KNN,SVM & RANDOM FOREST

Final Project Project

Course: CS-634

Name: Anusha Syed

Email: as445@njit.edu.

Libraries used:

numpy, random and pandas for reading files and display of data in data frame.

- Imported KNeighborsClassifier from sklearn.neighbors
- Imported StandardScaler , MinMaxScaler from sklearn.preprocessing
- Imported train_test_split from sklearn.model_selection
- Imported classification_report , confusion_matrix, accuracy_score, f1_score from sklearn.metrics
- Imported LabelEncoder, StandardScaler from sklearn.preprocessing
- Imported RandomForestClassifier from sklearn.ensemble
- Imported StratifiedKFold, RandomizedSearchCV from sklearn.model_selection

➔ Used all these libraries for classification according to given model

```
# Importing the important libraries

import numpy as np
import random
import pandas as pd

from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler , MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report , confusion_matrix, accuracy_score, f1_score

from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import StratifiedKFold, RandomizedSearchCV


import warnings
warnings.filterwarnings('ignore')
```

File (dataset) Input:

Here, I have loaded my data set using pandas library and displayed it.

➔ df.shape tells how many rows and columns are there

➔ df.isnull().sum() tells if there is any null value in data and if it is then given the sum of the number of items that has null values

➔ df.describe() shows the actual data stored in your csv file

```python
# Loading the dataset

df = pd.read_csv('heart.csv')
df.head()
```

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
0	40	M	ATA	140	289	0	Normal	172	N	0.0	Up	0
1	49	F	NAP	160	180	0	Normal	156	N	1.0	Flat	1
2	37	M	ATA	130	283	0	ST	98	N	0.0	Up	0
3	48	F	ASY	138	214	0	Normal	108	Y	1.5	Flat	1
4	54	M	NAP	150	195	0	Normal	122	N	0.0	Up	0


```python
print(df.shape)
print(df["HeartDisease"].value_counts())
```

```
(918, 12)
1    508
0    410
Name: HeartDisease, dtype: int64
```

[53]
```python
df.isnull().sum()
```

```
Age               0
Sex               0
ChestPainType     0
RestingBP         0
Cholesterol       0
FastingBS         0
RestingECG        0
MaxHR             0
ExerciseAngina    0
Oldpeak           0
ST_Slope          0
HeartDisease      0
dtype: int64
```

[54]
```python
df.describe()
```

	Age	RestingBP	Cholesterol	FastingBS	MaxHR	Oldpeak	HeartDisease
count	918.000000	918.000000	918.000000	918.000000	918.000000	918.000000	918.000000
mean	53.510893	132.396514	198.799564	0.233115	136.809368	0.887364	0.553377
std	9.432617	18.514154	109.384145	0.423046	25.460334	1.066570	0.497414
min	28.000000	0.000000	0.000000	0.000000	60.000000	-2.600000	0.000000
25%	47.000000	120.000000	173.250000	0.000000	120.000000	0.000000	0.000000
50%	54.000000	130.000000	223.000000	0.000000	138.000000	0.600000	1.000000
75%	60.000000	140.000000	267.000000	0.000000	156.000000	1.500000	1.000000
max	77.000000	200.000000	603.000000	1.000000	202.000000	6.200000	1.000000

```
[55] df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 918 entries, 0 to 917
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Age             918 non-null    int64
 1   Sex             918 non-null    object
 2   ChestPainType   918 non-null    object
 3   RestingBP       918 non-null    int64
 4   Cholesterol     918 non-null    int64
 5   FastingBS       918 non-null    int64
 6   RestingECG      918 non-null    object
 7   MaxHR           918 non-null    int64
 8   ExerciseAngina  918 non-null    object
 9   Oldpeak         918 non-null    float64
 10  ST_Slope        918 non-null    object
 11  HeartDisease    918 non-null    int64
dtypes: float64(1), int64(6), object(5)
memory usage: 86.2+ KB
```

Separating the dependent and independent columns

Here, X is my training data and Y is my label.

```
[56] # Seperating the dependent and the independent columns

    X = df.drop("HeartDisease" , axis = 1)
    y = df['HeartDisease']
```

Separating the numerical and the categorical columns

Here, cat_cols brings the categorical data like those that don't have any numerical value and num_cols prints those labels that has numerical values to it.

```
[58] print(cat_cols , num_cols)

    ['Sex', 'ChestPainType', 'RestingECG', 'ExerciseAngina', 'ST_Slope'] ['Age', 'RestingBP', 'Cholesterol', 'FastingBS', 'MaxHR', 'Oldpeak']
```

```
# Seperating the numerical and the categorical columns

cat_cols = list(df.select_dtypes(include=['object']).columns)
num_cols = []

for i in list(X.columns):
  if i not in cat_cols:
    num_cols.append(i)
```

Getting mean and percentile of data

```
normal_cols = ["Age","MaxHR","RestingBP"]
for col in normal_cols:
    mean = np.mean(df[col])
    std = np.std(df[col])
    lower_range = mean - (3*std)
    upper_range = mean + (3*std)
    df[col] = np.where(((df[col] < lower_range) | (df[col] > upper_range))
                        ,random.randint(int(lower_range),int(upper_range)),df[col])
```

```
[60] IQR = np.percentile(df["Cholesterol"],75) - np.percentile(df["Cholesterol"],25)
     lower_bound = np.percentile(df["Cholesterol"],25) - 1.5 * IQR
     upper_bound = np.percentile(df["Cholesterol"],75) + 1.5 * IQR
     median_cholesterol = np.median(df["Cholesterol"])

     df["Cholesterol"] = np.where(((df["Cholesterol"] > upper_bound) | (df["Cholesterol"] < lower_bound))
                                 ,random.randint(int(np.percentile(df["Cholesterol"],25)),
                                                int(np.percentile(df["Cholesterol"],75))),df["Cholesterol"])
```

Separating the test and training data set

Here, test_size=0.2 means that 20% is the testing data and rest 80% is the training data.

```
[61] # Seperating the train and test dataset
     x_train,x_test,y_train,y_test = train_test_split(X,y,stratify = y , random_state=42,test_size=0.2)
```

Standardadising the features of training and testing dataset

```
[63] # Standardadising the features of training and testing dataset

     scaler = StandardScaler()
     scaler.fit(X[num_cols])
     X[num_cols] = scaler.transform(X[num_cols])
```

```
[64] X = pd.get_dummies(data = X , drop_first=True)
```

Converted categorical data also into numerical data and now again separating the test and training data

```
# Seperating the train and test dataset
x_train,x_test,y_train,y_test = train_test_split(X,y,stratify = y , random_state=42,test_size=0.2)
```

```
x_train[:3]
```

	Age	RestingBP	Cholesterol	FastingBS	MaxHR	Oldpeak	Sex_M	ChestPainType_ATA	ChestPainType_NAP	ChestPainType_TA	RestingECG_
485	1.006537	0.356867	0.166481	1.813758	-0.346192	0.293283	1	1	0	0	
486	0.157954	-1.210356	0.139040	1.813758	1.697314	-0.457194	1	1	0	0	
117	0.582246	-0.129513	1.273277	1.813758	-0.267596	0.574711	0	0	0	0	

KNN Algorithm:

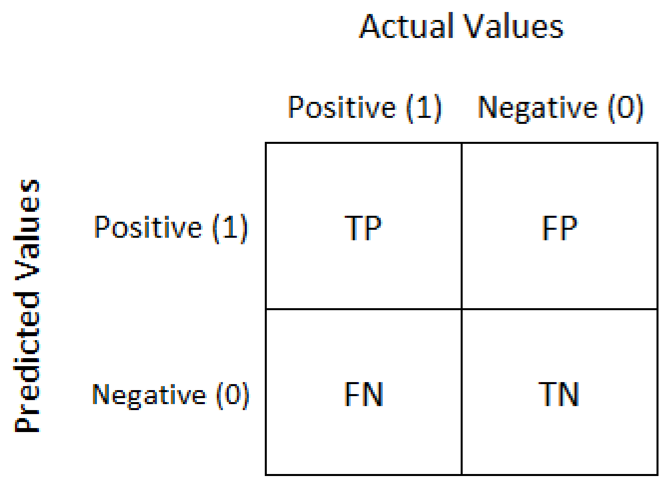
KNN Algorithm

```
[ ] knn = KNeighborsClassifier(n_neighbors=7)
    knn.fit(x_train,y_train)
    y_pred_knn = knn.predict(x_test)

    print(knn.score(x_test,y_test))
    print(classification_report(y_test,y_pred_knn))
    print(confusion_matrix(y_test,y_pred_knn))
```

```
0.6956521739130435
              precision    recall  f1-score   support

           0       0.67      0.62      0.65        82
           1       0.71      0.75      0.73       102

    accuracy                           0.70       184
   macro avg       0.69      0.69      0.69       184
weighted avg       0.69      0.70      0.69       184

[[51 31]
 [25 77]]
```

I have used the built in functions extracting from library for this algorithm to set up the training data and labels and printing out the report that shows performance metrics. It also prints the accuracy of the algorithm and prints out the confusion matrix.

Refer to this diagram to interpret results for confusion matrix:



Therefore, 51 is the TP, 31 is FP, 25 is FN and 77 is TN.

SVM Algorithm:

SVM Algorithm

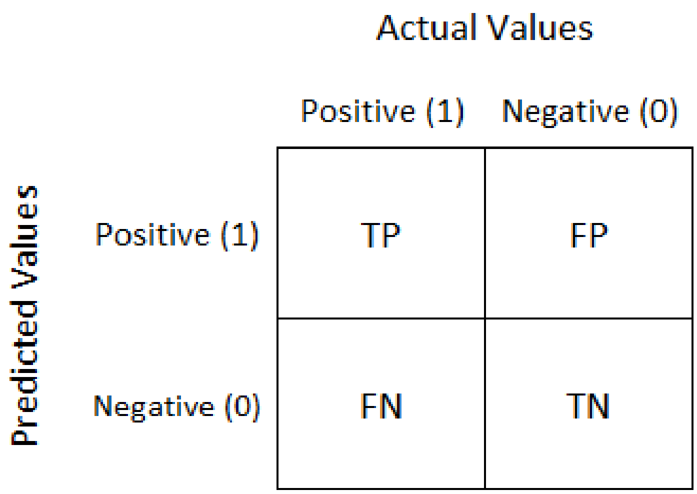
```
from sklearn.svm import SVC
classifier = SVC(kernel='rbf', random_state=27)
classifier.fit(x_train, y_train)
y_pred_svm = classifier.predict(x_test)
```

```
[20] print(classifier.score(x_test,y_test))
     print(classification_report(y_test, y_pred_svm))
     print(confusion_matrix(y_test,y_pred_svm))
```

```
0.717391304347826
              precision    recall  f1-score   support

           0       0.68      0.70      0.69        82
           1       0.75      0.74      0.74       102

    accuracy                           0.72       184
   macro avg       0.71      0.72      0.71       184
weighted avg       0.72      0.72      0.72       184

[[57 25]
 [27 75]]
```

I have used the built in functions extracting from library for this algorithm to set up the training data and labels and printing out the report that shows performance metrics. It also prints the accuracy of the algorithm and prints out the confusion matrix.

Refer to this diagram to interpret results for confusion matrix:



Therefore, 57 is the TP, 25 is FP, 27 is FN and 75 is TN.

Random Forest Algorithm:

Random Forest Algorithm

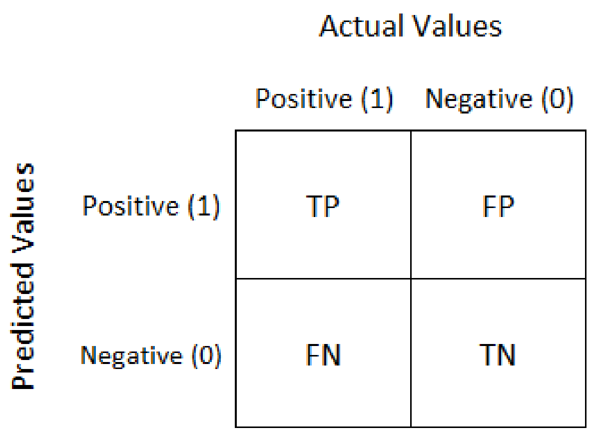
```
[21] randomforest = RandomForestClassifier()
     randomforest.fit(x_train,y_train)
     y_pred_rf = randomforest.predict(x_test)
```

```
[22] print('Train Accuracy %s' % round(accuracy_score(y_test, y_pred_rf),2))
     print('Train F1-score %s' % f1_score(y_test, y_pred_rf, average=None))
     print(classification_report(y_test, y_pred_rf))
     print("Confusion Matrix:\n",confusion_matrix(y_test, y_pred_rf))
```

```
Train Accuracy 0.88
Train F1-score [0.86419753 0.89320388]
              precision    recall  f1-score   support

           0       0.88      0.85      0.86        82
           1       0.88      0.90      0.89       102

    accuracy                           0.88       184
   macro avg       0.88      0.88      0.88       184
weighted avg       0.88      0.88      0.88       184

Confusion Matrix:
 [[70 12]
 [10 92]]
```

I have used the built in functions extracting from library for this algorithm to set up the training data and labels and printing out the report that shows performance metrics. It also prints the accuracy of the algorithm and prints out the confusion matrix.

Refer to this diagram to interpret results for confusion matrix:



Therefore, 70 is the TP, 12 is FP, 10 is FN and 92 is TN.

LSTM Algorithm:

LSTM - Long Short Term Memory

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers.core import Dense,Activation,Dropout
from keras.layers import LSTM
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier
from sklearn import svm
from xgboost import XGBClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier

from keras.callbacks import EarlyStopping
import math
```

```
X_train,Y_train,X_test,Y_test=train_test_split(X,y,test_size=0.5,random_state=0)
```

[65]
```
print(X_train.shape)
print(Y_train.shape)
print(X_test.shape)
print(Y_test.shape)
```

```
(459, 11)
(459, 11)
(459,)
(459,)
```

[66]
```
print(X_train[:1])
```

```
    Age  Sex  ChestPainType  ...  ExerciseAngina  Oldpeak  ST_Slope
46   37    1              0  ...               0      0.0         2

[1 rows x 11 columns]
```

```
X_train=np.expand_dims(X_train, axis=2)
Y_train=np.expand_dims(Y_train, axis=2)
es=EarlyStopping(patience=7)
model=Sequential()
model.add(LSTM(1,input_shape=(11,1)))
model.add(Dense(1,activation='softmax'))
model.add(Activation('sigmoid'))
model.compile(loss='binary_crossentropy',optimizer='Adam',metrics=['accuracy'])
model.fit(X_train,X_test,epochs=10,batch_size=1,verbose=1,callbacks=[es])
predict = model.predict(Y_train)
c = model.evaluate(Y_train,Y_test)
```

```
Epoch 1/10
449/459 [===========================>.] - ETA: 0s - loss: 0.7721 - accuracy: 0.5412WARNING:tensorflow:Early stopping condi
459/459 [==============================] - 3s 3ms/step - loss: 0.7686 - accuracy: 0.5447
Epoch 2/10
459/459 [==============================] - ETA: 0s - loss: 0.7686 - accuracy: 0.5447WARNING:tensorflow:Early stopping condi
459/459 [==============================] - 2s 3ms/step - loss: 0.7686 - accuracy: 0.5447
Epoch 3/10
446/459 [===========================>.] - ETA: 0s - loss: 0.7662 - accuracy: 0.5471WARNING:tensorflow:Early stopping condi
459/459 [==============================] - 1s 3ms/step - loss: 0.7686 - accuracy: 0.5447
Epoch 4/10
452/459 [===========================>.] - ETA: 0s - loss: 0.7690 - accuracy: 0.5442WARNING:tensorflow:Early stopping condi
459/459 [==============================] - 1s 3ms/step - loss: 0.7686 - accuracy: 0.5447
Epoch 5/10
456/459 [===========================>.] - ETA: 0s - loss: 0.7672 - accuracy: 0.5461WARNING:tensorflow:Early stopping condi
459/459 [==============================] - 1s 3ms/step - loss: 0.7686 - accuracy: 0.5447
Epoch 6/10
452/459 [===========================>.] - ETA: 0s - loss: 0.7668 - accuracy: 0.5465WARNING:tensorflow:Early stopping condi
459/459 [==============================] - 1s 3ms/step - loss: 0.7686 - accuracy: 0.5447
Epoch 7/10
450/459 [===========================>.] - ETA: 0s - loss: 0.7666 - accuracy: 0.5467WARNING:tensorflow:Early stopping condi
459/459 [==============================] - 1s 3ms/step - loss: 0.7686 - accuracy: 0.5447
Epoch 8/10
459/459 [==============================] - ETA: 0s - loss: 0.7686 - accuracy: 0.5447WARNING:tensorflow:Early stopping condi
459/459 [==============================] - 1s 3ms/step - loss: 0.7686 - accuracy: 0.5447
Epoch 9/10
452/459 [===========================>.] - ETA: 0s - loss: 0.7646 - accuracy: 0.5487WARNING:tensorflow:Early stopping condi
459/459 [==============================] - 1s 3ms/step - loss: 0.7686 - accuracy: 0.5447
Epoch 10/10
```



```
Epoch 10/10
446/459 [===========================>.] - ETA: 0s - loss: 0.7639 - accuracy: 0.5493WARNING:tensorflow:Early stopping con
459/459 [==============================] - 1s 3ms/step - loss: 0.7686 - accuracy: 0.5447
15/15 [==============================] - 0s 2ms/step - loss: 0.7512 - accuracy: 0.5621
```

```
[ ] len(predict)

459
```

I could not create accuracy for LSTM because confusion matrix is created for classification algorithms whereas LSTM is a regression algorithm.

10-folds execution Method:

Declaring list for accuracy, tss, precision:

```
knn_acc = []
knn_tss = []
knn_prec = []
knn_tn = []
knn_tp = []
knn_fn = []
knn_fp = []

svm_acc = []
svm_tss = []
svm_prec = []
svm_tn = []
svm_tp = []
svm_fn = []
svm_fp = []

randmf_acc = []
randmf_tss = []
randmf_prec = []
randmf_tn = []
randmf_tp = []
randmf_fn = []
randmf_fp = []
```

```
for i in range(0,11):

  x_train,x_test,y_train,y_test = train_test_split(X,y,stratify = y , random_state=42,test_size=0.3)

  ## Running KNN Algorithm
  knn.fit(x_train,y_train)
  y_pred_knn = knn.predict(x_test)

  tn, fp, fn, tp = confusion_matrix(y_test, y_pred_knn).ravel()
  knn_tn.append(tn)
  knn_tp.append(fp)
  knn_fn.append(fn)
  knn_fp.append(tp)

  acck = (tp + tn) / (tn + fp + fn + tp)
  tss = (tp / (tp - fn)) - (fp / (fp + tn))
  precision = tp / (tp + fp)

  knn_acc.append(acck)
  knn_tss.append(tss)
  knn_prec.append(precision)
```

```
## Running Random Forest Algorithm
randomforest.fit(x_train,y_train)
y_pred_rf = randomforest.predict(x_test)

tn, fp, fn, tp = confusion_matrix(y_test, y_pred_rf).ravel()
randmf_tn.append(tn)
randmf_tp.append(fp)
randmf_fn.append(fn)
randmf_fp.append(tp)

accrf = (tp + tn) / (tn + fp + fn + tp)
tss = (tp / (tp - fn)) - (fp / (fp + tn))
precision = tp / (tp + fp)

randmf_acc.append(accrf)
randmf_tss.append(tss)
randmf_prec.append(precision)
```

```
## Running SVM Algorithm
classifier.fit(x_train, y_train)
y_pred_svm = classifier.predict(x_test)

tn, fp, fn, tp = confusion_matrix(y_test, y_pred_svm).ravel()
svm_tn.append(tn)
svm_tp.append(fp)
svm_fn.append(fn)
svm_fp.append(tp)

accs = (tp + tn) / (tn + fp + fn + tp)
tss = (tp / (tp - fn)) - (fp / (fp + tn))
precision = tp / (tp + fp)

svm_acc.append(accs)
svm_tss.append(tss)
svm_prec.append(precision)
```

Average:

```
## Average:
avg_knn_acc = sum(knn_acc) / len(knn_acc)
avg_knn_tss = sum(knn_tss) / len(knn_tss)
avg_knn_prec = sum(knn_prec) / len(knn_prec)
avg_knn_tn = sum(knn_tn) / len(knn_tn)
avg_knn_tp = sum(knn_tp) / len(knn_tp)
avg_knn_fn = sum(knn_fn) / len(knn_fn)
avg_knn_fp = sum(knn_fp) / len(knn_fp)
```

```
[75] ## Average:
avg_svm_acc = sum(svm_acc) / len(svm_acc)
avg_svm_tss = sum(svm_tss) / len(svm_tss)
avg_svm_prec = sum(svm_prec) / len(svm_prec)
avg_svm_tn = sum(svm_tn) / len(svm_tn)
avg_svm_tp = sum(svm_tp) / len(svm_tp)
avg_svm_fn = sum(svm_fn) / len(svm_fn)
avg_svm_fp = sum(svm_fp) / len(svm_fp)
```

```
## Average:
avg_randmf_acc = sum(randmf_acc) / len(randmf_acc)
avg_randmf_tss = sum(randmf_tss) / len(randmf_tss)
avg_randmf_prec = sum(randmf_prec) / len(randmf_prec)
avg_randmf_tn = sum(randmf_tn) / len(randmf_tn)
avg_randmf_tp = sum(randmf_tp) / len(randmf_tp)
avg_randmf_fn = sum(randmf_fn) / len(randmf_fn)
avg_randmf_fp = sum(randmf_fp) / len(randmf_fp)
```

```
table_cols = {'Algorithm' : ['KNN' , 'SVM' , 'Random Forest'] , 'TP' : [avg_knn_tp , avg_svm_tp , avg_randmf_tp] , 'FP' : [a
```

```
[77] 'FN' : [avg_knn_fn, avg_svm_fn, avg_randmf_fn] , 'TN' : [avg_knn_tn, avg_svm_tn, avg_randmf_tn] , 'ACC' : [avg_knn_acc, avg
```

```
:c], 'TSS' : [avg_knn_tss , avg_svm_tss, avg_randmf_tss] , 'Precision' : [avg_knn_prec, avg_svm_prec, avg_randmf_prec]}
```

Table:

This is the final table that shows accuracy of algorithm after running it ten times.

```
[34] table_cols1 = pd.DataFrame.from_dict(table_cols)
```

```
table_cols1
```

	Algorithm	TP	FP	FN	TN	ACC	TSS	Precision
0	KNN	38.000000	112.000000	41.000000	85.000000	0.713768	1.268522	0.746667
1	SVM	34.000000	113.000000	40.000000	89.000000	0.731884	1.271522	0.768707
2	Random Forest	15.545455	140.727273	12.272727	107.454545	0.899209	0.969340	0.900618

Result:

Which algorithm performs better and why? Justify your answer

- ➔ Accuracy for KNN was coming 70 and after 10 folds it is coming as 74.6
- ➔ Accuracy for SVM was coming 72 and after 10 folds it is coming as 76.8
- ➔ Accuracy for Random Forest was coming 88 and after 10 folds it is coming as 89.9

Random Forest algorithm performs better and you can see that after training the algorithm 10 times the accuracy of it is better. It chooses randomly during the training process and is not dependent solely on any specific set of features. Therefore, this randomized feature selection makes this algorithm more accurate.

Also, this also depends upon the data.

LSTM's accuracy is not listed as confusion matrix cannot be created on regression type algorithm. It's meaningless.