Artificial Intelligence

Assignment 4

Presented by:

Ayman Ahmed Abdelaziz

Summary

It was required to compare between the following classifiers: KNN, Decision Tree, Random Forest, AdaBoost and Neural Network on the dataset MAGIC 4 which had 2 classes classified from the following features: fLength, fWidth, fSize, fConc, fConc1, fAsym, fM3Long,fM3Trans, fAlpha, fDist, and 2 classes g and h which are gamma and hadron.

Loading and Balancing The Data

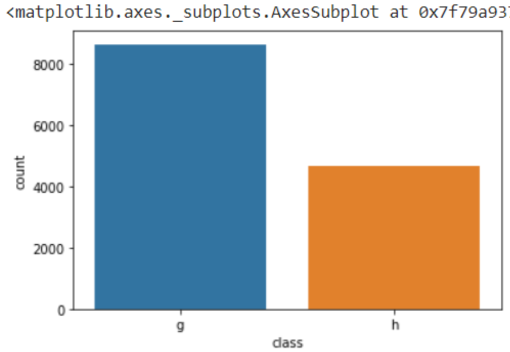
Loading the data was used with pandas.

The data was unbalanced, so to get good results and not being biased towards the dominant class, balancing the data was required using the undersampling technique, it removes random entries from the dominant class to equalize both classes.

Before Balancing:

```
train_data,test_data=load_split_data()
sns.countplot(x='class',data=train_data)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f79a9370320:
```

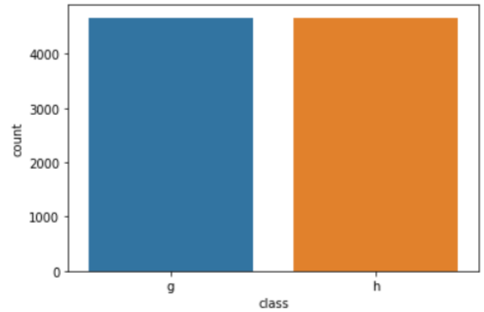


After balancing:

```
5] train_data=undersample(train_data)
```

```
7] sns.countplot(x='class',data=train_data)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f79a925b668>
```



Classifiers

KNN Classifier:

The KNN classifier uses K nearest neighbors to classify the entering example.

Decision Tree Classifier:

The Decision Tree classifier uses the gini impurity index to place the features in the tree with respect to its gini index the lower the closer to the root.

Random Forest Classifier:

The random forest classifier creates several trees inside the forest with less features inside each tree to reduce overfitting the output probability is averaged to get the final probability

Naïve Bayes Classifier:

The naïve bayes classifier uses conditional probability rule assuming that all features are independent.

AdaBoost Classifier

The AdaBoost can be used in conjunction with many other types of learning algorithms to improve performance.

Neural Network Classifier:

The neural network classifier uses a neural network to classify the entry given, using the optimizer Adam with learning rate 0.0001 and loss binary crossentropy as the labels are binary, then the neural network is trained for 25 epochs.

Classifiers

The structure starts with a normalization layer that normalizes the data adapted to the training set, and then 2 fully connected layers with an activation function ReLU and an output layer with one neuron with a sigmoid activation function to classify the entry either Gamma or Hadron.

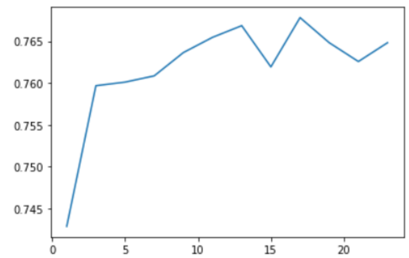
Tunning Classifiers

Tunning the classifiers is done with respect to the validation accuracy, we first generate several models of each classifier and then choose the one with the best validation accuracy.

KNN: Parameters to tune K

```
scores_list=get_knns()
plt.plot(range(1,25,2),scores_list)
```

```
[<matplotlib.lines.Line2D at 0x7f79a8687400>]
```



Metrics(Recall, Precision,F1,Accuracy) and Confusion Matrix:

```
0.8658172946597994 0.8251097907517437 0.844973544973545 0.7946021731510691
[[3194  495]
 [ 677 1340]]
```

Decision Tree: No parameters to tune

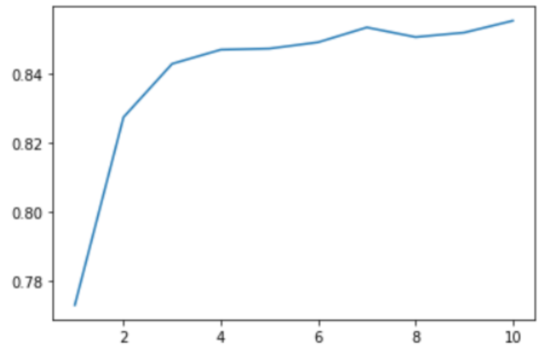
Metrics(Recall, Precision,F1,Accuracy) and Confusion Matrix:

```
0.787747357007319 0.8711031175059952 0.8273309608540925 0.7874167542937259
[[2906  783]
 [ 430 1587]]
```

Random Forest: parameters to tune number of estimators

```
plt.plot(range(1,11),scores_list)
```

```
[<matplotlib.lines.Line2D at 0x7f79a13e8160>]
```



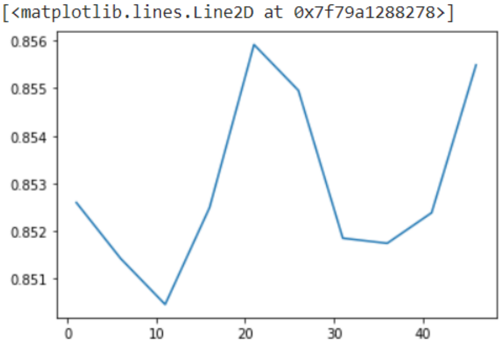
Metrics(Recall, Precision,F1,Accuracy) and Confusion Matrix:

```
0.8641908376253727 0.9026047565118913 0.8829801966486637 0.8519102698913424
[[3188  501]
 [ 344 1673]]
```

AdaBoost: parameters to tune number of estimators

```
plt.plot(range(1,50,5),scores_list)
```

```
[<matplotlib.lines.Line2D at 0x7f79a1288278>]
```



Metrics(Recall, Precision,F1,Accuracy) and Confusion Matrix:

```
0.8712388181078883 0.9028089887640449 0.8867429990343495 0.8561163687346652
[[3214  475]
 [ 346 1671]]
```

Tunning the classifiers

Naïve Bayes: No parameters to tune

```
0.8991596638655462 0.730777704340163 0.8062712688381137 0.7206449351559762
[[3317  372]
 [1222  795]]
```

Neural Network: parameters to tune number of neurons inside each layer

```
normalizer=Normalization()
#train_asnp=train_data.values
normalizer.adapt(train_asnp)
model=Sequential([normalizer,
                  Dense(8,'relu'),
                  Dense(8,'relu'),
                  Dense(1,'sigmoid')
                  ])
optimizer=Adam(learning_rate=0.0001)
model.compile(optimizer=optimizer,loss='binary_crossentropy',metrics=['accuracy'])
model.fit(train_asnp,Y_asnp,epochs=100,validation_split=0.2,batch_size=1)
```

```
Epoch 4/100
7473/7473 [==============================] - 8s 1ms/step - loss: 0.4357 - accuracy: 0.7989 - val_loss: 0.6340 - val_accuracy: 0.6319
Epoch 5/100
7473/7473 [==============================] - 8s 1ms/step - loss: 0.4195 - accuracy: 0.8071 - val_loss: 0.6366 - val_accuracy: 0.6303
Epoch 6/100
7473/7473 [==============================] - 8s 1ms/step - loss: 0.4064 - accuracy: 0.8093 - val_loss: 0.6145 - val_accuracy: 0.6447
Epoch 7/100
7473/7473 [==============================] - 9s 1ms/step - loss: 0.4036 - accuracy: 0.8118 - val_loss: 0.5932 - val_accuracy: 0.6699
Epoch 8/100
7473/7473 [==============================] - 8s 1ms/step - loss: 0.3933 - accuracy: 0.8225 - val_loss: 0.5671 - val_accuracy: 0.6950
Epoch 9/100
7473/7473 [==============================] - 8s 1ms/step - loss: 0.3849 - accuracy: 0.8283 - val_loss: 0.5543 - val_accuracy: 0.7132
Epoch 10/100
7473/7473 [==============================] - 8s 1ms/step - loss: 0.3949 - accuracy: 0.8276 - val_loss: 0.5418 - val_accuracy: 0.7250
Epoch 11/100
7473/7473 [==============================] - 8s 1ms/step - loss: 0.3741 - accuracy: 0.8340 - val_loss: 0.5411 - val_accuracy: 0.7245
Epoch 12/100
7473/7473 [==============================] - 8s 1ms/step - loss: 0.3794 - accuracy: 0.8332 - val_loss: 0.5176 - val_accuracy: 0.7437
Epoch 13/100
7473/7473 [==============================] - 8s 1ms/step - loss: 0.3810 - accuracy: 0.8365 - val_loss: 0.5445 - val_accuracy: 0.7266
```

Metrics(Recall, Precision,F1,Accuracy) and Confusion Matrix:

```
#Y_hat=model.predict(test_asnp)
f1_nn=f1_score(y_true=Y_asnp_test,y_pred=y_pred,pos_label=True)
precision_nn=precision_score(y_true=Y_asnp_test,y_pred=y_pred,pos_label=True)
recall_nn=recall_score(y_true=Y_asnp_test,y_pred=y_pred,pos_label=True)
print(str(recall_nn)+" "+str(precision_nn)+" "+str(f1_nn)+" "+str(eva[1]))
confusion_matrix_nn=confusion_matrix(y_true=Y_asnp_test,y_pred=y_pred)
print(confusion_matrix_nn)
```

```
0.9289780428300353 0.8630067992948879 0.8947780678851174 0.8587451577186584
[[1473  544]
 [ 262 3427]]
```

Code

Loading and splitting the data:

```
def load_split_data():
  df=pd.read_csv("/content/drive/MyDrive/dataset/magic04.csv")
  seed=0
  test_data=df.sample(frac=0.3,random_state=seed)
  train_data=df.drop(test_data.index)
  return train_data,test_data
```

Undersampling:

```
def sample_together(n, X, y):
    rows = random.sample(np.arange(0,len(X.index)).tolist(),n)
    return X.iloc[rows,], y.iloc[rows,]
def undersample(train_data, under = 'h'):
    y=train_data.pop('class')
    X=train_data
    y_min = y[y == under]
    y_max = y[y != under]
    X_min = X.filter(y_min.index,axis = 0)
    X_max = X.filter(y_max.index,axis = 0)

    X_under, y_under = sample_together(len(y_min.index), X_max, y_max)

    X = pd.concat([X_under, X_min])
    y = pd.concat([y_under, y_min])
    y_train=np.array(y)
    y_train=y_train.reshape((y_train.shape[0],1))
    train_data=np.concatenate((X,y_train),axis=1)
    train_data=pd.DataFrame(train_data)
    train_data.columns=['fLength', 'fWidth', 'fSize', 'fConc', 'fConc1', 'fAsym', 'fM3Long','fM3Trans', 'fAlpha', 'fDist', 'class']
    return train_data
```

Code

KNN:

```
def get_knns():
  scores_list=[]
  for i in range(1,25,2):
    knn=KNeighborsClassifier(n_neighbors=i)
    knn.fit(train_data,Y_train)
    scores=cross_val_score(knn,train_data,Y_train,cv=5)
    average_score=sum(scores)/len(scores)
    scores_list.append(average_score)
  return scores_list
```

```
def get_best_knn(scores_list):
  n=np.argmax(scores_list)
  knn_best=KNeighborsClassifier(n_neighbors=(2*n)+1)
  knn_best.fit(train_data,Y_train)
  return knn_best
```

```
Y_hat=knn_best.predict(test_data)
accuracy_knn=sum(Y_hat==Y_test)/len(Y_hat)
f1_knn=f1_score(y_true=Y_test,y_pred=Y_hat,pos_label='g')
precision_knn=precision_score(y_true=Y_test,y_pred=Y_hat,pos_label='g')
recall_knn=recall_score(y_true=Y_test,y_pred=Y_hat,pos_label='g')
confusion_matrix_knn=confusion_matrix(y_true=Y_test,y_pred=Y_hat)
print(str(recall_knn)+" "+str(precision_knn)+" "+str(f1_knn)+" "+str(accuracy_knn))
print(confusion_matrix_knn)
```

Naïve Bayes:

```
nb=GaussianNB()
nb.fit(train_data,Y_train)
Y_hat=nb.predict(test_data)
accuracy_nb=sum(Y_hat==Y_test)/len(Y_hat)
f1_nb=f1_score(y_true=Y_test,y_pred=Y_hat,pos_label='g')
precision_nb=precision_score(y_true=Y_test,y_pred=Y_hat,pos_label='g')
recall_nb=recall_score(y_true=Y_test,y_pred=Y_hat,pos_label='g')
print(str(recall_nb)+" "+str(precision_nb)+" "+str(f1_nb)+" "+str(accuracy_nb))
confusion_matrix_nb=confusion_matrix(y_true=Y_test,y_pred=Y_hat)
print(confusion_matrix_nb)
```

```
0.8991596638655462 0.730777704340163 0.8062712688381137 0.7206449351559762
[[3317  372]
 [1222  795]]
```

Random Forests:

```
def get_best_rf():
  scores_list=[]
  for i in range(1,50,5):
    rf=RandomForestClassifier(n_estimators=i)
    rf.fit(train_data,Y_train)
    scores=cross_val_score(rf,train_data,Y_train,cv=5)
    average_score=sum(scores)/len(scores)
    scores_list.append(average_score)
  return scores_list
```

```
def get_best_rf(scores_list):
  n=np.argmax(scores_list)
  rf_best=RandomForestClassifier(n_estimators=(5*n)+1)
  rf_best.fit(train_data,Y_train)
  return rf_best
```

```
rf_best=get_best_rf(scores_list)
Y_hat=rf_best.predict(test_data)
accuracy_rf=sum(Y_hat==Y_test)/len(Y_hat)
f1_rf=f1_score(y_true=Y_test,y_pred=Y_hat,pos_label='g')
precision_rf=precision_score(y_true=Y_test,y_pred=Y_hat,pos_label='g')
recall_rf=recall_score(y_true=Y_test,y_pred=Y_hat,pos_label='g')
print(str(recall_rf)+" "+str(precision_rf)+" "+str(f1_rf)+" "+str(accuracy_rf))
confusion_matrix_rf=confusion_matrix(y_true=Y_test,y_pred=Y_hat)
print(confusion_matrix_rf)
```

```
0.8641908376253727 0.9026047565118913 0.8829801966486637 0.8519102698913424
[[3188  501]
 [ 344 1673]]
```

Code

AdaBoost:

```
scores_list=[]
for i in range(1,50,5):
  ada=AdaBoostClassifier(n_estimators=i)
  ada.fit(train_data,Y_train)
  scores=cross_val_score(rf,train_data,Y_train,cv=5)
  average_score=sum(scores)/len(scores)
  scores_list.append(average_score)
```

```
] def get_best_ada(scores_list):
   n=np.argmax(scores_list)
   ada_best=RandomForestClassifier(n_estimators=(5*n)+1)
   ada_best.fit(train_data,Y_train)
   return ada_best
```

```
] ada_best=get_best_ada(scores_list)
  Y_hat=ada_best.predict(test_data)
  accuracy_ada=sum(Y_hat==Y_test)/len(Y_hat)
  f1_ada=f1_score(y_true=Y_test,y_pred=Y_hat,pos_label='g')
  precision_ada=precision_score(y_true=Y_test,y_pred=Y_hat,pos_label='g')
  recall_ada=recall_score(y_true=Y_test,y_pred=Y_hat,pos_label='g')
  print(str(recall_ada)+" "+str(precision_ada)+" "+str(f1_ada)+" "+str(accuracy_ada))
  confusion_matrix_ada=confusion_matrix(y_true=Y_test,y_pred=Y_hat)
  print(confusion_matrix_ada)
```

```
0.8712388181078883 0.9028089887640449 0.8867429990343495 0.8561163687346652
[[3214  475]
 [ 346 1671]]
```

Code

Neural Network:

```
normalizer=Normalization()
#train_asnp=train_data.values
normalizer.adapt(train_asnp)
model=Sequential([normalizer,
                  Dense(8,'relu'),
                  Dense(8,'relu'),
                  Dense(1,'sigmoid')
                  ])
optimizer=Adam(learning_rate=0.0001)
model.compile(optimizer=optimizer,loss='binary_crossentropy',metrics=['accuracy'])
model.fit(train_asnp,Y_asnp,epochs=100,validation_split=0.2,batch_size=1)
```

```
Epoch 4/100
7473/7473 [==============================] - 8s 1ms/step - loss: 0.4357 - accuracy: 0.7989 - val_loss: 0.6340 - val_accuracy: 0.6319
Epoch 5/100
7473/7473 [==============================] - 8s 1ms/step - loss: 0.4195 - accuracy: 0.8071 - val_loss: 0.6366 - val_accuracy: 0.6303
Epoch 6/100
7473/7473 [==============================] - 8s 1ms/step - loss: 0.4064 - accuracy: 0.8093 - val_loss: 0.6145 - val_accuracy: 0.6447
Epoch 7/100
7473/7473 [==============================] - 9s 1ms/step - loss: 0.4036 - accuracy: 0.8118 - val_loss: 0.5932 - val_accuracy: 0.6699
Epoch 8/100
7473/7473 [==============================] - 8s 1ms/step - loss: 0.3933 - accuracy: 0.8225 - val_loss: 0.5671 - val_accuracy: 0.6950
Epoch 9/100
7473/7473 [==============================] - 8s 1ms/step - loss: 0.3849 - accuracy: 0.8283 - val_loss: 0.5543 - val_accuracy: 0.7132
Epoch 10/100
7473/7473 [==============================] - 8s 1ms/step - loss: 0.3949 - accuracy: 0.8276 - val_loss: 0.5418 - val_accuracy: 0.7250
Epoch 11/100
7473/7473 [==============================] - 8s 1ms/step - loss: 0.3741 - accuracy: 0.8340 - val_loss: 0.5411 - val_accuracy: 0.7245
Epoch 12/100
7473/7473 [==============================] - 8s 1ms/step - loss: 0.3794 - accuracy: 0.8332 - val_loss: 0.5176 - val_accuracy: 0.7437
Epoch 13/100
7473/7473 [==============================] - 8s 1ms/step - loss: 0.3810 - accuracy: 0.8365 - val_loss: 0.5445 - val_accuracy: 0.7266
```

```
#Y_hat=model.predict(test_asnp)
f1_nn=f1_score(y_true=Y_asnp_test,y_pred=y_pred,pos_label=True)
precision_nn=precision_score(y_true=Y_asnp_test,y_pred=y_pred,pos_label=True)
recall_nn=recall_score(y_true=Y_asnp_test,y_pred=y_pred,pos_label=True)
print(str(recall_nn)+" "+str(precision_nn)+" "+str(f1_nn)+" "+str(eva[1]))
confusion_matrix_nn=confusion_matrix(y_true=Y_asnp_test,y_pred=y_pred)
print(confusion_matrix_nn)
```

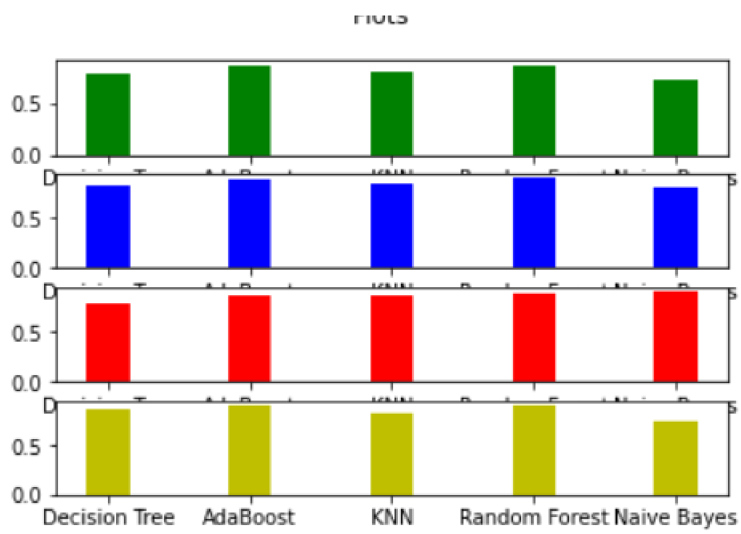
```
0.9289780428300353 0.8630067992948879 0.8947780678851174 0.8587451577186584
[[1473  544]
 [ 262 3427]]
```

Decision Tree:

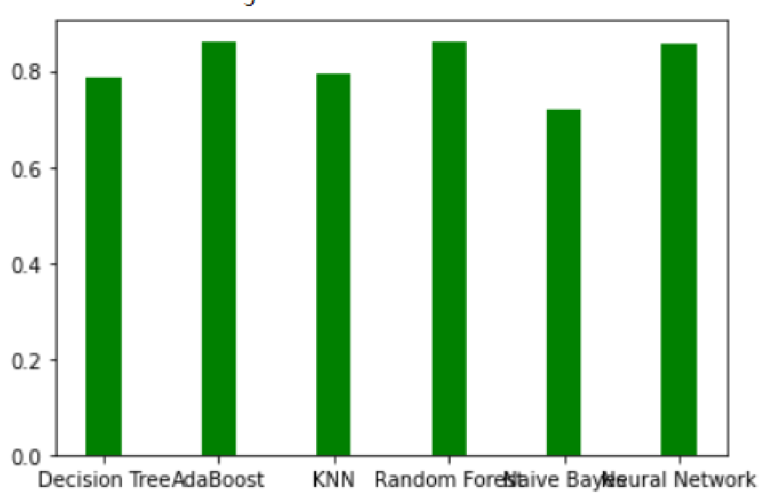
```
dt=DecisionTreeClassifier(criterion='gini')
dt.fit(train_data,Y_train)
Y_hat=dt.predict(test_data)
accuracy_dt=sum(Y_hat==Y_test)/len(Y_hat)
f1_dt=f1_score(y_true=Y_test,y_pred=Y_hat,pos_label='g')
precision_dt=precision_score(y_true=Y_test,y_pred=Y_hat,pos_label='g')
recall_dt=recall_score(y_true=Y_test,y_pred=Y_hat,pos_label='g')
print(str(recall_dt)+" "+str(precision_dt)+" "+str(f1_dt)+" "+str(accuracy_dt))
confusion_matrix_dt=confusion_matrix(y_true=Y_test,y_pred=Y_hat)
print(confusion_matrix_dt)
```

Comparisons

The following plots shows the comparisons of the different classifiers used, the first plot using the different metrics and the second plot using the accuracy after adding the neural network:



<BarContainer object of 6 artists>



Comparisons

As we can see in the above plots the dominant classifiers are AdaBoost, Random Forest and the Neural network in all the metrics as they avoid the overfitting more than the other classifiers and as they are eager learners.

The Neural Network dominated the other classifiers in the f1 score, recall and precision, but AdaBoost achieved the highest accuracy.