COP528 Applied Machine Learning Coursework

TASK 1 - MACHINE LEARNING PIPELINE

I. INTRODUCTION

This Report highlights the pipeline used to analyze a Student Performance dataset from an open URL. Algorithms were used to classify the [Performance] of the students using the other 10 attribute. Three algorithms were used and optimized, which include AdaBoost, Random Forest and k Nearest Neighbor. Random Forest gave best results with optimal **no. estimators = 350.** The results were unsatisfactory with low Accuracy (31%), Precision (31%) and F1(30.3%) scores, however it highlights the problems involved in the data.

II. DATA

A. DataSet

The data set chosen for this task was *'Student Performance on an entrance examination Data Set'* – this dataset contains participants data who have gained admission to medical colleges of Assam. This dataset has 666 instances with 11 attributes and was obtained from the UCI open Machine Learning Repository. This data's default task is classification. The data will be used to classify the [Performance] of the participants by the information provided from the other attributes.

After examining the data briefly, to obtain a basic understanding of the data, it was concluded that there is no missing data, or N/A values, and all data is nominal.

The data was further examined, and the nominal data was encoded, so all data was converted to numerical, i.e., gender was encoded from [female, male] to [0,1]. Normalization is unnecessary as the data used to be nominal, so the data cannot be skewed by anomalous data values.

B. Preliminary Analysis

This initial analysis is required to further evaluate the dataset and allow further insight for modelling a better analytical model.
Immediately after cleaning up the data and encoding into numeric data, a very simple correlation matrix was plotted which is shown in Figure 1.

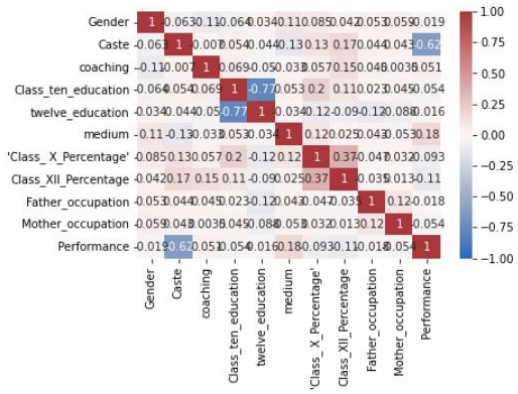

Figure 1- Correlation Matrix

This matrix shows how each attribute correlates with each other, from the plot shown in Figure 1. Analysis from the matrix shows that not all attributes are highly influential as a predictor, it can be seen that [Caste] does not correlate with our classification category [Performance]. Therefore, this can be attribute can be removed, theoretically this will improve the prediction model as well as reduce computational power required and produce minimal noise. Further dimension reduction will be necessary, this is carried out later.

III. METHEDOLOGY

The following workflow shown in Figure 2, displays the approach of completing this task. II Data-B- Preliminary

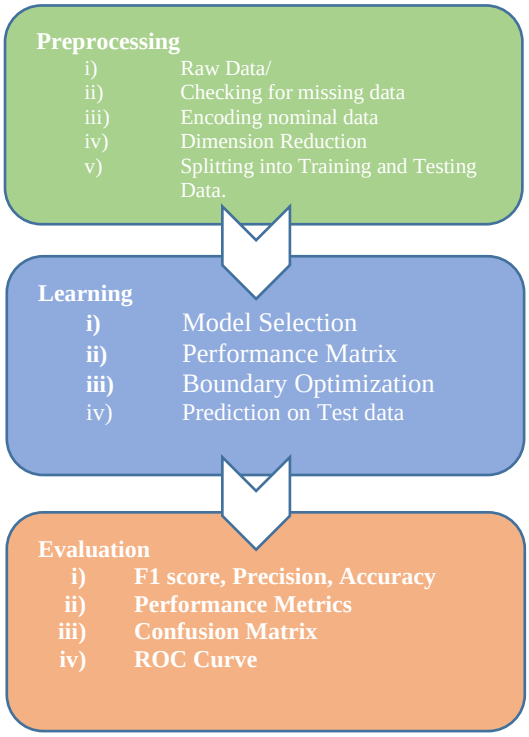

Figure 2- Machine Learning Pipeline workflow

Analysis covers the obtaining of the raw data, checking for missing data, encoding and down selection of best predictor attributes.

A. Feature Selection/Dimension Reduction

With [Caste] removed from our dataset, the other attributes can be individually examined to determine which is most influential when classifying the [Performance]. The data was first Standardized which scales the data with a mean of 0 and variance of 1, this is a requirement for an optimal performing machine learning algorithm.
Principal Component Analysis is used (PCA) which is a dimension reduction method that decomposes the covariance of the data into eigenvalues and eigenvectors, in order to project the data in a lower dimensional subspace. [1] This eliminates noise and lowers computational power. [2]. For this dataset, PCA was applied, and the dimension of the data was reduced to using only 3 principal components.

B. *Data Splitting*

The PCA data was then sampled, into training and testing sets, this was done using **sklearn** module in python. The ratio of training to testing was split into 70-30 %. The training data is further split into an addition 70-30 into training-validation datasets, this can be used in a loop function to optimize model and fine tuning it to avoid overfitting of the prediction. The models are trained on the training data, and then tested, the performances of the classifiers are then compared with other methods.[8]

C. *Machine Learning Algorithms*

i) *AdaBoost*

First classifying algorithm used was a supervised algorithm, Adaptive Boosting, better known as AdaBoost, which is used as an ensemble method in Machine Learning. The principal behind it is combines several different classifiers with weights being assigned to each instance after each iteration, with higher weights to misclassified instances, this improves the performance of the classifier as a whole. This utilizes many 'weak' classifiers to form 'strong' ones.[3][8]

For the algorithm to work well, it was susceptible to fine tuning, this was done by tuning the number of estimators and learning rate through optimization loop using validation test set. The results can be shown in Figure 3. The optimal model had, **n estimators** = 1150, and **learning rate** = 0.8, with algorithm 'SAMME', in DecisionTree Classifier AdaBoost pipeline.

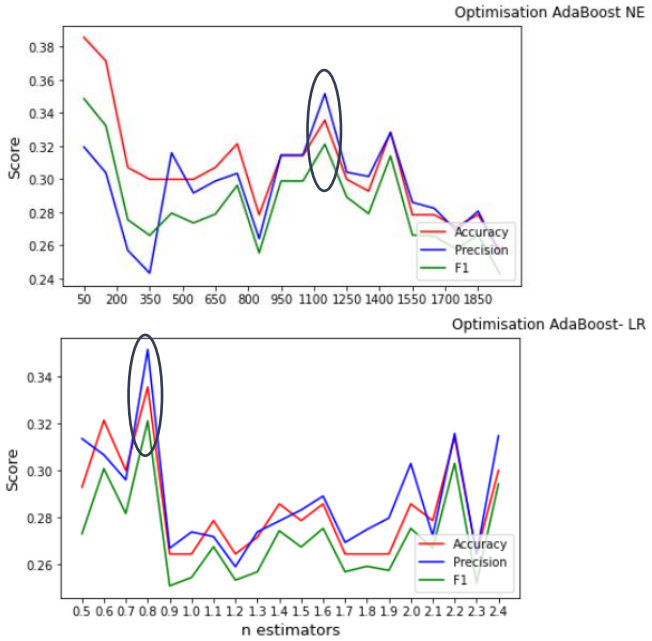

Figure 3- AdaBoost Optimisation

ii) *Random Forest*

Random Forest classifier was attempted next, this is an ensemble Machine Learning technique, that mixes many classifiers to offer solutions to complicated data structure and is commonly used in regression or classifying problems. This algorithm utilizes many decisions tress. The aggregation of bagging and bootstrapping of this algorithm, creates the 'forest' of decision trees.[8] This is when the samples can be used various times in a single tree as they are not removed from the data set when used. Different data samples are used to train the individual trees in the 'forest, this give as high variance for each tree, however as a whole 'forest' the variance is very low.

The results are established based totally on the prediction of decision trees, using mean value to drive its decisions, so is very good at resolving overfitting issues.[4]

For Optimisation of RFC, it is dependent on the number of trees that are used, as this determines the accuracy of the prediction, but will require a balance, as more trees require more computational power. Figure 4 shows the optimization graph using the validation test loop for hyper-parameter tuning. This shows that the best value to uses for **no. of estimators** (i.e., trees) is 350.

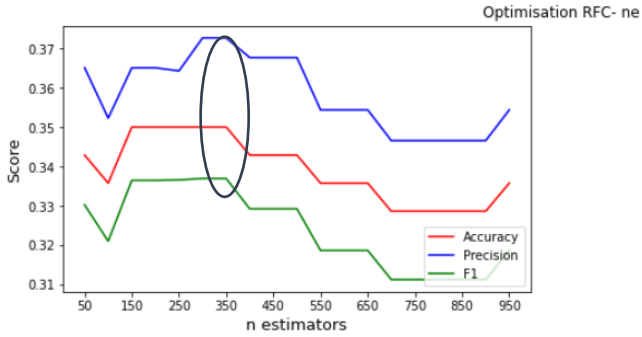

Figure 4- RFC Optimization

iii) *k Nearest Neighbor*

The final classifier was using k-Nearest Neighbor, which is also a very simple supervised Machine Learning technique. The classifier works by assuming the similarities between new data, and already available data. This means that when new data appears, it can be easily classified into a well-defined category using the kNN algorithm. This is also known as a 'lazy' classifier as it doesn't learn form the training data when present, but stores it instead, and only performs an action when new data is presented.[6][8]

To optimize the algorithm, a validation test loop was created which discovered the optimal 'k' value and weighing metric. Various values of 'k' were used ranging from 1-20, and Figure 5 shows that the optimal **'k' value** was 7, with **weight** = 'uniform'. This was done with PCA applied with 3 principal components.

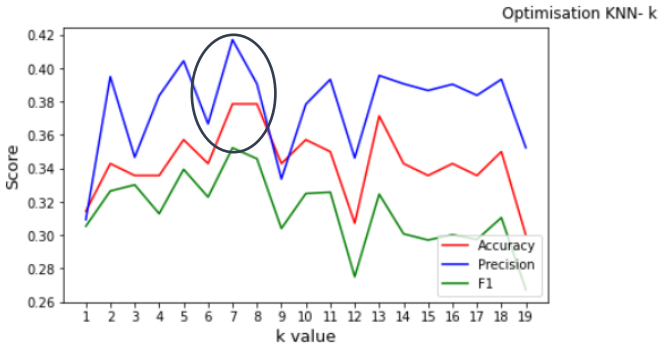

Figure 5 - kNN Optimization

IV. EXPERIMENTAL RESULTS

All 3 models were used on the test set with the original training data. The 3 algorithms were tuned to optimized hyper-parameters. The results are displayed below using the following scores Accuracy, Precision and F1. These are made using sklearn functions. These were then ranked accordingly in order of best performance.

Score	AdaBoost	RFC	kNN
Accuracy %	28.0	31.0	26.0
Precision %	25.3	31.0	25.5
F1 %	26.4	30.3	25.3
Ranking	**2**	**1**	**3**

Figure 6- Performance Metrics

Figure 6 shows the performance metrics of the algorithms. Random Forest performed best, followed by AdaBoost and kNN respectively. RFC had the highest F1 score so this means lower rates of false positives and negatives compared to the rest. This may be because the original data was nominal and decision trees perform better with discrete data.
However, even through RFC performed the best, overall predictions were very poor. This may mean that the attributes within this data are not good predictors of the classification.

V. EVALUTATION

From the results displayed above Optimized RFC was chosen as the best predictor for this data, with the highest F1 score. To understand the poor scoring and performance of the algorithm as confusion matrix was produced as shown in Figure 7 below.

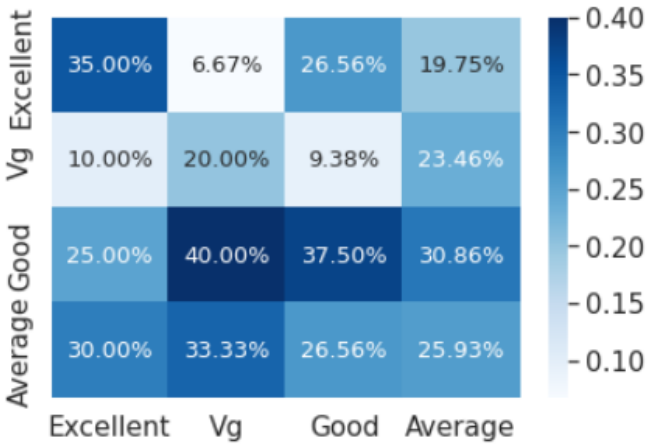

Figure 7- RFC Confusion Matrix

The matrix evidently shows that the model created was unsuccessful and not very effective, as it shows that the model was unbale to predict even 50% correctly in any of the categories, with the highest being 37.5%, this is close to our recorded data when comparing all the scores. The percentage of False Negatives is very high. [7]

A further evaluation was done with a ROC curve. This is shown in Figure 8.

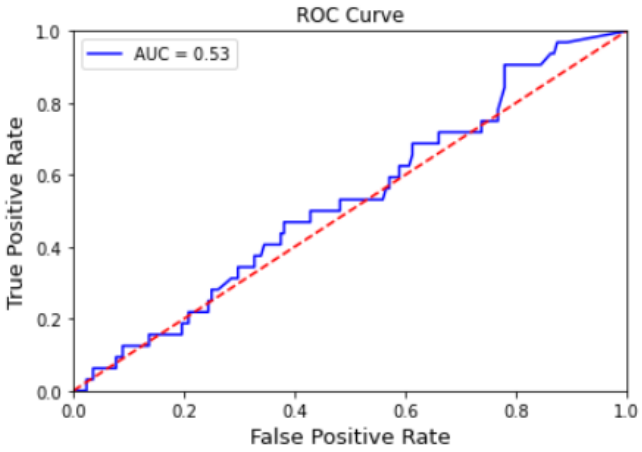

Figure 8- RFC ROC Curve

The ROC curve shows the performance of False Positive against True Positives of the model, the higher the area under the curve (AUC) the better the predictor model. However, as established previously, this is a poor model, even if it performed best out of the other 2 algorithms, with an AUC of 0.53, it is close to 0.5 which is basically random guessing.[8]

VI. REFLECTION

The report demonstrates a pipeline used to classify Student Performances using 3 Algorithms with best performance being achieved by Random Forest Classifier, which obtained an Accuracy and Precision on 31%? This is very poor performance.

This is probably due to the data being insufficient so a good model could not be made, even after hyper-parameter tuning. Likewise another reason for the poor results, maybe due to the data itself, as it was all discrete, it is difficult to make a prediction even after encoding into numeric values.

Next time, it may be necessary to remove a few more features which have no contribution to the Student Performance. However, this was done by dimensional reduction using PCA, but the results still faltered. This data might not fit the classification task.

To conclude even after following a thorough in-depth pipeline for classification the results were inadequate, further research and work is needed, maybe different types of algorithms (regression or clustering) are required to obtain better results for this data set,

TASK 2 – CONVOLUTIONAL NEURAL NETWORK

I. INTRODUCTION

A data folder was given with image set belonging to 10 different classes. The Task was to create a Deep Learning machine algorithm to correctly classify the images into the right classes.

A Convolutional Neural Network (CNN) which is a Deep Learning method, was used to tackle this task. A predefined simple CNN was used and modified appropriately to obtain a basic performing model which achieved a validation accuracy of 61.5% with 2.85 loss value.
An attempt was made to Optimize this model via fine tuning and data augmentation; however, the model was not operational.

II. DATA

A. Dataset

A dataset was provided with 13394 images belonging to 10 classes. The data was already split into a training and a validation set, with split of 70-30% respectively. The 10 classes had separate folders within the main dataset. The data contained images of Fish, Dog, Radio, Church, Chainsaw, French Horn, Garbage Truck, Fuel Pumps, Golf Balls and Parachute.

B. Preliminary Analysis

When examining the raw dataset, it was evident that the classes were missing labels, therefore they were manually labeled in the categories listed above.

After a random pick of 9 images within the data was printed as shown in Figure 1.

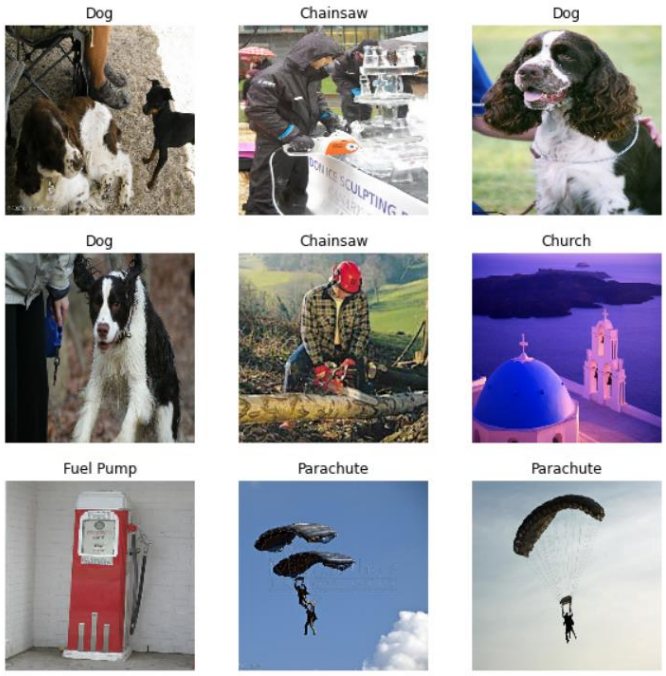

Figure 1- Random Sampled Images

Figure 1 shows that sampling was done randomly which mean same categories have been displayed multiple times. This initial evaluation of the images gives an insight of how features could be identified when training the model. For example, images of some of the Dogs can be easily distinguished, however some may be hard as the colors of the Dog are similar to the background noise, this may be misinterpreted by the CNN model.
Likewise, the images of parachutes displayed are shown to be very small in detail and would be difficult to identify. This analysis shows that a right kernel size is crucial that is to be used in the CNN layers, as this can determine if the CNN can distinguish the smaller less focused objects in the photographs.[6]

III. METHEDOOLOGY

C. Pre-Processing

The image data was imported using TensorFlow functions with a batch size of 32 and image size of 160x160, they were also shuffled so the algorithm can be trained and tested.
The initial image data set was encoded to 0-255 according to RGB pixel values. CNN works best with small values. Therefore, the sampling had to be rescaled before passing through the CNN layers.
Standardization is most vital part of the pre-processing of image dataset, this rescales the values ranging from 0-1 and confines the range of the data, this is beneficial as it helps with the ‘issue of propagating gradients. [1] [6]
A stochastic gradient descent optimization algorithm is used for training in CNN, this measure the error gradient, the difference between input and output, which is dependent of batch size. Therefore, it is important to choose an appropriate batch size when training the CNN algorithm.

D. Pipeline

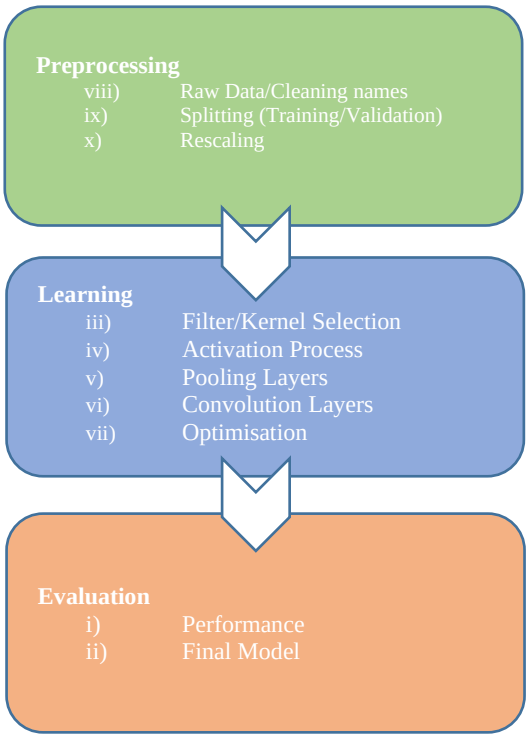

Figure 2- Machine Learning Pipeline workflow

The approach was very similar to Task 1

E. CNN Model

The model was created in Colab. Mainly using the TensorFlow and Keras packages in python to create a functioning neural network for the given task. Figure 2 shows the Architecture Modell of the CNN created, this is discussed in detail as follows:

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_3 (Rescaling)     (None, 160, 160, 3)       0

 conv2d_9 (Conv2D)           (None, 158, 158, 32)      896

 max_pooling2d_7 (MaxPooling  (None, 79, 79, 32)       0
 2D)

 conv2d_10 (Conv2D)          (None, 77, 77, 64)        18496

 max_pooling2d_8 (MaxPooling  (None, 38, 38, 64)       0
 2D)

 conv2d_11 (Conv2D)          (None, 36, 36, 64)        36928

 max_pooling2d_9 (MaxPooling  (None, 18, 18, 64)       0
 2D)

 flatten_3 (Flatten)         (None, 20736)             0

 dense_7 (Dense)             (None, 64)                1327168

 dense_8 (Dense)             (None, 10)                650

=================================================================
Total params: 1,384,138
Trainable params: 1,384,138
Non-trainable params: 0
```

Figure 2- CNN model

i) CNN (conv2d)

A very simple premade CNN template was used and was modified accordingly to fit the dataset. The main attribute is the number of kernels that would be used to extract features from the images.[3] Normal procedures were followed, by setting the padding to equal the 'same' as input and output, this is conventional for CNN.

A commonly used activation function was used 'relu'. Relu is commonly used because of its speed, and simplicity. Likewise, empirically it has been proven that when training a deep leaning network, the data converge faster and accurately when using Relu than using Sigmoid function.

Figure 2 shows the Convolution layer being used, with changes of dimensions which is always decreasing. 3 convolution layers are used all using activation function 'relu' but differing in shape.

ii) Pooling Layer(max_pooling2d)

These layers are used to reduce the size of the features. This in return lessens the computations needed by the network and lowers the parameters required for learning.
This layer is placed after the convolution layer and summarizes the features present within the region of the above layer. This means that the model now functions on the summarized data, rather than individual instances. [4]

MaxPooling is the most common method, which extracts the most prominent features of the previous layers. [6] As shown in Figure 2, 3 MaxPooling layers were used after every convolution layer, and these were halving the dimensions of the convolution each time.

iii) Flattening layer(flatten)

Flattening layers are placed just before the final output layer, as the layers above create the feature selection model, it outputs it as 3D, however the flattening convert all the value previously calculated by the other layers into vectors where the classification can start.[8]
As shown in Figure 2, this is used once after the 3 modules of CNN layers above it.

iv) *Dense Layers (dense)*

The dense layer is a deeply connected neural network layer, which receives information from all other neurons from the layers above. This layer is very common and is used at the end, as it performs various matrix-vector multiplications. These calculated values are the parameters that are used for training and updating using a method called backpropagation. [5]

Figure 2 shows the use of 2 dense layers, these should ideally end with the dense layer having dimensions equal to the number of classes present in the original dataset (10 in this case is valid).

IV) EXPERIMENTAL RESULTS

The model was compiled and fitted into the validation data. The results are shown below.

Experiment parameters –

- 32 images per batch
- 3 modules of CNN
- 2 dense layers
- Activation function = 'relu'
- Epoch = 10

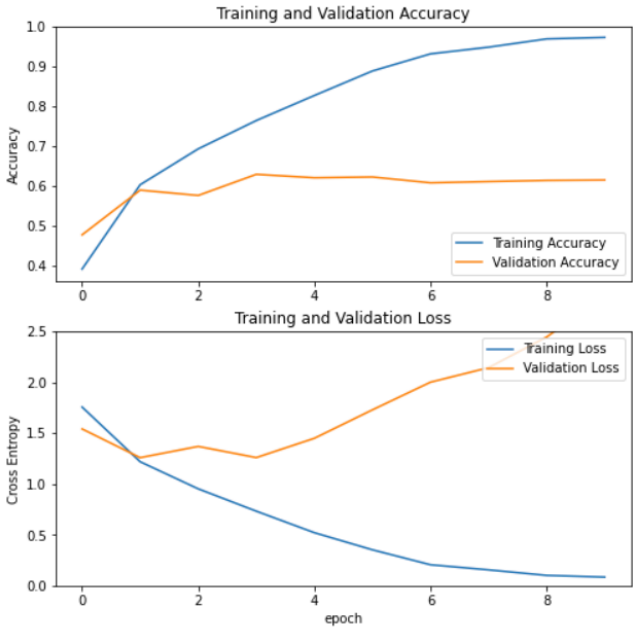

Figure 3- CNN Original Data Accuracy

Epoch is the number of iterations for the algorithm, this is a fairly low value due to the unavailable facilities as it would require high computations and a stronger GPU.

V) EVALUATION

Figure 3 shows decent results. The accuracy of the raining data is very high and is complemented by the minimal loss after 10 iterations. Final values were 97.3% accuracy and 0.08 loss value. However, the base model did not perform very well for the validation test, which had the best value of 61.5 % after 10 iteration and heavy loss value of 2.85.
It is very evident by the validation loss skewing up after the 10 iteration shows signs of heavy overfitting. [6]

There are various ways of combatting this problem

A. Data Augmentation

Data augmentation is normally done when the data set doesn't have enough instances, therefore data augmentation is done to increase the data and gives the algorithm more to learn from. The method behind this is to rotate, flip and scale the image in various ways, this helps to reveal additional features to the model [2][6]

B. Reduce Architecture Complexity

Regularization of the architecture can solve overfitting, using less hidden layers, and modules would help.

Dropout

Dropouts are another recognized technique to increase validation accuracy and decrease overfitting. This is when random neuros are deleted in the training so the wights of the attribute is not updated. This can be risky as deletion from initial layer will mean deletion from entire network, but this can be finetuned in CNN parameters.

These were attempted however the code would not function appropriately to provide any results for fine tuning.[5]

IV) REFLECTION

The initial model was created and gave promising results. However fine tuning the model with the researched methods described above to obtain better accuracy and fix the overfitting of the validation data, could not be achieved as the codes failed.

To develop a deeper understating of the CNN and fine tuning, further work is required, and research should be focused on the manipulation of the coding involved in creating the model. As a predefined model was used and modified the fundamentals of the models were unknow which have contributed to the failure of obtaining proficient results. Also, an evaluation can be improved by researching the use of other activation functions, and conclusions can be drawn and compared with previously researched empirical data.

TASK 1 REFERENCES -

1. *Brownlee, J., 2022. Feature Selection For Machine Learning in Python. [online] Machine Learning Mastery. Available at: <https://machinelearningmastery.com/feature-selection-machine-learning-python/> [Accessed 16 March 2022].*
2. *Medium. 2022. PCA using Python (scikit-learn). [online] Available at: <https://towardsdatascience.com/pca-using-python-scikit-learn-e653f8989e60> [Accessed 16 March 2022].*
3. *2022. [online] Available at: <https://www.mygreatlearning.com/blog/adaboost-algorithm/> [Accessed 16 March 2022].*
4. *Engineering Education (EngEd) Program | Section. 2022. Introduction to Random Forest in Machine Learning. [online] Available at: <https://www.section.io/engineering-education/introduction-to-random-forest-in-machine-learning/> [Accessed 16 March 2022].*
5. *Analytics Vidhya. 2022. Sruthi E R, Author at Analytics Vidhya. [online] Available at: <https://www.analyticsvidhya.com/blog/author/sruthi94/> [Accessed 16 March 2022].*
6. *www.javatpoint.com. 2022. K-Nearest Neighbor(KNN) Algorithm for Machine Learning - Javatpoint. [online] Available at: <https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning> [Accessed 16 March 2022].*
7. *Acutecaretesting.org. 2022. ROC curves – what are they and how are they used?. [online] Available at: <https://acutecaretesting.org/en/articles/roc-curves-what-are-they-and-how-are-they-used?BannerClosed&BannerClosed> [Accessed 16 March 2022].*
8. *Fang, H., 2022. Applied Machine Learning. [online] 21COP528 - Applied Machine Learning. Available at: <https://learn.lboro.ac.uk/course/view.php?id=21420> [Accessed 16 March 2022].*

TASK 2 REFERENCES -

1. *Medium. 2022. Data Preprocessing and Network Building in CNN. [online] Available at: <https://towardsdatascience.com/data-preprocessing-and-network-building-in-cnn-15624ef3a28b> [Accessed 18 March 2022].*
2. *Analytics Vidhya. 2022.* CNN Image Classification | Image Classification Using CNN. *[online] Available at: <https://www.analyticsvidhya.com/blog/2020/02/learn-image-classification-cnn-convolutional-neural-networks-3-datasets/> [Accessed 18 March 2022].*
3. *TensorFlow. 2022.* Transfer learning and fine-tuning | TensorFlow Core. *[online] Available at: <https://www.tensorflow.org/tutorials/images/transfer_learning> [Accessed 18 March 2022].*
4. *GeeksforGeeks. 2022.* CNN | Introduction to Pooling Layer - GeeksforGeeks. *[online] Available at: <https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/> [Accessed 18 March 2022].*
5. *MLK - Machine Learning Knowledge. 2022.* Keras Dense Layer Explained for Beginners - MLK - Machine Learning Knowledge. *[online] Available at: <https://machinelearningknowledge.ai/keras-dense-layer-explained-for-beginners/> [Accessed 18 March 2022].*
6. *Fang, H., 2022. Applied Machine Learning. [online] 21COP528 - Applied Machine Learning. Available at: <https://learn.lboro.ac.uk/course/view.php?id=21420> [Accessed 16 March 2022].*

APPENDIX

Colab Notebook - https://drive.google.com/drive/folders/1wUlZUgfe6PTMjnSvSqp7AEFj3wtbqHkO?usp=sharing

Task 1

```
#Getting the data ready
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from google.colab import drive
drive.mount('/content/gdrive')

df=pd.read_csv('gdrive/My Drive/Colab Notebooks/student_performance.csv') # Importing file from Google Drive
df.isnull().sum() # checking for incomplete values

print(df.describe) # gives idea of structure and inputs of dataset
from sklearn import preprocessing # encoding nominal data to numeric
enc = preprocessing.OrdinalEncoder()

enc.fit(df)
df_enc = enc.transform(df)
df_enc

import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

features = df.columns.values.tolist()          # creating a correleation matrix
corr_df = pd.DataFrame(df_enc, columns=features)
correlation_mat = corr_df.corr()
sns.heatmap(correlation_mat, annot=True, vmax=1, vmin=-1, center=0, cmap='vlag')
plt.show()

new_df = df.drop(columns='Caste')
from sklearn import preprocessing # encoding nominal data to numeric
enc = preprocessing.OrdinalEncoder()



enc.fit(new_df)
new_df_enc = enc.transform(new_df)
new_df_enc
X = new_df_enc[:,:9] # seperating the class attribute [Performance] as y
y = new_df_enc[:,9]


from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

X_std = StandardScaler().fit_transform(X)

pca = PCA(n_components=3) #Projection to 3 dimensions
principal_comp = pca.fit_transform(X_std)
principal_df = pd.DataFrame(data = principal_comp
        , columns = ['pc 1', 'pc 2','pc 3'])
```

```
y_df = pd.DataFrame(data=y,columns = ['Performance']) # putting dataset back together
finalDf = pd.concat([principal_df, y_df], axis = 1)

finalDf_enc = finalDf.to_numpy()
final_X = finalDf_enc[:,0:3] # converting back to numpy array
final_y = finalDf_enc[:,3]


from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
 final_X, final_y, test_size=0.3, random_state=0) # Splitting of intial data into training Set
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)

from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.preprocessing import MinMaxScaler
from sklearn.pipeline import make_pipeline

X_trainset, X_valset, y_trainset, y_valset = train_test_split(X_train,y_train,test_size=0.3, random_state=0)



AB_acc = []
AB_pr =[]
AB_f1= []
for ne in np.arange(50,2000,100): # chaninge no. of estimators in increments of 100 to get best results
  AB_clf = AdaBoostClassifier(DecisionTreeClassifier(max_depth=2, min_samples_split=25, min_samples_leaf=5),
 algorithm="SAMME",
 n_estimators=ne, learning_rate=0.8, random_state=0)

  AB_clf.fit(X_trainset, y_trainset)
  AB_pred = AB_clf.predict(X_valset)
  AB_acc.append(accuracy_score(y_valset,AB_pred))
  AB_pr.append(precision_score(y_valset,AB_pred, average='weighted'))
  AB_f1.append(f1_score(y_valset,AB_pred, average='weighted'))



x_plot = np.arange(50,2000,100)
plt.rcParams['axes.labelsize']= 13
fig1 = plt.figure(figsize=(15,8))
fig1.suptitle('Optimisation AdaBoost NE')
fig1.subplots_adjust(top=0.95)

ax1 = fig1.add_subplot(221)
ax1.plot(x_plot,AB_acc,'r-',label='Accuracy')
ax1.plot(x_plot,AB_pr,'b-',label='Precision')
ax1.plot(x_plot,AB_f1,'g-',label='F1')


plt.ylabel('Score')
plt.xlabel('n estimators')
plt.xticks(np.arange(50,2000,150))

plt.legend(loc='lower right')
plt.show()
```

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.preprocessing import MinMaxScaler
from sklearn.pipeline import make_pipeline

lAB_acc = []
lAB_pr =[]
lAB_f1= []

for lr in np.arange(0.5,2.5,0.1): # chaninge no. of estimators in increments of 25 to get best results
  AB_clf = AdaBoostClassifier(DecisionTreeClassifier(max_depth=2, min_samples_split=25, min_samples_leaf=5),
 algorithm="SAMME",
 n_estimators=1150, learning_rate=lr, random_state=0)

  AB_clf.fit(X_trainset, y_trainset)
  AB_pred = AB_clf.predict(X_valset)
  lAB_acc.append(accuracy_score(y_valset,AB_pred))
  lAB_pr.append(precision_score(y_valset,AB_pred, average='weighted'))
  lAB_f1.append(f1_score(y_valset,AB_pred, average='weighted'))


lx_plot = np.arange(0.5,2.5,0.1)
plt.rcParams['axes.labelsize']= 13
fig1 = plt.figure(figsize=(15,8))
fig1.suptitle('Optimisation AdaBoost- LR')
fig1.subplots_adjust(top=0.95)

ax1 = fig1.add_subplot(221)
ax1.plot(lx_plot,lAB_acc,'r-',label='Accuracy')
ax1.plot(lx_plot,lAB_pr,'b-',label='Precision')
ax1.plot(lx_plot,lAB_f1,'g-',label='F1')


plt.ylabel('Score')
plt.xlabel('n estimators')
plt.xticks(np.arange(0.5,2.5,0.1))

plt.legend(loc='lower right')
plt.show()

from sklearn.ensemble import RandomForestClassifier
rfc_acc = []
rfc_pr =[]
rfc_f1= []

for ne in np.arange(50,1000,50): # chaninge no. of estimators in increments of 25 to get best results
  rfc_clf =RandomForestClassifier(n_estimators=ne,n_jobs=-1,random_state=0)

  rfc_clf.fit(X_trainset, y_trainset)
  rfc_pred = rfc_clf.predict(X_valset)
  rfc_acc.append(accuracy_score(y_valset,rfc_pred))
  rfc_pr.append(precision_score(y_valset,rfc_pred, average='weighted'))
  rfc_f1.append(f1_score(y_valset,rfc_pred, average='weighted'))

rfcx_plot = np.arange(50,1000,50)
plt.rcParams['axes.labelsize']= 13
```

```
fig3 = plt.figure(figsize=(15,8))
fig3.suptitle('Optimisation RFC- ne')
fig3.subplots_adjust(top=0.95)

ax3 = fig3.add_subplot(221)
ax3.plot(rfcx_plot,rfc_acc,'r-',label='Accuracy')
ax3.plot(rfcx_plot,rfc_pr,'b-',label='Precision')
ax3.plot(rfcx_plot,rfc_f1,'g-',label='F1')


plt.ylabel('Score')
plt.xlabel('n estimators')
plt.xticks(np.arange(50,1000,100))

plt.legend(loc='lower right')
plt.show()

from pandas.core.common import random_state
from sklearn.neighbors import KNeighborsClassifier

knn_acc = []
knn_pr =[]
knn_f1= []

for k in np.arange(1,20,1): # chaninge no. of estimators in increments of 25 to get best results
  knn_clf = KNeighborsClassifier(n_neighbors=k, weights='uniform')

  knn_clf.fit(X_trainset, y_trainset)
  knn_pred = knn_clf.predict(X_valset)
  knn_acc.append(accuracy_score(y_valset,knn_pred))
  knn_pr.append(precision_score(y_valset,knn_pred, average='weighted'))
  knn_f1.append(f1_score(y_valset,knn_pred, average='weighted'))

knnx_plot = np.arange(1,20,1)
plt.rcParams['axes.labelsize']= 13
fig2 = plt.figure(figsize=(15,8))
fig2.suptitle('Optimisation KNN- k')
fig2.subplots_adjust(top=0.95)

ax2 = fig2.add_subplot(221)
ax2.plot(knnx_plot,knn_acc,'r-',label='Accuracy')
ax2.plot(knnx_plot,knn_pr,'b-',label='Precision')
ax2.plot(knnx_plot,knn_f1,'g-',label='F1')


plt.ylabel('Score')
plt.xlabel('k value')
plt.xticks(np.arange(1,20,1))

plt.legend(loc='lower right')
plt.show()


OPT_AB_clf = AdaBoostClassifier(DecisionTreeClassifier(max_depth=2, min_samples_split=25, min_samples_leaf=5),
 algorithm="SAMME",
 n_estimators=1150, learning_rate=0.8, random_state=0)

OPT_AB_clf.fit(X_train, y_train)
OPT_AB_pred = OPT_AB_clf.predict(X_test)
print(accuracy_score(y_test,OPT_AB_pred))
print(precision_score(y_test,OPT_AB_pred, average='weighted'))
print(f1_score(y_test,OPT_AB_pred, average='weighted'))
```

```
from sklearn.linear_model import LogisticRegression

OPT_rfc_clf = RandomForestClassifier(n_estimators=350,n_jobs=-1,random_state=0)

OPT_rfc_clf.fit(X_train, y_train)
OPT_rfc_pred = OPT_rfc_clf.predict(X_test)

print(accuracy_score(y_test,OPT_rfc_pred))
print(precision_score(y_test,OPT_rfc_pred, average='weighted'))
print(f1_score(y_test,OPT_rfc_pred, average='weighted'))

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, OPT_rfc_pred)
performance_label = ['Excellent', 'Vg', 'Good', 'Average']
import seaborn as sns
rfc_cm = pd.DataFrame(cm, index = performance_label, columns = performance_label)
# plt.figure(figsize=(10,7))
sns.set(font_scale=1.4) # for label size
sns.heatmap(rfc_cm/np.sum(rfc_cm), annot=True, annot_kws={"size": 13},
        cmap='Blues', fmt = '.2%') # font size

plt.show()



from sklearn.utils import multiclass
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
OPT_rfc_conf = OPT_rfc_clf.predict_proba(X_test)

fpr, tpr, thresholds = roc_curve(y_test, OPT_rfc_conf[:,1], pos_label=1)
roc_auc = metrics.auc(fpr, tpr)


import matplotlib.pyplot as plt
plt.title('ROC Curve')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'best')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()

OPT_knn_clf = KNeighborsClassifier(n_neighbors=7, weights='uniform')

OPT_knn_clf.fit(X_train, y_train)
OPT_knn_pred = OPT_knn_clf.predict(X_test)
print(accuracy_score(y_test,OPT_knn_pred))
print(precision_score(y_test,OPT_knn_pred, average='weighted'))
print(f1_score(y_test,OPT_knn_pred, average='weighted'))
```

Task 2

```
#Getting the data ready
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from google.colab import drive
drive.mount('/content/gdrive')

df=pd.read_csv('gdrive/My Drive/Colab Notebooks/student_performance.csv') # Importing file from Google Drive
df.isnull().sum() # checking for incomplete values

print(df.describe) # gives idea of structure and inputs of dataset
from sklearn import preprocessing # encoding nominal data to numeric
enc = preprocessing.OrdinalEncoder()

enc.fit(df)
df_enc = enc.transform(df)
df_enc

import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

features = df.columns.values.tolist()          # creating a correleation matrix
corr_df = pd.DataFrame(df_enc, columns=features)
correlation_mat = corr_df.corr()
sns.heatmap(correlation_mat, annot=True, vmax=1, vmin=-1, center=0, cmap='vlag')
plt.show()

new_df = df.drop(columns='Caste')
from sklearn import preprocessing # encoding nominal data to numeric
enc = preprocessing.OrdinalEncoder()



enc.fit(new_df)
new_df_enc = enc.transform(new_df)
new_df_enc
X = new_df_enc[:,:9] # seperating the class attribute [Performance] as y
y = new_df_enc[:,9]


from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

X_std = StandardScaler().fit_transform(X)

pca = PCA(n_components=3) #Projection to 3 dimensions
principal_comp = pca.fit_transform(X_std)
principal_df = pd.DataFrame(data = principal_comp
        , columns = ['pc 1', 'pc 2','pc 3'])

y_df = pd.DataFrame(data=y,columns = ['Performance']) # putting dataset back together
finalDf = pd.concat([principal_df, y_df], axis = 1)
```

```
finalDf_enc = finalDf.to_numpy()
final_X = finalDf_enc[:,0:3] # converting back to numpy array
final_y = finalDf_enc[:,3]


from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
 final_X, final_y, test_size=0.3, random_state=0) # Splitting of intial data into training Set
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)

from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.preprocessing import MinMaxScaler
from sklearn.pipeline import make_pipeline

X_trainset, X_valset, y_trainset, y_valset = train_test_split(X_train,y_train,test_size=0.3, random_state=0)



AB_acc = []
AB_pr =[]
AB_f1= []
for ne in np.arange(50,2000,100): # chaninge no. of estimators in increments of 100 to get best results
  AB_clf = AdaBoostClassifier(DecisionTreeClassifier(max_depth=2, min_samples_split=25, min_samples_leaf=5),
 algorithm="SAMME",
 n_estimators=ne, learning_rate=0.8, random_state=0)

  AB_clf.fit(X_trainset, y_trainset)
  AB_pred = AB_clf.predict(X_valset)
  AB_acc.append(accuracy_score(y_valset,AB_pred))
  AB_pr.append(precision_score(y_valset,AB_pred, average='weighted'))
  AB_f1.append(f1_score(y_valset,AB_pred, average='weighted'))



x_plot = np.arange(50,2000,100)
plt.rcParams['axes.labelsize']= 13
fig1 = plt.figure(figsize=(15,8))
fig1.suptitle('Optimisation AdaBoost NE')
fig1.subplots_adjust(top=0.95)

ax1 = fig1.add_subplot(221)
ax1.plot(x_plot,AB_acc,'r-',label='Accuracy')
ax1.plot(x_plot,AB_pr,'b-',label='Precision')
ax1.plot(x_plot,AB_f1,'g-',label='F1')


plt.ylabel('Score')
plt.xlabel('n estimators')
plt.xticks(np.arange(50,2000,150))

plt.legend(loc='lower right')
plt.show()
```

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.preprocessing import MinMaxScaler
from sklearn.pipeline import make_pipeline

lAB_acc = []
lAB_pr =[]
lAB_f1= []

for lr in np.arange(0.5,2.5,0.1): # chaninge no. of estimators in increments of 25 to get best results
  AB_clf = AdaBoostClassifier(DecisionTreeClassifier(max_depth=2, min_samples_split=25, min_samples_leaf=5),
 algorithm="SAMME",
 n_estimators=1150, learning_rate=lr, random_state=0)

  AB_clf.fit(X_trainset, y_trainset)
  AB_pred = AB_clf.predict(X_valset)
  lAB_acc.append(accuracy_score(y_valset,AB_pred))
  lAB_pr.append(precision_score(y_valset,AB_pred, average='weighted'))
  lAB_f1.append(f1_score(y_valset,AB_pred, average='weighted'))


lx_plot = np.arange(0.5,2.5,0.1)
plt.rcParams['axes.labelsize']= 13
fig1 = plt.figure(figsize=(15,8))
fig1.suptitle('Optimisation AdaBoost- LR')
fig1.subplots_adjust(top=0.95)

ax1 = fig1.add_subplot(221)
ax1.plot(lx_plot,lAB_acc,'r-',label='Accuracy')
ax1.plot(lx_plot,lAB_pr,'b-',label='Precision')
ax1.plot(lx_plot,lAB_f1,'g-',label='F1')


plt.ylabel('Score')
plt.xlabel('n estimators')
plt.xticks(np.arange(0.5,2.5,0.1))

plt.legend(loc='lower right')
plt.show()

from sklearn.ensemble import RandomForestClassifier
rfc_acc = []
rfc_pr =[]
rfc_f1= []

for ne in np.arange(50,1000,50): # chaninge no. of estimators in increments of 25 to get best results
  rfc_clf =RandomForestClassifier(n_estimators=ne,n_jobs=-1,random_state=0)

  rfc_clf.fit(X_trainset, y_trainset)
  rfc_pred = rfc_clf.predict(X_valset)
  rfc_acc.append(accuracy_score(y_valset,rfc_pred))
  rfc_pr.append(precision_score(y_valset,rfc_pred, average='weighted'))
  rfc_f1.append(f1_score(y_valset,rfc_pred, average='weighted'))

rfcx_plot = np.arange(50,1000,50)
plt.rcParams['axes.labelsize']= 13
fig3 = plt.figure(figsize=(15,8))
fig3.suptitle('Optimisation RFC- ne')
fig3.subplots_adjust(top=0.95)
```

```
ax3 = fig3.add_subplot(221)
ax3.plot(rfcx_plot,rfc_acc,'r-',label='Accuracy')
ax3.plot(rfcx_plot,rfc_pr,'b-',label='Precision')
ax3.plot(rfcx_plot,rfc_f1,'g-',label='F1')


plt.ylabel('Score')
plt.xlabel('n estimators')
plt.xticks(np.arange(50,1000,100))

plt.legend(loc='lower right')
plt.show()

from pandas.core.common import random_state
from sklearn.neighbors import KNeighborsClassifier

knn_acc = []
knn_pr =[]
knn_f1= []

for k in np.arange(1,20,1): # chaninge no. of estimators in increments of 25 to get best results
  knn_clf = KNeighborsClassifier(n_neighbors=k, weights='uniform')

  knn_clf.fit(X_trainset, y_trainset)
  knn_pred = knn_clf.predict(X_valset)
  knn_acc.append(accuracy_score(y_valset,knn_pred))
  knn_pr.append(precision_score(y_valset,knn_pred, average='weighted'))
  knn_f1.append(f1_score(y_valset,knn_pred, average='weighted'))

knnx_plot = np.arange(1,20,1)
plt.rcParams['axes.labelsize']= 13
fig2 = plt.figure(figsize=(15,8))
fig2.suptitle('Optimisation KNN- k')
fig2.subplots_adjust(top=0.95)

ax2 = fig2.add_subplot(221)
ax2.plot(knnx_plot,knn_acc,'r-',label='Accuracy')
ax2.plot(knnx_plot,knn_pr,'b-',label='Precision')
ax2.plot(knnx_plot,knn_f1,'g-',label='F1')


plt.ylabel('Score')
plt.xlabel('k value')
plt.xticks(np.arange(1,20,1))

plt.legend(loc='lower right')
plt.show()


OPT_AB_clf = AdaBoostClassifier(DecisionTreeClassifier(max_depth=2, min_samples_split=25, min_samples_leaf=5),
 algorithm="SAMME",
 n_estimators=1150, learning_rate=0.8, random_state=0)

OPT_AB_clf.fit(X_train, y_train)
OPT_AB_pred = OPT_AB_clf.predict(X_test)
print(accuracy_score(y_test,OPT_AB_pred))
print(precision_score(y_test,OPT_AB_pred, average='weighted'))
print(f1_score(y_test,OPT_AB_pred, average='weighted'))

from sklearn.linear_model import LogisticRegression
```

```
OPT_rfc_clf = RandomForestClassifier(n_estimators=350,n_jobs=-1,random_state=0)

OPT_rfc_clf.fit(X_train, y_train)
OPT_rfc_pred = OPT_rfc_clf.predict(X_test)

print(accuracy_score(y_test,OPT_rfc_pred))
print(precision_score(y_test,OPT_rfc_pred, average='weighted'))
print(f1_score(y_test,OPT_rfc_pred, average='weighted'))

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, OPT_rfc_pred)
performance_label = ['Excellent', 'Vg', 'Good', 'Average']
import seaborn as sns
rfc_cm = pd.DataFrame(cm, index = performance_label, columns = performance_label)
# plt.figure(figsize=(10,7))
sns.set(font_scale=1.4) # for label size
sns.heatmap(rfc_cm/np.sum(rfc_cm), annot=True, annot_kws={"size": 13},
        cmap='Blues', fmt = '.2%') # font size

plt.show()



from sklearn.utils import multiclass
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
OPT_rfc_conf = OPT_rfc_clf.predict_proba(X_test)

fpr, tpr, thresholds = roc_curve(y_test, OPT_rfc_conf[:,1], pos_label=1)
roc_auc = metrics.auc(fpr, tpr)


import matplotlib.pyplot as plt
plt.title('ROC Curve')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'best')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()

OPT_knn_clf = KNeighborsClassifier(n_neighbors=7, weights='uniform')

OPT_knn_clf.fit(X_train, y_train)
OPT_knn_pred = OPT_knn_clf.predict(X_test)
print(accuracy_score(y_test,OPT_knn_pred))
print(precision_score(y_test,OPT_knn_pred, average='weighted'))
print(f1_score(y_test,OPT_knn_pred, average='weighted'))
```