CS 634 Data Mining - Fall 2021

Professor Yasser Abduallah

Final term Project Report (Option 1)

Supervised Data Mining (Classification) And Evaluations

Nishi Rani (nr36@njit.edu)

NJIT ID : 31524700

Contents:

Option 1:

2 Machine Learning Algorithms selected for supervised classification were :

a) Naïve Bayes
b) Random Forest

1 Deep learning algorithm selected:

c) LSTM (Long Short Term Memory)

Data was taken from https://archive.ics.uci.edu/ml/datasets/Wine
For Tensorflow installation I used :
!pip install tensorflow in jupyter notebook.

Data was available in .data format I read the binary file using below python command as:

Code:

```
file_handler=open('wine.data','rb')
lines_array =  file_handler.readlines()
for l in lines_array:
   print(l)
```

Sample output:

```
b'1,14.23,1.71,2.43,15.6,127,2.8,3.06,.28,2.29,5.64,1.04,3.92,1065\n'
b'1,13.2,1.78,2.14,11.2,100,2.65,2.76,.26,1.28,4.38,1.05,3.4,1050\n'
b'1,13.16,2.36,2.67,18.6,101,2.8,3.24,.3,2.81,5.68,1.03,3.17,1185\n'
b'1,14.37,1.95,2.5,16.8,113,3.85,3.49,.24,2.18,7.8,.86,3.45,1480\n'
b'1,13.24,2.59,2.87,21,118,2.8,2.69,.39,1.82,4.32,1.04,2.93,735\n'
```

Saved the data file as .csv using notepad++ to easily access the rows and columns using pandas dataframe. Moreover Wine.data had 3 labels for which multi-label classification/ confusion matrix was required which was bit too complex to handle for each fold of cross validation, So with the suggestion from professor, I took the data for only 2 labels i.e. to go ahead with the binary classification.

data=pd.read_csv('wine.csv',header=None)
df=data.sample(frac=1) # To shuffle the data I have used the sample function.
print(df.head)

```
<bound method NDFrame.head of      0      1     2     3     4    5     6     7     8     9     10    11  \
54   1  13.74  1.67  2.25  16.4  118  2.60  2.90  0.21  1.62  5.85  0.92
100  2  12.08  2.08  1.70  17.5   97  2.23  2.17  0.26  1.40  3.30  1.27
104  2  12.51  1.73  1.98  20.5   85  2.20  1.92  0.32  1.48  2.94  1.04
52   1  13.82  1.75  2.42  14.0  111  3.88  3.74  0.32  1.87  7.05  1.01
108  2  12.22  1.29  1.94  19.0   92  2.36  2.04  0.39  2.08  2.70  0.86
..  ..    ...   ...   ...   ...  ...   ...   ...   ...   ...   ...   ...
47   1  13.90  1.68  2.12  16.0  101  3.10  3.39  0.21  2.14  6.10  0.91
55   1  13.56  1.73  2.46  20.5  116  2.96  2.78  0.20  2.45  6.25  0.98
25   1  13.05  2.05  3.22  25.0  124  2.63  2.68  0.47  1.92  3.58  1.13
61   2  12.64  1.36  2.02  16.8  100  2.02  1.41  0.53  0.62  5.75  0.98
18   1  14.19  1.59  2.48  16.5  108  3.30  3.93  0.32  1.86  8.70  1.23

       12    13
54   3.20  1060
100  2.96   710
104  3.57   672
52   3.26  1190
108  3.02   312
..    ...   ...
47   3.33   985
```

Next, separate the features and labels, There are overall 14 columns, first column represents the quality of wine i.e. “Label” with index 0. And rest of the columns are “features” which helps in predicting the label i.e. index 1 to index 13.

labels=df.iloc[:,0]

features=df.iloc[:,1:14]

X=features

y=labels

```
X:           1     2     3      4     5     6     7     8     9     10    11    12  \
105  12.42  2.55  2.27  22.0   90  1.68  1.84  0.66  1.42  2.70  0.86  3.30
84   11.84  0.89  2.58  18.0   94  2.20  2.21  0.22  2.35  3.05  0.79  3.08
113  11.41  0.74  2.50  21.0   88  2.48  2.01  0.42  1.44  3.08  1.10  2.31
68   13.34  0.94  2.36  17.0  110  2.53  1.30  0.55  0.42  3.17  1.02  1.93
59   12.37  0.94  1.36  10.6   88  1.98  0.57  0.28  0.42  1.95  1.05  1.82
..     ...   ...   ...   ...  ...   ...   ...   ...   ...   ...   ...   ...
82   12.08  1.13  2.51  24.0   78  2.00  1.58  0.40  1.40  2.20  1.31  2.72
100  12.08  2.08  1.70  17.5   97  2.23  2.17  0.26  1.40  3.30  1.27  2.96
42   13.88  1.89  2.59  15.0  101  3.25  3.56  0.17  1.70  5.43  0.88  3.56
118  12.77  3.43  1.98  16.0   80  1.63  1.25  0.43  0.83  3.40  0.70  2.12
30   13.73  1.50  2.70  22.5  101  3.00  3.25  0.29  2.38  5.70  1.19  2.71

       13
105   315
84    520
113   434
68    750
59    520
..    ...
82    630
100   710
42   1095
118   372
30   1285

[130 rows x 13 columns]
y :  105     2
84      2
113     2
```

Next step was to import all the necessary python libraries and implementing the all three models for each fold of k-fold cross validation:

I have used k=10.

Below is the snapshot consisting of importing all the essential libraries and implementing the models.

Model 1 : Overview of Naïve Bayes for each fold of CV

I have created a list of all metrics for each of the model, Here let's see for Naïve Bayes model.

```
TP_NB=[]
FP_NB=[]
FN_NB=[]
TN_NB=[]
TPR_NB=[]
FPR_NB=[]
TNR_NB=[]
FNR_NB=[]
RECALL_NB=[]
PRECISION_NB=[]
F1_SCORE_NB=[]
ACCURACY_NB=[]
ERROR_RATE_NB=[]
BACC_NB=[]
TSS_NB=[]
HSS_NB=[]
BS_NB=[]
BSS_NB=[]
```

And then function for finding evaluation metrics is created for each of the model,

```
def evaluation_metricsNB(TP,TN,FP,FN):

    TP=TP
    TN=TN
    FP=FP
    FN=FN
    TPR=TP/(TP+FN)
    TNR=TN/(TN+FP)
    FPR=FP/(FP+TN)
    FNR=FN/(FN+TP)
    RECALL=TPR
    PRECISION=TP/(TP+FP)
    F1_SCORE=(2*TP)/(2*TP+FP+FN)
    ACCURACY=(TP+TN)/(TP+FP+TN+FN)
    ERROR_RATE=1-ACCURACY
    BACC=(TPR+TNR)/2
    TSS=TPR-FPR
    HSS=2*(TP*TN-FP*FN)/(((TP+FN)*(FN+TN))+((TP+FP)*(FP+TN)))
    sum_y=0
    for n in range(len(y_test)):
        sum_y+=(y_test[n]-y_predNB[n])**2
    BS=sum_y/len(y_test)

    y_meantemp=0
    for i in range(len(y_test)):
        y_meantemp+=y_test[i]
    ymean=y_meantemp/len(y_test)
    #BSS
    temp=0
```

```
    for i in range(len(y_test)):
        temp+=(y_test[i]-ymean)**2
    temp=temp/len(y_test)
    BSS=BS/temp

    TP_NB.append(TP)
    FP_NB.append(FP)
    FN_NB.append(FN)
    TN_NB.append(TN)
    TPR_NB.append(TPR)
    FPR_NB.append(FPR)
    TNR_NB.append(TNR)
    FNR_NB.append(FNR)
    RECALL_NB.append(RECALL)
    PRECISION_NB.append(PRECISION)
    F1_SCORE_NB.append(F1_SCORE)
    ACCURACY_NB.append(ACCURACY)
    ERROR_RATE_NB.append(ERROR_RATE)
    BACC_NB.append(BACC)
    TSS_NB.append(TSS)
    HSS_NB.append(HSS)
    BS_NB.append(BS)
    BSS_NB.append(BSS)
```

Then Naïve Bayes model was implemented using the following libraries and functions:

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

X=np.array(X)
y=np.array(y)
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix


kf = KFold(n_splits=10)

TN_TOTALNB=0
TP_TOTALNB=0
FP_TOTALNB=0
FN_TOTALNB=0
```

```
for train_index, test_index in kf.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

###########################      Model1 Naive Bayes     ###################################

    modelNB = GaussianNB()
    modelNB.fit(X_train, y_train)
    y_predNB = modelNB.predict(X_test)
    cnf_matrixNB = confusion_matrix(y_test, y_predNB)
    [[TNNB, FPNB],
    [FNNB, TPNB]]=cnf_matrixNB

    evaluation_metricsNB(TPNB,TNNB,FPNB,FNNB)

    TN_TOTALNB+=TNNB
    TP_TOTALNB+=TPNB
    FP_TOTALNB+=FPNB
    FN_TOTALNB+=FNNB
```

Then the metrics calculated by Naïve-Bayes in each fold of cross validation stored in a dataframe.

```
dfa=pd.DataFrame({
                "TP": TP_NB,
                "FP": FP_NB,
                "FN": FN_NB,
                "TN": TN_NB,
                "TPR":TPR_NB,
                "FPR":FPR_NB,
                "TNR":TNR_NB,
                "FNR":FNR_NB,
                "RECALL":RECALL_NB,
                'PRECISION':PRECISION_NB,
                'F1_SCORE':F1_SCORE_NB,
                'Accuracy':ACCURACY_NB,
                'Error rate':ERROR_RATE_NB,
                'BACC':BACC_NB,
                'TSS':TSS_NB,
                'HSS':HSS_NB,
                'BS' :BS_NB,
                'BSS':BSS_NB},
                index=['Naive Bayes','Naive Bayes','Naive Bayes','Naive Bayes','Naive Bayes'
```

Model 2: Overview of Random Forest for each fold of CV

I have created a list of all metrics for each of the model, Here let's see for Random Forest model.

```
TP_RF=[]
FP_RF=[]
FN_RF=[]
TN_RF=[]
TPR_RF=[]
FPR_RF=[]
TNR_RF=[]
FNR_RF=[]
RECALL_RF=[]
PRECISION_RF=[]
F1_SCORE_RF=[]
ACCURACY_RF=[]
ERROR_RATE_RF=[]
BACC_RF=[]
TSS_RF=[]
HSS_RF=[]
BS_RF=[]
BSS_RF=[]
```

And then function for finding evaluation metrics is created for each of th e model,

```
def evaluation_metricsRF(TP,TN,FP,FN):

    TP=TP
    TN=TN
    FP=FP
    FN=FN
    TPR=TP/(TP+FN)
    TNR=TN/(TN+FP)
    FPR=FP/(FP+TN)
    FNR=FN/(FN+TP)
    RECALL=TPR
    PRECISION=TP/(TP+FP)
    F1_SCORE=(2*TP)/(2*TP+FP+FN)
    ACCURACY=(TP+TN)/(TP+FP+TN+FN)
    ERROR_RATE=1-ACCURACY
    BACC=(TPR+TNR)/2
    TSS=TPR-FPR
    HSS=2*(TP*TN-FP*FN)/(((TP+FN)*(FN+TN))+((TP+FP)*(FP+TN)))
    sum_y=0
    for n in range(len(y_test)):
        sum_y+=(y_test[n]-y_predRF[n])**2
    BS=sum_y/len(y_test)

    y_meantemp=0
    for i in range(len(y_test)):
        y_meantemp+=y_test[i]
    ymean=y_meantemp/len(y_test)

    temp=0
```

```
    for i in range(len(y_test)):
        temp+=(y_test[i]-ymean)**2
    temp=temp/len(y_test)
    BSS=BS/temp

    TP_RF.append(TP)
    FP_RF.append(FP)
    FN_RF.append(FN)
    TN_RF.append(TN)
    TPR_RF.append(TPR)
    FPR_RF.append(FPR)
    TNR_RF.append(TNR)
    FNR_RF.append(FNR)
    RECALL_RF.append(RECALL)
    PRECISION_RF.append(PRECISION)
    F1_SCORE_RF.append(F1_SCORE)
    ACCURACY_RF.append(ACCURACY)
    ERROR_RATE_RF.append(ERROR_RATE)
    BACC_RF.append(BACC)
    TSS_RF.append(TSS)
    HSS_RF.append(HSS)
    BS_RF.append(BS)
    BSS_RF.append(BSS)
```

Then Random forest model was implemented using the following librari es and functions:

```
############################      Model2 Random Forest     ####################################

    rf= RandomForestClassifier(n_estimators=20, random_state=0)
    rf.fit(X_train, y_train)
    y_predRF=rf.predict(X_test)
    cnf_matrixRF = confusion_matrix(y_test, y_predRF)
    [[TNRF, FPRF],
    [FNRF, TPRF]]=cnf_matrixRF

    evaluation_metricsRF(TPRF,TNRF,FPRF,FNRF)

    TN_TOTALRF+=TNRF
    TP_TOTALRF+=TPRF
    FP_TOTALRF+=FPRF
    FN_TOTALRF+=FNRF
```

Then the metrics calculated by Random forest in each fold of cross valid ation stored in a dataframe.

```
dfb=pd.DataFrame({
                "TP": TP_RF,
                "FP": FP_RF,
                "FN": FN_RF,
                "TN": TN_RF,
                "TPR":TPR_RF,
                "FPR":FPR_RF,
                "TNR":TNR_RF,
                "FNR":FNR_RF,
                "RECALL":RECALL_RF,
                'PRECISION':PRECISION_RF,
                'F1_SCORE':F1_SCORE_RF,
                'Accuracy':ACCURACY_RF,
                'Error rate':ERROR_RATE_RF,
                'BACC':BACC_RF,
                'TSS':TSS_RF,
                'HSS':HSS_RF,
                'BS' :BS_RF,
                'BSS':BSS_RF},
                index=['Random-Forest','Random-Forest','Random-Forest','Random-Forest','Random-Forest'
```

Model 3: Overview of LSTM for each fold of CV

I have created a list of all metrics for each of the model, Here let's see for LSTM model.

```
TP_LSTM=[]
FP_LSTM=[]
FN_LSTM=[]
TN_LSTM=[]
TPR_LSTM=[]
FPR_LSTM=[]
TNR_LSTM=[]
FNR_LSTM=[]
RECALL_LSTM=[]
PRECISION_LSTM=[]
F1_SCORE_LSTM=[]
ACCURACY_LSTM=[]
ERROR_RATE_LSTM=[]
BACC_LSTM=[]
TSS_LSTM=[]
HSS_LSTM=[]
BS_LSTM=[]
BSS_LSTM=[]
```

And then function for finding evaluation metrics is created for each of th e model,

```
def evaluation_metrics_lstm(TP,TN,FP,FN):

    TP=TP
    TN=TN
    FP=FP
    FN=FN
    TPR=TP/(TP+FN)
    TNR=TN/(TN+FP)
    FPR=FP/(FP+TN)
    FNR=FN/(FN+TP)
    RECALL=TPR
    PRECISION=TP/(TP+FP)
    if math.isnan(PRECISION):
        PRECISION_LSTM.append(np.nan)

    F1_SCORE=(2*TP)/(2*TP+FP+FN)
    ACCURACY=(TP+TN)/(TP+FP+TN+FN)
    ERROR_RATE=1-ACCURACY
    BACC=(TPR+TNR)/2
    TSS=TPR-FPR
    HSS=2*(TP*TN-FP*FN)/(((TP+FN)*(FN+TN))+((TP+FP)*(FP+TN)))
    sum_y=0
    for n in range(len(y_test)):
        sum_y+=(y_test[n]-y_predLSTM[n])**2
    BS=sum_y/len(y_test)

    y_meantemp=0
    for i in range(len(y_test)):
        y_meantemp+=y_test[i]
    ymean=y_meantemp/len(y_test)

    temp=0
    for i in range(len(y_test)):
        temp+=(y_test[i]-ymean)**2
    temp=temp/len(y_test)
    BSS=BS/temp

    TP_LSTM.append(TP)
    FP_LSTM.append(FP)
    FN_LSTM.append(FN)
    TN_LSTM.append(TN)
    TPR_LSTM.append(TPR)
    FPR_LSTM.append(FPR)
    TNR_LSTM.append(TNR)
    FNR_LSTM.append(FNR)
    RECALL_LSTM.append(RECALL)

    F1_SCORE_LSTM.append(F1_SCORE)
    ACCURACY_LSTM.append(ACCURACY)
    ERROR_RATE_LSTM.append(ERROR_RATE)
    BACC_LSTM.append(BACC)
    TSS_LSTM.append(TSS)
    HSS_LSTM.append(HSS)
    BS_LSTM.append(BS)
    BSS_LSTM.append(BSS)
```

Then LSTM model was implemented using the following libraries and functions:

```
############################      Model 3 LSTM      ##########################################


#    Reshape the data to match 3 dimension for LSTM layers.

    X_train1 = X_train.reshape(X_train.shape[0], X_train.shape[1],1)
    X_test1 = X_test.reshape(X_test.shape[0], X_test.shape[1],1)

#     print('X_train.shape:', X_train.shape)
#     print('y_train.shape:', y_train.shape)
#     print('X_test.shape:', X_test.shape)
#     print('y_test.shape:', y_test.shape)

    lstm_model = tf.keras.Sequential()
    lstm_model.add(tf.keras.layers.LSTM(64,return_sequences=True, return_state=False,input_shape=(X_test1.shape[1],X_test1.sh
    lstm_model.add(tf.keras.layers.LSTM(64, return_sequences=True, return_state=False))
    lstm_model.add(tf.keras.layers.LSTM(64, return_sequences=True, return_state=False))
    lstm_model.add(tf.keras.layers.Flatten())
    lstm_model.add(tf.keras.layers.Dense(1, activation='sigmoid'))

    # Compile the Model
    optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)
    lstm_model.compile(optimizer='adam', loss="binary_crossentropy", metrics=['accuracy'])
    #lstm_model.summary()
    lstm_model.fit(X_train1, y_train,batch_size=1, verbose = 0)
    y_predLSTM = lstm_model.predict(X_test1)
    score = lstm_model.evaluate(X_test1, y_test,verbose=0)
    cnf_matrix_LSTM = confusion_matrix(y_test, y_predLSTM)
    [[TNlstm, FPlstm],
    [FNlstm, TPlstm]]=cnf_matrix_LSTM

    evaluation_metrics_lstm(TPlstm,TNlstm,FPlstm,FNlstm)

    TN_TOTAL_LSTM+=TNlstm
    TP_TOTAL_LSTM+=TPlstm
```

Then the metrics calculated by LSTM in each fold of cross validation stored in a da taframe.

```
dfc=pd.DataFrame({
                "TP": TP_LSTM,
                "FP": FP_LSTM,
                "FN": FN_LSTM,
                "TN": TN_LSTM,
                "TPR":TPR_LSTM,
                "FPR":FPR_LSTM,
                "TNR":TNR_LSTM,
                "FNR":FNR_LSTM,
                "RECALL":RECALL_LSTM,
                'PRECISION':PRECISION_LSTM,
                'F1_SCORE':F1_SCORE_LSTM,
                'Accuracy':ACCURACY_LSTM,
                'Error rate':ERROR_RATE_LSTM,
                'BACC':BACC_LSTM,
                'TSS':TSS_LSTM,
                'HSS':HSS_LSTM,
                'BS' :BS_LSTM,
                'BSS':BSS_LSTM},
```

Dataframe for Each-Fold output of 3 models.

```
d1=pd.concat([dfa.iloc[0:1],dfb.iloc[0:1],dfc.iloc[0:1]])
d2=pd.concat([dfa.iloc[1:2],dfb.iloc[1:2],dfc.iloc[1:2]])
d3=pd.concat([dfa.iloc[2:3],dfb.iloc[2:3],dfc.iloc[2:3]])
d4=pd.concat([dfa.iloc[3:4],dfb.iloc[3:4],dfc.iloc[3:4]])
d5=pd.concat([dfa.iloc[4:5],dfb.iloc[4:5],dfc.iloc[4:5]])
d6=pd.concat([dfa.iloc[5:6],dfb.iloc[5:6],dfc.iloc[5:6]])
d7=pd.concat([dfa.iloc[6:7],dfb.iloc[6:7],dfc.iloc[6:7]])
d8=pd.concat([dfa.iloc[7:8],dfb.iloc[7:8],dfc.iloc[7:8]])
d9=pd.concat([dfa.iloc[8:9],dfb.iloc[8:9],dfc.iloc[8:9]])
d10=pd.concat([dfa.iloc[9:10],dfb.iloc[9:10],dfc.iloc[9:10]])

dfEachFold=pd.concat([d1,d2,d3,d4,d5,d6,d7,d8,d9,d10],keys=('KFOLD-1','KFOLD-2','KFOLD-3','KFOLD-4','KFOLD-5','KFOLD-6','KFOL
                'KFOLD-8','KFOLD-9','KFOLD-10'))

display(dfEachFold)
```

Output Table of each fold comparison:

		TP	FP	FN	TN	TPR	FPR	TNR	FNR	RECALL	PRECISION	F1_SCORE	Accuracy	Error rate	BACC	TSS	HS
KFOLD-1	Naive-Bayes	7	0	0	6	1.000000	0.000	1.000	0.000000	1.000000	1.000000	1.000000	1.000000	0.000000	1.000000	1.000000	1.00000
	Random-Forest	7	0	0	6	1.000000	0.000	1.000	0.000000	1.000000	1.000000	1.000000	1.000000	0.000000	1.000000	1.000000	1.00000
	LSTM	0	0	7	6	0.000000	0.000	1.000	1.000000	0.000000	NaN	0.000000	0.461538	0.538462	0.500000	0.000000	0.00000
KFOLD-2	Naive-Bayes	6	0	0	7	1.000000	0.000	1.000	0.000000	1.000000	1.000000	1.000000	1.000000	0.000000	1.000000	1.000000	1.00000
	Random-Forest	5	0	1	7	0.833333	0.000	1.000	0.166667	0.833333	1.000000	0.909091	0.923077	0.076923	0.916667	0.833333	0.84337
	LSTM	0	0	6	7	0.000000	0.000	1.000	1.000000	0.000000	NaN	0.000000	0.538462	0.461538	0.500000	0.000000	0.00000
KFOLD-3	Naive-Bayes	7	0	0	6	1.000000	0.000	1.000	0.000000	1.000000	1.000000	1.000000	1.000000	0.000000	1.000000	1.000000	1.00000
	Random-Forest	7	0	0	6	1.000000	0.000	1.000	0.000000	1.000000	1.000000	1.000000	1.000000	0.000000	1.000000	1.000000	1.00000
	LSTM	0	0	7	6	0.000000	0.000	1.000	1.000000	0.000000	NaN	0.000000	0.461538	0.538462	0.500000	0.000000	0.00000
KFOLD-4	Naive-Bayes	7	0	0	6	1.000000	0.000	1.000	0.000000	1.000000	1.000000	1.000000	1.000000	0.000000	1.000000	1.000000	1.00000
	Random-Forest	7	0	0	6	1.000000	0.000	1.000	0.000000	1.000000	1.000000	1.000000	1.000000	0.000000	1.000000	1.000000	1.00000
	LSTM	0	0	7	6	0.000000	0.000	1.000	1.000000	0.000000	NaN	0.000000	0.461538	0.538462	0.500000	0.000000	0.00000
	Naive-Bayes	6	0	0	7	1.000000	0.000	1.000	0.000000	1.000000	1.000000	1.000000	1.000000	0.000000	1.000000	1.000000	1.00000

KFOLD-5	Naive-Bayes	6	0	0	7	1.000000	0.000	1.000	0.000000	1.000000	1.000000	1.000000	1.000000	0.000000	1.000000	1.000000	1.00000
	Random-Forest	6	0	0	7	1.000000	0.000	1.000	0.000000	1.000000	1.000000	1.000000	1.000000	0.000000	1.000000	1.000000	1.00000
	LSTM	0	0	6	7	0.000000	0.000	1.000	1.000000	0.000000	NaN	0.000000	0.538462	0.461538	0.500000	0.000000	0.00000
KFOLD-6	Naive-Bayes	9	0	0	4	1.000000	0.000	1.000	0.000000	1.000000	1.000000	1.000000	1.000000	0.000000	1.000000	1.000000	1.00000
	Random-Forest	9	0	0	4	1.000000	0.000	1.000	0.000000	1.000000	1.000000	1.000000	1.000000	0.000000	1.000000	1.000000	1.00000
	LSTM	0	0	9	4	0.000000	0.000	1.000	1.000000	0.000000	NaN	0.000000	0.307692	0.692308	0.500000	0.000000	0.00000
KFOLD-7	Naive-Bayes	8	0	0	5	1.000000	0.000	1.000	0.000000	1.000000	1.000000	1.000000	1.000000	0.000000	1.000000	1.000000	1.00000
	Random-Forest	8	0	0	5	1.000000	0.000	1.000	0.000000	1.000000	1.000000	1.000000	1.000000	0.000000	1.000000	1.000000	1.00000
	LSTM	0	0	8	5	0.000000	0.000	1.000	1.000000	0.000000	NaN	0.000000	0.384615	0.615385	0.500000	0.000000	0.00000
KFOLD-8	Naive-Bayes	8	0	0	5	1.000000	0.000	1.000	0.000000	1.000000	1.000000	1.000000	1.000000	0.000000	1.000000	1.000000	1.00000
	Random-Forest	8	0	0	5	1.000000	0.000	1.000	0.000000	1.000000	1.000000	1.000000	1.000000	0.000000	1.000000	1.000000	1.00000
	LSTM	0	0	8	5	0.000000	0.000	1.000	1.000000	0.000000	NaN	0.000000	0.384615	0.615385	0.500000	0.000000	0.00000
KFOLD-9	Naive-Bayes	5	2	0	6	1.000000	0.250	0.750	0.000000	1.000000	0.714286	0.833333	0.846154	0.153846	0.875000	0.750000	0.69767
	Random-Forest	5	1	0	7	1.000000	0.125	0.875	0.000000	1.000000	0.833333	0.909091	0.923077	0.076923	0.937500	0.875000	0.84337

Code for average cross validation metrics:

1. Naïve Bayes

```
# Aggregating for Naive Bayes

TN=TN_TOTALNB/10
TP=TP_TOTALNB/10
FN=FN_TOTALNB/10
FP=FP_TOTALNB/10

TPR=TP/(TP+FN)
TNR=TN/(TN+FP)
FPR=FP/(FP+TN)
FNR=FN/(FN+TP)
RECALL=TPR
PRECISION=TP/(TP+FP)
F1_SCORE=(2*TP)/(2*TP+FP+FN)
ACCURACY=(TP+TN)/(TP+FP+TN+FN)
ERROR_RATE=1-ACCURACY
BACC=(TPR+TNR)/2
TSS=TPR-FPR
HSS=2*(TP*TN-FP*FN)/(((TP+FN)*(FN+TN))+((TP+FP)*(FP+TN)))
sum_y=0
for n in range(len(y_test)):
    sum_y+=(y_test[n]-y_predNB[n])**2
BS=sum_y/len(y_test)

y_meantemp=0
for i in range(len(y_test)):
    y_meantemp+=y_test[i]
ymean=y_meantemp/len(y_test)
temp=0
```

```
for i in range(len(y_test)):
    temp+=(y_test[i]-ymean)**2
temp=temp/len(y_test)
BSS=BS/temp

dfavg1=pd.DataFrame({"TP": TP,
                     "FP": FP,
                     "FN": FN,
                     "TN": TN,
                     "TPR":TPR,
                     "FPR":FPR,
                     "TNR":TNR,
                     "FNR":FNR,
                     "RECALL":RECALL,
                     'PRECISION':PRECISION,
                     'F1_SCORE':F1_SCORE,
                     'Accuracy':ACCURACY,
                     'Error rate':ERROR_RATE,
                     'BACC':BACC,
                     'TSS':TSS,
                     'HSS':HSS,
                     'BS' :BS,
                     'BSS':BSS
                      },
                      index=["NAIVE BAYES"])
```

2. Random Forest

```
#Averaging for random forest model

TN=TN_TOTALRF/10
TP=TP_TOTALRF/10
FP=FP_TOTALRF/10
FN=FN_TOTALRF/10

TPR=TP/(TP+FN)
TNR=TN/(TN+FP)
FPR=FP/(FP+TN)
FNR=FN/(FN+TP)
RECALL=TPR
PRECISION=TP/(TP+FP)
F1_SCORE=(2*TP)/(2*TP+FP+FN)
ACCURACY=(TP+TN)/(TP+FP+TN+FN)
ERROR_RATE=1-ACCURACY
BACC=(TPR+TNR)/2
TSS=TPR-FPR
HSS=2*(TP*TN-FP*FN)/(((TP+FN)*(FN+TN))+((TP+FP)*(FP+TN)))
sum_y=0
for n in range(len(y_test)):
    sum_y+=(y_test[n]-y_predRF[n])**2
BS=sum_y/len(y_test)

y_meantemp=0
for i in range(len(y_test)):
    y_meantemp+=y_test[i]
ymean=y_meantemp/len(y_test)
temp=0
```

```
for i in range(len(y_test)):
    temp+=(y_test[i]-ymean)**2
temp=temp/len(y_test)
BSS=BS/temp

dfavg2=pd.DataFrame({"TP": TP,
                "FP": FP,
                "FN": FN,
                "TN": TN,
                "TPR":TPR,
                "FPR":FPR,
                "TNR":TNR,
                "FNR":FNR,
                "RECALL":RECALL,
                'PRECISION':PRECISION,
                'F1_SCORE':F1_SCORE,
                'Accuracy':ACCURACY,
                'Error rate':ERROR_RATE,
                'BACC':BACC,
                'TSS':TSS,
                'HSS':HSS,
                'BS' :BS,
                'BSS':BSS
                 },
                 index=["RANDOM FOREST"])
```

3. LSTM

```
# Aggregating for LSTM model

TN=TN_TOTAL_LSTM/10
TP=TP_TOTAL_LSTM/10
FP=FP_TOTAL_LSTM/10
FN=FN_TOTAL_LSTM/10

TPR=TP/(TP+FN)
TNR=TN/(TN+FP)
FPR=FP/(FP+TN)
FNR=FN/(FN+TP)
RECALL=TPR
PRECISION=TP/(TP+FP)
F1_SCORE=(2*TP)/(2*TP+FP+FN)
ACCURACY=(TP+TN)/(TP+FP+TN+FN)
ERROR_RATE=1-ACCURACY
BACC=(TPR+TNR)/2
TSS=TPR-FPR
HSS=2*(TP*TN-FP*FN)/(((TP+FN)*(FN+TN))+((TP+FP)*(FP+TN)))
sum_y=0
for n in range(len(y_test)):
    sum_y+=(y_test[n]-y_predLSTM[n])**2
BS=sum_y/len(y_test)

y_meantemp=0
for i in range(len(y_test)):
    y_meantemp+=y_test[i]
ymean=y_meantemp/len(y_test)
temp=0
```

```python
for i in range(len(y_test)):
    y_meantemp+=y_test[i]
ymean=y_meantemp/len(y_test)
temp=0
for i in range(len(y_test)):
    temp+=(y_test[i]-ymean)**2
temp=temp/len(y_test)
BSS=BS/temp

dfavg3=pd.DataFrame({"TP": TP,
                "FP": FP,
                "FN": FN,
                "TN": TN,
                "TPR":TPR,
                "FPR":FPR,
                "TNR":TNR,
                "FNR":FNR,
                "RECALL":RECALL,
                'PRECISION':PRECISION,
                'F1_SCORE':F1_SCORE,
                'Accuracy':ACCURACY,
                'Error rate':ERROR_RATE,
                'BACC':BACC,
                'TSS':TSS,
                'HSS':HSS,
                'BS' :BS,
                'BSS':BSS
                 },
                 index=["LSTM"])
```

Output of average cross validation outputs of all 3 models:

```python
df_avg = pd.concat([dfavg1, dfavg2,dfavg3])
display(df_avg)
```

	TP	FP	FN	TN	TPR	FPR	TNR	FNR	RECALL	PRECISION	F1_SCORE	Accuracy	Error rate	BACC	TSS	HSS
NAIVE BAYES	7.1	0.2	0.0	5.7	1.000000	0.033898	0.966102	0.000000	1.000000	0.972603	0.986111	0.984615	0.015385	0.983051	0.966102	0.968877
RANDOM FOREST	7.0	0.1	0.1	5.8	0.985915	0.016949	0.983051	0.014085	0.985915	0.985915	0.985915	0.984615	0.015385	0.984483	0.968966	0.968966
LSTM	0.0	0.0	7.1	5.9	0.000000	0.000000	1.000000	1.000000	0.000000	NaN	0.000000	0.453846	0.546154	0.500000	0.000000	0.000000

Saving output table to .xlsx file

```
import openpyxl
import xlsxwriter
import xlwt
writer = pd.ExcelWriter('FinalResult.xlsx', engine='xlsxwriter')

#write each DataFrame to a specific sheet
dfEachFold.to_excel(writer, sheet_name='EachFold')
df_avg.to_excel(writer, sheet_name='Overall')

#close the Pandas Excel writer and output the Excel file
writer.save()
```

Complete Source code:

import pandas as pd

import numpy as np

import tensorflow as tf

from tensorflow.python.ops.math_ops import reduce_prod

import warnings

warnings.filterwarnings("ignore")

import math

print('tensorflow version : ',tf. __version__)

print('numpy version : ', np. __version__)

data=pd.read_csv('wine.csv',header=None)

df=data.sample(frac=1)

print(df.head)

First column represents the quality of wine(1 or 2) so it is selected as label

labels=df.iloc[:,0]

features=df.iloc[:,1:14]

X=features

y=labels

TP_NB=[]

FP_NB=[]

FN_NB=[]

TN_NB=[]

TPR_NB=[]

FPR_NB=[]

TNR_NB=[]

FNR_NB=[]

RECALL_NB=[]

PRECISION_NB=[]

F1_SCORE_NB=[]

ACCURACY_NB=[]

ERROR_RATE_NB=[]

BACC_NB=[]

TSS_NB=[]

HSS_NB=[]

BS_NB=[]

BSS_NB=[]

TP_RF=[]

FP_RF=[]

FN_RF=[]

TN_RF=[]

TPR_RF=[]

FPR_RF=[]

TNR_RF=[]

FNR_RF=[]

RECALL_RF=[]

PRECISION_RF=[]

F1_SCORE_RF=[]

ACCURACY_RF=[]

ERROR_RATE_RF=[]

BACC_RF=[]

TSS_RF=[]

HSS_RF=[]

BS_RF=[]

BSS_RF=[]

TP_LSTM=[]

FP_LSTM=[]

FN_LSTM=[]

TN_LSTM=[]

TPR_LSTM=[]

FPR_LSTM=[]

TNR_LSTM=[]

FNR_LSTM=[]

RECALL_LSTM=[]

PRECISION_LSTM=[]

F1_SCORE_LSTM=[]

ACCURACY_LSTM=[]

ERROR_RATE_LSTM=[]

BACC_LSTM=[]

```
TSS_LSTM=[]

HSS_LSTM=[]

BS_LSTM=[]

BSS_LSTM=[]


def evaluation_metricsNB(TP,TN,FP,FN):


    TP=TP
    TN=TN
    FP=FP
    FN=FN
    TPR=TP/(TP+FN)
    TNR=TN/(TN+FP)
    FPR=FP/(FP+TN)
    FNR=FN/(FN+TP)
    RECALL=TPR
    PRECISION=TP/(TP+FP)
    F1_SCORE=(2*TP)/(2*TP+FP+FN)
    ACCURACY=(TP+TN)/(TP+FP+TN+FN)
    ERROR_RATE=1-ACCURACY
    BACC=(TPR+TNR)/2
    TSS=TPR-FPR
    HSS=2*(TP*TN-FP*FN)/(((TP+FN)*(FN+TN))+((TP+FP)*(FP+TN)))
    sum_y=0
    for n in range(len(y_test)):
        sum_y+=(y_test[n]-y_predNB[n])**2
```

```
BS=sum_y/len(y_test)


y_meantemp=0
for i in range(len(y_test)):
    y_meantemp+=y_test[i]
ymean=y_meantemp/len(y_test)
#BSS
temp=0
for i in range(len(y_test)):
    temp+=(y_test[i]-ymean)**2
temp=temp/len(y_test)
BSS=BS/temp


TP_NB.append(TP)
FP_NB.append(FP)
FN_NB.append(FN)
TN_NB.append(TN)
TPR_NB.append(TPR)
FPR_NB.append(FPR)
TNR_NB.append(TNR)
FNR_NB.append(FNR)
RECALL_NB.append(RECALL)
PRECISION_NB.append(PRECISION)
F1_SCORE_NB.append(F1_SCORE)
ACCURACY_NB.append(ACCURACY)
ERROR_RATE_NB.append(ERROR_RATE)
```

```
    BACC_NB.append(BACC)

    TSS_NB.append(TSS)

    HSS_NB.append(HSS)

    BS_NB.append(BS)

    BSS_NB.append(BSS)


def evaluation_metricsRF(TP,TN,FP,FN):


    TP=TP

    TN=TN

    FP=FP

    FN=FN

    TPR=TP/(TP+FN)

    TNR=TN/(TN+FP)

    FPR=FP/(FP+TN)

    FNR=FN/(FN+TP)

    RECALL=TPR

    PRECISION=TP/(TP+FP)

    F1_SCORE=(2*TP)/(2*TP+FP+FN)

    ACCURACY=(TP+TN)/(TP+FP+TN+FN)

    ERROR_RATE=1-ACCURACY

    BACC=(TPR+TNR)/2

    TSS=TPR-FPR

    HSS=2*(TP*TN-FP*FN)/(((TP+FN)*(FN+TN))+((TP+FP)*(FP+TN)))

    sum_y=0

    for n in range(len(y_test)):
```

```
    sum_y+=(y_test[n]-y_predRF[n])**2
BS=sum_y/len(y_test)

y_meantemp=0
for i in range(len(y_test)):
    y_meantemp+=y_test[i]
ymean=y_meantemp/len(y_test)

temp=0
for i in range(len(y_test)):
    temp+=(y_test[i]-ymean)**2
temp=temp/len(y_test)
BSS=BS/temp

TP_RF.append(TP)
FP_RF.append(FP)
FN_RF.append(FN)
TN_RF.append(TN)
TPR_RF.append(TPR)
FPR_RF.append(FPR)
TNR_RF.append(TNR)
FNR_RF.append(FNR)
RECALL_RF.append(RECALL)
PRECISION_RF.append(PRECISION)
F1_SCORE_RF.append(F1_SCORE)
ACCURACY_RF.append(ACCURACY)
```

```
    ERROR_RATE_RF.append(ERROR_RATE)
    BACC_RF.append(BACC)
    TSS_RF.append(TSS)
    HSS_RF.append(HSS)
    BS_RF.append(BS)
    BSS_RF.append(BSS)

def evaluation_metrics_lstm(TP,TN,FP,FN):

    TP=TP
    TN=TN
    FP=FP
    FN=FN
    TPR=TP/(TP+FN)
    TNR=TN/(TN+FP)
    FPR=FP/(FP+TN)
    FNR=FN/(FN+TP)
    RECALL=TPR
    PRECISION=TP/(TP+FP)
    if math.isnan(PRECISION):
        PRECISION_LSTM.append(np.nan)

    F1_SCORE=(2*TP)/(2*TP+FP+FN)
    ACCURACY=(TP+TN)/(TP+FP+TN+FN)
    ERROR_RATE=1-ACCURACY
    BACC=(TPR+TNR)/2
```

```
TSS=TPR-FPR
HSS=2*(TP*TN-FP*FN)/(((TP+FN)*(FN+TN))+((TP+FP)*(FP+TN)))
sum_y=0
for n in range(len(y_test)):
    sum_y+=(y_test[n]-y_predLSTM[n])**2
BS=sum_y/len(y_test)

y_meantemp=0
for i in range(len(y_test)):
    y_meantemp+=y_test[i]
ymean=y_meantemp/len(y_test)

temp=0
for i in range(len(y_test)):
    temp+=(y_test[i]-ymean)**2
temp=temp/len(y_test)
BSS=BS/temp

TP_LSTM.append(TP)
FP_LSTM.append(FP)
FN_LSTM.append(FN)
TN_LSTM.append(TN)
TPR_LSTM.append(TPR)
FPR_LSTM.append(FPR)
TNR_LSTM.append(TNR)
FNR_LSTM.append(FNR)
```

```
    RECALL_LSTM.append(RECALL)


    F1_SCORE_LSTM.append(F1_SCORE)
    ACCURACY_LSTM.append(ACCURACY)
    ERROR_RATE_LSTM.append(ERROR_RATE)
    BACC_LSTM.append(BACC)
    TSS_LSTM.append(TSS)
    HSS_LSTM.append(HSS)
    BS_LSTM.append(BS)
    BSS_LSTM.append(BSS)


import warnings
warnings.filterwarnings("ignore")


X=np.array(X)
y=np.array(y)
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix



kf = KFold(n_splits=10)
```

```
TN_TOTALNB=0
TP_TOTALNB=0
FP_TOTALNB=0
FN_TOTALNB=0

TN_TOTALRF=0
TP_TOTALRF=0
FP_TOTALRF=0
FN_TOTALRF=0

TN_TOTAL_LSTM=0
TP_TOTAL_LSTM=0
FP_TOTAL_LSTM=0
FN_TOTAL_LSTM=0


for train_index, test_index in kf.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

###########################      Model1 Naive Bayes     ###################################

    modelNB = GaussianNB()
    modelNB.fit(X_train, y_train)
    y_predNB = modelNB.predict(X_test)
    cnf_matrixNB = confusion_matrix(y_test, y_predNB)
```

```
[[TNNB, FPNB],
[FNNB, TPNB]]=cnf_matrixNB

evaluation_metricsNB(TPNB,TNNB,FPNB,FNNB)

TN_TOTALNB+=TNNB
TP_TOTALNB+=TPNB
FP_TOTALNB+=FPNB
FN_TOTALNB+=FNNB

##########################   Model2 Random Forest
#################################

rf= RandomForestClassifier(n_estimators=20, random_state=0)
rf.fit(X_train, y_train)
y_predRF=rf.predict(X_test)
cnf_matrixRF = confusion_matrix(y_test, y_predRF)
[[TNRF, FPRF],
[FNRF, TPRF]]=cnf_matrixRF

evaluation_metricsRF(TPRF,TNRF,FPRF,FNRF)

TN_TOTALRF+=TNRF
TP_TOTALRF+=TPRF
FP_TOTALRF+=FPRF
FN_TOTALRF+=FNRF
```

########################### Model 3 LSTM
###

```
#   Reshape the data to match 3 dimension for LSTM layers.

    X_train1 = X_train.reshape(X_train.shape[0], X_train.shape[1],1)

    X_test1 = X_test.reshape(X_test.shape[0], X_test.shape[1],1)

#    print('X_train.shape:', X_train.shape)

#    print('y_train.shape:', y_train.shape)

#    print('X_test.shape:', X_test.shape)

#    print('y_test.shape:', y_test.shape)

    lstm_model = tf.keras.Sequential()

    lstm_model.add(tf.keras.layers.LSTM(64,return_sequences=True,
return_state=False,input_shape=(X_test1.shape[1],X_test1.shape[2])))

    lstm_model.add(tf.keras.layers.LSTM(64, return_sequences=True, return_state=False))

    lstm_model.add(tf.keras.layers.LSTM(64, return_sequences=True, return_state=False))

    lstm_model.add(tf.keras.layers.Flatten())

    lstm_model.add(tf.keras.layers.Dense(1, activation='sigmoid'))

    # Compile the Model

    optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)

    lstm_model.compile(optimizer='adam', loss="binary_crossentropy", metrics=['accuracy'])

    #lstm_model.summary()

    lstm_model.fit(X_train1, y_train,batch_size=1, verbose = 0)
```

```
    y_predLSTM = lstm_model.predict(X_test1)
    score = lstm_model.evaluate(X_test1, y_test,verbose=0)
    cnf_matrix_LSTM = confusion_matrix(y_test, y_predLSTM)
    [[TNlstm, FPlstm],
    [FNlstm, TPlstm]]=cnf_matrix_LSTM

    evaluation_metrics_lstm(TPlstm,TNlstm,FPlstm,FNlstm)

    TN_TOTAL_LSTM+=TNlstm
    TP_TOTAL_LSTM+=TPlstm
    FP_TOTAL_LSTM+=FPlstm
    FN_TOTAL_LSTM+=FNlstm

dfa=pd.DataFrame({
            "TP": TP_NB,
            "FP": FP_NB,
            "FN": FN_NB,
            "TN": TN_NB,
            "TPR":TPR_NB,
            "FPR":FPR_NB,
            "TNR":TNR_NB,
            "FNR":FNR_NB,
            "RECALL":RECALL_NB,
            'PRECISION':PRECISION_NB,
            'F1_SCORE':F1_SCORE_NB,
            'Accuracy':ACCURACY_NB,
```

```
            'Error rate':ERROR_RATE_NB,
            'BACC':BACC_NB,
            'TSS':TSS_NB,
            'HSS':HSS_NB,
            'BS' :BS_NB,
            'BSS':BSS_NB},
            index=['Naive-Bayes','Naive-Bayes','Naive-Bayes','Naive-Bayes','Naive-Bayes',
                'Naive-Bayes','Naive-Bayes','Naive-Bayes','Naive-Bayes','Naive-Bayes',])
dfb=pd.DataFrame({
            "TP": TP_RF,
            "FP": FP_RF,
            "FN": FN_RF,
            "TN": TN_RF,
            "TPR":TPR_RF,
            "FPR":FPR_RF,
            "TNR":TNR_RF,
            "FNR":FNR_RF,
            "RECALL":RECALL_RF,
            'PRECISION':PRECISION_RF,
            'F1_SCORE':F1_SCORE_RF,
            'Accuracy':ACCURACY_RF,
            'Error rate':ERROR_RATE_RF,
            'BACC':BACC_RF,
            'TSS':TSS_RF,
            'HSS':HSS_RF,
            'BS' :BS_RF,
```

```
        'BSS':BSS_RF},
        index=['Random-Forest','Random-Forest','Random-Forest','Random-Forest','Random-Forest',
           'Random-Forest','Random-Forest','Random-Forest','Random-Forest','Random-Forest',])

dfc=pd.DataFrame({
        "TP": TP_LSTM,
        "FP": FP_LSTM,
        "FN": FN_LSTM,
        "TN": TN_LSTM,
        "TPR":TPR_LSTM,
        "FPR":FPR_LSTM,
        "TNR":TNR_LSTM,
        "FNR":FNR_LSTM,
        "RECALL":RECALL_LSTM,
        'PRECISION':PRECISION_LSTM,
        'F1_SCORE':F1_SCORE_LSTM,
        'Accuracy':ACCURACY_LSTM,
        'Error rate':ERROR_RATE_LSTM,
        'BACC':BACC_LSTM,
        'TSS':TSS_LSTM,
        'HSS':HSS_LSTM,
        'BS' :BS_LSTM,
        'BSS':BSS_LSTM},
        index=['LSTM','LSTM','LSTM','LSTM','LSTM','LSTM',
           'LSTM','LSTM','LSTM','LSTM',])
```

```
d1=pd.concat([dfa.iloc[0:1],dfb.iloc[0:1],dfc.iloc[0:1]])

d2=pd.concat([dfa.iloc[1:2],dfb.iloc[1:2],dfc.iloc[1:2]])

d3=pd.concat([dfa.iloc[2:3],dfb.iloc[2:3],dfc.iloc[2:3]])

d4=pd.concat([dfa.iloc[3:4],dfb.iloc[3:4],dfc.iloc[3:4]])

d5=pd.concat([dfa.iloc[4:5],dfb.iloc[4:5],dfc.iloc[4:5]])

d6=pd.concat([dfa.iloc[5:6],dfb.iloc[5:6],dfc.iloc[5:6]])

d7=pd.concat([dfa.iloc[6:7],dfb.iloc[6:7],dfc.iloc[6:7]])

d8=pd.concat([dfa.iloc[7:8],dfb.iloc[7:8],dfc.iloc[7:8]])

d9=pd.concat([dfa.iloc[8:9],dfb.iloc[8:9],dfc.iloc[8:9]])

d10=pd.concat([dfa.iloc[9:10],dfb.iloc[9:10],dfc.iloc[9:10]])


dfEachFold=pd.concat([d1,d2,d3,d4,d5,d6,d7,d8,d9,d10],keys=('KFOLD-1','KFOLD-2','KFOLD-3','KFOLD-4','KFOLD-5','KFOLD-6','KFOLD-7',

                 'KFOLD-8','KFOLD-9','KFOLD-10'))


display(dfEachFold)


# Aggregating for Naive Bayes


TN=TN_TOTALNB/10

TP=TP_TOTALNB/10

FN=FN_TOTALNB/10

FP=FP_TOTALNB/10


TPR=TP/(TP+FN)

TNR=TN/(TN+FP)

FPR=FP/(FP+TN)
```

```
FNR=FN/(FN+TP)
RECALL=TPR
PRECISION=TP/(TP+FP)
F1_SCORE=(2*TP)/(2*TP+FP+FN)
ACCURACY=(TP+TN)/(TP+FP+TN+FN)
ERROR_RATE=1-ACCURACY
BACC=(TPR+TNR)/2
TSS=TPR-FPR
HSS=2*(TP*TN-FP*FN)/(((TP+FN)*(FN+TN))+((TP+FP)*(FP+TN)))
sum_y=0
for n in range(len(y_test)):
    sum_y+=(y_test[n]-y_predNB[n])**2
BS=sum_y/len(y_test)

y_meantemp=0
for i in range(len(y_test)):
    y_meantemp+=y_test[i]
ymean=y_meantemp/len(y_test)
temp=0
for i in range(len(y_test)):
    temp+=(y_test[i]-ymean)**2
temp=temp/len(y_test)
BSS=BS/temp

dfavg1=pd.DataFrame({"TP": TP,
                "FP": FP,
```

```
            "FN": FN,
            "TN": TN,
            "TPR":TPR,
            "FPR":FPR,
            "TNR":TNR,
            "FNR":FNR,
            "RECALL":RECALL,
            'PRECISION':PRECISION,
            'F1_SCORE':F1_SCORE,
            'Accuracy':ACCURACY,
            'Error rate':ERROR_RATE,
            'BACC':BACC,
            'TSS':TSS,
            'HSS':HSS,
            'BS' :BS,
            'BSS':BSS
             },
             index=["NAIVE BAYES"])


#Averaging for random forest model


TN=TN_TOTALRF/10
TP=TP_TOTALRF/10
FP=FP_TOTALRF/10
FN=FN_TOTALRF/10
```

```
TPR=TP/(TP+FN)

TNR=TN/(TN+FP)

FPR=FP/(FP+TN)

FNR=FN/(FN+TP)

RECALL=TPR

PRECISION=TP/(TP+FP)

F1_SCORE=(2*TP)/(2*TP+FP+FN)

ACCURACY=(TP+TN)/(TP+FP+TN+FN)

ERROR_RATE=1-ACCURACY

BACC=(TPR+TNR)/2

TSS=TPR-FPR

HSS=2*(TP*TN-FP*FN)/(((TP+FN)*(FN+TN))+((TP+FP)*(FP+TN)))

sum_y=0

for n in range(len(y_test)):

    sum_y+=(y_test[n]-y_predRF[n])**2

BS=sum_y/len(y_test)


y_meantemp=0

for i in range(len(y_test)):

    y_meantemp+=y_test[i]

ymean=y_meantemp/len(y_test)

temp=0

for i in range(len(y_test)):

    temp+=(y_test[i]-ymean)**2

temp=temp/len(y_test)
```

```
BSS=BS/temp

dfavg2=pd.DataFrame({"TP": TP,
                "FP": FP,
                "FN": FN,
                "TN": TN,
                "TPR":TPR,
                "FPR":FPR,
                "TNR":TNR,
                "FNR":FNR,
                "RECALL":RECALL,
                'PRECISION':PRECISION,
                'F1_SCORE':F1_SCORE,
                'Accuracy':ACCURACY,
                'Error rate':ERROR_RATE,
                'BACC':BACC,
                'TSS':TSS,
                'HSS':HSS,
                'BS' :BS,
                'BSS':BSS
                 },
                 index=["RANDOM FOREST"])

# Aggregating for LSTM model

TN=TN_TOTAL_LSTM/10
```

```
TP=TP_TOTAL_LSTM/10
FP=FP_TOTAL_LSTM/10
FN=FN_TOTAL_LSTM/10

TPR=TP/(TP+FN)
TNR=TN/(TN+FP)
FPR=FP/(FP+TN)
FNR=FN/(FN+TP)
RECALL=TPR
PRECISION=TP/(TP+FP)
F1_SCORE=(2*TP)/(2*TP+FP+FN)
ACCURACY=(TP+TN)/(TP+FP+TN+FN)
ERROR_RATE=1-ACCURACY
BACC=(TPR+TNR)/2
TSS=TPR-FPR
HSS=2*(TP*TN-FP*FN)/(((TP+FN)*(FN+TN))+((TP+FP)*(FP+TN)))
sum_y=0
for n in range(len(y_test)):
    sum_y+=(y_test[n]-y_predLSTM[n])**2
BS=sum_y/len(y_test)

y_meantemp=0
for i in range(len(y_test)):
    y_meantemp+=y_test[i]
ymean=y_meantemp/len(y_test)
temp=0
```

```
for i in range(len(y_test)):
    temp+=(y_test[i]-ymean)**2
temp=temp/len(y_test)
BSS=BS/temp


dfavg3=pd.DataFrame({"TP": TP,
            "FP": FP,
            "FN": FN,
            "TN": TN,
            "TPR":TPR,
            "FPR":FPR,
            "TNR":TNR,
            "FNR":FNR,
            "RECALL":RECALL,
            'PRECISION':PRECISION,
            'F1_SCORE':F1_SCORE,
            'Accuracy':ACCURACY,
            'Error rate':ERROR_RATE,
            'BACC':BACC,
            'TSS':TSS,
            'HSS':HSS,
            'BS' :BS,
            'BSS':BSS
             },
             index=["LSTM"])
```

```
df_avg = pd.concat([dfavg1, dfavg2,dfavg3])

display(df_avg)

import openpyxl

import xlsxwriter

import xlwt

writer = pd.ExcelWriter('FinalResult.xlsx', engine='xlsxwriter')

#write each DataFrame to a specific sheet

dfEachFold.to_excel(writer, sheet_name='EachFold')

df_avg.to_excel(writer, sheet_name='Overall')

#close the Pandas Excel writer and output the Excel file

writer.save()
```

Github Link: https://github.com/nr36/CS634-FinalProject

Comparison/Discussion:

Random Forest outperforms among the three models.

Below are the references made while comparing the three model evaluation metrics:

- Data was almost balanced and no missing values were there that's why accuracy and balanced accuracy i.e. BACC are almost same.
- The result from random forest and Naïve Bayes are close but few thing should be noted
    a. In Naïve Bayes model, In 10 folds there were 2 False positives noted i.e. there is a chance of 20% false positive values. For eg. The output label was 0 but predicted as 1.
    b. Whereas in Random forest model, Out of 10 folds only in one fold, one false negative was detected i.e. there is a chance of 10% false negative values For eg. The output label was 1 but predicted as 0.
    c. Random forest Model performed best out of the 3 models I have selected in all the aspects like accuracy, BACC, F1-score and more. The random Forest and Naïve bayes almost take similar amount of execution time.
    d. While using LSTM model, I have used LSTM with 4 hidden layers other than input and the output layers each with 64 hidden units and activation function as sigmoid and using Adam optimizer with learning rate 0.0001 and loss as binary_crossentropy but the deep neural network couldn't perform well with 100 epochs in each of the 10 folds of cross validation.
       It got too slow in my laptop and I couldn't use GPU as my laptop doesn't support that. May be if we increase number of layers to some higher numbers It could have performed decently.
       <u>But with the result of LSTM we can conclude that:</u>
       No true positive or false positive values have been detected only true negative and false negatives labels were predicted. So it can be

inferred that LSTM only has predictions with probability less than 0.5 for all the data so it considered predictions of all the data as 0. LSTM didn't perform well here.

e. So, For the given wine dataset I would preferably choose Random Forest over Naïve Bayes and LSTM.

Conclusion:

For the given wine dataset I would preferably choose Random Forest over Naïve Bayes and LSTM after comparing the evaluation metrics.