2 NLP (60 pts.)

Download the Sentiment Labelled Sentences Data Set. Using N-gram models, employ ID3 algorithm to learn decision trees that are able to detect the sentiment of a sentence. Randomly divide the data set into two parts, one part (part A) contains 70% of total items and the other one (part B) contains 30%. Train your models using part A and test them using part B. You are free to use any tools and 3rd-party libraries(Weka, Matlab, R, scikit-learn, etc.).

1. (20 pts.) Train a decision tree using bigram model. Report what you do and what you find with regard to data processing, parameter tuning, learned model, confusion matrix, and so on. Give one example (if it exists) of false positive and one example of a false negative (if it exists). Try to explain why they are incorrectly classified.

2. (20 pts.) Train decision trees using a unigram model and a trigram model separately. Compare them with the bigram model in terms of precision, recall, and F-score.

3. (20 pts.) Employ data-cleansing approaches such as removing stop words to improve your result. Report what you do and what you find.

Note: Submit the code along with a readme file indicating what packages we need to install to run the script. Please also submit your training and testing dataset.

Solution:

1.

```
#READING ALL THE THREE FILES
file_imdb=open("imdb_labelled.txt","r")
imdb=[]
for line in file_imdb:
    line = line.split('\n')
    line = line[0].split('\t')
    line[1] = int(line[1])
    imdb.append(line)
file_imdb.close()
imdb_df = pd.DataFrame(imdb, columns=["Summary", "Score"])
file_amazon=open("amazon_cells_labelled.txt","r")
amazon=[]
for line in file_amazon:
    line = line.split('\n')
    line = line[0].split('\t')
    line[1] = int(line[1])
    amazon.append(line)
file_amazon.close()
amazon_df = pd.DataFrame(amazon, columns=["Summary", "Score"])
file_yelp=open("yelp_labelled.txt","r")
yelp=[]
for line in file_yelp:
    line = line.split('\n')
    line = line[0].split('\t')
    line[1] = int(line[1])
    yelp.append(line)
file_yelp.close()
yelp_df = pd.DataFrame(yelp, columns=["Summary", "Score"])
```

In this step we read the three files and add the contents into three respective dataframes (imdb_df, amazon_df, yelp_df) using pandas. Each dataframe has a Summary column having the reviews and a Score column having respective sentiment score.

```
#MERGING ALL FILES INTO ONE DATAFRAME
final_data_df = pd.concat([imdb_df,amazon_df,yelp_df], ignore_index= True)
print ("Count of final table: ", final_data_df["Summary"].count())
```

```
Count of final table:  3000
```

In this step we merge the 3 dataframes into one – final_data_df. The count of the frame can be seen to be 3000.

```
In [109]: #DATA CLEANSING APPROACHES
data=[]
#data_df['Summary']=data_df['Summary'].str.lower()
for index in range(len(final_data_df['Summary'])):
    #NORMAL
    data.append([final_data_df['Summary'][index],final_data_df['Score'][index]])
    #LOWERCASE
    #data.append([final_data_df['Summary'][index].lower(),final_data_df['Score'][index]])
    #No Numbers
    #data.append([final_data_df['Summary'][index].translate(None, digits),final_data_df['Score'][index]])
    #No Punctuation
    #data.append([final_data_df['Summary'][index].translate(None, string.punctuation),final_data_df['Score'][index]])
    #No Numbers AND Punctuation
    #data.append([final_data_df['Summary'][index].translate(None, string.punctuation).translate(None, digits),final_dat
    #Word Tokenize
    #words = word_tokenize(final_data_df['Summary'][index])
    #data.append([' '.join(word for word in words),final_data_df['Score'][index]])
    #StopWords
    #words = [word for word in final_data_df['Summary'][index].split() if word not in stopwords.words('english')]
    #data.append([' '.join(word for word in words),final_data_df['Score'][index]])


data_df = pd.DataFrame(data, columns=["Summary", "Score"])

print (data_df[:10])
```

```
                                            Summary  Score
0  A very, very, very slow-moving, aimless movie ...      0
1  Not sure who was more lost - the flat characte...      0
2  Attempting artiness with black & white and cle...      0
3       Very little music or anything to speak of.      0
4  The best scene in the movie was when Gerardo i...      1
5  The rest of the movie lacks art, charm, meanin...      0
6                                 Wasted two hours.      0
7  Saw the movie today and thought it was a good ...      1
8                                A bit predictable.      0
9  Loved the casting of Jimmy Buffet as the scien...      1
```

In this step, we perform various Data Cleansing Steps and store the result into the final data frame – data_df which will then be used for the subsequent analysis.

Data Cleansing Approaches Used:

- Normal
- All lowercase words
- Removing any usage of numbers and special characters from the reviews
- Removing punctuations from the reviews
- Removing “stop words” from the reviews
- Lemmatization
- Stemming

```
#SPLITTING FILES RANDOMLY - 70% TRAINING & 30% TESTING
[Data_train,Data_test,Train_labels,Test_labels] = train_test_split(data_df['Summary'],
                                                                   data_df['Score'] ,
                                                                   test_size=0.3,
                                                                   random_state=42)

print "Data train count = ",
print Data_train.count()
print "Data test count = ",
print Data_test.count()
print "\nTraining Data :"

print Data_train[:5],
print "\nTesting Data :"
print Data_test[:5]
```

```
Data train count =  2100
Data test count =  900

Training Data :
611             I believe that Pitch Black was done well
530     there are so many problems i dont know where t...
2787    I dont have very many words to say about this ...
49      The film succeeds despite or perhaps because o...
1883                                  WARNING  DO NOT BUY
Name: Summary, dtype: object
Testing Data :
1801                   For the price this was a great deal
1190                   The replacement died in a few weeks
1817          Gets a signal when other Verizon phones wont
251     The cinematographyif it can be called thatsuck...
2505                      I would not recommend this place
Name: Summary, dtype: object
```

In this step, we randomly SPLIT the data_df data frame into Training Data and Testing Data.
Training Data is 70% of the given reviews.
Testing Data is 30% of the given reviews.

Output shows the count of Training data to be 2100 and Testing data to be 900. Output also shows that Training Data and Testing Data are randomly distributed.

```
classifier = DecisionTreeClassifier(random_state=20160121, criterion='entropy')
```

In this step, the classifier is used as provided in the problem. We use a DecisionTreeClassifier with parameters as ‘entropy’ which ensures that the data is classified via the ID3 algorithm.

BIGRAM Analysis:

```
#Bigram Model
bigram_vec = TfidfVectorizer(ngram_range=(1, 2),
                             strip_accents='unicode',
                             min_df=2,
                             norm='l2')

bigram_model = bigram_vec.fit(data_df['Summary'])
bigram_train = bigram_model.transform(Data_train)
bigram_test  = bigram_model.transform(Data_test)

bigram_clf = classifier.fit(bigram_train, Train_labels)
bigram_prediction = bigram_clf.predict(bigram_test)

print (metrics.classification_report(Test_labels.values, bigram_prediction))

bi_conf_mat = metrics.confusion_matrix(Test_labels.values, bigram_prediction)

print ("\nConfusion Matrix for BIGRAM:")
print ('\t\tPrediction')
print ("\t\tNEG\tPOS")
print ("Actual\tNEG\t", bi_conf_mat[0][0], "\t", bi_conf_mat[0][1])
print ("\tPOS\t", bi_conf_mat[1][0], "\t", bi_conf_mat[1][1])

bi_accuracy = (float(bi_conf_mat[0][0]+bi_conf_mat[1][1])*100/900)
print ("\nAccuracy for BIGRAM")
print (bi_accuracy)
```

```
             precision    recall  f1-score   support

          0       0.70      0.75      0.72       443
          1       0.74      0.69      0.72       457

avg / total       0.72      0.72      0.72       900


Confusion Matrix for BIGRAM:
                Prediction
                NEG     POS
Actual  NEG      331     112
        POS      140     317

Accuracy for BIGRAM
72.0
```

In this step, we performed the analysis and prediction using the bigram model and the saw the results.
Output shows the precision, recall, f-score, Confusion matrix, and accuracy of the model.

```
i=0
fp_count=0
fn_count=0
false_pos=[]
false_neg=[]
for idx, value in Test_labels.iteritems():
    if(value==0 and bigram_prediction[i]==1):
        false_pos.append("False Positive: "+Data_test[idx])
        fp_count+=1
    if(value==1 and bigram_prediction[i]==0):
        false_neg.append("False Negative: "+Data_test[idx])
        fn_count+=1
    i+=1
#print (fp_count)
#print (fn_count)
print (false_pos[1])
print (false_neg[1])
```

```
False Positive: But other than that the movie seemed to drag and the heroes didn't really work for their freedom.
False Negative: Overall, a delight!
```

In this step, we see an example of one False Positive and one False Negative case.

Output shoes one example of the false positive generated and one example of the false negative generated.

2.
UNIGRAM Analysis:

```
#Unigram Model
unigram_vec = TfidfVectorizer(ngram_range=(1, 1),
                              strip_accents='unicode',
                              min_df=2,
                              norm='l2')

unigram_model = unigram_vec.fit(data_df['Summary'])
unigram_train = unigram_model.transform(Data_train)
unigram_test  = unigram_model.transform(Data_test)

unigram_clf = classifier.fit(unigram_train, Train_labels)
unigram_prediction = unigram_clf.predict(unigram_test)
print (metrics.classification_report(Test_labels.values, unigram_prediction))

uni_conf_mat = metrics.confusion_matrix(Test_labels.values, unigram_prediction)
print ("\nConfusion Matrix for UNIGRAM:")
print ('\t\tPrediction')
print ("\t\tNEG\tPOS")
print ("Actual\tNEG\t", uni_conf_mat[0][0], "\t", uni_conf_mat[0][1])
print ("\tPOS\t", uni_conf_mat[1][0], "\t", uni_conf_mat[1][1])

uni_accuracy = (float(uni_conf_mat[0][0]+uni_conf_mat[1][1])*100/900)
print ("\nAccuracy for UNIGRAM")
print (uni_accuracy)
```

```
             precision    recall  f1-score   support

          0       0.68      0.75      0.71       443
          1       0.73      0.65      0.69       457

avg / total       0.70      0.70      0.70       900


Confusion Matrix for UNIGRAM:
                Prediction
                NEG     POS
Actual  NEG      331     112
        POS      158     299

Accuracy for UNIGRAM
70.0
```

In this step, we performed the analysis and prediction using the unigram model and the saw the results.
Output shows the precision, recall, f-score, Confusion matrix, and accuracy of the model.

TRIGRAM Analysis:

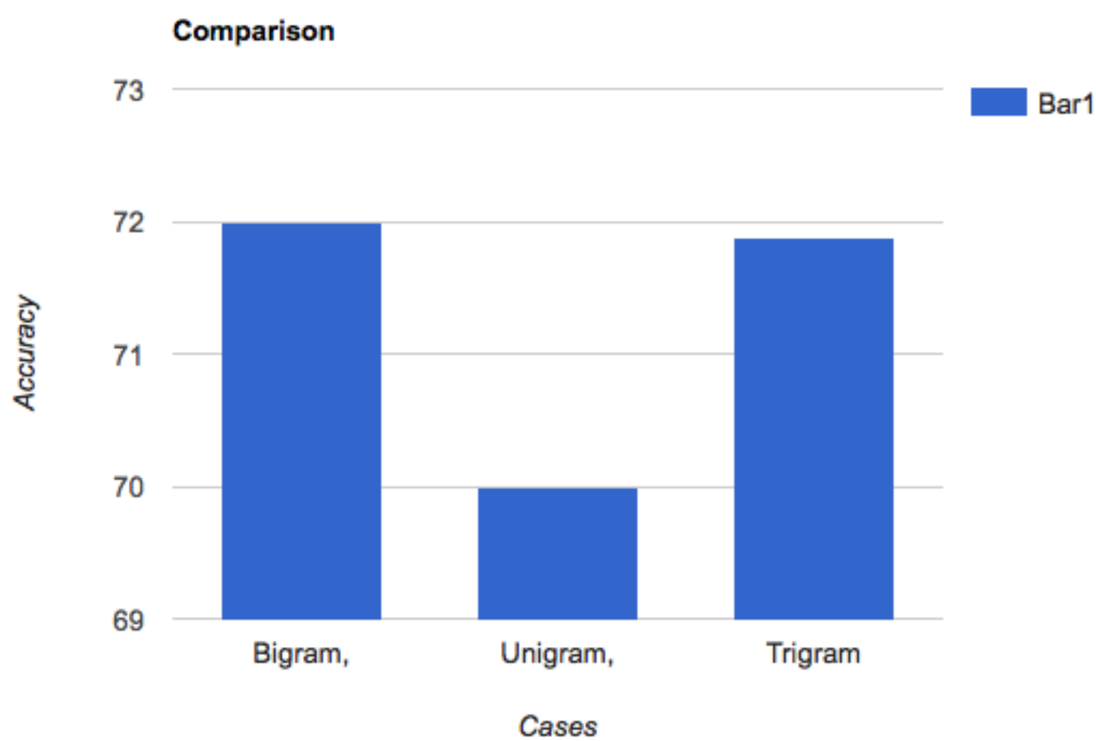
```
#Trinigram Model
trigram_vec = TfidfVectorizer(ngram_range=(1, 3),
                             strip_accents='unicode',
                             min_df=2,
                             norm='l2')

trigram_model = trigram_vec.fit(data_df['Summary'])
trigram_train = trigram_model.transform(Data_train)
trigram_test  = trigram_model.transform(Data_test)

trigram_clf = classifier.fit(trigram_train, Train_labels)
trigram_prediction = trigram_clf.predict(trigram_test)
print (metrics.classification_report(Test_labels.values, trigram_prediction))

tri_conf_mat = metrics.confusion_matrix(Test_labels.values, trigram_prediction)
print ("\nConfusion Matrix for TRIGRAM:")
print ('\t\tPrediction')
print ("\t\tNEG\tPOS")
print ("Actual\tNEG\t", tri_conf_mat[0][0], "\t", tri_conf_mat[0][1])
print ("\tPOS\t", tri_conf_mat[1][0], "\t", tri_conf_mat[1][1])

accuracy = (float(tri_conf_mat[0][0]+tri_conf_mat[1][1])*100/900)
print ("\nAccuracy for TRIGRAM")
print (accuracy)
```

```
             precision    recall  f1-score   support

          0       0.70      0.75      0.73       443
          1       0.74      0.68      0.71       457

avg / total       0.72      0.72      0.72       900


Confusion Matrix for TRIGRAM:
                Prediction
                NEG     POS
Actual  NEG      334     109
        POS      144     313

Accuracy for TRIGRAM
71.88888888888889
```

In this step, we performed the analysis and prediction using the trigram model and the saw the results.
Output shows the precision, recall, f-score, Confusion matrix, and accuracy of the model.

COMPARISON:

	Bigram	Unigram	Trigram
Precision	0.72	0.70	0.72
Recall	0.72	0.70	0.72
F-score	0.72	0.70	0.72
Accuracy	72.0%	70.0%	71.88%



3. DATA CLEANSING Results:

a. All words being Lowercase:

```
#DATA CLEANSING APPROACHES
data=[]
#data_df['Summary']=data_df['Summary'].str.lower()
for index in range(len(final_data_df['Summary'])):
    #NORMAL
    #data.append([final_data_df['Summary'][index],final_data_df['Score'][index]])
    #LOWERCASE
    data.append([final_data_df['Summary'][index].lower(),final_data_df['Score'][index]])
    #No Numbers
    #data.append([final_data_df['Summary'][index].translate(None, digits),final_data_df['Score'][index]])
    #No Punctuation
    #data.append([final_data_df['Summary'][index].translate(None, string.punctuation),final_data_df['Score'][index]])
    #No Numbers AND Punctuation
    #data.append([final_data_df['Summary'][index].translate(None, string.punctuation).translate(None, digits),final_dat
    #Word Tokenize
    #words = word_tokenize(final_data_df['Summary'][index])
    #data.append([' '.join(word for word in words),final_data_df['Score'][index]])
    #StopWords
    #words = [word for word in final_data_df['Summary'][index].split() if word not in stopwords.words('english')]
    #data.append([' '.join(word for word in words),final_data_df['Score'][index]])


data_df = pd.DataFrame(data, columns=["Summary", "Score"])

print (data_df[:10])
```

```
                                         Summary  Score
0  a very, very, very slow-moving, aimless movie ...      0
1  not sure who was more lost - the flat characte...      0
2  attempting artiness with black & white and cle...      0
3         very little music or anything to speak of.      0
```

```
             precision    recall  f1-score   support

          0       0.70      0.75      0.72       443
          1       0.74      0.69      0.72       457

avg / total       0.72      0.72      0.72       900


Confusion Matrix for BIGRAM:
                Prediction
                NEG     POS
Actual   NEG    331     112
         POS    140     317

Accuracy for BIGRAM
72.0
```

b. No Numbers in reviews:

	precision	recall	f1-score	support
0	0.69	0.69	0.69	443
1	0.70	0.70	0.70	457
avg / total	0.70	0.70	0.70	900

Confusion Matrix for BIGRAM:

		Prediction	
		NEG	POS
Actual	NEG	306	137
	POS	137	320

Accuracy for BIGRAM
69.5555555556

c. No Punctuation in reviews:

	precision	recall	f1-score	support
0	0.69	0.74	0.71	443
1	0.73	0.68	0.70	457
avg / total	0.71	0.71	0.71	900

Confusion Matrix for BIGRAM:

		Prediction	
		NEG	POS
Actual	NEG	327	116
	POS	148	309

Accuracy for BIGRAM
70.666666667

d. No Numbers AND Punctuation in reviews:

	precision	recall	f1-score	support
0	0.72	0.67	0.70	443
1	0.70	0.75	0.73	457
avg / total	0.71	0.71	0.71	900

Confusion Matrix for BIGRAM:

		Prediction	
		NEG	POS
Actual	NEG	299	144
	POS	114	343

Accuracy for BIGRAM
71.3333333333

e. Using NLTK Word Tokenizer

	precision	recall	f1-score	support
0	0.69	0.73	0.71	443
1	0.72	0.68	0.70	457
avg / total	0.71	0.71	0.71	900

Confusion Matrix for BIGRAM:

		Prediction	
		NEG	POS
Actual	NEG	324	119
	POS	146	311

Accuracy for BIGRAM
70.5555555556

f. Excluding STOPWRODS (nltk library) from the reviews

	precision	recall	f1-score	support
0	0.73	0.78	0.75	443
1	0.77	0.72	0.74	457
avg / total	0.75	0.75	0.75	900

Confusion Matrix for BIGRAM:

		Prediction	
		NEG	POS
Actual	NEG	345	98
	POS	129	328

Accuracy for BIGRAM
74.7777777778

g. Stemming:

	precision	recall	f1-score	support
0	0.71	0.73	0.72	443
1	0.73	0.72	0.73	457
avg / total	0.72	0.72	0.72	900

Confusion Matrix for BIGRAM:

		Prediction	
		NEG	POS
Actual	NEG	325	118
	POS	130	327

Accuracy for BIGRAM
72.4444444444

COMPARISON IN GRAPH: