IST 664 – Natural Language Processing

HW 3

Kshitij Sankesara

SU ID: 913789324

Introduction:

For this assignment, we are analysing the product reviews from Amazon Product Data. It is provided by Julian McAuley. The dataset contains of close to 143 million reviews from May 1996 – July 2014. It contains product reviews and Metadata from Amazon. The product reviews contains of ratings, text, helpfulness votes. The Metadata consists of description, category information, price, brand and Image features. It also consists of links (viewed, also bought). We will use only the 5 core subsets for our assignment. The 5-core subset of three categories (Baby/ Clothing, Shoes and Jewellery/ Health and Personal Care). I have extracted review data of the year 2013 for my analysis.

Data Pre-Processing:

The review texts were first extracted from baby text file using Regular Expressions. Then, it were processed so that we can retrieve reviews only from the year 2013. All reviews from the year 2013 were obtained. For further processing, I removed all the punctuations and converted all the words into lowercase. Tokenization was performed. As it is a review data, we can't remove all the stopwords. For example, we can't remove the word 'not' as it is a negative word, so we need it for our sentiment analysis. That's why I have removed other stopwords which doesn't show positive or negative meaning.

```
In [1]: #Importing the libraries
        import nltk
        import random
        import collections
        from nltk.tokenize import sent_tokenize
        from nltk.corpus import sentence_polarity
        from nltk.metrics import *
        from sklearn.feature_extraction.text import TfidfVectorizer
        from sklearn import svm
```

Importing the libraries

```
In [2]: #Reading the text file
        Textdata = open('baby.txt').readlines()

In [3]: #Creating lists for review texts and years
        reviewtextlist = []
        reviewyearlist = []

In [4]: #Using regex to extract review text
        textpattern=r'''(?x)
              reviewText[:](.+)
              '''

In [5]: #Using regex to extract review years
        yearpattern = r''' (?x)
            reviewTime[:](.+)
        '''
```

Using RegEx to extract text and years

```
for line in Textdata:
    sentlist = []
    reviewyear_tokens = nltk.regexp_tokenize(line, yearpattern)
    reviewtext_tokens = nltk.regexp_tokenize(line, textpattern)

    if(len(reviewtext_tokens)>0):
        current_review = reviewtext_tokens[0].strip("\n")
        reviewtextlist.append(current_review)
    if(len(reviewyear_tokens)>0):
        current_year = reviewyear_tokens[0].split(',')[1].strip('\n').strip(' ')
        reviewyearlist.append(current_year)
```

For Line in TextData

```
#For year 2013
final = []
for i in range(0, len(reviewyearlist)):
    if(reviewyearlist[i]=='2013'):
        final.append(reviewtextlist[i])
```

```
print("number of reviews",len(final))
```

```
number of reviews 62223
```

```
final= final[:10000]
```

```
print(len(final))
```

```
10000
```

For Year 2013

```
print("number of reviews",len(final))
```

```
number of reviews 62223
```

```
final= final[:10000]
```

```
print(len(final))
```

```
10000
```

```
final[:5]
```

```
["Perfect for new parents. We were able to keep track of baby's feeding, sleep and diaper change schedule for t
he first two and a half months of her life. Made life easier when the doctor would ask questions about habits b
ecause we had it all right there!",
 'This book is such a life saver.  It has been so helpful to be able to go back to track trends, answer pediatr
ician questions, or communicate with each other when you are up at different times of the night with a newborn.
I think it is one of those things that everyone should be required to have before they leave the hospital.  We
went through all the pages of the newborn version, then moved to the infant version, and will finish up the sec
ond infant book (third total) right as our baby turns 1.  See other things that are must haves for baby at [...
]',
 'I bought this a few times for my older son and have bought it again for my newborn. This is super easy to use
and helps me keep track of his daily routine. When he started going to the sitter when I went back to work, it
helped me know how his day went to better prepare me for how the evening would most likely go. When he was sick
, it help me keep track of how many diapers a day he was producing to make sure he was getting dehydrated. The
note sections to the side and bottom are useful too because his sitter writes in small notes about whether or n
ot he liked his lunch or if the playtime included going for a walk, etc.Excellent for moms who are wanting to k
eep track of their kids daily routine even though they are at work. Excellent for dads to keep track as my husb
and can quickly forget what time he fed our son. LOL',
 'My 3 month old son spend half of his days with my mother and half with a neighbor while I worked.  I had them
track his activity (loosely) in this which allowed me to get an idea of how his schedule was developing and how
much milk he was eating.  It was the best way to have some cohesion in his life while I was at work.',
 "This book is perfect!  I'm a first time new mom, and this book made it so easy to keep track of feedings, dia
per changes, sleep.  Definitely would recommend this for new moms.  Plus it's small enough that I throw in the
diaper back for doctor visits."]
```

Displaying the reviews

```
finalsentlist = []
for review in final:
    curr_sent_list = nltk.sent_tokenize(review)
    for sent in curr_sent_list:
        finalsentlist.append(sent)
```

```
finalsentlist[:5]
```

```
['Perfect for new parents.',
 "We were able to keep track of baby's feeding, sleep and diaper change schedule for the first two and a half m
onths of her life.",
 'Made life easier when the doctor would ask questions about habits because we had it all right there!',
 'This book is such a life saver.',
 'It has been so helpful to be able to go back to track trends, answer pediatrician questions, or communicate w
ith each other when you are up at different times of the night with a newborn.']
```

```
print(len(finalsentlist))
```

```
46361
```

Displaying the reviews after tokenization

```
pattern_2 = r''' (?x)
    [][.,;"'?!():-_%']
'''
```

```
temp_textList = []
for sent in final_sentlist:
    sent = sent.lower()
    words = nltk.word_tokenize(sent)
    if(len(words)==1):
        if(len(nltk.regexp_tokenize(words[0], pattern_2))<0):
            temp_textList.append(sent)
    else:
        temp_textList.append(sent)
```

```
len(temp_textList)
```

```
46176
```

```
temp_textList[:5]
```

```
['perfect for new parents.',
 "we were able to keep track of baby's feeding, sleep and diaper change schedule for the first two and a half m
onths of her life.",
 'made life easier when the doctor would ask questions about habits because we had it all right there!',
 'this book is such a life saver.',
 'it has been so helpful to be able to go back to track trends, answer pediatrician questions, or communicate w
ith each other when you are up at different times of the night with a newborn.']
```

```
finalsentlist=temp_textList
```

Displaying the reviews after Lowercase words

```
from nltk.corpus import stopwords
```

```
stopwords = stopwords.words('english')
negationwords = ['no', 'not', 'never', 'none', 'nowhere', 'nothing', 'noone', 'rather', 'hardly', 'scarcely', 'ra

for w in stopwords:
    if (w in negationwords) or (w.endswith("n't")):
        stopwords.remove(w)
```

Removing Stopwords

Feature extraction from Sentence Polarity:

It is a dataset with already tokenized sentences. Tokenized sentences are with ‘pos’ and ‘neg’ tags. Then, frequency distribution was calculated, and 2000 most common words were extracted. Then, I divided the dataset into two. One was testing and the another was training data set. Naïve Bayes classifier was built on the training set and later applied on the test dataset. The accuracy of the Naïve Bayes classifier was found to be 0.748. The review texts were then used to classify them as ‘neg’ Two files were then created (positive and negative text files). The resulting positive and negative sentences were written respectively.

```
bag_of_words = sentence_polarity.sents()
sentlabels = [(sent, label) for label in sentence_polarity.categories()
                      for sent in sentence_polarity.sents(categories=label)]

random.shuffle(sentlabels)
```

```
sentlabels
```

```
    '.'],
   'pos'),
  (["it's",
    'not',
    'a',
    'classic',
    'spy-action',
    'or',
    'buddy',
    'movie',
    ',',
    'but',
    "it's",
    'entertaining',
    'enough',
    'and',
    'worth',
    'a',
    'look',
    '.'],
   'pos'),
```

```
#Generating word features
wordsList = [word for (sent,label) in sentlabels for word in sent if word not in stopwords]
words = nltk.FreqDist(wordsList)
wordItems = words.most_common(2000)
wordFeatures = [word for (word, freq) in wordItems]
```

Creating word features

```
#Words as Features

def documentFeatures(document, wordFeatures):
    documentWords = set(document)
    features = {}
    for word in wordFeatures:
        if word in documentWords:
            features['contains(%s)'% word] = True
        else:
            features['contains(%s)'% word] = False
    return features
```

```
featuresets = [(documentFeatures(sent,wordFeatures), labels) for (sent,labels) in sentlabels]
```

```
featuresets
```

```
[({'contains(.)': True,
   'contains(,)': True,
   'contains(film)': False,
   'contains(movie)': False,
   'contains(not)': False,
   'contains(one)': False,
   'contains(like)': False,
   'contains(")': False,
   'contains(--)': False,
   'contains(story)': False,
   'contains(no)': False,
   'contains(much)': False,
   'contains(even)': False,
   'contains(good)': False,
   'contains(comedy)': False,
   'contains(time)': False,
   'contains(characters)': False,
   'contains(little)': False,
   'contains(way)': False,
   'contains(funny)': False,
   'contains(make)': False,
```

Creating Feature sets

```
trainSet, testSet = featuresets[1000:], featuresets[:1000]
classifier1 = nltk.NaiveBayesClassifier.train(trainSet)
print ("Accuracy:", nltk.classify.accuracy(classifier1, testSet))
```

```
Accuracy: 0.748
```

Accuracy

```
def display_accuracy_measures(actual_set, input_classifier):
    reference_sets = collections.defaultdict(set)
    predicted_sets = collections.defaultdict(set)
    for i, (feats, label) in enumerate(actual_set):
        reference_sets[label].add(i)
        predicted = input_classifier.classify(feats)
        predicted_sets[predicted].add(i)
    print('Positive precision:', precision(reference_sets['pos'], predicted_sets['pos']))
    print('Positive recall:', recall(reference_sets['pos'], predicted_sets['pos']))
    print('Positive F-measure:', f_measure(reference_sets['pos'], predicted_sets['pos']))
    print('Negative precision:', precision(reference_sets['neg'], predicted_sets['neg']))
    print ('Negative recall:', recall(reference_sets['neg'], predicted_sets['neg']))
    print ('Negative F-measure:', f_measure(reference_sets['neg'], predicted_sets['neg']))
```

```
pos_sentencelist = []
neg_sentencelist = []
```

```
for sent in final_sentlist:
    if(classifier1.classify(documentFeatures(nltk.word_tokenize(sent),wordFeatures)) == 'pos'):
        pos_sentencelist.append(sent)
    elif(classifier1.classify(documentFeatures(nltk.word_tokenize(sent), wordFeatures)) == 'neg'):
        neg_sentencelist.append(sent)
```

```
display_accuracy_measures(featuresets, classifier1)
```

```
Positive precision: 0.8034171626031867
Positive recall: 0.7850309510410804
Positive F-measure: 0.7941176470588235
Negative precision: 0.7898404547955254
Negative recall: 0.8079159632339148
Negative F-measure: 0.7987759643916914
```

```
print("positive reviews text:",len(pos_sentencelist))
print("negative reviews text:",len(neg_sentencelist))
```

```
positive reviews text: 18184
negative reviews text: 28177
```

```
print(pos_sentencelist[:4])
print(neg_sentencelist[:4])
```

```
['Perfect for new parents.', "We were able to keep track of baby's feeding, sleep and diaper change schedule fo
r the first two and a half months of her life.", 'This book is such a life saver.', 'It has been so helpful to
be able to go back to track trends, answer pediatrician questions, or communicate with each other when you are
up at different times of the night with a newborn.']
['Made life easier when the doctor would ask questions about habits because we had it all right there!', 'I thi
nk it is one of those things that everyone should be required to have before they leave the hospital.', 'We wen
t through all the pages of the newborn version, then moved to the infant version, and will finish up the second
infant book (third total) right as our baby turns 1.', 'See other things that are must haves for baby at [...]'
]
```

Positive and Negative Words – First 4

```
negationwords = ['no', 'not', 'never', 'none', 'nowhere', 'nothing', 'rather', 'hardly', 'scarcely', 'rarely', 'n
```

```
#Negation Features

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

```
not_featuresets = [(notFeatures(sent, wordFeatures, negationwords), labels) for (sent, labels) in sentlabels]

trainSet, testSet = not_featuresets[1000:], not_featuresets[:1000]
Classifier2 = nltk.NaiveBayesClassifier.train(trainSet)
print("Accuracy:", nltk.classify.accuracy(Classifier2, testSet))
```

```
Accuracy: 0.778
```

Not features

```
display_accuracy_measures(not_featuresets, Classifier2)
```

```
Positive precision: 0.9485053037608486
Positive recall: 0.922528606265241
Positive F-measure: 0.9353366298972994
Negative precision: 0.9245937557056783
Negative recall: 0.9499155880697805
Negative F-measure: 0.9370836417468541
```

```
pos_sentencelist2 = []
neg_sentencelist2 = []
```

```
for sent in final_sentlist:
    if(Classifier2.classify(notFeatures(nltk.word_tokenize(sent),wordFeatures,negationwords)) == 'pos'):
        pos_sentencelist2.append(sent)
    elif(Classifier2.classify(notFeatures(nltk.word_tokenize(sent),wordFeatures,negationwords)) == 'neg'):
        neg_sentencelist2.append(sent)
```

```
display_accuracy_measures(not_featuresets, Classifier2)
```

```
Positive precision: 0.9485053037608486
Positive recall: 0.922528606265241
Positive F-measure: 0.9353366298972994
Negative precision: 0.9245937557056783
Negative recall: 0.9499155880697805
Negative F-measure: 0.9370836417468541
```

```
print(len(pos_sentencelist2))
print(len(neg_sentencelist2))
```

```
15403
30958
```

```
print(pos_sentencelist2[:5])
print(neg_sentencelist2[:5])
```

```
['Perfect for new parents.', 'This book is such a life saver.', 'We went through all the pages of the newborn v
ersion, then moved to the infant version, and will finish up the second infant book (third total) right as our
baby turns 1.', 'See other things that are must haves for baby at [...]', 'I bought this a few times for my old
er son and have bought it again for my newborn.']
["We were able to keep track of baby's feeding, sleep and diaper change schedule for the first two and a half m
onths of her life.", 'Made life easier when the doctor would ask questions about habits because we had it all r
ight there!', 'It has been so helpful to be able to go back to track trends, answer pediatrician questions, or
communicate with each other when you are up at different times of the night with a newborn.', 'I think it is on
e of those things that everyone should be required to have before they leave the hospital.', 'This is super eas
y to use and helps me keep track of his daily routine.']
```

Positive and Negative Words - 5

```
#Subjectivity Count features

def readSubjectivity(path):
    flexicon = open(path, 'r')
    sldict = { }
    for line in flexicon:
        fields = line.split()
        strength = fields[0].split("=")[1]
        word = fields[2].split("=")[1]
        posTag = fields[3].split("=")[1]
        stemmed = fields[4].split("=")[1]
        polarity = fields[5].split("=")[1]
        if (stemmed == 'y'):
            isStemmed = True
        else:
            isStemmed = False
        sldict[word] = [strength, posTag, isStemmed, polarity]
    return sldict
```

```
SLpath = 'subjclueslen1-HLTEMNLP05.tff'
SL = readSubjectivity(SLpath)
print(SL['absolute'])
```

```
['strongsubj', 'adj', False, 'neutral']
```

Subjectivity Features

```
SLpath = 'subjclueslen1-HLTEMNLP05.tff'
SL = readSubjectivity(SLpath)
print(SL['absolute'])
```

```
['strongsubj', 'adj', False, 'neutral']
```

```
def SLFeatures(document, wordFeatures, SL):
    document_words = set(document)
    features = {}
    for word in wordFeatures:
        features['contains(%s)' % word] = (word in document_words)
        weakPos = 0
        strongPos = 0
        weakNeg = 0
        strongNeg = 0
        for word in document_words:
            if word in SL:
                strength, posTag, isStemmed, polarity = SL[word]
                if strength == 'weaksubj' and polarity == 'positive':
                    weakPos += 1
                if strength == 'strongsubj' and polarity == 'positive':
                    strongPos += 1
                if strength == 'weaksubj' and polarity == 'negative':
                    weakNeg += 1
                if strength == 'strongsubj' and polarity == 'negative':
                    strongNeg += 1
                features['positivecount'] = weakPos + (2 * strongPos)
                features['negativecount'] = weakNeg + (2 * strongNeg)
    return features
```

```
SL_featuresets = [(SLFeatures(sent, wordFeatures, SL), labels) for (sent,labels) in sentlabels]

print(SL_featuresets[0][0]['positivecount'])
print(SL_featuresets[0][0]['negativecount'])

trainSet, testSet = SL_featuresets[1000:], SL_featuresets[:1000]
Classifier3 = nltk.NaiveBayesClassifier.train(trainSet)
print("Accuracy:",nltk.classify.accuracy(Classifier3, testSet))
```

```
2
6
Accuracy: 0.76
```

Features Extraction

```
#Importing the required packages
dir(nltk.metrics)
from nltk.metrics import *
```

```
naiveBayesClassifier = nltk.NaiveBayesClassifier.train(trainSet)
```

```
new_list = []
test_list = []
for (features, label) in testSet:
    new_list.append(label)
    test_list.append(naiveBayesClassifier.classify(features))
```

```
#Confusion Matrix
confusion_matrix = ConfusionMatrix(new_list, test_list)
print(confusion_matrix)
```

```
     |   n   p |
     |   e   o |
     |   g   s |
-----+---------+
neg  |<380>119 |
pos  | 121<380>|
-----+---------+
(row = reference; col = test)
```

```
from sklearn.svm import LinearSVC
from nltk.classify.scikitlearn import SklearnClassifier
sklearnClassifier=nltk.classify.SklearnClassifier(LinearSVC()).train(trainSet)
nltk.classify.accuracy(sklearnClassifier, testSet)
```

```
0.744
```

```
from sklearn.metrics import confusion_matrix
tn, fp, fn, tp = confusion_matrix(new_list, test_list).ravel()
(tn, fp, fn, tp)
```

```
(380, 119, 121, 380)
```

Confusion Matrix

```
precision=(tp/(tp+fp))
```

```
precision
```

```
0.7615230460921844
```

```
recall=(tp/(tp+fn))
```

```
recall
```

```
0.7584830339321357
```

```
score=(precision*recall/(precision+recall))
```

```
f_measure_score=2*score
```

```
f_measure_score
```

```
0.76
```

Precision, Recall and F-Measure

```
def sentenceCreation(temp_textList):
    sentence = ""
    for i in range(len(temp_textList)):
        sentence+=  temp_textList[i] + " "
    return sentence
```

```
positivefile = open("positive.txt","w")
negativefile = open("negative.txt","w")

for i in range(len(temp_textList)):
    sent, orig_sent = temp_textList[i], finalsentlist[i]
    if((classifier1.classify(documentFeatures(sent,wordFeatures))) == 'pos'):
        positivefile.write(sentenceCreation(orig_sent))
        positivefile.write("\n")
    else:
        negativefile.write(sentenceCreation(orig_sent))
        negativefile.write("\n")

positivefile.close()
negativefile.close()
```

Creating 2 text files