HR Analytics Project Documentation

Introduction :

This project is based on a hypothetical dataset downloaded from IBM HR Analytics Employee Attrition & Performance. It has 1,470 data points (rows) and 35 features (columns) describing each employee's background and characteristics; and labelled (supervised learning) with whether they are still in the company or whether they have gone to work somewhere else. Machine Learning models can help to understand and determine how these factors relate to workforce attrition.

- Data Pre-processing Done

The purpose of preprocessing is to convert raw data into a form that fits machine learning. Structured and clean data allows a data scientist to get more precise results from an applied machine learning model. The technique includes data formatting, cleaning, And sampling.

- Model/s Development and Evaluation

I have used the below models for classification:

1. LogisticRegression
2. SVC
3. RandomForestClassifier
4. AdaBoostClassifier
5. DecisionTreeClassifier

- Identification of possible problem-solving approaches (methods)

- Read the data (from csv)
- Identify the dependent and independent variables.
- Check if the data has missing values or the data is categorical or not.
- If yes, apply basic data preprocessing operations to bring the data in a go to go format.
- Now split the data into the groups of training and testing for the respective purpose.
- After splitting data, fit it to a most suitable model. (How to find a suitable model is answered below)
- Validate the model. If satisfactory, then go with it, else tune the parameters and keep testing. In a few cases, you can also try different algorithms for the same problem to understand the difference between the accuracies.
- From step 7 one can also learn about accuracy paradox.
- Visualize the data.

- Testing of Identified Approaches (Algorithms)

1. Naive Bayes
2. Cross validation
3. Confusion matrix
4. Accuracy score
5. Classification report

- Run and Evaluate selected models

I have used the below models for classification:

DecisionTreeClassifier:

```
dt=DecisionTreeClassifier()
dt.fit(x_train,y_train)
pred=dt.predict(x_test)
dt_accu=accuracy_score(y_test,pred)

print(accuracy_score(y_test,pred))
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

```
0.7716049382716049
[[350  50]
 [ 61  25]]
              precision    recall  f1-score   support

           0       0.85      0.88      0.86       400
           1       0.33      0.29      0.31        86

    accuracy                           0.77       486
   macro avg       0.59      0.58      0.59       486
weighted avg       0.76      0.77      0.77       486
```

KNeighborsClassifier:

```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier()
knn.fit(x_train,y_train)
knn_pred=knn.predict(x_test)
knn_accu=accuracy_score(y_test,pred)

print(accuracy_score(y_test,pred))
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

```
0.7716049382716049
[[350  50]
 [ 61  25]]
              precision    recall  f1-score   support

           0       0.85      0.88      0.86       400
           1       0.33      0.29      0.31        86

    accuracy                           0.77       486
   macro avg       0.59      0.58      0.59       486
weighted avg       0.76      0.77      0.77       486
```

Random Forest Classifier:

```
rf=RandomForestClassifier()
rf.fit(x_train,y_train)
pred=rf.predict(x_test)
rf_accu=accuracy_score(y_test,pred)
print(accuracy_score(y_test,pred))
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

```
0.8353909465020576
[[394   6]
 [ 74  12]]
              precision    recall  f1-score   support

           0       0.84      0.98      0.91       400
           1       0.67      0.14      0.23        86

    accuracy                           0.84       486
   macro avg       0.75      0.56      0.57       486
weighted avg       0.81      0.84      0.79       486
```

SVC:

```
svc=SVC()
svc.fit(x_train,y_train)
pred=svc.predict(x_test)
svc_accu=accuracy_score(y_test,pred)
print(accuracy_score(y_test,pred))
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

```
0.823045267489712
[[400   0]
 [ 86   0]]
              precision    recall  f1-score   support

           0       0.82      1.00      0.90       400
           1       0.00      0.00      0.00        86

    accuracy                           0.82       486
   macro avg       0.41      0.50      0.45       486
weighted avg       0.68      0.82      0.74       486
```

AdaBoost Classifier:

```
ad=AdaBoostClassifier()
ad.fit(x_train,y_train)
pred=ad.predict(x_test)
ad_accu=accuracy_score(y_test,pred)
print(accuracy_score(y_test,pred))
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

```
0.8395061728395061
[[381  19]
 [ 59  27]]
              precision    recall  f1-score   support

           0       0.87      0.95      0.91       400
           1       0.59      0.31      0.41        86

    accuracy                           0.84       486
   macro avg       0.73      0.63      0.66       486
weighted avg       0.82      0.84      0.82       486
```

GaussianNB:

```
gnb=GaussianNB()
gnb.fit(x_train,y_train)
pred=gnb.predict(x_test)
gnb_accu=accuracy_score(y_test,pred)
print(accuracy_score(y_test,pred))
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

```
0.7489711934156379
[[317  83]
 [ 39  47]]
              precision    recall  f1-score   support

           0       0.89      0.79      0.84       400
           1       0.36      0.55      0.44        86

    accuracy                           0.75       486
   macro avg       0.63      0.67      0.64       486
weighted avg       0.80      0.75      0.77       486
```

MultinomialNB:

```
from sklearn.naive_bayes import MultinomialNB
mnb=MultinomialNB()
mnb.fit(x_train,y_train)
pred=mnb.predict(x_test)
mnb_accu=accuracy_score(y_test,pred)
print(accuracy_score(y_test,pred))
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

```
0.5473251028806584
[[223 177]
 [ 43  43]]
              precision    recall  f1-score   support

           0       0.84      0.56      0.67       400
           1       0.20      0.50      0.28        86

    accuracy                           0.55       486
   macro avg       0.52      0.53      0.48       486
weighted avg       0.72      0.55      0.60       486
```

CONCLUSION:

AdaBoost Classifier has emerged as the final winning model with **F1-score 91%** and highest **Recall 95%**. This could be the highest possible score achieved with the inherent limitations in the dataset.

AdaBoost Classifier has accuracy is **83%.**

Hence, AdaBost Classifier has the best model of this dataset.