New Jersey's Science & Technology University

CS 634(Section 104) Data Mining - Spring2023
Date:4/19/2024
Professor Yasser Abduallah

Final Project Report
Supervised Data Mining (Classification) using Random Forest, Naive Bayes, LTSM Algorithms

Aakash Siricilla (as4592)

NJIT ID : 233370

Contents:

Option 5

Supervised Data Mining (Classification) Binary Classification Only

The chosen machine learning algorithms for supervised classification include:
a) Naïve Bayes
b) Random Forest

The deep learning algorithm chosen is:
c) LSTM (Long Short Term Memory)

The data was sourced from https://archive.ics.uci.edu/ml/datasets/Wine.
To install TensorFlow in a Jupyter notebook, I used the command:

```
!pip install tensorflow
```

The data was available in .data format, and I read the binary file using the following Python code:

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

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'
```

I converted the data file to .csv format using Notepad for better organization into rows and columns, facilitating its manipulation with the pandas dataframe. However, the original Wine.data comprised three labels, necessitating multi-label classification and confusion matrix analysis, which proved overly intricate to manage for each cross-validation fold. Consequently, I opted to work with only two labels, simplifying the task to binary classification. I then imported the modified data using the pandas **`read_csv`** function:

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

To randomize the data order, I utilized the `sample` function with a fraction of 1:

df = data.sample(frac=1)

Printing the head of the data frame for a preview:

print(df.head())

```
data=pd.read_csv('wine.csv',header=None)
df=data.sample(frac=1)
print(df.head)
```

```
<bound method NDFrame.head of      0      1     2     3     4     5     6     7     8     9    10    11  \
118  2  12.77  3.43  1.98  16.0   80  1.63  1.25  0.43  0.83  3.40  0.70
17   1  13.83  1.57  2.62  20.0  115  2.95  3.40  0.40  1.72  6.60  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
99   2  12.29  3.17  2.21  18.0   88  2.85  2.99  0.45  2.81  2.30  1.42
95   2  12.47  1.52  2.20  19.0  162  2.50  2.27  0.32  3.28  2.60  1.16
..  ..    ...   ...   ...   ...  ...   ...   ...   ...   ...   ...   ...
9    1  13.86  1.35  2.27  16.0   98  2.98  3.15  0.22  1.85  7.22  1.01
57   1  13.29  1.97  2.68  16.8  102  3.00  3.23  0.31  1.66  6.00  1.07
35   1  13.48  1.81  2.41  20.5  100  2.70  2.98  0.26  1.86  5.10  1.04
41   1  13.41  3.84  2.12  18.8   90  2.45  2.68  0.27  1.48  4.28  0.91
15   1  13.63  1.81  2.70  17.2  112  2.85  2.91  0.30  1.46  7.30  1.28

       12    13
118  2.12   372
17   2.57  1130
61   1.59   450
99   2.83   406
95   2.63   937
..    ...   ...
9    3.55  1045
57   2.84  1270
35   3.47   920
41   3.00  1035
15   2.88  1310

[130 rows x 14 columns]>
```

To proceed, I segmented the dataset into features and labels. The dataset comprises a total of 14 columns, with the first column denoting the wine quality, labeled as "Label" with index 0. The subsequent columns are considered as features, contributing to the prediction of the label, spanning from index 1 to index 13. Below is the code to accomplish this segmentation:

```
labels = df.iloc[:, 0] # Extracting labels from the first column
features = df.iloc[:, 1:14] # Extracting features from columns 1 to 13
X = features # Assigning features to X
y = labels # Assigning labels to y
```

```
X:          1     2     3     4    5     6     7     8     9     10    11    12  \
118  12.77  3.43  1.98  16.0   80  1.63  1.25  0.43  0.83  3.40  0.70  2.12
17   13.83  1.57  2.62  20.0  115  2.95  3.40  0.40  1.72  6.60  1.13  2.57
61   12.64  1.36  2.02  16.8  100  2.02  1.41  0.53  0.62  5.75  0.98  1.59
99   12.29  3.17  2.21  18.0   88  2.85  2.99  0.45  2.81  2.30  1.42  2.83
95   12.47  1.52  2.20  19.0  162  2.50  2.27  0.32  3.28  2.60  1.16  2.63
..     ...   ...   ...   ...  ...   ...   ...   ...   ...   ...   ...   ...
9    13.86  1.35  2.27  16.0   98  2.98  3.15  0.22  1.85  7.22  1.01  3.55
57   13.29  1.97  2.68  16.8  102  3.00  3.23  0.31  1.66  6.00  1.07  2.84
35   13.48  1.81  2.41  20.5  100  2.70  2.98  0.26  1.86  5.10  1.04  3.47
41   13.41  3.84  2.12  18.8   90  2.45  2.68  0.27  1.48  4.28  0.91  3.00
15   13.63  1.81  2.70  17.2  112  2.85  2.91  0.30  1.46  7.30  1.28  2.88

       13
118   372
17   1130
61    450
99    406
95    937
..    ...
9    1045
57   1270
35    920
41   1035
15   1310

[130 rows x 13 columns]
y :  118    2
17     1
61     2
99     2
95     2
      ..
9      1
57     1
35     1
41     1
15     1
Name: 0, Length: 130, dtype: int64
```

The next step involved importing all the necessary Python libraries and implementing all three models for each fold of k-fold cross-validation, where k is set to 10. Below is a snapshot illustrating the process of importing the essential libraries and implementing the models:

Model 1: Summary of Naïve Bayes for each fold of Cross-Validation

For the Naïve Bayes model, a list of all metrics has been created to assess its performance across each fold of cross-validation. Let's delve into the details of these metrics specifically for the 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 subsequently, a function has been crafted to compute evaluation metrics for each of the models.

```
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)
```

The Naïve Bayes model was then implemented utilizing the following libraries and functions:

```
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
```

Subsequently, the metrics computed by the Naïve Bayes model in each fold of cross-validation were 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',
                       'Naive-Bayes','Naive-Bayes','Naive-Bayes','Naive-Bayes','Naive-Bayes',])
```

Model 2: Summary of Random Forest for each fold of Cross-Validation

For the Random Forest model, a list of all metrics has been prepared to assess its performance across each fold of cross-validation. Let's examine these metrics specifically for the 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=[]
```

Following that, a function has been devised to compute evaluation metrics for each of the models.

```
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, the Random Forest model was implemented utilizing the following libraries 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
```

Subsequently, the metrics computed by the Random Forest model in each fold of cross-validation were 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',
                       'Random-Forest','Random-Forest','Random-Forest','Random-Forest','Random-Forest',])
```

Model 3: Summary of LSTM for each fold of Cross-Validation

For the LSTM model, a list of all metrics has been compiled to evaluate its performance across each fold of cross-validation. Let's explore these metrics specifically for the 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=[]
```

Following that, a function has been developed to calculate evaluation metrics for each of the models.

```
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, the LSTM model was implemented using the following libraries and functions:
Also used Confusion matrix:

```
#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
```

Afterwards, the metrics computed by the LSTM model in each fold of cross-validation were stored in a dataframe.

```
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',])
```

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','KFOLD-7',
           '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	HSS	
KFOLD-1	Naive-Bayes	8	0	0	5	1.000000	0.000000	1.000000	0.000000	1.000000	1.000000	1.000000	1.000000	0.000000	1.000000	1.000000	1.000000	
	Random-Forest	8	0	0	5	1.000000	0.000000	1.000000	0.000000	1.000000	1.000000	1.000000	1.000000	0.000000	1.000000	1.000000	1.000000	
	LSTM	0	0	8	5	0.000000	0.000000	1.000000	1.000000	0.000000	NaN	0.000000	0.384615	0.615385	0.500000	0.000000	0.000000	[0.6153
KFOLD-2	Naive-Bayes	7	0	0	6	1.000000	0.000000	1.000000	0.000000	1.000000	1.000000	1.000000	1.000000	0.000000	1.000000	1.000000	1.000000	
	Random-Forest	7	0	0	6	1.000000	0.000000	1.000000	0.000000	1.000000	1.000000	1.000000	1.000000	0.000000	1.000000	1.000000	1.000000	
	LSTM	0	0	7	6	0.000000	0.000000	1.000000	1.000000	0.000000	NaN	0.000000	0.461538	0.538462	0.500000	0.000000	0.000000	[0.5384
KFOLD-3	Naive-Bayes	8	0	0	5	1.000000	0.000000	1.000000	0.000000	1.000000	1.000000	1.000000	1.000000	0.000000	1.000000	1.000000	1.000000	
	Random-Forest	8	0	0	5	1.000000	0.000000	1.000000	0.000000	1.000000	1.000000	1.000000	1.000000	0.000000	1.000000	1.000000	1.000000	
	LSTM	0	0	8	5	0.000000	0.000000	1.000000	1.000000	0.000000	NaN	0.000000	0.384615	0.615385	0.500000	0.000000	0.000000	[0.6153
KFOLD-4	Naive-Bayes	6	0	0	7	1.000000	0.000000	1.000000	0.000000	1.000000	1.000000	1.000000	1.000000	0.000000	1.000000	1.000000	1.000000	
	Random-Forest	5	0	1	7	0.833333	0.000000	1.000000	0.166667	0.833333	1.000000	0.909091	0.923077	0.076923	0.916667	0.833333	0.843373	0.0
	LSTM	0	0	6	7	0.000000	0.000000	1.000000	1.000000	0.000000	NaN	0.000000	0.538462	0.461538	0.500000	0.000000	0.000000	[0.4615
KFOLD-5	Naive-Bayes	8	0	0	5	1.000000	0.000000	1.000000	0.000000	1.000000	1.000000	1.000000	1.000000	0.000000	1.000000	1.000000	1.000000	
	Random-Forest	8	0	0	5	1.000000	0.000000	1.000000	0.000000	1.000000	1.000000	1.000000	1.000000	0.000000	1.000000	1.000000	1.000000	
	LSTM	0	0	8	5	0.000000	0.000000	1.000000	1.000000	0.000000	NaN	0.000000	0.384615	0.615385	0.500000	0.000000	0.000000	[0.6153
KFOLD-6	Naive-Bayes	6	1	0	6	1.000000	0.142857	0.857143	0.000000	1.000000	0.857143	0.923077	0.923077	0.076923	0.928571	0.857143	0.847059	0.0
	Random-Forest	6	0	0	7	1.000000	0.000000	1.000000	0.000000	1.000000	1.000000	1.000000	1.000000	0.000000	1.000000	1.000000	1.000000	
	LSTM	0	0	6	7	0.000000	0.000000	1.000000	1.000000	0.000000	NaN	0.000000	0.538462	0.461538	0.500000	0.000000	0.000000	[0.4615
KFOLD-7	Naive-Bayes	6	1	0	6	1.000000	0.142857	0.857143	0.000000	1.000000	0.857143	0.923077	0.923077	0.076923	0.928571	0.857143	0.847059	0.0
	Random-Forest	6	0	0	7	1.000000	0.000000	1.000000	0.000000	1.000000	1.000000	1.000000	1.000000	0.000000	1.000000	1.000000	1.000000	
	LSTM	0	0	6	7	0.000000	0.000000	1.000000	1.000000	0.000000	NaN	0.000000	0.538462	0.461538	0.500000	0.000000	0.000000	[0.4615
KFOLD-8	Naive-Bayes	4	0	0	9	1.000000	0.000000	1.000000	0.000000	1.000000	1.000000	1.000000	1.000000	0.000000	1.000000	1.000000	1.000000	
	Random-Forest	4	0	0	9	1.000000	0.000000	1.000000	0.000000	1.000000	1.000000	1.000000	1.000000	0.000000	1.000000	1.000000	1.000000	
	LSTM	0	0	4	9	0.000000	0.000000	1.000000	1.000000	0.000000	NaN	0.000000	0.692308	0.307692	0.500000	0.000000	0.000000	[0.3076
KFOLD-9	Naive-Bayes	8	0	0	5	1.000000	0.000000	1.000000	0.000000	1.000000	1.000000	1.000000	1.000000	0.000000	1.000000	1.000000	1.000000	
	Random-Forest	8	0	0	5	1.000000	0.000000	1.000000	0.000000	1.000000	1.000000	1.000000	1.000000	0.000000	1.000000	1.000000	1.000000	
	LSTM	0	0	8	5	0.000000	0.000000	1.000000	1.000000	0.000000	NaN	0.000000	0.384615	0.615385	0.500000	0.000000	0.000000	[0.6153
KFOLD-10	Naive-Bayes	10	0	0	3	1.000000	0.000000	1.000000	0.000000	1.000000	1.000000	1.000000	1.000000	0.000000	1.000000	1.000000	1.000000	
	Random-Forest	10	0	0	3	1.000000	0.000000	1.000000	0.000000	1.000000	1.000000	1.000000	1.000000	0.000000	1.000000	1.000000	1.000000	
	LSTM	0	0	10	3	0.000000	0.000000	1.000000	1.000000	0.000000	NaN	0.000000	0.230769	0.769231	0.500000	0.000000	0.000000	[0.769

Code for calculating the average cross-validation metrics:

1. Naïve Bayes Model

```
# 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 Model

```
#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 Model

```
# 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"])
```

Summary of the average cross-validation results from all three models:

```
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	BS	
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	0.000000	0.00
RANDOM FOREST	7.0	0.0	0.1	5.9	0.985915	0.000000	1.000000	0.014085	0.985915	1.000000	0.992908	0.992308	0.007692	0.992958	0.985915	0.984505	0.000000	0.00
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	0.769231	4.33

Saving the output table as an .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.close()
```

Full 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

data=pd.read_csv('wine.csv',header=None)
df=data.sample(frac=1)
print(df.head)

labels=df.iloc[:,0]
features=df.iloc[:,1:14]
X=features
y=labels
print('X: ',X)
print('y : ',y)

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.close()
```

Github Link:https://github.com/Aakashnjit/Aakash_Siricilla_DMFinalProject

Comparison/Discussion:
Random Forest emerges as the top performer among the trio of models. Here are the observations made while scrutinizing the evaluation metrics of the three models:
• The dataset was well-balanced with no missing values, resulting in similar accuracy and balanced accuracy (BACC) scores across the board.
• While Random Forest and Naïve Bayes yielded comparable results, some distinctions are noteworthy:

a. Naïve Bayes exhibited a higher rate of false positives, with 2 instances out of 10 folds, indicating a potential 20% occurrence of misclassification (e.g., predicting label 1 when it's actually 0).
b. Conversely, Random Forest showcased superior performance, with only one false negative detected in one fold, implying a lower 10% chance of misclassifying label 1 as 0.
c. Random Forest excelled across various metrics including accuracy, BACC, and F1-score. Both Random Forest and Naïve Bayes incurred similar execution times.
d. The implementation of LSTM with 4 hidden layers and 64 hidden units each, employing the sigmoid activation function and Adam optimizer with a learning rate of 0.0001, proved inefficient. The model struggled with computational demands, particularly on a laptop without GPU support. Although increasing the number of layers might have improved performance, the model underperformed across all folds of cross-validation.
- LSTM's poor performance is evident from its inability to detect true or false positives, exclusively predicting true negatives and false negatives. This indicates that LSTM assigns probabilities below 0.5 for all predictions, effectively labelling all data as 0.

e. Consequently, for the given wine dataset, Random Forest emerges as the preferred choice over Naïve Bayes and LSTM.

Conclusion:
Based on the comparison of evaluation metrics for the wine dataset, Random Forest emerges as the preferred choice over Naïve Bayes and LSTM.