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)

Evaluation metrics used were :

TPR, FPR, Recall, Precision,

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 first model 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 1st model

Model 1 : Naïve Bayes

```
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
from sklearn.metrics import multilabel_confusion_matrix

kf = KFold(n_splits=10)

 ############################ Model 1 Naive Bayes ##########################################

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

    #Naive bayes model
    model = GaussianNB()
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    print('Accuracy achieved by Naive bayes in this fold: ',accuracy_score(y_test, y_pred))
```

```
cnf_matrix = confusion_matrix(y_test, y_pred)
print('Confusion matrix :\n',cnf_matrix)
[[TN, FP],
[FN, TP]]=cnf_matrix
print('True Negative: ',TN)
print('False Positive: ',FP)
print('False Negative: ',FN)
print('True Positive: ',TP)

TPR=TP/(TP+FN)
print('TPR : ',TPR)
TNR=TN/(TN+FP)
print('TNR : ',TNR)
FPR=FP/(FP+TN)
print('FPR : ',FPR)
FNR=FN/(FN+TP)
print('FNR : ',FNR)
RECALL=TPR
print('Recall : ',RECALL)
PRECISION=TP/(TP+FP)
print('Precision : ',PRECISION)
F1_SCORE=(2*TP)/(2*TP+FP+FN)
print('F1_Score : ',F1_SCORE)
ACCURACY=(TP+TN)/(TP+FP+TN+FN)
print('Accuracy : ',ACCURACY)
ERROR_RATE=1-ACCURACY
print('Error rate : ',ERROR_RATE)
BACC=(TPR+TNR)/2
print('BACC : ',BACC)
```

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

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
print('BSS : ',BSS)

print('\n')
```

Output Snapshot:

```
    print('\n')

#     from sklearn.metrics import classification_report
#     print(classification_report(y_test,y_pred))
```

```
Accuracy achieved by Naive bayes in this fold:  1.0
Confusion matrix :
 [[ 3  0]
 [ 0 10]]
True Negative:  3
False Positive:  0
False Negative:  0
True Positive:  10
TPR :  1.0
TNR :  1.0
FPR :  0.0
FNR :  0.0
Recall :  1.0
Precision :  1.0
F1_Score :  1.0
Accuracy :  1.0
Error rate :  0.0
BACC :  1.0
TSS :  1.0
```

Complete Output:

```
Accuracy achieved by Naive bayes in 1st fold:  1.0
Confusion matrix :
 [[6 0]
 [0 7]]
True Negative:  6
```

```
False Positive:  0
False Negative:  0
True Positive:  7
TPR :  1.0
TNR :  1.0
FPR :  0.0
FNR :  0.0
Recall :  1.0
Precision :  1.0
F1_Score :  1.0
Accuracy :  1.0
Error rate :  0.0
BACC :  1.0
TSS :  1.0
HSS :  1.0
BS :  0.0
BSS :  0.0



Accuracy achieved by Naive bayes in 2nd fold:  1.0
Confusion matrix :
 [[9 0]
 [0 4]]
True Negative:  9
False Positive:  0
False Negative:  0
True Positive:  4
TPR :  1.0
TNR :  1.0
FPR :  0.0
FNR :  0.0
Recall :  1.0
Precision :  1.0
F1_Score :  1.0
Accuracy :  1.0
Error rate :  0.0
BACC :  1.0
TSS :  1.0
HSS :  1.0
BS :  0.0
BSS :  0.0


Accuracy achieved by Naive bayes in 3rd fold:  0.9230769230769231
Confusion matrix :
 [[4 0]
 [1 8]]
True Negative:  4
False Positive:  0
False Negative:  1
True Positive:  8
TPR :  0.8888888888888888
TNR :  1.0
```

```
FPR :  0.0
FNR :  0.1111111111111111
Recall :  0.8888888888888888
Precision :  1.0
F1_Score :  0.9411764705882353
Accuracy :  0.9230769230769231
Error rate :  0.07692307692307687
BACC :  0.9444444444444444
TSS :  0.8888888888888888
HSS :  0.8311688311688312
BS :  0.07692307692307693
BSS :  0.3611111111111111


Accuracy achieved by Naive bayes in 4th fold:  0.9230769230769231
Confusion matrix :
 [[7 1]
 [0 5]]
True Negative:  7
False Positive:  1
False Negative:  0
True Positive:  5
TPR :  1.0
TNR :  0.875
FPR :  0.125
FNR :  0.0
Recall :  1.0
Precision :  0.8333333333333334
F1_Score :  0.9090909090909091
Accuracy :  0.9230769230769231
Error rate :  0.07692307692307687
BACC :  0.9375
TSS :  0.875
HSS :  0.8433734939759037
BS :  0.07692307692307693
BSS :  0.32499999999999999


Accuracy achieved by Naive bayes in 5th fold:  1.0
Confusion matrix :
 [[4 0]
 [0 9]]
True Negative:  4
False Positive:  0
False Negative:  0
True Positive:  9
TPR :  1.0
TNR :  1.0
FPR :  0.0
FNR :  0.0
Recall :  1.0
Precision :  1.0
F1_Score :  1.0
Accuracy :  1.0
```

```
Error rate :  0.0
BACC :  1.0
TSS :  1.0
HSS :  1.0
BS :  0.0
BSS :  0.0


Accuracy achieved by Naive bayes in 6th fold:  1.0
Confusion matrix :
 [[6 0]
 [0 7]]
True Negative:  6
False Positive:  0
False Negative:  0
True Positive:  7
TPR :  1.0
TNR :  1.0
FPR :  0.0
FNR :  0.0
Recall :  1.0
Precision :  1.0
F1_Score :  1.0
Accuracy :  1.0
Error rate :  0.0
BACC :  1.0
TSS :  1.0
HSS :  1.0
BS :  0.0
BSS :  0.0


Accuracy achieved by Naive bayes in 7th fold:  0.9230769230769231
Confusion matrix :
 [[4 0]
 [1 8]]
True Negative:  4
False Positive:  0
False Negative:  1
True Positive:  8
TPR :  0.8888888888888888
TNR :  1.0
FPR :  0.0
FNR :  0.1111111111111111
Recall :  0.8888888888888888
Precision :  1.0
F1_Score :  0.9411764705882353
Accuracy :  0.9230769230769231
Error rate :  0.07692307692307687
BACC :  0.9444444444444444
TSS :  0.8888888888888888
HSS :  0.8311688311688312
BS :  0.07692307692307693
BSS :  0.3611111111111111
```

```
Accuracy achieved by Naive bayes in 8th fold:  1.0
Confusion matrix :
 [[6 0]
 [0 7]]
True Negative:  6
False Positive:  0
False Negative:  0
True Positive:  7
TPR :  1.0
TNR :  1.0
FPR :  0.0
FNR :  0.0
Recall :  1.0
Precision :  1.0
F1_Score :  1.0
Accuracy :  1.0
Error rate :  0.0
BACC :  1.0
TSS :  1.0
HSS :  1.0
BS :  0.0
BSS :  0.0


Accuracy achieved by Naive bayes in 9th fold:  1.0
Confusion matrix :
 [[5 0]
 [0 8]]
True Negative:  5
False Positive:  0
False Negative:  0
True Positive:  8
TPR :  1.0
TNR :  1.0
FPR :  0.0
FNR :  0.0
Recall :  1.0
Precision :  1.0
F1_Score :  1.0
Accuracy :  1.0
Error rate :  0.0
BACC :  1.0
TSS :  1.0
HSS :  1.0
BS :  0.0
BSS :  0.0


Accuracy achieved by Naive bayes in 10th fold:  0.9230769230769231
Confusion matrix :
 [[6 1]
 [0 6]]
```

```
True Negative:  6
False Positive:  1
False Negative:  0
True Positive:  6
TPR :  1.0
TNR :  0.8571428571428571
FPR :  0.14285714285714285
FNR :  0.0
Recall :  1.0
Precision :  0.8571428571428571
F1_Score :  0.923076923076923l
Accuracy :  0.9230769230769231
Error rate :  0.07692307692307687
BACC :  0.9285714285714286
TSS :  0.8571428571428572
HSS :  0.8470588235294118
BS :  0.07692307692307693
BSS :  0.30952380952380965
```

Model 2: Random Forest

```
#################### MOdel 2 Random Forest ######################################


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

    #Random Forest Model
    rf= RandomForestClassifier(n_estimators=20, random_state=0)
    rf.fit(X_train, y_train)
    y_predrf=rf.predict(X_test)

    print('Accuracy achieved by Random Forest in this fold : ',accuracy_score(y_test, y_predrf))

    cnf_matrix = confusion_matrix(y_test, y_predrf)
    print('Confusion matrix :\n',cnf_matrix)
    [[TN, FP],
    [FN, TP]]=cnf_matrix
    print('True Negative: ',TN)
    print('False Positive: ',FP)
    print('False Negative: ',FN)
    print('True Positive: ',TP)

    TPR=TP/(TP+FN)
    print('TPR : ',TPR)
    TNR=TN/(TN+FP)
    print('TNR : ',TNR)
```

```
FPR=FP/(FP+TN)
print('FPR : ',FPR)
FNR=FN/(FN+TP)
print('FNR : ',FNR)
RECALL=TPR
print('Recall : ',RECALL)
PRECISION=TP/(TP+FP)
print('Precision : ',PRECISION)
F1_SCORE=(2*TP)/(2*TP+FP+FN)
print('F1_Score : ',F1_SCORE)
ACCURACY=(TP+TN)/(TP+FP+TN+FN)
print('Accuracy : ',ACCURACY)
ERROR_RATE=1-ACCURACY
print('Error rate : ',ERROR_RATE)
BACC=(TPR+TNR)/2
print('BACC : ',BACC)
TSS=TPR-FPR
print('TSS : ',TSS)
HSS=2*(TP*TN-FP*FN)/(((TP+FN)*(FN+TN))+((TP+FP)*(FP+TN)))
print('HSS : ',HSS)
sum_y=0
for n in range(len(y_test)):
    sum_y+=(y_test[n]-y_pred[n])**2
BS=sum_y/len(y_test)
print('BS : ',BS)
```

```
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
print('BSS : ',BSS)
```

Output Snapshot:

```
    temp=temp/len(y_test)
    BSS=BS/temp
    print('BSS : ',BSS)
```

```
Accuracy achieved by Random Forest in this fold :  1.0
Confusion matrix :
 [[ 3  0]
 [ 0 10]]
True Negative:  3
False Positive:  0
False Negative:  0
True Positive:  10
TPR :  1.0
TNR :  1.0
FPR :  0.0
FNR :  0.0
Recall :  1.0
Precision :  1.0
F1_Score :  1.0
Accuracy :  1.0
Error rate :  0.0
BACC :  1.0
TSS :  1.0
```

Output:

```
Accuracy achieved by Random Forest in 1st fold :  1.0
Confusion matrix :
 [[6 0]
 [0 7]]
True Negative:  6
False Positive:  0
False Negative:  0
True Positive:  7
TPR :  1.0
TNR :  1.0
FPR :  0.0
FNR :  0.0
Recall :  1.0
Precision :  1.0
F1_Score :  1.0
Accuracy :  1.0
Error rate :  0.0
BACC :  1.0
TSS :  1.0
HSS :  1.0
BS :  0.6153846153846154
BSS :  2.476190476190477
Accuracy achieved by Random Forest in 2nd fold :  1.0
Confusion matrix :
 [[9 0]
 [0 4]]
True Negative:  9
False Positive:  0
False Negative:  0
True Positive:  4
TPR :  1.0
TNR :  1.0
FPR :  0.0
FNR :  0.0
```

```
Recall :  1.0
Precision :  1.0
F1_Score :  1.0
Accuracy :  1.0
Error rate :  0.0
BACC :  1.0
TSS :  1.0
HSS :  1.0
BS :  0.38461538461538464
BSS :  1.8055555555555556
Accuracy achieved by Random Forest in 3rd fold :  0.9230769230769231
Confusion matrix :
 [[4 0]
 [1 8]]
True Negative:  4
False Positive:  0
False Negative:  1
True Positive:  8
TPR :  0.8888888888888888
TNR :  1.0
FPR :  0.0
FNR :  0.1111111111111111
Recall :  0.8888888888888888
Precision :  1.0
F1_Score :  0.9411764705882353
Accuracy :  0.9230769230769231
Error rate :  0.07692307692307687
BACC :  0.9444444444444444
TSS :  0.8888888888888888
HSS :  0.8311688311688312
BS :  0.46153846153846156
BSS :  2.1666666666666665
Accuracy achieved by Random Forest in 4th fold :  0.9230769230769231
Confusion matrix :
 [[7 1]
 [0 5]]
True Negative:  7
False Positive:  1
False Negative:  0
True Positive:  5
TPR :  1.0
TNR :  0.875
FPR :  0.125
FNR :  0.0
Recall :  1.0
Precision :  0.8333333333333334
F1_Score :  0.9090909090909091
Accuracy :  0.9230769230769231
Error rate :  0.07692307692307687
BACC :  0.9375
TSS :  0.875
HSS :  0.8433734939759037
BS :  0.3076923076923077
BSS :  1.2999999999999996
```

```
Accuracy achieved by Random Forest in 5th fold :  1.0
Confusion matrix :
 [[4 0]
 [0 9]]
True Negative:  4
False Positive:  0
False Negative:  0
True Positive:  9
TPR :  1.0
TNR :  1.0
FPR :  0.0
FNR :  0.0
Recall :  1.0
Precision :  1.0
F1_Score :  1.0
Accuracy :  1.0
Error rate :  0.0
BACC :  1.0
TSS :  1.0
HSS :  1.0
BS :  0.46153846153846156
BSS :  2.1666666666666665
Accuracy achieved by Random Forest in 6th fold :  1.0
Confusion matrix :
 [[6 0]
 [0 7]]
True Negative:  6
False Positive:  0
False Negative:  0
True Positive:  7
TPR :  1.0
TNR :  1.0
FPR :  0.0
FNR :  0.0
Recall :  1.0
Precision :  1.0
F1_Score :  1.0
Accuracy :  1.0
Error rate :  0.0
BACC :  1.0
TSS :  1.0
HSS :  1.0
BS :  0.3076923076923077
BSS :  1.2380952380952384
Accuracy achieved by Random Forest in 7th fold :  0.9230769230769231
Confusion matrix :
 [[4 0]
 [1 8]]
True Negative:  4
False Positive:  0
False Negative:  1
True Positive:  8
TPR :  0.8888888888888888
TNR :  1.0
```

```
FPR :  0.0
FNR :  0.1111111111111111
Recall :  0.8888888888888888
Precision :  1.0
F1_Score :  0.9411764705882353
Accuracy :  0.9230769230769231
Error rate :  0.07692307692307687
BACC :  0.9444444444444444
TSS :  0.8888888888888888
HSS :  0.8311688311688312
BS :  0.3076923076923077
BSS :  1.4444444444444444
Accuracy achieved by Random Forest in 8th fold :  1.0
Confusion matrix :
 [[6 0]
 [0 7]]
True Negative:  6
False Positive:  0
False Negative:  0
True Positive:  7
TPR :  1.0
TNR :  1.0
FPR :  0.0
FNR :  0.0
Recall :  1.0
Precision :  1.0
F1_Score :  1.0
Accuracy :  1.0
Error rate :  0.0
BACC :  1.0
TSS :  1.0
HSS :  1.0
BS :  0.46153846153846156
BSS :  1.8571428571428577
Accuracy achieved by Random Forest in 9th fold :  1.0
Confusion matrix :
 [[5 0]
 [0 8]]
True Negative:  5
False Positive:  0
False Negative:  0
True Positive:  8
TPR :  1.0
TNR :  1.0
FPR :  0.0
FNR :  0.0
Recall :  1.0
Precision :  1.0
F1_Score :  1.0
Accuracy :  1.0
Error rate :  0.0
BACC :  1.0
TSS :  1.0
HSS :  1.0
```

```
BS :  0.5384615384615384
BSS :  2.2749999999999995
Accuracy achieved by Random Forest in 10th fold :  1.0
Confusion matrix :
 [[7 0]
 [0 6]]
True Negative:  7
False Positive:  0
False Negative:  0
True Positive:  6
TPR :  1.0
TNR :  1.0
FPR :  0.0
FNR :  0.0
Recall :  1.0
Precision :  1.0
F1_Score :  1.0
Accuracy :  1.0
Error rate :  0.0
BACC :  1.0
TSS :  1.0
HSS :  1.0
BS :  0.07692307692307693
BSS :  0.30952380952380965
```

Model 3 : LSTM

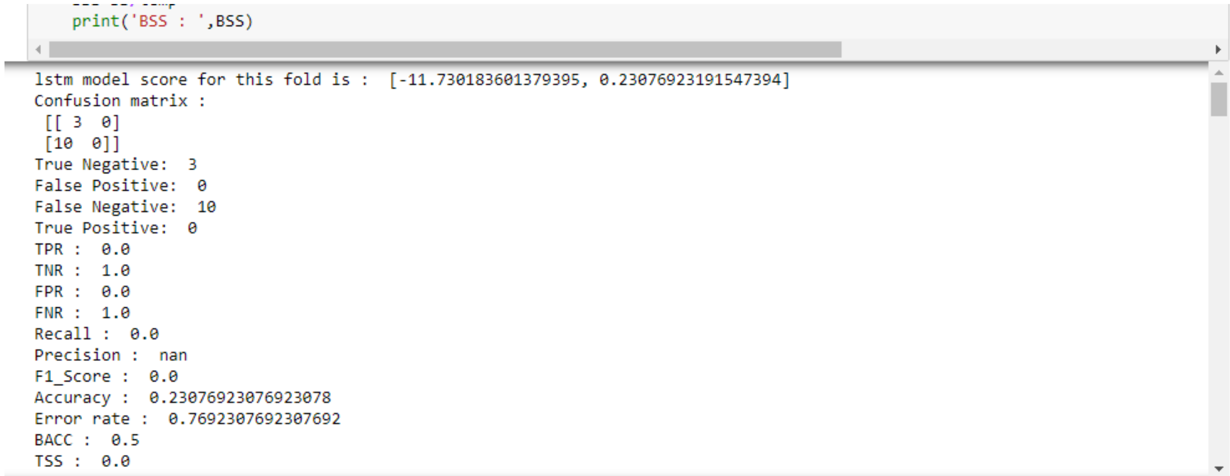
LSTM-Long Short Term Memory

```
###################################### Model 3 LSTM (Deep Learning) #########################################

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]

    #reshape the data to match 3 dimension for LSTM layers.
    X_train = X_train.reshape(X_train.shape[0], X_train.shape[1],1)
    X_test = 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_test.shape[1],X_test.shap
    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))
    #when we return the sequence, we change the shape, so last layer should not return sequnce to the Dense layer
    lstm_model.add(tf.keras.layers.Flatten())
    lstm_model.add(tf.keras.layers.Dense(1, activation='softmax'))

# Compile the Model
    optimizer = tf.keras.optimizers.Adam(learning_rate=0.0001)
    lstm_model.compile(optimizer='adam', loss="binary_crossentropy", metrics=['accuracy'])
    #lstm_model.summary()
    lstm_model.fit(X_train, y_train,epochs=5,batch_size=2, verbose = 0)
```

```
y_pred = lstm_model.predict(X_test)
#print(y_pred)
score = lstm_model.evaluate(X_test, y_test,verbose=0)
print('lstm model score for this fold is : ',score)

cnf_matrix = confusion_matrix(y_test, y_pred)
print('Confusion matrix :\n',cnf_matrix)
[[TN, FP],
[FN, TP]]=cnf_matrix
print('True Negative: ',TN)
print('False Positive: ',FP)
print('False Negative: ',FN)
print('True Positive: ',TP)

TPR=TP/(TP+FN)
print('TPR : ',TPR)
TNR=TN/(TN+FP)
print('TNR : ',TNR)
FPR=FP/(FP+TN)
print('FPR : ',FPR)
FNR=FN/(FN+TP)
print('FNR : ',FNR)
RECALL=TPR
print('Recall : ',RECALL)
PRECISION=TP/(TP+FP)
print('Precision : ',PRECISION)
F1_SCORE=(2*TP)/(2*TP+FP+FN)
print('F1_Score : ',F1_SCORE)
```

```
ACCURACY=(TP+TN)/(TP+FP+TN+FN)
print('Accuracy : ',ACCURACY)
ERROR_RATE=1-ACCURACY
print('Error rate : ',ERROR_RATE)
BACC=(TPR+TNR)/2
print('BACC : ',BACC)
TSS=TPR-FPR
print('TSS : ',TSS)
HSS=2*(TP*TN-FP*FN)/(((TP+FN)*(FN+TN))+((TP+FP)*(FP+TN)))
print('HSS : ',HSS)
sum_y=0
for n in range(len(y_test)):
    sum_y+=(y_test[n]-y_pred[n])**2
BS=sum_y/len(y_test)
print('BS : ',BS)

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
print('BSS : ',BSS)
```

Snapshot of output:


```
print('BSS : ',BSS)
```
```
lstm model score for this fold is :  [-11.730183601379395, 0.23076923191547394]
Confusion matrix :
 [[ 3  0]
 [10  0]]
True Negative:  3
False Positive:  0
False Negative:  10
True Positive:  0
TPR :  0.0
TNR :  1.0
FPR :  0.0
FNR :  1.0
Recall :  0.0
Precision :  nan
F1_Score :  0.0
Accuracy :  0.23076923076923078
Error rate :  0.7692307692307692
BACC :  0.5
TSS :  0.0
```

Complete Output:

```
Evaluation metrics for lstm model is :
 [-5.865091800689697, 0.6153846383094788]
Confusion matrix :
 [[8 0]
 [5 0]]
True Negative:  8
False Positive:  0
False Negative:  5
True Positive:  0
TPR :  0.0
TNR :  1.0
FPR :  0.0
FNR :  1.0
Recall :  0.0
Precision :  nan
F1_Score :  0.0
Accuracy :  0.6153846153846154
Error rate :  0.3846153846153846
BACC :  0.5
TSS :  0.0
HSS :  0.0
BS :  [0.3846154]
BSS :  [1.625]


Evaluation metrics for lstm model is :
 [-9.384146690368652, 0.38461539149284363]
```

```
Confusion matrix :
 [[5 0]
 [8 0]]
True Negative:  5
False Positive:  0
False Negative:  8
True Positive:  0
TPR :  0.0
TNR :  1.0
FPR :  0.0
FNR :  1.0
Recall :  0.0
Precision :  nan
F1_Score :  0.0
Accuracy :  0.38461538461538464
Error rate :  0.6153846153846154
BACC :  0.5
TSS :  0.0
HSS :  0.0
BS :  [0.61538464]
BSS :  [2.6000001]


Evaluation metrics for lstm model is :
 [-10.557165145874023, 0.3076923191547394]
Confusion matrix :
 [[4 0]
 [9 0]]
True Negative:  4
False Positive:  0
False Negative:  9
True Positive:  0
TPR :  0.0
TNR :  1.0
FPR :  0.0
FNR :  1.0
Recall :  0.0
Precision :  nan
F1_Score :  0.0
Accuracy :  0.3076923076923077
Error rate :  0.6923076923076923
BACC :  0.5
TSS :  0.0
HSS :  0.0
```

```
BS :  [0.6923077]
BSS :  [3.2500002]


Evaluation metrics for lstm model is :
 [-7.03810977935791, 0.5384615659713745]
Confusion matrix :
 [[7 0]
 [6 0]]
True Negative:  7
False Positive:  0
False Negative:  6
True Positive:  0
TPR :  0.0
TNR :  1.0
FPR :  0.0
FNR :  1.0
Recall :  0.0
Precision :  nan
F1_Score :  0.0
Accuracy :  0.5384615384615384
Error rate :  0.46153846153846156
BACC :  0.5
TSS :  0.0
HSS :  0.0
BS :  [0.46153846]
BSS :  [1.8571428]


Evaluation metrics for lstm model is :
 [-9.384146690368652, 0.38461539149284363]
Confusion matrix :
 [[5 0]
 [8 0]]
True Negative:  5
False Positive:  0
False Negative:  8
True Positive:  0
TPR :  0.0
TNR :  1.0
FPR :  0.0
FNR :  1.0
Recall :  0.0
Precision :  nan
```

```
F1_Score :  0.0
Accuracy :  0.38461538461538464
Error rate :  0.6153846153846154
BACC :  0.5
TSS :  0.0
HSS :  0.0
BS :  [0.61538464]
BSS :  [2.6000001]


Evaluation metrics for lstm model is :
 [-7.03810977935791, 0.5384615659713745]
Confusion matrix :
 [[7 0]
 [6 0]]
True Negative:  7
False Positive:  0
False Negative:  6
True Positive:  0
TPR :  0.0
TNR :  1.0
FPR :  0.0
FNR :  1.0
Recall :  0.0
Precision :  nan
F1_Score :  0.0
Accuracy :  0.5384615384615384
Error rate :  0.46153846153846156
BACC :  0.5
TSS :  0.0
HSS :  0.0
BS :  [0.46153846]
BSS :  [1.8571428]


Evaluation metrics for lstm model is :
 [-7.03810977935791, 0.5384615659713745]
Confusion matrix :
 [[7 0]
 [6 0]]
True Negative:  7
False Positive:  0
False Negative:  6
True Positive:  0
```

```
TPR :  0.0
TNR :  1.0
FPR :  0.0
FNR :  1.0
Recall :  0.0
Precision :  nan
F1_Score :  0.0
Accuracy :  0.5384615384615384
Error rate :  0.46153846153846156
BACC :  0.5
TSS :  0.0
HSS :  0.0
BS :  [0.46153846]
BSS :  [1.8571428]


Evaluation metrics for lstm model is :
 [-8.211128234863281, 0.4615384638309479]
Confusion matrix :
 [[6 0]
 [7 0]]
True Negative:  6
False Positive:  0
False Negative:  7
True Positive:  0
TPR :  0.0
TNR :  1.0
FPR :  0.0
FNR :  1.0
Recall :  0.0
Precision :  nan
F1_Score :  0.0
Accuracy :  0.46153846153846156
Error rate :  0.5384615384615384
BACC :  0.5
TSS :  0.0
HSS :  0.0
BS :  [0.53846157]
BSS :  [2.1666667]


Evaluation metrics for lstm model is :
 [-8.211128234863281, 0.4615384638309479]
Confusion matrix :
```

```
 [[6 0]
 [7 0]]
True Negative:  6
False Positive:  0
False Negative:  7
True Positive:  0
TPR :  0.0
TNR :  1.0
FPR :  0.0
FNR :  1.0
Recall :  0.0
Precision :  nan
F1_Score :  0.0
Accuracy :  0.46153846153846156
Error rate :  0.5384615384615384
BACC :  0.5
TSS :  0.0
HSS :  0.0
BS :  [0.53846157]
BSS :  [2.1666667]


Evaluation metrics for lstm model is :
 [-10.557165145874023, 0.3076923191547394]
Confusion matrix :
 [[4 0]
 [9 0]]
True Negative:  4
False Positive:  0
False Negative:  9
True Positive:  0
TPR :  0.0
TNR :  1.0
FPR :  0.0
FNR :  1.0
Recall :  0.0
Precision :  nan
F1_Score :  0.0
Accuracy :  0.3076923076923077
Error rate :  0.6923076923076923
BACC :  0.5
TSS :  0.0
HSS :  0.0
BS :  [0.6923077]
```

BSS : [3.2500002]

Evaluation metrics for the average 10 fold cross validation using LSTM :

```
TN_AVG :  5.9
TP_AVG :  0.0
FN_AVG :  7.1
FP_AVG :  0.0
TPR :  0.0
TNR :  1.0
FPR :  0.0
FNR :  1.0
Recall :  0.0
Precision :  nan
F1_Score :  0.0
Accuracy :  0.4538461538461539
Error rate :  0.5461538461538461
BACC :  0.5
TSS :  0.0
HSS :  0.0
BS :  [0.6923077]
BSS :  [3.2500002]
```

Complete Source code:

```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.python.ops.math_ops import reduce_prod
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)
def evaluation_metrics(TP,TN,FP,FN):
    TPR=TP/(TP+FN)
    print('TPR : ',TPR)
    TNR=TN/(TN+FP)
    print('TNR : ',TNR)
    FPR=FP/(FP+TN)
    print('FPR : ',FPR)
    FNR=FN/(FN+TP)
    print('FNR : ',FNR)
    RECALL=TPR
    print('Recall : ',RECALL)
    PRECISION=TP/(TP+FP)
    print('Precision : ',PRECISION)
    F1_SCORE=(2*TP)/(2*TP+FP+FN)
    print('F1_Score : ',F1_SCORE)
    ACCURACY=(TP+TN)/(TP+FP+TN+FN)
    print('Accuracy : ',ACCURACY)
    ERROR_RATE=1-ACCURACY
    print('Error rate : ',ERROR_RATE)
    BACC=(TPR+TNR)/2
    print('BACC : ',BACC)
    TSS=TPR-FPR
    print('TSS : ',TSS)
    HSS=2*(TP*TN-FP*FN)/(((TP+FN)*(FN+TN))+((TP+FP)*(FP+TN)))
    print('HSS : ',HSS)
    sum_y=0
    for n in range(len(y_test)):
        sum_y+=(y_test[n]-y_pred[n])**2
```

```
    BS=sum_y/len(y_test)
    print('BS : ',BS)

    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
    print('BSS : ',BSS)
    print('\n')
    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
from sklearn.metrics import multilabel_confusion_matrix

kf = KFold(n_splits=10)

 ################# Model 1 Naive Bayes #####################
TN_TOTAL=0
TP_TOTAL=0
FP_TOTAL=0
FN_TOTAL=0

for train_index, test_index in kf.split(X):
    #print("TRAIN:", train_index, "TEST:", test_index)
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
    model = GaussianNB()
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    print('Evaluation metrics for each fold :\n')
    cnf_matrix = confusion_matrix(y_test, y_pred)
    print('Confusion matrix :\n',cnf_matrix)
    [[TNs, FPs],
```

```
    [FNs, TPs]]=cnf_matrix
    print('True Negative: ',TNs)
    print('False Positive: ',FPs)
    print('False Negative: ',FNs)
    print('True Positive: ',TPs)
    evaluation_metrics(TPs,TNs,FPs,FNs)
    TN_TOTAL+=TNs
    TP_TOTAL+=TPs
    FP_TOTAL+=FPs
    FN_TOTAL+=FNs

TN=TN_TOTAL/10
TP=TP_TOTAL/10
FN=FN_TOTAL/10
FP=FP_TOTAL/10
print('Evaluation metrics for the average 10 fold cross validation using Naiv
e Bayes : \n')
print('TN_AVG : ',TN)
print('TP_AVG : ',TP)
print('FN_AVG : ',FN)
print('FP_AVG : ',FP)

TPR=TP/(TP+FN)
print('TPR AVG : ',TPR)
TNR=TN/(TN+FP)
print('TNR AVG: ',TNR)
FPR=FP/(FP+TN)
print('FPR AVG: ',FPR)
FNR=FN/(FN+TP)
print('FNR AVG: ',FNR)
RECALL=TPR
print('Recall : ',RECALL)
PRECISION=TP/(TP+FP)
print('Precision : ',PRECISION)
F1_SCORE=(2*TP)/(2*TP+FP+FN)
print('F1_Score : ',F1_SCORE)
ACCURACY=(TP+TN)/(TP+FP+TN+FN)
print('Accuracy : ',ACCURACY)
ERROR_RATE=1-ACCURACY
print('Error rate : ',ERROR_RATE)
BACC=(TPR+TNR)/2
print('BACC : ',BACC)
TSS=TPR-FPR
print('TSS : ',TSS)
```

```
HSS=2*(TP*TN-FP*FN)/(((TP+FN)*(FN+TN))+((TP+FP)*(FP+TN)))
print('HSS : ',HSS)
sum_y=0
for n in range(len(y_test)):
    sum_y+=(y_test[n]-y_pred[n])**2
BS=sum_y/len(y_test)
print('BS : ',BS)

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
print('BSS : ',BSS)
print('\n')

df1=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"])
#################### MOdel 2 Random Forest ######################
TN_TOTAL=0
TP_TOTAL=0
FP_TOTAL=0
```

```
FN_TOTAL=0

for train_index, test_index in kf.split(X):
    #print("TRAIN:", train_index, "TEST:", test_index)
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
    rf= RandomForestClassifier(n_estimators=20, random_state=0)
    rf.fit(X_train, y_train)
    y_predrf=rf.predict(X_test)
    print('Evaluation metrics achieved by Random Forest in each fold : ')
    cnf_matrix = confusion_matrix(y_test, y_predrf)
    print('Confusion matrix :\n',cnf_matrix)
    [[TNs, FPs],
    [FNs, TPs]]=cnf_matrix
    print('True Negative: ',TNs)
    print('False Positive: ',FPs)
    print('False Negative: ',FNs)
    print('True Positive: ',TPs)
    evaluation_metrics(TPs,TNs,FPs,FNs)
    TN_TOTAL+=TNs
    TP_TOTAL+=TPs
    FP_TOTAL+=FPs
    FN_TOTAL+=FNs

TN=TN_TOTAL/10
TP=TP_TOTAL/10
FN=FN_TOTAL/10
FP=FP_TOTAL/10
print('Evaluation metrics for the average 10 fold cross validation using Rand
om Forest : \n')
print('TN_AVG : ',TN)
print('TP_AVG : ',TP)
print('FN_AVG : ',FN)
print('FP_AVG : ',FP)

TPR=TP/(TP+FN)
print('TPR AVG : ',TPR)
TNR=TN/(TN+FP)
print('TNR AVG: ',TNR)
FPR=FP/(FP+TN)
print('FPR AVG: ',FPR)
FNR=FN/(FN+TP)
print('FNR AVG: ',FNR)
RECALL=TPR
```

```
print('Recall : ',RECALL)
PRECISION=TP/(TP+FP)
print('Precision : ',PRECISION)
F1_SCORE=(2*TP)/(2*TP+FP+FN)
print('F1_Score : ',F1_SCORE)
ACCURACY=(TP+TN)/(TP+FP+TN+FN)
print('Accuracy : ',ACCURACY)
ERROR_RATE=1-ACCURACY
print('Error rate : ',ERROR_RATE)
BACC=(TPR+TNR)/2
print('BACC : ',BACC)
TSS=TPR-FPR
print('TSS : ',TSS)
HSS=2*(TP*TN-FP*FN)/(((TP+FN)*(FN+TN))+((TP+FP)*(FP+TN)))
print('HSS : ',HSS)
sum_y=0
for n in range(len(y_test)):
    sum_y+=(y_test[n]-y_pred[n])**2
BS=sum_y/len(y_test)
print('BS : ',BS)

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
print('BSS : ',BSS)
print('\n')

df2=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"])

import warnings
warnings.filterwarnings("ignore")
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'

try:
    tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.ERROR)
except Exception as e:
    print('')
################# Model 3 LSTM (Deep Learning) #####################
TN_TOTAL=0
TP_TOTAL=0
FP_TOTAL=0
FN_TOTAL=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]

    #reshape the data to match 3 dimension for LSTM layers.
    X_train = X_train.reshape(X_train.shape[0], X_train.shape[1],1)
    X_test = 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_stat
e=False,input_shape=(X_test.shape[1],X_test.shape[2]))) # Instruction: in cas
e you change the data format
    lstm_model.add(tf.keras.layers.LSTM(64, return_sequences=True, return_sta
te=False))
```

```
    lstm_model.add(tf.keras.layers.LSTM(64, return_sequences=True, return_sta
te=False))
    lstm_model.add(tf.keras.layers.LSTM(64, return_sequences=True, return_sta
te=False))
    lstm_model.add(tf.keras.layers.LSTM(64, return_sequences=True, return_sta
te=False))
    #when we return the sequence, we change the shape, so last layer should n
ot return sequnce to the Dense layer
    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=0.0001)
    lstm_model.compile(optimizer='adam', loss="binary_crossentropy", metrics=
['accuracy'])
    #lstm_model.summary()
    lstm_model.fit(X_train, y_train,epochs=100,batch_size=1, verbose = 0)
    #more epochs the better and we can add EarlyStopping in callbacks so that
 if the accuracy is not improving, it will stop.
    y_pred = lstm_model.predict(X_test)
    #print(y_pred)
    score = lstm_model.evaluate(X_test, y_test,verbose=0)
    print('Evaluation metrics for lstm model is : \n',score)
    cnf_matrix = confusion_matrix(y_test, y_pred)
    print('Confusion matrix :\n',cnf_matrix)
    [[TNs, FPs],
    [FNs, TPs]]=cnf_matrix
    print('True Negative: ',TNs)
    print('False Positive: ',FPs)
    print('False Negative: ',FNs)
    print('True Positive: ',TPs)
    evaluation_metrics(TPs,TNs,FPs,FNs)
    TN_TOTAL+=TNs
    TP_TOTAL+=TPs
    FP_TOTAL+=FPs
    FN_TOTAL+=FNs

TN=TN_TOTAL/10
TP=TP_TOTAL/10
FN=FN_TOTAL/10
FP=FP_TOTAL/10
print('Evaluation metrics for the average 10 fold cross validation using LSTM
 : \n')
print('TN_AVG : ',TN)
```

```
print('TP_AVG : ',TP)
print('FN_AVG : ',FN)
print('FP_AVG : ',FP)

TPR=TP/(TP+FN)
print('TPR AVG : ',TPR)
TNR=TN/(TN+FP)
print('TNR AVG: ',TNR)
FPR=FP/(FP+TN)
print('FPR AVG: ',FPR)
FNR=FN/(FN+TP)
print('FNR AVG: ',FNR)
RECALL=TPR
print('Recall : ',RECALL)
PRECISION=TP/(TP+FP)
print('Precision : ',PRECISION)
F1_SCORE=(2*TP)/(2*TP+FP+FN)
print('F1_Score : ',F1_SCORE)
ACCURACY=(TP+TN)/(TP+FP+TN+FN)
print('Accuracy : ',ACCURACY)
ERROR_RATE=1-ACCURACY
print('Error rate : ',ERROR_RATE)
BACC=(TPR+TNR)/2
print('BACC : ',BACC)
TSS=TPR-FPR
print('TSS : ',TSS)
HSS=2*(TP*TN-FP*FN)/(((TP+FN)*(FN+TN))+((TP+FP)*(FP+TN)))
print('HSS : ',HSS)
sum_y=0
for n in range(len(y_test)):
    sum_y+=(y_test[n]-y_pred[n])**2
BS=sum_y/len(y_test)
print('BS : ',BS)

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
print('BSS : ',BSS)
```

```
print('\n')

df3=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"])

# Creating Table to have all the evaluation metrics in one place.
frames=[df1,df2,df3]
result=pd.concat(frames)
result_transpose=result.T
print(result_transpose)
```

Result Table:

	Naive Bayes	Random Forest	LSTM
TP	7.100000	7.000000	0.000000
FP	0.200000	0.000000	0.000000
FN	0.000000	0.100000	7.100000
TN	5.700000	5.900000	5.900000
TPR	1.000000	0.985915	0.000000
FPR	0.033898	0.000000	0.000000
TNR	0.966102	1.000000	1.000000
FNR	0.000000	0.014085	1.000000
RECALL	1.000000	0.985915	0.000000
PRECISION	0.972603	1.000000	NaN
F1_SCORE	0.986111	0.992908	0.000000
Accuracy	0.984615	0.992308	0.453846
Error rate	0.015385	0.007692	0.546154
BACC	0.983051	0.992958	0.500000
TSS	0.966102	0.985915	0.000000
HSS	0.968877	0.984505	0.000000
BS	0.076923	0.076923	0.538462
BSS	0.309524	0.309524	2.166667

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.