report:
               precision    recall  f1-score   support

         0.0       0.99      1.00      1.00       565
         1.0       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:1437: UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 d
ue to no predicted samples.
  'precision', 'predicted', average, warn_for)
/conda/envs/rapids/lib/python3.6/site-packages/sklearn/metrics/classification.
py:1437: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0
due to no predicted samples.
  'precision', 'predicted', average, warn_for)
/conda/envs/rapids/lib/python3.6/site-packages/sklearn/metrics/classification.
py:1437: UndefinedMetricWarning: Precision and F-score are ill-defined and bei
ng set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
/conda/envs/rapids/lib/python3.6/site-packages/sklearn/metrics/classification.
py:1437: UndefinedMetricWarning: Precision and F-score are ill-defined and bei
ng set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
/conda/envs/rapids/lib/python3.6/site-packages/sklearn/metrics/classification.
py:1437: UndefinedMetricWarning: Precision and F-score are ill-defined and bei
ng set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
```

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.

In [0]:

```
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
```

In [0]:

```
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
            })
```

In [0]:

```python
# 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}')
```

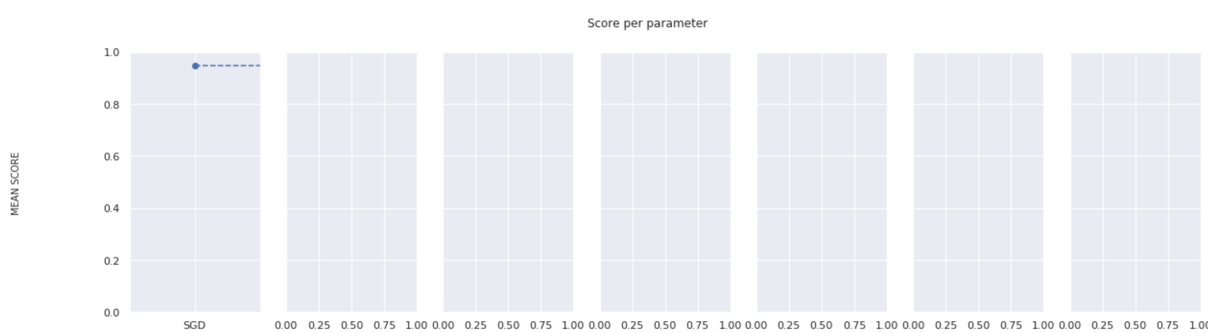
```
Best Accuracy for 0.9473390928539741 using {'optimizer': 'Adam', 'neurons': 6
4, 'init_mode': 'he_uniform', 'filters': 16, 'dropout_rate': 0.4, 'batch_siz
e': 32, 'activation': 'sigmoid'}

Top results and params:
 mean=0.8468, std=0.2112 using {'optimizer': 'Adadelta', 'neurons': 64, 'init_
mode': 'lecun_uniform', 'filters': 8, 'dropout_rate': 0.4, 'batch_size': 32,
'activation': 'hard_sigmoid'}
 mean=0.7029, std=0.4185 using {'optimizer': 'Adam', 'neurons': 32, 'init_mod
e': 'uniform', 'filters': 8, 'dropout_rate': 0.4, 'batch_size': 64, 'activatio
n': 'tanh'}
 mean=0.3398, std=0.4652 using {'optimizer': 'Adamax', 'neurons': 64, 'init_mo
de': 'lecun_uniform', 'filters': 16, 'dropout_rate': 0.4, 'batch_size': 32, 'a
ctivation': 'softmax'}
 mean=0.9473, std=0.06852 using {'optimizer': 'Adam', 'neurons': 64, 'init_mod
e': 'he_uniform', 'filters': 16, 'dropout_rate': 0.4, 'batch_size': 32, 'activ
ation': 'sigmoid'}
 mean=0.8991, std=0.1421 using {'optimizer': 'RMSprop', 'neurons': 64, 'init_m
ode': 'he_uniform', 'filters': 16, 'dropout_rate': 0.4, 'batch_size': 64, 'act
ivation': 'softsign'}
```

In [0]:

```python
#For plotting the results when tuning several hyperparameters,
#what I did was fixed all parameters to their best value except for one
#and plotted the mean score for the other parameter for each of its values.

def plot_search_results(grid_result):
    """
    Params:
        grid: A trained GridSearchCV object.
    """
    ## Results from grid search
    results = grid.cv_results_
    means_test = results['mean_test_score']
    stds_test = results['std_test_score']
    #means_train = results['mean_train_score']
    #stds_train = results['std_train_score']

    ## Getting indexes of values per hyper-parameter
    masks=[]
    masks_names= list(grid.best_params_.keys())
    for p_k, p_v in grid.best_params_.items():
        masks.append(list(results['param_'+p_k].data==p_v))

    #params=grid.param_grid
    params=grid.param_distributions

    ## Ploting results
    fig, ax = plt.subplots(1,len(params),sharex='none', sharey='all',figsize=(20,5
    fig.suptitle('Score per parameter')
    fig.text(0.04, 0.5, 'MEAN SCORE', va='center', rotation='vertical')
    pram_preformace_in_best = {}
    for i, p in enumerate(masks_names):
        m = np.stack(masks[:i] + masks[i+1:])
        pram_preformace_in_best
        best_parms_mask = m.all(axis=0)
        best_index = np.where(best_parms_mask)[0]
        x = np.array(params[p])
        y_1 = np.array(means_test[best_index])
        e_1 = np.array(stds_test[best_index])
        #y_2 = np.array(means_train[best_index])
        #e_2 = np.array(stds_train[best_index])
        ax[i].errorbar(x, y_1, e_1, linestyle='--', marker='o', label='test')
        #ax[i].errorbar(x, y_1, e_1, linestyle='--', marker='o', label='train')
        #ax[i].errorbar(x, y_2, e_2, linestyle='-', marker='^',label='test' )
        ax[i].set_xlabel(p.upper())

    plt.show()

plot_search_results(grid_result)
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-123-833e71c60cc4> in <module>
     46     plt.show()
     47 
---> 48 plot_search_results(grid_result)
```

```
<ipython-input-123-833e71c60cc4> in plot_search_results(grid_result)
     39         #y_2 = np.array(means_train[best_index])
     40         #e_2 = np.array(stds_train[best_index])
---> 41         ax[i].errorbar(x, y_1, e_1, linestyle='--', marker='o', label=
'test')
     42         #ax[i].errorbar(x, y_1, e_1, linestyle='--', marker='o', label
='train')
     43         #ax[i].errorbar(x, y_2, e_2, linestyle='-', marker='^',label
='test' )

/conda/envs/rapids/lib/python3.6/site-packages/matplotlib/__init__.py in inner
(ax, data, *args, **kwargs)
   1808                         "the Matplotlib list!)" % (label_namer, func._
_name__),
   1809                         RuntimeWarning, stacklevel=2)
-> 1810             return func(ax, *args, **kwargs)
   1811 
   1812         inner.__doc__ = _add_data_doc(inner.__doc__,

/conda/envs/rapids/lib/python3.6/site-packages/matplotlib/axes/_axes.py in err
orbar(self, x, y, yerr, xerr, fmt, ecolor, elinewidth, capsize, barsabove, lol
ims, uplims, xlolims, xuplims, errorevery, capthick, **kwargs)
   3258             noylims = ~(lolims | uplims)
   3259             if noylims.any() or len(noylims) == 0:
-> 3260                 xo, _ = xywhere(x, lower, noylims & everymask)
   3261                 lo, uo = xywhere(lower, upper, noylims & everymask)
   3262                 barcols.append(self.vlines(xo, lo, uo, **eb_lines_styl
e))

/conda/envs/rapids/lib/python3.6/site-packages/matplotlib/axes/_axes.py in xyw
here(xs, ys, mask)
   3159             ys are not arrays
   3160             """
-> 3161             assert len(xs) == len(ys)
   3162             assert len(xs) == len(mask)
   3163             xs = [thisx for thisx, b in zip(xs, mask) if b]

AssertionError: 
```



In [0]:

```
train = pd.read_csv('exoTrain.csv')
test = pd.read_csv('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_arra


#X_train, X_test, y_train, y_test = train_test_split(X, y.values,train_size=0.90,
#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_arr
```

AUTOML Platforms democratizes machine learning by making it accessible for everyone. AUTOML platforms involves automation of feature preprocessing, feature engineering, algorithm selection, hyperparameter optimization, model tuning, etc. TPOT specifically is based on Genetic Alogrithms where it performs genetic representation of solution domain and a fitness function to evaluate the solution domain. TPOT will consider a population in solution domain and apply simplified evolution laws to have them optimize the objective function called fitness. For each generation it will select the best (meaning here the fittest) individuals using fitness function and use genetic operations to reproduce next generation. Recombination (exploitation) combines parent features to form children solutions. Mutations introduce random (exploration) perturbations. This way, the average population's fitness is supposed to improve from one generation to the next to arrive at fittest individual. However the model two was still performing better than the model from TPOT.

In [0]:
```
!pwd
!mkdir tpot_checkpoint
```

In [0]:
```
exoplanet_model_tpot = TPOTClassifier(
                          generations = 5,     #100: Number of iterations to the ru
                          population_size=10, #100: Number of individuals to retai
                          offspring_size = 2, #None: Number of offspring to produc
                          mutation_rate=0.8,  #0.9: Mutation rate for the genetic
                          crossover_rate=0.2, #0.1: Crossover rate for the genetic
                          scoring='accuracy', #accuracy: One of the available scor
                          cv=5,               #5: Cross-validation strategy used w
                          subsample = 1.0,    #1.0: Fraction of training samples t
                          n_jobs = 2,         #1: Number of processes to use in pa
                          max_time_mins=120,  #None: How many minutes TPOT has to
                          max_eval_time_mins=5,     #5: How many minutes TPOT has t
                          random_state=3,     #None: Integer. Use this parameter t
                          config_dict = None, #None: 'TPOT light', 'TPOT Sparse', '
                       #   template = None,     #None: Template of predefined pipel
                          warm_start = True,  #False: Flag indicating whether the
                          memory = 'auto',    #None: 'auto' or a caching dir  path
                          use_dask = False,   #False: Whether to use Dask-ML's pip
                          periodic_checkpoint_folder = './tpot_checkpoint', #None:
                          early_stop=None,    #None: Ends the optimization process
                          verbosity=3,        #0: 0 to 4.  How much information TP
                       )
```

In [0]:
```
exoplanet_model_tpot.fit(X_train_array, y_train)
```

In [0]:
```
y_pred=exoplanet_model_tpot.predict(X_test_array)
y_train_pred=exoplanet_model_tpot.predict(X_train_array)
```

In [0]:
```
from sklearn.metrics import precision_score, recall_score, confusion_matrix, class

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))
```

In [0]:
```
print(exoplanet_model_tpot.score(X_test_array, y_test))
```

In [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('\n clasification report:\n', classification_report(y_train, y_train_pred))
print('\n confussion matrix:\n',confusion_matrix(y_train, y_train_pred))
```

In [0]:
```
exoplanet_model_tpot.export('exoplanet_model_tpot.py')

#tpot_model2.export('tpot_loanpred_model2_30mins.py')

!cat exoplanet_model_tpot.py
```

In [0]:
```
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_pick

# 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))
```

In [0]:
```
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
```

In [0]:
```
!mkdir /rapids/notebooks/utils/hostdir/capstone/tpot_checkpoint2
```

In [0]:

```
exoplanet_model_tpot_f1 = TPOTClassifier(
                          generations = 5,    #100: Number of iterations to the ru
                          population_size=10, #100: Number of individuals to retai
                          offspring_size = 2, #None: Number of offspring to produc
                          mutation_rate=0.8,  #0.9: Mutation rate for the genetic
                          crossover_rate=0.2, #0.1: Crossover rate for the genetic
                          scoring='f1_weighted', #accuracy: One of the available s
                          cv=5,               #5: Cross-validation strategy used w
                          subsample = 1.0,    #1.0: Fraction of training samples t
                          n_jobs = 4,         #1: Number of processes to use in pa
                          max_time_mins=120,  #None: How many minutes TPOT has to
                          max_eval_time_mins=10,     #5: How many minutes TPOT has
                          random_state=3,     #None: Integer. Use this parameter t
                          config_dict = 'TPOT light', #None: 'TPOT light', 'TPOT S
                     #   template = None,     #None: Template of predefined pipel
                          warm_start = False,  #False: Flag indicating whether the
                          memory = 'auto',    #None: 'auto' or a caching dir  path
                          use_dask = False,   #False: Whether to use Dask-ML's pip
                          periodic_checkpoint_folder = '/rapids/notebooks/utils/ho
                          early_stop=None,    #None: Ends the optimization process
                          verbosity=3,        #0: 0 to 4.  How much information TP
                     )
```

In [0]:

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

In [0]:

```
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_p

# load the model from disk
exoplanet_model_tpot1_f1_load = pickle.load(open('exoplanet_model_tpot_pickle_f1.p

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

In [0]:

```
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))
```

In [0]:

In [0]:

In [0]:

In [0]:

In [0]:

In [0]:

In [0]:

In [0]:

In [0]:

In [0]:

In [0]: