NJIT
New Jersey Institute of Technology

CS 634

FinalTerm project implementation

Himani Patel

Hpp5@njit.edu

Table Of Contents:

Random Forest Classifier: **It is an ensemble tree-based learning algorithm. The Random Forest Classifier is a set of decision trees from randomly selected subsets of the training set. It aggregates the votes from different decision trees to decide the final class of the test object.**
Random Forest Prediction for a classification problem:
f(x) = majority vote of all predicted classes over B trees

a) Importing python libraries and loading our dataset into dataframe:

import numpy as np
import matplotlib.pyplot as mlt
import pandas as pd
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import KFold, cross_validate, cross_val_predict
from sklearn.metrics import confusion_matrix,make_scorer
from sklearn.neighbors import KNeighborsClassifier

```
In [218]: import numpy as np
          import matplotlib.pyplot as mlt
          import pandas as pd
          from sklearn import preprocessing
          from sklearn.model_selection import train_test_split
          from sklearn.preprocessing import StandardScaler
          from sklearn.ensemble import RandomForestClassifier
          from sklearn.model_selection import KFold, cross_validate, cross_val_predict
          from sklearn.metrics import confusion_matrix,make_scorer
          from sklearn.neighbors import KNeighborsClassifier
```

b) Loading the CSV file.
I've downloaded the dataset file from the:
https://www.kaggle.com/jeffreybraun/chipotle-locations

```
In [219]: dataset = pd.read_csv('/Users/himanipatel/Downloads/archive/chipotle_stores.csv')
```

c) With the dataset.info(), we'll get to know how many columns and rows will be there in the dataset.

```
In [220]: dataset.head()
Out[220]:
```

	state	location	address	latitude	longitude
0	Alabama	Auburn	346 W Magnolia Ave Auburn, AL 36832 US	32.606813	-85.487328
1	Alabama	Birmingham	300 20th St S Birmingham, AL 35233 US	33.509721	-86.802756
2	Alabama	Birmingham	3220 Morrow Rd Birmingham, AL 35235 US	33.595581	-86.647437
3	Alabama	Birmingham	4719 Highway 280 Birmingham, AL 35242 US	33.422582	-86.698279
4	Alabama	Cullman	1821 Cherokee Ave SW Cullman, AL 35055 US	34.154134	-86.841220

d) Dropping the dataset 'address' and 'state'.

```
In [7]: A=dataset.drop(['address'],axis=1)
        B=dataset['state'].copy()

In [8]: A.head()
Out[8]:
```

	state	location	latitude	longitude
0	Alabama	Auburn	32.606813	-85.487328
1	Alabama	Birmingham	33.509721	-86.802756
2	Alabama	Birmingham	33.595581	-86.647437
3	Alabama	Birmingham	33.422582	-86.698279
4	Alabama	Cullman	34.154134	-86.841220

```
In [9]: B.head()
Out[9]: 0    Alabama
        1    Alabama
        2    Alabama
        3    Alabama
        4    Alabama
        Name: state, dtype: object
```

```
In [10]: A["latitude"]=A["latitude"].fillna(A["latitude"].mean())

In [11]: A["longitude"]=A["longitude"].fillna(A["longitude"].mode()[0])

In [12]: A.info()
         <class 'pandas.core.frame.DataFrame'>
         RangeIndex: 2629 entries, 0 to 2628
         Data columns (total 4 columns):
          #   Column     Non-Null Count  Dtype
         ---  ------     --------------  -----
          0   state      2629 non-null   object
          1   location   2629 non-null   object
          2   latitude   2629 non-null   float64
          3   longitude  2629 non-null   float64
         dtypes: float64(2), object(2)
         memory usage: 82.3+ KB
```

Create a random forest classifier

```
In [229]: ##feature scaling

          num_col = A._get_numeric_data().columns
          col=A.columns
          cat_col=list(set(col) - set(num_col))

          for col in cat_col:
              le = preprocessing.LabelEncoder()
              A[col] = le.fit_transform(A[col])
          A.shape

Out[229]: (2629, 4)
```

In [230]:
```
##splitting the dataset into the training set and the test set

A_train, A_test, B_train, B_test = train_test_split(A, B, test_size = 0.30, random_state = 0)
```

In [231]:
```
sc = StandardScaler()
A_train = sc.fit_transform(A_train)
A_test = sc.transform(A_test)
```

e) Creating a Random Forest Classification model and fitting it to the training data and also predicting the test set result and and from sklearn.metrics importing confusion_matrix,make_scorer

In [232]:
```
##training the Random Forest model on the training set

rndm_frst=RandomForestClassifier(n_estimators=10,criterion='entropy',random_state=0)
rndm_frst.fit(A_train , B_train)
```

Out[232]:
```
RandomForestClassifier(criterion='entropy', n_estimators=10, random_state=0)
```

In [233]:
```
##predecting the test set results
B_rndm = rndm_frst.predict(A_test)
```

In [234]:
```
##making the confusion matrix

cm = confusion_matrix(B_test, B_rndm)
print(cm)
```

```
[[ 7  0  0 ...  0  0  0]
 [ 0 22  0 ...  0  0  0]
 [ 0  0  1 ...  0  0  0]
 ...
 [ 0  0  0 ...  2  0  0]
 [ 0  0  0 ...  0  7  0]
 [ 0  0  0 ...  0  0  0]]
```

f)Applying the logic of the training set results in the random forest classifier

```
def tn(B_test,B_rndm): return confusion_matrix(B_test,B_rndm)[0,0]
def tn(B_test,B_rndm): return confusion_matrix(B_test,B_rndm)[0,1]
def tn(B_test,B_rndm): return confusion_matrix(B_test,B_rndm)[1,1]
def tn(B_test,B_rndm): return confusion_matrix(B_test,B_rndm)[1,0]

def tpr(B_test,B_rndm):
    no_tp = confusion_matrix(B_test,B_rndm)[1,1]
    no_fn = confusion_matrix(B_test,B_rndm)[1,0]
    return round((no_tp / (no_tp + no_fn)),2)

def tnr(B_test,B_rndm):
    no_tn = confusion_matrix(B_test,B_rndm)[0,0]
    no_fp = confusion_matrix(B_test,B_rndm)[0,1]

    return round((no_tn / (no_tn + no_fp)),2)

def fpr(B_test,B_rndm):
    no_tn = confusion_matrix(B_test,B_rndm)[0,0]
    no_fp = confusion_matrix(B_test,B_rndm)[0,1]

    return round((no_fp / (no_tn + no_fp)),2)

def fnr(B_test,B_rndm):
    no_tp = confusion_matrix(B_test,B_rndm)[1,1]
    no_fn = confusion_matrix(B_test,B_rndm)[1,0]
    return round((no_fn / (no_tp + no_fn)),2)
def Recall(B_test,B_rndm):
    no_tp = confusion_matrix(B_test,B_rndm)[1,1]
    no_fn = confusion_matrix(B_test,B_rndm)[1,0]
    return round((no_tp / (no_tp + no_fn)),2)
def Precision(B_test,B_rndm):
    no_tp = confusion_matrix(B_test,B_rndm)[1,1]
    no_fp = confusion_matrix(B_test,B_rndm)[0,1]
    return round((no_tp / (no_tp + no_fp)),2)
```

```
def F1Score(B_test,B_rndm):
    no_tp = confusion_matrix(B_test,B_rndm)[1,1]
    no_fp = confusion_matrix(B_test,B_rndm)[0,1]
    no_fn = confusion_matrix(B_test,B_rndm)[1,0]
    return round(((2*no_tp) / ((2*no_tp) + no_fp+no_fn)),2)
def Accuracy(B_test,B_rndm):
    no_tn = confusion_matrix(B_test,B_rndm)[0,0]
    no_fp = confusion_matrix(B_test,B_rndm)[0,1]
    no_tp = confusion_matrix(B_test,B_rndm)[1,1]
    no_fn = confusion_matrix(B_test,B_rndm)[1,0]
    return round(((no_tp + no_tn) / (no_tp + no_fp + no_fn + no_tn)),2)
def Error(B_test,B_rndm):
    no_tn = confusion_matrix(B_test,B_rndm)[0,0]
    no_fp = confusion_matrix(B_test,B_rndm)[0,1]
    no_tp = confusion_matrix(B_test,B_rndm)[1,1]
    no_fn = confusion_matrix(B_test,B_rndm)[1,0]
    return round(((no_fp + no_fn) / (no_tp + no_fp + no_fn + no_tn)),2)
def BACC(B_test,B_rndm):
    no_tn = confusion_matrix(B_test,B_rndm)[0,0]
    no_fp = confusion_matrix(B_test,B_rndm)[0,1]
    no_tp = confusion_matrix(B_test,B_rndm)[1,1]
    no_fn = confusion_matrix(B_test,B_rndm)[1,0]
    return round(0.5*((no_tp / (no_tp + no_fn))+(no_tn / (no_fp + no_tn))),2)
def TSS(B_test,B_rndm):
    no_tn = confusion_matrix(B_test,B_rndm)[0,0]
    no_fp = confusion_matrix(B_test,B_rndm)[0,1]
    no_tp = confusion_matrix(B_test,B_rndm)[1,1]
    no_fn = confusion_matrix(B_test,B_rndm)[1,0]
    return round((no_tp / (no_tp + no_fn))-(no_fp / (no_fp + no_tn)),2)
def HSS(B_test,B_rndm):
    no_tn = confusion_matrix(B_test,B_rndm)[0,0]
    no_fp = confusion_matrix(B_test,B_rndm)[0,1]
    no_tp = confusion_matrix(B_test,B_rndm)[1,1]
    no_fn = confusion_matrix(B_test,B_rndm)[1,0]
    return round((2*((no_tp * no_tn)-(no_fp * no_fn)))/(((no_tp + no_fn)*(no_fn + no_tn))+((no_tp + no_fp)*(no_fp + no_
```

```
#def tpr(ans_tp,ans_fn): return ans_tp/(ans_tp+ans_fn)
scoring = {'tp': make_scorer(tp),'tn': make_scorer(tn),'fp': make_scorer(fp),'fn': make_scorer(fn),'tpr': make_scorer(t
           'tnr':make_scorer(tnr),'fpr':make_scorer(fpr),'fnr':make_scorer(fnr),'recall':make_scorer(Recall),
           'precision':make_scorer(Precision),'F1Score':make_scorer(F1Score),
           'Accuracy':make_scorer(Accuracy),'Error':make_scorer(Error),'BACC':make_scorer(BACC),'TSS':make_scorer(TSS),
           'HSS':make_scorer(HSS)}
```

g) Splitting the dataset into the KFolds

```
In [238]: ## Splits the dataset in the Kfolds
cv = KFold(n_splits=10,random_state=1,shuffle=True)
scores = cross_validate(rndm_frst,A_train,B_train,scoring = scoring,cv=cv)
print(scores)
```

```
{'fit_time': array([0.16637421, 0.05789685, 0.04876494, 0.04847693, 0.04774284,
       0.06654572, 0.08344316, 0.07150507, 0.05427527, 0.05329204]), 'score_time': array([0.05997586, 0.04992104, 0.0
3793001, 0.0414722 , 0.03859401,
       0.152426  , 0.07557583, 0.04768896, 0.04427171, 0.06298184]), 'test_tp': array([28,  3,  5,  1, 29,  9, 33,
7,  1,  8]), 'test_tn': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0]), 'test_fp': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0]), 'tes
t_fn': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0]), 'test_tpr': array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.]), 'test_tn
r': array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.]), 'test_fpr': array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]), 'te
st_fnr': array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]), 'test_recall': array([1., 1., 1., 1., 1., 1., 1., 1., 1.,
1.]), 'test_precision': array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.]), 'test_F1Score': array([1., 1., 1., 1., 1.,
1., 1., 1., 1., 1.]), 'test_Accuracy': array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.]), 'test_Error': array([0., 0.,
0., 0., 0., 0., 0., 0., 0., 0.]), 'test_BACC': array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.]), 'test_TSS': array
([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.]), 'test_HSS': array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])}
```

```
In [239]: column = ['Fold 1','Fold 2','Fold 3','Fold 4','Fold 5','Fold 6','Fold 7','Fold 8','Fold 9','Fold 10']
row = ['TP','TN','FP','FN','TPR','TNR','FPR','FNR','Recall','Precision','Error','BACC','TSS','HSS','F1 Score','Accuracy


data_rf = [value for value in scores.values()]
data_rf = data_rf[3:]
```

h)Creating the table for the result.

```
In [240]: for i in range(len(data_rf)):
              avg = round(sum(data_rf[i])/len(data_rf[i]),2)
              temp = list(data_rf[i])
              temp.append(avg)
              data_rf[i]=temp
          print(data_rf)
```

```
[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.0], [1.0,
1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0], [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0], [0.0, 0.
0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], [1.0, 1.0,
1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0], [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0], [1.0, 1.0, 1.
0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0], [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0], [0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0], [1.0, 1.0, 1.0, 1.
0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0], [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]]
```

```
In [277]: import matplotlib.pyplot as plt
          fig3,ax3 = mlt.subplots()


          fig3.patch.set_visible(False)
          ax3.axis('off')


          the_table3=ax3.table(cellText=data_svm, rowLabels=row, colLabels=column,loc='center',colWidths=[0.25 for x in column])
          the_table3.auto_set_font_size(False)
          the_table3.set_fontsize(13)
          fig3.tight_layout()
          plt.show()
```

Output of the Random Forest Classifier:

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Fold 6	Fold 7	Fold 8	Fold 9	Fold 10
TP	28	3	5	1	29	9	33	7	1	8
TN	0	0	0	0	0	0	0	0	0	0
FP	0	0	0	0	0	0	0	0	0	0
FN	0	0	0	0	0	0	0	0	0	0
TPR	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
TNR	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
FPR	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
FNR	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Recall	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Precision	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Error	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
BACC	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
TSS	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
HSS	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
F1 Score	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Accuracy	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

Conclusion: Random forest classifiers have many applications. They are among the most robust machine learning algorithms and are a must-have in any AI and ML professional.

Support Vector Machine: A support vector machine (SVM) is a supervised machine learning model that uses classification algorithms for two-group classification problems. After giving an SVM model sets of labeled training data for each category, they're able to categorize new text.

a) Importing the python required libraries. Splitting the dataset into the training set and test set. Training the support vector machine classification model on the training set. Importing confusion_matrix from the sklearn.metrics.

```
In [40]: ###SVM

In [64]: from sklearn.svm import SVC
         from sklearn import svm
         svm_clf=SVC(kernel='linear',random_state=0)
         svm_clf.fit(A_train,B_train)

Out[64]: SVC(kernel='linear', random_state=0)

In [65]: B_pred = classifier.predict(A_test)

In [66]: from sklearn.metrics import confusion_matrix
         cm = confusion_matrix(B_test, B_pred)
         print(cm)

         [[ 7  0  0 ...  0  0  0]
          [ 0 22  0 ...  0  0  0]
          [ 0  0  1 ...  0  0  0]
          ...
          [ 0  0  0 ...  0  0  0]
          [ 0  0  0 ...  0  7  0]
          [ 0  0  0 ...  0  0  0]]
```

b) Displaying the results (confusion matrix and accuracy) and applying the svm model on the test Data

```
In [69]: #Apply model on test data
         y_pred_test=svm_clf.predict(A_test)
         y_pred_test

Out[69]: array(['Pennsylvania', 'California', 'Virginia', 'Maryland', 'Oklahoma',
                'Washington', 'California', 'Texas', 'Illinois', 'Pennsylvania',
                'California', 'Ohio', 'Florida', 'Missouri', 'Pennsylvania',
                'New York', 'Pennsylvania', 'Ohio', 'Virginia', 'Florida',
                'Illinois', 'Minnesota', 'Illinois', 'Texas', 'California',
                'Florida', 'Indiana', 'Kansas', 'Texas', 'Missouri', 'Maryland',
                'Texas', 'Texas', 'Illinois', 'Minnesota', 'New York',
                'Washington', 'South Carolina', 'Pennsylvania', 'Washington DC',
                'Massachusetts', 'Texas', 'Colorado', 'Arizona', 'Ohio',
                'California', 'Minnesota', 'Texas', 'Maryland', 'Pennsylvania',
                'Ohio', 'Ohio', 'Ohio', 'Florida', 'Florida', 'Virginia',
                'Missouri', 'California', 'Texas', 'Texas', 'California',
                'Florida', 'Texas', 'Maryland', 'Washington', 'California',
                'California', 'Rhode Island', 'Oregon', 'Missouri', 'California',
                'California', 'California', 'Connecticut', 'Indiana', 'California',
                'South Carolina', 'Ohio', 'Texas', 'Colorado', 'New York',
                'Arizona', 'California', 'California', 'Kentucky', 'Ohio',
                'California', 'Ohio', 'New York', 'California', 'Ohio',
                'California', 'Washington', 'Pennsylvania', 'New York', 'Utah',
```

c) Calculating the accuracy model by importing metrics from sklearn and creating the confusion matrix

```
In [257]: #Accuracy calculation
          from sklearn import metrics
          metrics.accuracy_score(B_test,B_test)

Out[257]: 1.0
```

```
In [258]: #Create confusion matrix
          conf=metrics.confusion_matrix(B_test,B_test)
          conf

Out[258]: array([[ 7,  0,  0, ...,  0,  0,  0],
                 [ 0, 22,  0, ...,  0,  0,  0],
                 [ 0,  0,  1, ...,  0,  0,  0],
                 ...,
                 [ 0,  0,  0, ...,  2,  0,  0],
                 [ 0,  0,  0, ...,  0,  7,  0],
                 [ 0,  0,  0, ...,  0,  0,  1]])
```

```
In [259]: #Precision, Recall, FScore
          pr_rcl=metrics.precision_recall_fscore_support(B_test,B_test,average='weighted')
          pr_rcl

Out[259]: (1.0, 1.0, 1.0, None)
```

```
In [260]: print(scores.values)

          <built-in method values of dict object at 0x7f939f6db6c0>
```

```
In [261]: data_svm = [value for value in scores.values()]
          data_svm = data_svm[2:]
```

```
In [276]: fig3,ax3 = mlt.subplots()


          fig3.patch.set_visible(False)
          ax3.axis('off')


          the_table3=ax3.table(cellText=data_svm, rowLabels=row, colLabels=column,loc='center',colWidths=[0.25 for x in column])
          the_table3.auto_set_font_size(False)
          the_table3.set_fontsize(13)
          fig3.tight_layout()
          plt.show()
```

Output of the SVM model accuracy

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Fold 6	Fold 7	Fold 8	Fold 9	Fold 10
TP	28	3	5	1	29	9	33	7	1	8
TN	0	0	0	0	0	0	0	0	0	0
FP	0	0	0	0	0	0	0	0	0	0
FN	0	0	0	0	0	0	0	0	0	0
TPR	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
TNR	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
FPR	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
FNR	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Recall	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Precision	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Error	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
BACC	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
TSS	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
HSS	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
F1 Score	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Accuracy	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

LONG SHORT-TERM MEMORY (LSTM)

LSTM stands for Long-Short Term Memory. LSTM is a type of recurrent neural network but is better than traditional recurrent neural networks in terms of memory. Having a good hold over memorizing certain patterns LSTMs perform fairly better. As with every other NN, LSTM can have multiple hidden layers and as it passes through every layer, the relevant information is kept and all the irrelevant information gets discarded in every single cell.

a) Importing required libraries for the algorithm

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.utils import to_categorical
from keras.layers import Dense,RNN,LSTM,Activation,Dropout
from keras.models import Sequential

```
In [92]: ##LSTM

In [290]: import tensorflow as tf
          from tensorflow import keras
          from tensorflow.keras import layers
          from tensorflow.keras.utils import to_categorical
          from keras.layers import Dense,RNN,LSTM,Activation,Dropout
          from keras.models import Sequential

In [291]: A.shape

Out[291]: (2629, 4)

In [292]: B.shape

Out[292]: (2629,)

In [293]: A_train = np.reshape(A_train, (A_train.shape[0],A_train.shape[1],1))

In [305]: A_train = np.reshape(A_train, (A_train.shape[0],A_train.shape[1],1))
```

In [306]:
```
model = Sequential() # initializing model

model.add(LSTM(units=60, return_sequences=False, input_shape=(A_train.shape[1],1)))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
model.fit(A_train, B_train, epochs=101, batch_size=4000,validation_split=0.3)
```

Github Link: https://github.com/Hpp5/CS634-Finaltermproject