APPENDIX

IMPORT LIBRARY

```
1. from google.colab import files
2. import os
3. import numpy as np
4. import pandas as pd
5. from sklearn.model_selection import train_test_split, cross_val_score,
   Kfold
6. from google.colab import drive
7. import keras
8. import tensorflow as tf
9. from keras.models import Sequential
10. from keras.layers import Dense
11. import xgboost as xgb
12. from bayes_opt import BayesianOptimization
```

SET SEED TO 42

```
13. seed = 42
```

MOUNT DRIVE AND IMPORT DATASET

```
14. drive.mount('/content/drive')
15. !ls "/content/drive/My Drive/TA"
16. data = pd.read_csv('/content/drive/My Drive/TA/Dataset/diabetes.csv')
```

DATASET SHAPE

```
17. data.shape
```

DATASET INFORMATION

```
18. data.info()
```

PRINT TOP 10 OF THE DATASET

```
19. data.head(10)
```

LABEL ENCODING

```
20. labels = data.columns[1:]
21. for i in labels:
22.             data[i].loc[data[i].isin(['Yes'])] = 1
23.             data[i].loc[data[i].isin(['No'])] = 0
24.             data[i].loc[data[i].isin(['Positive'])] = 1
25.             data[i].loc[data[i].isin(['Negative'])] = 0
26.             data[i].loc[data[i].isin(['Male'])] = 1
27.             data[i].loc[data[i].isin(['Female'])] = 0
28.             data[i] = data[i].astype(int)
```

FUNCTION TO CALCULATE THE CONFUSION MATRIX

```
29. def cm(target, prediction):
30.     TP = 0
31.     TN = 0
32.     FP = 0
33.     FN = 0
34.     j = 0
35.     for i in enumerate(target):
36.         if i[1] == 1:
37.             if prediction[j] == 1:
38.                 TP += 1
39.             else:
40.                 FN += 1
41.         elif i[1] == 0:
42.             if prediction[j] == 1:
43.                 FP += 1
44.             else:
45.                 TN += 1
46.         j += 1
47.     return TP, TN, FP, FN
```

FUNCTION TO CALCULATE THE RESULT FOR ANALYSIS

```
48. def result(target, prediction):
49.     TP, TN, FP, FN = cm( target, prediction)
50.     accuracy = (TP+TN)/(TP+TN+FP+FN)
51.     recall = TP/(TP+FN)
52.     precision = TP/(TP+FP)
53.     f1 = 2*((precision*recall)/(precision+recall))
54.     return accuracy, recall, precision, f1
```

FUNCTION FOR SPLIT THE DATASET INTO TRAINING AND TEST SET

```
55. def split(testsize, X, Y):
56.     print('Train set = ', (100-testsize),'% , Test set = ', testsize,
    '%')
57.     testsize = testsize/100
58.     X_train,  X_test,  Y_train,  Y_test  =  train_test_split(X,  Y,
    test_size=testsize, random_state=seed)
59.     return X_train, X_test, Y_train, Y_test
```

FUNCTION TO CALL THE FIRST ANN MODEL AND GET THE RESULT

```
60. def ann_model1(X_train, Y_train, X_test, Y_test):
61.     tf.keras.utils.set_random_seed(seed)
62.     ann1 = Sequential()
63.     ann1.add(Dense(units=     200,     kernel_initializer='uniform',
    activation = 'relu', input_dim=16))
64.     ann1.add(Dense(units= 200, kernel_initializer='uniform', activation
    = 'relu'))
65.     ann1.add(Dense(units= 150, kernel_initializer='uniform', activation
    = 'relu'))
66.     ann1.add(Dense(units = 1, kernel_initializer='uniform', activation
    = 'sigmoid')) #output
67.
```

```
    opt = keras.optimizers.Adam(learning_rate=0.001)
    ann1.compile(optimizer = opt, loss = 'binary_crossentropy', metrics = ['accuracy'])

    ann1.fit(X_train, Y_train, batch_size = 500, epochs = 500, verbose=0)
    ann_pred_test = ann1.predict(X_test)
    ann_pred_test=ann_pred_test.round()

    ann_accuracy, ann_recall, ann_precision, ann_f1 = result(Y_test, ann_pred_test)
    return ann_accuracy, ann_recall, ann_precision, ann_f1
```

FUNCTION TO CALL THE SECOND ANN MODEL AND GET THE RESULT

```
def ann_model2(X_train, Y_train, X_test, Y_test):
    tf.keras.utils.set_random_seed(seed)
    ann2 = Sequential()
    ann2.add(Dense(units= 200, kernel_initializer='uniform', activation = 'relu', input_dim=16))
    ann2.add(Dense(units= 200, kernel_initializer='uniform', activation = 'relu'))
    ann2.add(Dense(units= 200, kernel_initializer='uniform', activation = 'relu'))
    ann2.add(Dense(units= 150, kernel_initializer='uniform', activation = 'relu'))
    ann2.add(Dense(units = 1, kernel_initializer='uniform', activation = 'sigmoid')) #output

    opt = keras.optimizers.Adam(learning_rate=0.001)
    ann2.compile(optimizer = opt, loss = 'binary_crossentropy', metrics = ['accuracy'])

    ann2.fit(X_train, Y_train, batch_size = 500, epochs = 500, verbose=0)
    ann_pred_test2 = ann2.predict(X_test)
    ann_pred_test2=ann_pred_test2.round()
    ann_accuracy2, ann_recall2, ann_precision2, ann_f12 = result(Y_test, ann_pred_test2)
    return ann_accuracy2, ann_recall2, ann_precision2, ann_f12
```

FUNCTION TO CALL THE THIRD ANN MODEL AND GET THE RESULT

```
def ann_model3(X_train, Y_train, X_test, Y_test):
    tf.keras.utils.set_random_seed(seed)
    ann3 = Sequential()
    ann3.add(Dense(units= 200, kernel_initializer='uniform', activation = 'relu', input_dim=16))
    ann3.add(Dense(units= 200, kernel_initializer='uniform', activation = 'relu'))
    ann3.add(Dense(units= 150, kernel_initializer='uniform', activation = 'relu'))
    ann3.add(Dense(units= 150, kernel_initializer='uniform', activation = 'relu'))
    ann3.add(Dense(units= 150, kernel_initializer='uniform', activation = 'relu'))
```

```
    ann3.add(Dense(units = 1, kernel_initializer='uniform', activation
  = 'sigmoid')) #output

    opt = keras.optimizers.Adam(learning_rate=0.001)
    ann3.compile(optimizer = opt, loss = 'binary_crossentropy', metrics
  = ['accuracy'])

    ann3.fit(X_train, Y_train, batch_size = 500, epochs = 500,
  verbose=0)
    ann_pred_test3 = ann3.predict(X_test)
    ann_pred_test3=ann_pred_test3.round()
    ann_accuracy3, ann_recall3, ann_precision3, ann_f13 = result(Y_test,
  ann_pred_test3)
    return ann_accuracy3, ann_recall3, ann_precision3, ann_f13
```

FUNCTION TO CALL THE FOURTH ANN MODEL AND GET THE RESULT

```
def ann_model4(X_train, Y_train, X_test, Y_test):
    tf.keras.utils.set_random_seed(seed)
    ann4 = Sequential()
    ann4.add(Dense(units=      200,      kernel_initializer='uniform',
  activation = 'relu', input_dim=16))
    ann4.add(Dense(units= 200, kernel_initializer='uniform', activation
  = 'relu'))
    ann4.add(Dense(units= 200, kernel_initializer='uniform', activation
  = 'relu'))
    ann4.add(Dense(units= 150, kernel_initializer='uniform', activation
  = 'relu'))
    ann4.add(Dense(units= 150, kernel_initializer='uniform', activation
  = 'relu'))
    ann4.add(Dense(units= 150, kernel_initializer='uniform', activation
  = 'relu'))
    ann4.add(Dense(units = 1, kernel_initializer='uniform', activation
  = 'sigmoid')) #output

    opt = keras.optimizers.Adam(learning_rate=0.001)
    ann4.compile(optimizer = opt, loss = 'binary_crossentropy', metrics
  = ['accuracy'])

    ann4.fit(X_train, Y_train, batch_size = 500, epochs = 500,
  verbose=0)
    ann_pred_test4 = ann4.predict(X_test)
    ann_pred_test4=ann_pred_test4.round()
    ann_accuracy4, ann_recall4, ann_precision4, ann_f14 = result(Y_test,
  ann_pred_test4)
    return ann_accuracy4, ann_recall4, ann_precision4, ann_f14
```

FUNCTION TO CALL AND GET THE RESULT FROM THE FIRST XGBOOST MODEL WITH THE DEFAULT HYPERPARAMATERS

```
def xgb_model1(X_train, Y_train, X_test, Y_test):
    xgb_clf   =   xgb.XGBClassifier(seed=seed,   max_depth=6,   gamma=0,
  n_estimators=100, learning_rate=0.3, colsample_bytree=1)
    xgb_model = xgb_clf.fit(X_train, Y_train)
    xgb_pred = xgb_model.predict(X_test)
```

```
135.    xgb_accuracy, xgb_recall, xgb_precision, xgb_f1 = result(Y_test,
   xgb_pred)
136.    return xgb_accuracy, xgb_recall, xgb_precision, xgb_f1
```

FUNTION TO DEFINE THE ESTIMATOR FOR THE BAYESIAN OPTIMIZATIONS

```
137.def    xgb_tune(max_depth,    gamma,    n_estimators    ,learning_rate,
   min_child_weight,  colsample_bytree):
```

SET THE 10 FOLDS VALIDATION WITH FIXED RANDOM STATE

```
138.    cv = KFold(n_splits=10, random_state=seed, shuffle=True)
```

SET THE ESTIMATOR FOR CROSS VALIDATION WITH THE HYPER PARAMETERS

```
139.    estimators = xgb.XGBClassifier(seed=seed, max_depth=int(max_depth),
   gamma=gamma,                              n_estimators=int(n_estimators),
   learning_rate=learning_rate, colsample_bytree=colsample_bytree)
```

CROSS VALIDATE THE ESTIMATOR THEN RETURN THE MEAN

```
140.    result = cross_val_score(estimators, X, Y, cv=cv, scoring='roc_auc')
141.    result = result.mean()
142.    return result
```

FUNCTION TO CALL THE SECOND XGBOOST MODEL WITH NEW PARAMETERS AND GET THE RESULT

```
143.def xgb_model2(X_train, Y_train, X_test, Y_test, parameters):
144.    clf    =    xgb.XGBClassifier(seed=seed,    max_depth=6,    gamma=0,
   n_estimators=100,  learning_rate=0.3,  colsample_bytree=1).fit(X_train,
   Y_train)
145.    clf    =    xgb.XGBClassifier(**parameters,seed=seed).fit(X_train,
   Y_train)
146.    xgb_pred2 = clf.predict(X_test)
147.    xgb_accuracy, xgb_recall, xgb_precision, xgb_f1 = result(Y_test,
   xgb_pred2)
148.    return xgb_accuracy, xgb_recall, xgb_precision, xgb_f1
```

CALL THE FUNCTION TO SPLIT INTO TRAINING AND TEST SET WITH DESIRED RATIO

```
149.X_train, X_test, Y_train, Y_test = split(20, X, Y)
150.X_train2, X_test2, Y_train2, Y_test2 = split(25, X, Y)
151.X_train3, X_test3, Y_train3, Y_test3 = split(30, X, Y)
152.X_train4, X_test4, Y_train4, Y_test4 = split(40, X, Y)
153.X_train5, X_test5, Y_train5, Y_test5 = split(50, X, Y)
```

CALL THE FIRST ANN MODEL AND TRAIN IT WITH THE TRAINING AND TEST SETS AND SAVE THE RESULT INTO VARIABLES

```
154.ann_accuracy1_1,    ann_recall1_1,    ann_precision1_1,    ann_f11_1    =
   ann_model1(X_train, Y_train, X_test, Y_test)
155.ann_accuracy1_2,    ann_recall1_2,    ann_precision1_2,    ann_f11_2    =
   ann_model1(X_train2, Y_train2, X_test2, Y_test2)
156.ann_accuracy1_3,    ann_recall1_3,    ann_precision1_3,    ann_f11_3    =
   ann_model1(X_train3, Y_train3, X_test3, Y_test3)
157.ann_accuracy1_4,    ann_recall1_4,    ann_precision1_4,    ann_f11_4    =
   ann_model1(X_train4, Y_train4, X_test4, Y_test4)
158.ann_accuracy1_5,    ann_recall1_5,    ann_precision1_5,    ann_f11_5    =
   ann_model1(X_train5, Y_train5, X_test5, Y_test5)
```

CALL THE SECOND ANN MODEL AND TRAIN IT WITH THE TRAINING AND TEST SETS AND SAVE THE RESULT INTO VARIABLES

```
159.ann_accuracy2_1,    ann_recall2_1,    ann_precision2_1,    ann_f12_1    =
   ann_model2(X_train, Y_train, X_test, Y_test)
160.ann_accuracy2_2,    ann_recall2_2,    ann_precision2_2,    ann_f12_2    =
   ann_model2(X_train2, Y_train2, X_test2, Y_test2)
161.ann_accuracy2_3,    ann_recall2_3,    ann_precision2_3,    ann_f12_3    =
   ann_model2(X_train3, Y_train3, X_test3, Y_test3)
162.ann_accuracy2_4,    ann_recall2_4,    ann_precision2_4,    ann_f12_4    =
   ann_model2(X_train4, Y_train4, X_test4, Y_test4)
163.ann_accuracy2_5,    ann_recall2_5,    ann_precision2_5,    ann_f12_5    =
   ann_model2(X_train5, Y_train5, X_test5, Y_test5)
```

CALL THE THIRD ANN MODEL AND TRAIN IT WITH THE TRAINING AND TEST SETS AND SAVE THE RESULT INTO VARIABLES

```
164.ann_accuracy3_1,    ann_recall3_1,    ann_precision3_1,    ann_f13_1    =
   ann_model3(X_train, Y_train, X_test, Y_test)
165.ann_accuracy3_2,    ann_recall3_2,    ann_precision3_2,    ann_f13_2    =
   ann_model3(X_train2, Y_train2, X_test2, Y_test2)
166.ann_accuracy3_3,    ann_recall3_3,    ann_precision3_3,    ann_f13_3    =
   ann_model3(X_train3, Y_train3, X_test3, Y_test3)
167.ann_accuracy3_4,    ann_recall3_4,    ann_precision3_4,    ann_f13_4    =
   ann_model3(X_train4, Y_train4, X_test4, Y_test4)
168.ann_accuracy3_5,    ann_recall3_5,    ann_precision3_5,    ann_f13_5    =
   ann_model3(X_train5, Y_train5, X_test5, Y_test5)
```

CALL THE FOURTH ANN MODEL AND TRAIN IT WITH THE TRAINING AND TEST SETS AND SAVE THE RESULT INTO VARIABLES

```
169.ann_accuracy4_1,    ann_recall4_1,    ann_precision4_1,    ann_f14_1    =
   ann_model4(X_train, Y_train, X_test, Y_test)
170.ann_accuracy4_2,    ann_recall4_2,    ann_precision4_2,    ann_f14_2    =
   ann_model4(X_train2, Y_train2, X_test2, Y_test2)
171.ann_accuracy4_3,    ann_recall4_3,    ann_precision4_3,    ann_f14_3    =
   ann_model4(X_train3, Y_train3, X_test3, Y_test3)
172.ann_accuracy4_4,    ann_recall4_4,    ann_precision4_4,    ann_f14_4    =
   ann_model4(X_train4, Y_train4, X_test4, Y_test4)
```

```
ann_accuracy4_5,    ann_recall4_5,    ann_precision4_5,    ann_f14_5    =
   ann_model4(X_train5, Y_train5, X_test5, Y_test5)
```

RUN THE BAYESIAN SEARCH OPTIMIZATION TO GET THE OPTIMAL HYPERPARAMETERS

```
tune = BayesianOptimization(xgb_tune, {'max_depth': (3, 11),
                                'gamma': (1e-09, 0.5),
                                'n_estimators': (100, 250),
                                'learning_rate': (0.01,1.0),
                                'min_child_weight' : (1, 10),
                                'colsample_bytree' : (0.1, 0.8),
                            }, random_state=seed)
tune.maximize()
```

GET THE OPTIMAL HYPERPAREMETERS WITH THE MAX SCORE

```
parameters = tune.max['params']
```

ROUND THE MAX DEPTH, NUMBER OF ESTIMATORS, MINIMUM WEIGHT OF CHILD AND PRINT THE RESULT

```
parameters['max_depth']= round(parameters['max_depth'])
parameters['n_estimators']= round(parameters['n_estimators'])
parameters['min_child_weight']= round(parameters['min_child_weight'])
print("Best params :")
print(parameters)
```

CALL THE FIRST XGBOOST MODEL AND TRAIN IT WITH THE TRAINING AND TEST SETS AND SAVE THE RESULT INTO VARIABLES

```
xgb_accuracy1_1,    xgb_recall1_1,    xgb_precision1_1,    xgb_f11_1    =
   xgb_model1(X_train, Y_train, X_test, Y_test)
xgb_accuracy1_2,    xgb_recall1_2,    xgb_precision1_2,    xgb_f11_2    =
   xgb_model1(X_train2, Y_train2, X_test2, Y_test2)
xgb_accuracy1_3,    xgb_recall1_3,    xgb_precision1_3,    xgb_f11_3    =
   xgb_model1(X_train3, Y_train3, X_test3, Y_test3)
xgb_accuracy1_4,    xgb_recall1_4,    xgb_precision1_4,    xgb_f11_4    =
   xgb_model1(X_train4, Y_train4, X_test4, Y_test4)
xgb_accuracy1_5,    xgb_recall1_5,    xgb_precision1_5,    xgb_f11_5    =
   xgb_model1(X_train5, Y_train5, X_test5, Y_test5)
```

CALL THE SECOND XGBOOST MODEL AND TRAIN IT WITH THE NEW PARAMETERS. TRAINING AND TEST SETS AND SAVE THE RESULT INTO VARIABLES

```
xgb_accuracy2_1,    xgb_recall2_1,    xgb_precision2_1,    xgb_f12_1    =
   xgb_model2(X_train, Y_train, X_test, Y_test, parameters)
xgb_accuracy2_2,    xgb_recall2_2,    xgb_precision2_2,    xgb_f12_2    =
   xgb_model2(X_train2, Y_train2, X_test2, Y_test2, parameters)
xgb_accuracy2_3,    xgb_recall2_3,    xgb_precision2_3,    xgb_f12_3    =
   xgb_model2(X_train3, Y_train3, X_test3, Y_test3, parameters)
```

```
196.xgb_accuracy2_4,   xgb_recall2_4,   xgb_precision2_4,   xgb_f12_4   =
   xgb_model2(X_train4, Y_train4, X_test4, Y_test4, parameters)
197.xgb_accuracy2_5,   xgb_recall2_5,   xgb_precision2_5,   xgb_f12_5   =
   xgb_model2(X_train5, Y_train5, X_test5, Y_test5, parameters)
```

PRINT RESULT FOR 80:20 RATIO

```
198.print("Accuracy")
199.print("ANN1   :   ",   round(ann_accuracy1_1,3),   "   ||   ANN2   :   ",
   round(ann_accuracy2_1,3), " || ANN3 : ", round(ann_accuracy3_1,3) , " ||
   ANN4 : ", round(ann_accuracy4_1,3)  ," || XGBoost without tuning : ",
   round(xgb_accuracy1_1,3),"    ||    XGBoost    with    tuning    :    ",
   round(xgb_accuracy2_1,3))
200.print("Recall")
201.print("ANN1   :   ",   round(ann_recall1_1,3),   "   ||   ANN2   :   ",
   round(ann_recall2_1,3), " || ANN3 : ", round(ann_recall3_1,3)   , " ||
   ANN4  :  ",  round(ann_recall4_1,3)," ||  XGBoost  without  tuning  :  ",
   round(xgb_recall1_1,3),"    ||    XGBoost    with    tuning    :    ",
   round(xgb_recall2_1,3))
202.print("Precision")
203.print("ANN1   :   ",   round(ann_precision1_1,3),   "   ||   ANN2   :   ",
   round(ann_precision2_1,3), " || ANN3 : ", round(ann_precision3_1,3)   , "
   || ANN4 : ", round(ann_precision4_1,3)," || XGBoost without tuning : ",
   round(xgb_precision1_1,3),"    ||    XGBoost    with    tuning    :    ",
   round(xgb_precision2_1,3))
204.print("F1 Score")
205.print("ANN1 : ", round(ann_f11_1,3), " || ANN2 : ", round(ann_f12_1,3),
   " || ANN3 : ", round(ann_f13_1,3)  , " || ANN4 : ", round(ann_f14_1,3),"
   || XGBoost without tuning : ", round(xgb_f11_1,3)," || XGBoost with tuning
   : ", round(xgb_f12_1,3))
```

PRINT RESULT FOR 75:25 RATIO

```
206.print("Accuracy")
207.print("ANN1   :   ",   round(ann_accuracy1_2,3),   "   ||   ANN2   :   ",
   round(ann_accuracy2_2,3), " || ANN3 : ", round(ann_accuracy3_2,3) , " ||
   ANN4 : ", round(ann_accuracy4_2,3)  ," || XGBoost without tuning : ",
   round(xgb_accuracy1_2,3),"    ||    XGBoost    with    tuning    :    ",
   round(xgb_accuracy2_2,3))
208.print("Recall")
209.print("ANN1   :   ",   round(ann_recall1_2,3),   "   ||   ANN2   :   ",
   round(ann_recall2_2,3), " || ANN3 : ", round(ann_recall3_2,3), " || ANN4
   :  ",  round(ann_recall4_2,3)      ," ||  XGBoost  without  tuning  :  ",
   round(xgb_recall1_2,3),"    ||    XGBoost    with    tuning    :    ",
   round(xgb_recall2_2,3))
210.print("Precision")
211.print("ANN1   :   ",   round(ann_precision1_2,3),   "   ||   ANN2   :   ",
   round(ann_precision2_2,3), " || ANN3 : ", round(ann_precision3_2,3), " ||
   ANN4 : ", round(ann_precision4_2,3)    ," || XGBoost without tuning : ",
   round(xgb_precision1_2,3),"    ||    XGBoost    with    tuning    :    ",
   round(xgb_precision2_2,3))
212.print("F1 Score")
213.print("ANN1 : ", round(ann_f11_2,3), " || ANN2 : ", round(ann_f12_2,3),
   " || ANN3 : ", round(ann_f13_2,3), " || ANN4 : ", round(ann_f14_2,3)   ,"
   || XGBoost without tuning : ", round(xgb_f11_2,3)," || XGBoost with tuning
   : ", round(xgb_f12_2,3))
```

PRINT RESULT FOR 70:30 RATIO

```
print("Accuracy")
print("ANN1 : ", round(ann_accuracy1_3,3), " || ANN2 : ", round(ann_accuracy2_3,3), " || ANN3 : ", round(ann_accuracy3_3,3), " || ANN4 : ", round(ann_accuracy4_3,3) ," || XGBoost without tuning : ", round(xgb_accuracy1_3,3)," || XGBoost with tuning : ", round(xgb_accuracy2_3,3))
print("Recall")
print("ANN1 : ", round(ann_recall1_3,3), " || ANN2 : ", round(ann_recall2_3,3), " || ANN3 : ", round(ann_recall3_3,3) , " || ANN4 : ", round(ann_recall4_3,3) ," || XGBoost without tuning : ", round(xgb_recall1_3,3)," || XGBoost with tuning : ", round(xgb_recall2_3,3))
print("Precision")
print("ANN1 : ", round(ann_precision1_3,3), " || ANN2 : ", round(ann_precision2_3,3), " || ANN3 : ", round(ann_precision3_3,3), " || ANN4 : ", round(ann_precision4_3,3) ," || XGBoost without tuning : ", round(xgb_precision1_3,3)," || XGBoost with tuning : ", round(xgb_precision2_3,3))
print("F1 Score")
print("ANN1 : ", round(ann_f11_3,3), " || ANN2 : ", round(ann_f12_3,3), " || ANN3 : ", round(ann_f13_3,3), " || ANN4 : ", round(ann_f14_3,3) ," || XGBoost without tuning : ", round(xgb_f11_3,3)," || XGBoost with tuning : ", round(xgb_f12_3,3))
```

PRINT RESULT FOR 60:40 RATIO

```
print("Accuracy")
print("ANN1 : ", round(ann_accuracy1_4,3), " || ANN2 : ", round(ann_accuracy2_4,3), " || ANN3 : ", round(ann_accuracy3_4,3), " || ANN4 : ", round(ann_accuracy4_4,3) ," || XGBoost without tuning : ", round(xgb_accuracy1_4,3)," || XGBoost with tuning : ", round(xgb_accuracy2_4,3))
print("Recall")
print("ANN1 : ", round(ann_recall1_4,3), " || ANN2 : ", round(ann_recall2_4,3), " || ANN3 : ", round(ann_recall3_4,3), " || ANN4 : ", round(ann_recall4_4,3) ," || XGBoost without tuning : ", round(xgb_recall1_4,3)," || XGBoost with tuning : ", round(xgb_recall2_4,3))
print("Precision")
print("ANN1 : ", round(ann_precision1_4,3), " || ANN2 : ", round(ann_precision2_4,3), " || ANN3 : ", round(ann_precision3_4,3), " || ANN4 : ", round(ann_precision4_4,3) ," || XGBoost without tuning : ", round(xgb_precision1_4,3)," || XGBoost with tuning : ", round(xgb_precision2_4,3))
print("F1 Score")
print("ANN1 : ", round(ann_f11_4,3), " || ANN2 : ", round(ann_f12_4,3), " || ANN3 : ", round(ann_f13_4,3), " || ANN4 : ", round(ann_f14_4,3) ," || XGBoost without tuning : ", round(xgb_f11_4,3)," || XGBoost with tuning : ", round(xgb_f12_4,3))
```

PRINT RESULT FOR 50:50 RATIO

```
print("Accuracy")
print("ANN1 : ", round(ann_accuracy1_5,3), " || ANN2 : ", round(ann_accuracy2_5,3), " || ANN3 : ", round(ann_accuracy3_5,3), " || ANN4 : ", round(ann_accuracy4_5,3) ," || XGBoost without tuning : ", round(xgb_accuracy1_5,3)," || XGBoost with tuning : ", round(xgb_accuracy2_5,3))
print("Recall")
print("ANN1 : ", round(ann_recall1_5,3), " || ANN2 : ", round(ann_recall2_5,3), " || ANN3 : ", round(ann_recall3_5,3), " || ANN4 : ", round(ann_recall4_5,3) ," || XGBoost without tuning : ", round(xgb_recall1_5,3)," || XGBoost with tuning : ", round(xgb_recall2_5,3))
print("Precision")
print("ANN1 : ", round(ann_precision1_5,3), " || ANN2 : ", round(ann_precision2_5,3), " || ANN3 : ", round(ann_precision3_5,3), " || ANN4 : ", round(ann_precision4_5,3) ," || XGBoost without tuning : ", round(xgb_precision1_5,3)," || XGBoost with tuning : ", round(xgb_precision2_5,3))
print("F1 Score")
print("ANN1 : ", round(ann_f11_5,3), " || ANN2 : ", round(ann_f12_5,3), " || ANN3 : ", round(ann_f13_5,3), " || ANN4 : ", round(ann_f14_5,3) ," || XGBoost without tuning : ", round(xgb_f11_5,3)," || XGBoost with tuning : ", round(xgb_f12_5,3))
```

PAPER NAME

TA 19.K1.0017.docx

WORD COUNT

8589 Words

CHARACTER COUNT

45825 Characters

PAGE COUNT

32 Pages

FILE SIZE

384.4KB

SUBMISSION DATE

Jan 9, 2023 10:52 AM GMT+7

REPORT DATE

Jan 9, 2023 10:53 AM GMT+7

15% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

- 9% Internet database
- 8% Publications database
- Crossref database
- Crossref Posted Content database
- 9% Submitted Works database

Excluded from Similarity Report

- Bibliographic material
- Quoted material
- Cited material
- Small Matches (Less then 10 words)