**IST 664 – Natural Language Processing**
**Homework 3**

In this assignment, we conducted sentiment analysis to gain some understanding about the emotions reflected from the text that we have worked on in the previous two homework. We explored the sentiment polarity of the comments at the sentence level.

Here I am building three classifiers, compare their accuracy and use the best performing classifier for sentiment analysis.

## Data Preprocessing

### *Training on sentiment polarity*

For the purpose of training on sentiment polarity I have used the Airlines Tweets dataset (http://help.sentiment140.com/for-students) as it has the three required sentiments i.e., positive, negative and neutral. Also, the dataset is small and sufficient enough for training the classifier.

```
#using the Tweets dataset for training on sentiment polarity

train_dataset = pd.read_csv('Tweets.csv')
```

```
#removing @ and # from the training data

def remove_at(x):
    x = str(x).replace('@', '')
    x = str(x).replace('#', '')
    return x
```

I have removed @ and # symbols form the tweets for precise training of the model.

We need to tokenize the tweets for extracting features. There are plenty of tokenizers provided by NLTK which are available for us to use. For tokenizing the text, we can use the RegexpTokenzier available in the NLTK's tokenize library, which is a Regular Expression tokenizer. The text data contains symbols like *, :, etc. which are of no use thus, we use RegularExpression tokenizer to consider only the words and numbers in the text. The following lines of code were used for the same:

```
#tokenizing using RegexTokenizer by consideringing only alphabets and numbers

import nltk

tokenizer = nltk.RegexpTokenizer('\w+')

def senti_doc(x):
    t = tokenizer.tokenize(x)
    return t
```

We then create tuples of the tokens with their corresponding sentiments using the code below.

```
#creating tuples of tokens and corresponding sentiments

docs = []

for i in range(0, len(train_dataset['airline_sentiment'])):
    docs.append((doc[i], train_dataset['airline_sentiment'][i]))
```

Since the documents are in order by label, we mix them up for later separation into training and test sets using the random,shuffle.

We need to define the set of words that will be used for features. This is essentially all the words in the entire document collection, we will limit it to the 2000 most frequent words. This is done using the following code segment.

```
#defining set of words that will be used for features

all_words = [word for (sentance,category) in docs for word in sentance]
top_words = nltk.FreqDist(all_words)
most_common_words = top_words.most_common(2000)
word_features = [word for (word,count) in most_common_words]
```

### *1. Unigram word features*

I have defined the features for each document, using unigram features. The feature label will be 'contains(keyword)' for each keyword in the word_features set, and the value of the feature will be Boolean, according to whether the word is contained in that document.

```
#defining features for each document using unigram features

def document_features(document, word_features):
    document_words = set(document)
    features = {}
    for word in word_features:
        features['contains({})'.format(word)] = (word in document_words)
    return features
```

We create the training and test sets, train a Naïve Bayes classifier with 3-fold cross validation, and look at the accuracy for the unigram features.

```
#Naive Bayes classifier with 3-fold cross validation for training on sentiments using unigrams

import numpy as np
from sklearn.model_selection import KFold

kf = KFold(n_splits = 3)
sum = 0

for train, test in kf.split(featuresets):
    train_data = np.array(featuresets)[train]
    test_data = np.array(featuresets)[test]
    classifier = nltk.NaiveBayesClassifier.train(train_data)
    sum += nltk.classify.accuracy(classifier, test_data)

acc1 = sum/3
```

```
#accuracy of classifier1

acc1
```

```
0.7669398907103826
```

The classifier when tested shows 76% accuracy, which is pretty good for such a simple feature set.

### *2. Bigram word features*

In this approach, I am going to use the bigrams as the feature around which the Naïve Bayes will classify the sentences. Like last approach, to get the list of features, I first create all the possible bigrams of the corpus. Then remove all the non-alpha bigrams to remove all the full stops and the exclamation marks. Then all the stop words are removed to also create better set of bigrams. The bigrams are sorted by their frequency and take top 2000 bigrams as the features around which the classifier will sort.

```
#creating bigram features

all_bigrams = list(nltk.bigrams(all_words))

from nltk.collocations import *
import re

stopwords = nltk.corpus.stopwords.words('english')

def alpha(w):
    pattern = re.compile('^[^a-z]+$')
    if(pattern.match(w)):
        return True
    else:
        return False

bigram_measures = nltk.collocations.BigramAssocMeasures()
finder = BigramCollocationFinder.from_words(all_words)
finder.apply_word_filter(alpha)
finder.apply_word_filter(lambda w: w in stopwords)
scored = finder.score_ngrams(bigram_measures.raw_freq)
scored[:10]
```

Then we will convert the formatted sentences in to list of tuples, where first value of the tuple will be converted from the sentence to the feature format as shown below.

```
#defining features for each document using bigram features

def bi_document_features(document, bigram_features):
    document_words = list(nltk.bigrams(document))
    features = {}
    for word in bigram_features:
        features['contains({})'.format(word)] = (word in document_words)
    return features
```

```
featuresets2 = [(bi_document_features(d, bigram_features), c) for (d, c) in docs]

featuresets2[0]
```

```
  "contains(('With', 'New'))": False,
  "contains(('You', 'suck'))": False,
  "contains(('Your', 'staff'))": False,
  "contains(('Your', 'website'))": False,
  "contains(('account', 'united'))": False,
  "contains(('ago', 'AmericanAir'))": False,
  "contains(('already', 'Cancelled'))": False,
  "contains(('amazing', 'customer'))": False,
  "contains(('among', 'teens'))": False,
  "contains(('another', 'day'))": False,
  "contains(('anyone', 'else'))": False,
  "contains(('automated', 'message'))": False,
  "contains(('away', 'USAirways'))": False,
  "contains(('awesome', 'Thanks'))": False,
  "contains(('awesome', 'flight'))": False,
  "contains(('awful', 'united'))": False,
  "contains(('b', 'f'))": False,
  ...},
 'negative')
```

We then run the Naïve Bayes classifier with 3-fold cross validation and check the accuracy.

```
#Naive Bayes classifier with 3-fold cross validation for training on sentiments using bigram features

kf = KFold(n_splits = 3)
sum = 0

for train, test in kf.split(featuresets2):
    train_data2 = np.array(featuresets2)[train]
    test_data2 = np.array(featuresets2)[test]
    classifier2 = nltk.NaiveBayesClassifier.train(train_data2)
    sum += nltk.classify.accuracy(classifier2, test_data2)

acc2 = sum/3
```

```
#accuracy of classifier2

acc2
```

```
0.6756147540983607
```

The classifier when tested shows 67%% accuracy, which isn't great. So will try another approach.

### *3. Representing negation*

In this approach, we are going to take account of the negation in a sentence. we will go through the document words in order adding the word features, but if the word follows a negation words, we will change the feature to negated word. Then we basically follow the first techniques method.

```
#considering negation words with unigram approach

negationwords = ['no', 'not', 'never', 'none', 'nowhere', 'nothing', 'noone', 'rather', 'hardly', 'scarcely', 'rarel

def NOT_features(document, word_features, negationwords):
    features = {}
    for word in word_features:
        features['contains({})'.format(word)] = False
        features['contains(NOT{})'.format(word)] = False

    # go through document words in order
    for i in range(0, len(document)):
        word = document[i]
        if ((i + 1) < len(document)) and ((word in negationwords) or (word.endswith("n't"))):
            i += 1
            features['contains(NOT{})'.format(document[i])] = (document[i] in word_features)
        else:
            features['contains({})'.format(word)] = (word in word_features)
    return features
```

```
#Naive Bayes classifier with 3-fold cross validation for training on sentiments using negation words and unigram fea

kf = KFold(n_splits = 3)
sum = 0

for train, test in kf.split(NOT_featuresets):
    train_data3 = np.array(NOT_featuresets)[train]
    test_data3 = np.array(NOT_featuresets)[test]
    classifier3 = nltk.NaiveBayesClassifier.train(train_data3)
    sum += nltk.classify.accuracy(classifier3, test_data3)

acc3 = sum/3
```

```
#accuracy of classifier3

acc3
```

```
0.7594262295081968
```

In this approach, we get an accuracy of 75% which is less compared to the first classifier. Thus, we will use the first classifier to categorize our sentences.

**Determining Sentence Polarity**

Since the number of sentences in the comments is very large, I have used only $1/4^{th}$ of the comments for determining the polarity at sentence level. We tokenize each of the comments using sentence tokenizer and then classify them based on the first classifier. The author, title and country of the corresponding comments are also stored along with the number of positive, negative and neutral sentences in each comment. This entire process is carried out using the code below.

```
#determining the sentiments of the 1/4th of the comments using classifier1

pos_sent = []
neg_sent = []
neu_sent = []

title = []
author = []
country = []
total_pos = []
total_neg = []
total_neu = []

for i in range(0, int(len(covid['text'])/4)):
    sentences = nltk.sent_tokenize(covid['text'][i])
    pos_count = 0
    neg_count = 0
    neu_count = 0
    for sents in sentences:
        senti = classifier.classify(document_features(nltk.word_tokenize(sents), word_features))

        if senti == 'positive':
            pos_sent.append(sents)
            pos_count += 1

        elif senti == 'negative':
            neg_sent.append(sents)
            neg_count += 1

        else:
            neu_sent.append(sents)
            neu_count += 1

    title.append(covid['title'][i])
    author.append(covid['author'][i])
    country.append(covid['country'][i])

    total_pos.append(pos_count)
    total_neg.append(neg_count)
    total_neu.append(neu_count)
```

**Interpretation of Results**

For the first 1/4$^{th}$ of the comments analyzed, the results are as follows.
Number of positive sentences = 106295
Number of negative sentences = 302118
Number of neutral sentences = 175037

Some of the positive sentences are:

'All district hospitals will have five beds isolated for patients carrying the virus.'
'104 Arogya Sahayavani helpline run by the Health Department will take all calls related to the vi rus.'
'The international agency noted that China was doing everything it could to contain the outbreak.'

Some of the negative sentences are:

'Bengaluru: Isolation wards in hospitals across Karnataka and helpline to take calls on coronaviru s-related queries are ready to prevent any further spread of the virus after the first case in India w as reported from Kerala yesterday.'
'The Chief Secretary of the state government on Thursday held a meeting with the Additional Chi ef Secretary (Health), Health Commissioner, Mission Director of the National Health Mission an d other health department officials and reviewed the state preparedness to tackle any cases of cor onavirus whenever reported.'
'Rajiv Gandhi Institute of Chest Diseases (RGICD) with 15 beds and Wenlock Hospital at Manga luru with 10 beds have been selected for the treatment of the virus.'

Some of the neutral sentences are:

'Along with this, at least ten private hospitals in Bengaluru will be setting up similar isolation war ds.'
'Udayavani English'
'Aside from monitoring suspected cases, authorities are also preparing plans to repatriate Filipino s in the Chinese province of Hubei.'

**Analysis**

In order to analyze positive and negative sentences, each sentence is tagged with part-of-speech tags, which will be helpful in entity detection. POS tagging is the process of marking up a word in a corpus to a corresponding part of a speech tag, based on its context and definition. This task is not straightforward, as a particular word may have a different part of speech based on the context in which the word is used. This is implemented using the following line of code:

```
#POS tagging the positive sentences

tokens_pos = [nltk.word_tokenize(sent) for sent in pos_sent]
tags_pos = [nltk.pos_tag(tok) for tok in tokens_pos]
```

```
#POS tagging the negative sentences

tokens_neg = [nltk.word_tokenize(sent) for sent in neg_sent]
tags_neg = [nltk.pos_tag(tok) for tok in tokens_neg]
```

The top 50 adjectives, adverbs and verbs in positive sentences, we first identify most frequent positive words and then the required phrases are obtained using the following block of code:

```
#finding most frequent positive words
import itertools
all_tags_pos = list(itertools.chain.from_iterable(tags_pos))
cfd_pos = nltk.ConditionalFreqDist(((tags_pos, word) for (word, tags_pos) in all_tags_pos))
```

```
#top 50 adjectives in positive sentences
list(cfd_pos['JJ'])[0:50]
```

```
#top 50 adverbs in positive sentences
list(cfd_pos['RB'])[0:50]
```

```
#top 50 verbs in positive sentences
list(cfd_pos['VB'])[0:50]
```

The same is applied on the set of negative sentences to obtain the corresponding top 50 adverbs, adjectives and verbs.

```
#finding most frequent negative words
all_tags_neg = list(itertools.chain.from_iterable(tags_neg))
cfd_neg = nltk.ConditionalFreqDist(((tags_neg, word) for (word, tags_neg) in all_tags_neg))
```

```
#top 50 adjectives in negative sentences
list(cfd_neg['JJ'])[0:50]
```

```
#top 50 verbs in negative sentences
list(cfd_neg['VB'])[0:50]
```

```
#top 50 adverbs in negative sentences
list(cfd_neg['RB'])[0:50]
```

The following are the results obtained for top 50 adjectives, adverbs and verbs in positive sentences.

```
['new',
 'Chinese',
 'other',
 'first',
 'last',
 '“',
 'good',
 's',
 'global',
 'medical',
 'many',
 'such',
 'great',
 'major',
 'social',
 'novel',
 'top',
 'much',
 'local',
 'positive',
 'economic',
 'high',
 'big',
 'second',
 'early',
 'few',
 'same',
 'next',
 'confirmed',
 'international',
 'Canadian',
 'safe',
 'recent',
 'important',
 'strong',
 'total',
 'further',
 'public',
 'negative',
 'free',
 'late',
 'epidemic',
 'central',
 'foreign',
 'own',
 'little',
 'potential',
 'American',
 'severe',
 '’']
```

Adjectives

```
['also',
 'very',
 'well',
 'so',
 'as',
 'not',
 'just',
 'far',
 'here',
 'now',
 'much',
 'up',
 'back',
 'really',
 'even',
 'too',
 'first',
 "n't",
 'So',
 'ahead',
 'down',
 'still',
 'quickly',
 'always',
 'However',
 'especially',
 'later',
 'already',
 'closely',
 'away',
 'only',
 'often',
 'Meanwhile',
 'however',
 'recently',
 'forward',
 'soon',
 'again',
 'globally',
 'definitely',
 'nearly',
 'mostly',
 'then',
 'Not',
 'together',
 'Also',
 'there',
 'particularly',
 'Here',
 'Still']
```

Adverbs

```
['be',
 '”',
 'help',
 'see',
 'keep',
 'make',
 'have',
 'work',
 'continue',
 'take',
 'contain',
 'get',
 'prevent',
 'spread',
 'Read',
 'stay',
 'ensure',
 'go',
 'follow',
 'protect',
 'win',
 'support',
 'stop',
 'become',
 'do',
 'raise',
 'say',
 'identify',
 'remain',
 '”',
 'treat',
 'monitor',
 're',
 'See',
 'end',
 'give',
 'meet',
 'announce',
 'move',
 'face',
 'come',
 'reduce',
 'look',
 'report',
 'read',
 'feel',
 'Let',
 'Keep',
 'join',
 'start']
```

Verbs

The following are the results obtained for top 50 adjectives, adverbs and verbs in negative sentences.

```
['new',
 'Chinese',
 'other',
 'last',
 'first',
 'global',
 'public',
 '“',
 'due',
 'many',
 'medical',
 's',
 'such',
 'novel',
 '’',
 'confirmed',
 'local',
 'international',
 'positive',
 'economic',
 'same',
 'next',
 'further',
 'early',
 'major',
 'British',
 'past',
 'central',
 'close',
 'several',
 'epidemic',
 'good',
 'deadly',
 'recent',
 'social',
 'full',
 'total',
 '2019-nCoV',
 'few',
 'second',
 'financial',
 'high',
 '”',
 'potential',
 'severe',
 'possible',
 'infected',
 'likely',
 'able',
 'own']
```

Adjectives

```
['not',
 'also',
 'now',
 'still',
 'just',
 'so',
 'only',
 'even',
 'as',
 "n't",
 'back',
 'very',
 'well',
 'far',
 'already',
 'then',
 'ago',
 'here',
 'there',
 'nearly',
 'up',
 'yet',
 'too',
 'recently',
 'almost',
 'again',
 'first',
 'really',
 'down',
 'later',
 'currently',
 'However',
 'much',
 'So',
 'especially',
 'away',
 'ahead',
 'never',
 'right',
 'soon',
 'however',
 'previously',
 'together',
 'globally',
 'Meanwhile',
 'closely',
 'Now',
 'actually',
 'about',
 'long']
```

Adverbs

```
['be',
 'have',
 'get',
 'take',
 '“',
 'make',
 '[',
 'help',
 'do',
 'keep',
 'see',
 'go',
 'prevent',
 'contain',
 'stay',
 'work',
 'spread',
 'ensure',
 'continue',
 'leave',
 'provide',
 'find',
 'stop',
 'know',
 'come',
 'say',
 'protect',
 'avoid',
 'remain',
 'travel',
 'give',
 'use',
 'return',
 'put',
 '”',
 'reduce',
 'deal',
 'support',
 'buy',
 'bring',
 'close',
 'cause',
 'improve',
 'need',
 'allow',
 'respond',
 '’',
 'start',
 'try',
 'evacuate']
```

Verbs