IST 664
Natural Language Processing

Assignment:
Homework 4

Rashmitha Varma Pandati
SUID: 622666081

Table of Contents

SENTIMENT ANALYSIS OF AMAZON PRODUCT REVIEWS

1. Introduction:

The sentiment analysis on amazon product reviews helps in gaining an understanding of the context of the reviews to better modelling of online social behavior.

2. Descriptive Analysis of the Dataset

The dataset used in this homework is Amazon Products Data, provided by Julian McAuley at http://jmcauley.ucsd.edu/data/amazon/. This dataset contains product reviews and metadata from Amazon, including 142.8 million reviews spanning May 1996 – July 2014. It includes reviews (ratings, text, helpfulness votes), product metadata (descriptions, category information, price, brand, and image features), and links (viewed/ bought graphs). We will be using the **"Baby.txt"** file which is dataset of Baby products on Amazon. For our tasks, we will use only 5-core subsets of dataset. 5-core subsets mean that all users and items in the dataset have at least 5 reviews.

3. Modified Review File of the Dataset

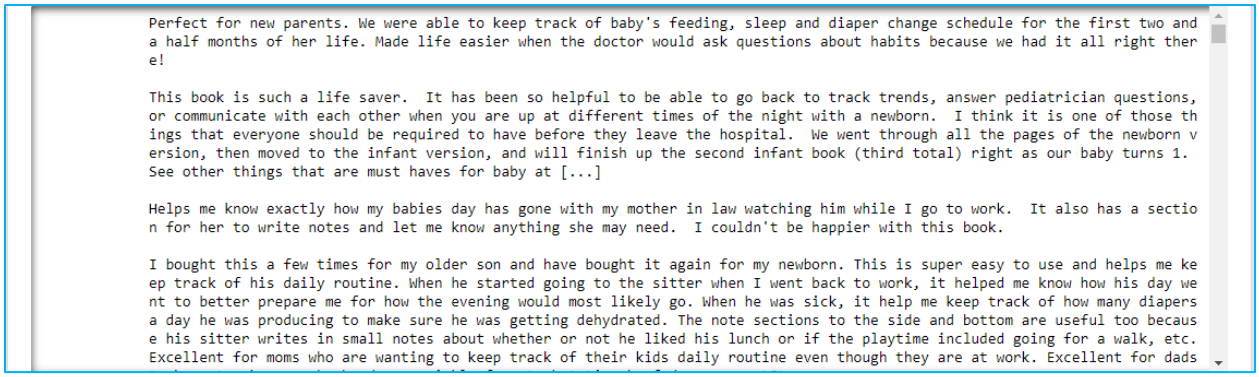
The raw data has been transformed into the following details for easy extraction:

- **reviewerID:** ID of the reviewer
- **asin:** ID of the product
- **reviewerName:** name of the reviewer
- **helpful:** helpfulness rating of the review, e.g. 2/3
- **reviewText:** text of the product
- **overall:** rating of the product
- **summary:** summary of the review
- **unixReviewTime:** time of the review (unix time)
- **reviewTime:** time of the review (raw)

4. Data Pre-processing

- We perform various data pre-processing tasks before classifying the trained model.
- We tokenize the sentences using Regular expression to remove non alpha characters like punctuation and other characters by converting them to lower cases.
- We remove Stopwords from the sentences since it occupies most part of the sentences. We use Stopwords defined for English.

❖ **Reviews before Pre-processing**



```
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 months of her life. Made life easier when the doctor would ask questions about habits because we had it all right ther
e!

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.  I think it is one of those th
ings that everyone should be required to have before they leave the hospital.  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 [...]

Helps me know exactly how my babies day has gone with my mother in law watching him while I go to work.  It also has a sectio
n for her to write notes and let me know anything she may need.  I couldn't be happier with this book.

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 ke
ep 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 we
nt 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 becaus
e his sitter writes in small notes about whether or not he liked his lunch or if the playtime included going for a walk, etc.
Excellent for moms who are wanting to keep track of their kids daily routine even though they are at work. Excellent for dads
```

❖ **Reviews after Pre-processing**

```
('perfect', 'new', 'parents.', 'able', 'keep', 'track', "baby's", 'feeding,', 'sleep', 'diaper', 'change', 'schedule', 'firs
t', 'two', 'half', 'months', 'life.', 'made', 'life', 'easier', 'doctor', 'ask', 'questions', 'habits', 'right', 'there!')

('book', 'life', 'saver.', 'helpful', 'able', 'go', 'back', 'track', 'trends,', 'answer', 'pediatrician', 'questions,', 'comm
unicate', 'different', 'times', 'night', 'newborn.', 'think', 'one', 'things', 'everyone', 'required', 'leave', 'hospital.',
'went', 'pages', 'newborn', 'version,', 'moved', 'infant', 'version,', 'finish', 'second', 'infant', 'book', '(third', 'tota
l)', 'right', 'baby', 'turns', '1.', 'see', 'things', 'haves', 'baby', '[...]')

('helps', 'know', 'exactly', 'babies', 'day', 'gone', 'mother', 'law', 'watching', 'go', 'work.', 'also', 'section', 'write',
'notes', 'let', 'know', 'anything', 'may', 'need.', 'happier', 'book.')

('bought', 'times', 'older', 'son', 'bought', 'newborn.', 'super', 'easy', 'use', 'helps', 'keep', 'track', 'daily', 'routin
e.', 'started', 'going', 'sitter', 'went', 'back', 'work,', 'helped', 'know', 'day', 'went', 'better', 'prepare', 'evening',
'likely', 'go.', 'sick,', 'help', 'keep', 'track', 'many', 'diapers', 'day', 'producing', 'make', 'sure', 'getting', 'dehydra
ted.', 'note', 'sections', 'side', 'bottom', 'useful', 'sitter', 'writes', 'small', 'notes', 'whether', 'liked', 'lunch', 'pl
aytime', 'included', 'going', 'walk,', 'etc.excellent', 'moms', 'wanting', 'keep', 'track', 'kids', 'daily', 'routine', 'eve
n', 'though', 'work.', 'excellent', 'dads', 'keep', 'track', 'husband', 'quickly', 'forget', 'time', 'fed', 'son.', 'lol')

('wanted', 'alternative', 'printing', 'daily', 'log', 'sheets', 'nanny', 'fill', 'out,', 'worked', 'great!', "i'm", 'longer',
```

5. Sentiment Analysis

- We perform sentiment analysis at sentence level by training the model using sentence_polarity corpus.
- Sentence_polarity corpus provides the set of positive sentences and negative sentences related to movie reviews and in addition, it also contains some of labeled positive and negative sentences.
- Here, we compare two models trained using two different feature sets which are:

➢ **Feature set without removal of Stopwords**

- We use Bag Of Words Feature/ Unigram feature and document vectorization to define our feature sets.
- First, we define each sentences in sentence_polarity corpus to be documents associated with its category. (either positive or negative).
- We collect most frequent words among the documents defined (sentence_polarity corpus) that define our entire corpus. These most frequent common words are our features. Here we select around 2000 features.

```
In [18]: #Categories of sentences
         sentences = sentence_polarity.sents()
         print(sentence_polarity.categories())

         ['neg', 'pos']

In [19]: #Positive and Negative Sentences
         positiveSentences = sentence_polarity.sents(categories="pos")
         print(len(positiveSentences))
         negativeSentences = sentence_polarity.sents(categories= "neg")
         print(len(negativeSentences))

         5331
         5331

In [20]: #Combining both categories
         documents = [(sent, cat) for cat in sentence_polarity.categories()
             for sent in sentence_polarity.sents(categories=cat)]

In [21]: #Shuffling
         random.shuffle(documents)
```

```
In [22]: #Not removing stopwords approach
         all_words_list = [word for (sent,cat) in documents for word in sent]
         all_words = nltk.FreqDist(all_words_list)
         word_items = all_words.most_common(2000)
         word_features = [word for (word, freq) in word_items]

In [26]: word_features[:30]

Out[26]: ['.',
          'the',
          ',',
          'a',
          'and',
          'of',
          'to',
          'is',
          'in',
          'that',
          'it',
          'as',
          'but',
          'with',
          'film',
          'this',
          'for',
          'its',
          'an',
          'movie',
          "it's",
          'be',
```

- **Feature set with removal of Stopwords**
  - We use Bag Of Words Feature/ Unigram feature and document vectorization to define our feature sets.
  - First, we define each sentences in sentence_polarity corpus to be documents associated with its category. (either positive or negative).
  - We collect most frequent words among the documents defined (sentence_polarity corpus) that define our entire corpus. These most frequent common words are our features. Here we select around 2000 features. However, the most frequent common words are filtered with stop words and hence contain only words which are not present in stopwords.

```
In [18]: #Categories of sentences
         sentences = sentence_polarity.sents()
         print(sentence_polarity.categories())

         ['neg', 'pos']

In [19]: #Positive and Negative Sentences
         positiveSentences = sentence_polarity.sents(categories="pos")
         print(len(positiveSentences))
         negativeSentences = sentence_polarity.sents(categories= "neg")
         print(len(negativeSentences))

         5331
         5331

In [20]: #Combining both categories
         documents = [(sent, cat) for cat in sentence_polarity.categories()
             for sent in sentence_polarity.sents(categories=cat)]

In [21]: #Shuffling
         random.shuffle(documents)
```

```
In [30]: #Removing stopwords approach
         SW_all_words_list = [word for (sent,cat) in documents for word in sent if word not in stoplist]
         SW_all_words = nltk.FreqDist(SW_all_words_list)
         SW_word_items = SW_all_words.most_common(2000)
         SW_word_features = [word for (word, freq) in SW_word_items]

In [38]: SW_word_features[:30]

Out[38]: ['.',
          ',',
          'film',
          'movie',
          'one',
          'like',
          '"',
          '--',
          'story',
          'much',
          'even',
          'good',
          'comedy',
          'time',
          'characters',
          'little',
          'way',
          'funny',
          'make',
          'enough',
          'never',
          'makes',
          'may',
          'us',
```

6. Classification of Feature Sets

We classify the obtained two feature sets using Naïve Bayes Classifier:

- **Feature set without removal of stop words**
  - We define our featuresets having the features defined for each sentences or documents. The BOW approach is used. BOW featureset contains word followed by its presence or absence resulting in sparse matrix or array.
  - We train the featureset using naïve bayes classifier. The approach is based on 90/10 split ratio (corresponding to train and test split). The model generated is used for further process.
  - The accuracy of the model is calculated using classifier accuracy provided by nltk on test set
  - Final step is to classify our preprocessed baby reviews/tokens using the trained model. And print the statistics involving no of positive or negative sentences classified using the trained model.

```
In [23]: 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

In [24]: featuresets = [(document_features(d,word_features), c) for (d,c) in documents]

In [25]: train_set, test_set = featuresets[1000:], featuresets[:1000]
         classifier = nltk.NaiveBayesClassifier.train(train_set)
         print (nltk.classify.accuracy(classifier, test_set))

         0.754
```

```
In [27]: positivePrediction=[]
         negativePrediction=[]
         print("Classifying the Clothes_Shoes_Jewellery reviews without removing stopwords:")
         for review in mylist2:
             reviewFeatures=document_features(review, word_features)
             prediction = classifier.classify(reviewFeatures)
             if prediction == "pos":
                 positivePrediction.append({review,prediction})
             else:
                 negativePrediction.append({review,prediction})
             print("\n.....................................................")
             print(review)
             print(prediction)

         Classifying the Clothes_Shoes_Jewellery reviews without removing stopwords:

         .....................................................
         ('perfect', 'new', 'parents.', 'able', 'keep', 'track', "baby's", 'feeding,', 'sleep', 'diaper', 'change', 'schedule', 'firs
         t', 'two', 'half', 'months', 'life.', 'made', 'life', 'easier', 'doctor', 'ask', 'questions', 'habits', 'right', 'there!')
         pos

         .....................................................
         ('book', 'life', 'saver.', 'helpful', 'able', 'go', 'back', 'track', 'trends,', 'answer', 'pediatrician', 'questions,', 'comm
         unicate', 'different', 'times', 'night', 'newborn.', 'think', 'one', 'things', 'everyone', 'required', 'leave', 'hospital.',
         'went', 'pages', 'newborn', 'version,', 'moved', 'infant', 'version,', 'finish', 'second', 'infant', 'book', '(third', 'tota
         l)', 'right', 'baby', 'turns', '1.', 'see', 'things', 'haves', 'baby', '[...]')
         pos

         .....................................................
         ('helps', 'know', 'exactly', 'babies', 'day', 'gone', 'mother', 'law', 'watching', 'go', 'work.', 'also', 'section', 'write',
         'notes', 'let', 'know', 'anything', 'may', 'need.', 'happier', 'book.')
         neg
```

- **Feature set with removal of stop words**
  - We define our featuresets having the features defined for each sentences or documents. The BOW approach is used. BOW featureset contains word followed by its presence or absence resulting in sparse matrix or array.
  - We train the featureset using naïve bayes classifier. The approach is based on 90/10 split ratio (corresponding to train and test split). The model generated is used for further process.
  - The accuracy of the model is calculated using classifier accuracy provided by nltk on test set.
  - Final step is to classify our preprocessed baby reviews/tokens using the trained model. And print the statistics involving no of positive or negative sentences classified using the trained model.
  - Total number of reviews of Baby Products on Amazon: **160792**
  - Total number of positive reviews: **68502**
  - Total number of negative reviews: **92290**

```
In [31]: # bag of words approach
def SW_document_features(SW_document, SW_word_features):
    SW_document_words = set(SW_document)
    SW_features = {}
    for word in SW_word_features:
        SW_features['contains({})'.format(word)] = (word in SW_document_words)
    return SW_features

In [32]: # define the feature sets using the document_features
SW_featuresets = [(SW_document_features(d,SW_word_features), c) for (d,c) in documents]

In [33]: # Train and test your model for accuracy
SW_train_set, SW_test_set = SW_featuresets[1000:], SW_featuresets[:1000]
SW_classifier = nltk.NaiveBayesClassifier.train(SW_train_set)
print (nltk.classify.accuracy(classifier, SW_test_set))

0.742
```

```
In [36]: SW_positivePrediction=[]
SW_negativePrediction=[]
print("Classifying the Clothes_Shoes_Jewellery reviews using stopwords:")
for review in mylist2:
    SW_reviewFeatures=SW_document_features(review, SW_word_features)
    SW_prediction = SW_classifier.classify(SW_reviewFeatures)
    if prediction == "pos":
        SW_positivePrediction.append({review,SW_prediction})
    else:
        SW_negativePrediction.append({review,SW_prediction})
    print("\n.....................................................")
    print(review)
    print(SW_prediction)

Classifying the Clothes_Shoes_Jewellery reviews using stopwords:

.....................................................
('perfect', 'new', 'parents.', 'able', 'keep', 'track', "baby's", 'feeding,', 'sleep', 'diaper', 'change', 'schedule', 'firs
t', 'two', 'half', 'months', 'life.', 'made', 'life', 'easier', 'doctor', 'ask', 'questions', 'habits', 'right', 'there!')
pos

.....................................................
('book', 'life', 'saver.', 'helpful', 'able', 'go', 'back', 'track', 'trends,', 'answer', 'pediatrician', 'questions,', 'comm
unicate', 'different', 'times', 'night', 'newborn.', 'think', 'one', 'things', 'everyone', 'required', 'leave', 'hospital.',
'went', 'pages', 'newborn', 'version,', 'moved', 'infant', 'version,', 'finish', 'second', 'infant', 'book', '(third', 'tota
l)', 'right', 'baby', 'turns', '1.', 'see', 'things', 'haves', 'baby', '[...]')
pos

.....................................................
('helps', 'know', 'exactly', 'babies', 'day', 'gone', 'mother', 'law', 'watching', 'go', 'work.', 'also', 'section', 'write',
'notes', 'let', 'know', 'anything', 'may', 'need.', 'happier', 'book.')
neg
```

7. Subjectivity

- These words are often used as a feature themselves or in conjunction with some other information.
- Two more features that involve counting the positive and negative subjectivity words present in the document are created.
- These features hold the counts of all the positive and negative subjective words, where each weakly subjective word is counted once, and each strongly subjective word is counted twice.
- After doing this, I again constructed the new feature set and created the training and test sets for this new word_features and calculated the accuracy again.

In [59]:
```
from nltk.corpus import subjectivity
```

In [60]:
```
def readSubjectivity(path):
    flexicon = open(path, 'r')
    # initialize an empty dictionary
    sldict = { }
    for line in flexicon:
        fields = line.split()   # default is to split on whitespace
        # split each field on the '=' and keep the second part as the value
        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
        # put a dictionary entry with the word as the keyword
        #     and a list of the other values
        sldict[word] = [strength, posTag, isStemmed, polarity]
    return sldict
```

In [62]:
```
SLpath = 'subjclueslen1-HLTEMNLP05.tff'
SL = readSubjectivity(SLpath)
print(SL['absolute'])
```

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

In [66]:
```
# define the features, to find out the feature_set
def SL_features(document, word_features, SL):
    document_words = set(document)
    features = {}
    for word in word_features:
        features['contains(%s)' % word] = (word in document_words)
        # count variables for the 4 classes of subjectivity
        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
```

In [67]:
```
#define the feature set for performinh the classification
# word features here is the revised word features after removing the stop words
SL_featuresets = [(SL_features(d, word_features, SL), c) for (d,c) in documents]
```

In [64]:
```
print(SL_featuresets[0][0]['positivecount'])
print(SL_featuresets[0][0]['negativecount'])
```

```
1
7
```

In [65]:
```
train_set, test_set = SL_featuresets[1000:], SL_featuresets[:1000]
classifier = nltk.NaiveBayesClassifier.train(train_set)
print(nltk.classify.accuracy(classifier, test_set))
```

```
0.772
```

8. Representing Negation

This classification technique is based on the negation of opinions. There are different ways to handle these negative words ending with n't. One way to deal with these words is to negate all the words after the negative word. Other technique is to negate the word following the negative word which has been used in this program.

- **Here is a sample list of negation words, including some adverbs called Approximate Negators:**
  { “no”, “not”, “never”, “none”, “rather”, “hardly”, “scarcely”, “rarely”, “seldom”, “neither”, “nor”, “couldn't”, “wasn't”, “didn't”, “wouldn't”, “shouldn't”, “weren't”, “don't”, “doesn't”, “haven't”, “hasn't”, “won't”, “hadn't” }

- **Process:**
  - Go through the document words in order of adding the word_features, but with a condition that if the word follows a negation word, then change the feature of that word to negated word.
  - Rerun the classifier by defining the word_features considering the negation words.
  - Accuracy on randomly split sentences this time will give the highest as compared to previous approaches.

```
In [39]: #Negation words
negationwords = ['no', 'not', 'never', 'none', 'nowhere', 'nothing', 'noone', 'rather', 'hardly', 'scarcely', 'rarely', 'seldom',

In [40]: #Negation features
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
```

```
In [42]: #Negation Classification
train_set, test_set = NOT_featuresets[1000:], NOT_featuresets[:1000]
classifier = nltk.NaiveBayesClassifier.train(train_set)
print(nltk.classify.accuracy(classifier, test_set))

0.77

In [43]: #Most Informatiovefeatures of Negation Classification
classifier.show_most_informative_features(30)

Most Informative Features
    contains(engrossing) = True              pos : neg    =     19.0 : 1.0
      contains(captures) = True              pos : neg    =     17.0 : 1.0
       contains(routine) = True              neg : pos    =     15.7 : 1.0
       contains(generic) = True              neg : pos    =     15.7 : 1.0
      contains(mediocre) = True              neg : pos    =     14.4 : 1.0
      contains(intimate) = True              pos : neg    =     14.3 : 1.0
            contains(90) = True              neg : pos    =     13.7 : 1.0
          contains(flat) = True              neg : pos    =     13.0 : 1.0
     contains(inventive) = True              pos : neg    =     13.0 : 1.0
        contains(boring) = True              neg : pos    =     12.7 : 1.0
     contains(wonderful) = True              pos : neg    =     12.6 : 1.0
          contains(warm) = True              pos : neg    =     12.6 : 1.0
     contains(NOTenough) = True              neg : pos    =     12.4 : 1.0
   contains(refreshingly) = True             pos : neg    =     12.3 : 1.0
      contains(haunting) = True              pos : neg    =     12.3 : 1.0
         contains(vivid) = True              pos : neg    =     12.3 : 1.0
     contains(realistic) = True              pos : neg    =     11.6 : 1.0
       contains(culture) = True              pos : neg    =     11.5 : 1.0
         contains(stale) = True              neg : pos    =     11.0 : 1.0
         contains(jokes) = True              neg : pos    =     11.0 : 1.0
```

9. References

- Course Contents