



APRIL 21, 2019


# NATURAL LANGUAGE PROCESSING

## HOMEWORK ASSIGNMENT 3

RAHUL NARPATRAJ RATHOD

MS INFORMATION MANAGEMENT

[rrathod@syr.edu](mailto:rrathod@syr.edu)



# Sentiment Analysis of Amazon Product Reviews

## Data Exploration

I went through the data that was provided to us on blackboard (baby.txt) to get sense of the data. I explored the format of the text it was a raw data with information which includes reviews (ratings, text, helpfulness votes), product metadata (descriptions, category information, price, brand, and image features), and links (also viewed/also bought graphs). As per the requirement of this assignment I extract only the reviewText from the text file.

## Data Pre-processing

First, I read the data from baby.txt to baby.txt. To extract just the review texts from the text file I used regular expression.

Below is the python code that extract review texts.

```
In [2]:
bt = open('baby.txt','r')
babytxt = bt.read()
bt.close()

In [3]:
r = re.compile('reviewerID:(.*)\nasin:(.*)\nreviewerName:(.*)\nhelpful:(.*)\nreviewText:(.*)\n(overall:.*\nsummary:.*\n)'
reviews = re.findall(r, babytxt)

reviewsText = []
for i in range(len(reviews)):
    reviewsText.append(reviews[i][4])

In [4]:
for i in reviewsText:
    print(i)

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 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 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 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 section 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 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 helped me keep track of how many diapers a day he was producing to make sure he was getting hydrated. The note sections to the side and bottom are useful too because 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 to keep track as my husband can quickly forget what time he fed our son. LOL
I wanted an alternative to printing out daily log sheets for the nanny to fill out, and this has worked out great! I'
```

## Sentimental Analysis

First, I tokenized the review texts on sentence level. Then after tokenizing the text at sentence level I further tokenize the sentences at the word level.

```
In [6]: sents = [nltk.sent_tokenize(sent) for sent in reviewsText]
```

```
In [7]: Sentencetokens = []
for review in sents:
    sent_wordlevel = [nltk.word_tokenize(sent) for sent in review]
    for token in sent_wordlevel:
        Sentencetokens.append(token)
```

```
In [8]: for i in Sentencetokens:
        print(i)
```

```
['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', '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', 'with', 'each', 'other', 'when', 'you', 'are', 'up', 'at', 'a',
different', 'times', 'of', 'the', 'night', 'with', 'a', 'newborn', '.']
['I', 'think', 'it', 'is', 'one', 'of', 'those', 'things', 'that', 'everyone', 'should', 'be', 'required', 'to', 'hav
e', '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', 'second', 'infant', 'book', '(', 'third', 'total',
```

I converted all the tokens to lowercase and obtain the following tokens.

```
In [9]: lowercasetokens = []
for tokens in Sentencetokens:
    lowercasetokens.append([t.lower() for t in tokens])
```

```
In [58]: lowercasetokens[:10]
```

```
Out[58]: [['perfect', 'for', 'new', 'parents', '.'],
['we',
'were',
'able',
'to',
'keep',
'track',
'of',
'baby',
"'s",
'feeding',
',',
'sleep',
'and',
'diaper',
'change',
'schedule',
'for',
'the',
'infant',
'book',
'(',
'third',
'total',
```

## Sentiment Analysis

Now that all the text preprocessing is done, and I have my tokens ready I can move on to sentiment analysis. I will be using NavieBayesClassifier. I started by loading the sentence\_polarity corpus. I also imported the random module to shuffle my document in future.

```
In [60]: from nltk.corpus import sentence_polarity
nltk.download('sentence_polarity')
```

The review text sentences are not labeled individually but can be retrieved by category. I first create the list of documents where each document(sentence) is paired with its label. After that I randomly shuffled the document using random module because all the documents were labelled in order.

```
In [11]: sentences = sentence_polarity.sents()
documents = [(sent, cat) for cat in sentence_polarity.categories()
              for sent in sentence_polarity.sents(categories=cat)]

#randomly shuffling the list
random.shuffle(documents)
```

```
In [12]: for i in documents:
          print(i)

('young', 'guys', 'doing', 'strange', 'guy', 'things', '.'], 'neg')
(['may', 'take', 'its', 'sweet', 'time', 'to', 'get', 'wherever', "it's", 'going', ',', 'but', 'if', 'you', 'have', 'the', 'patience', 'for', 'it', ',', 'you', "won't", 'feel', 'like', "it's", 'wasted', 'yours', '.'], 'pos')
(['according', 'to', 'wendigo', ',', '"nature"', 'loves', 'the', 'members', 'of', 'the', 'upper', 'class', 'almost', 'as', 'much', 'as', 'they', 'love', 'themselves', '.'], 'neg')
(["it's", 'about', 'issues', 'most', 'adults', 'have', 'to', 'face', 'in', 'marriage', 'and', 'i', 'think', "that's", 'what', 'i', 'liked', 'about', 'it', '--', 'the', 'real', 'issues', 'tucked', 'between', 'the', 'silly', 'and', 'crude', 'storyline', '.'], 'pos')
(['ringu', 'is', 'a', 'disaster', 'of', 'a', 'story', ',', 'full', 'of', 'holes', 'and', 'completely', 'lacking', 'in', 'chills', ',', 'ignore', 'the', 'reputation', ',', 'and', 'ignore', 'the', 'film', '.'], 'neg')
(['in', 'this', 'film', 'we', 'at', 'least', 'see', 'a', 'study', 'in', 'contrasts', ',', 'the', 'wide', 'range', 'of', 'one', 'actor', ',', 'and', 'the', 'limited', 'range', 'of', 'a', 'comedian', '.'], 'neg')
(['this', 'is', 'an', 'extraordinary', 'film', ',', 'not', 'least', 'because', 'it', 'is', 'japanese', 'and', 'yet', 'feels', 'universal', '.'], 'pos')
(['there', 'are', 'now', 'two', 'signs', 'that', 'm', 'night', "shyamalan's", 'debut', 'feature', 'sucked', 'up', 'all', 'he', 'has', 'to', 'give', 'to', 'the', 'mystic', 'genres', 'of', 'cinema', ':', 'unbreakable', 'and', 'sign
```

```
In [13]: 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 [14]: print(len(word_features))

2000
```

I defined the set of words that will be used for features. I choose the top 2000 words

```
In [13]: 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 [14]: print(len(word_features))

2000
```

## Bag of Words

I defined the features for each document, using Bag of Word approach. 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. Then I created the feature sets for the documents and applied to a NaiveBayesClassification. Thereafter defined the training and test sets.

```

In [14]: 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

In [15]: featuresets = [(documentFeatures(d,wordFeatures), c) for (d,c) in documents]

In [16]: trainSet, testSet = featuresets[1000:], featuresets[:1000]
          classifier = nltk.NaiveBayesClassifier.train(trainSet)
          print (nltk.classify.accuracy(classifier, testSet))

0.752

```

## Subjectivity Count

I first read subjectivity words from subjectivity lexicon file provided by our instructor. Then I created two features to count positive and negative words present in the document

```

: 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'])

```

I created feature extraction function that has all the word features as before, but also has two features 'positivecount' and 'negativecount'. These features contain counts of all the positive and negative subjectivity words. Then I created feature sets as before, but using this feature extraction function. Further I trained my features on the model and there was an increase in the accuracy

```

]: SL_featuresets = [(SLFeatures(d, wordFeatures, SL), c) for (d,c) in documents]

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

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

2
2
0.761

```

## Negation Features

In this method I defined a function that negates the word following the negation word. Here is one list of negation words that I have used to define the function ('no', 'not', 'never', 'none', 'nowhere', 'nothing', 'noone', 'rather', 'hardly', 'scarcely', 'rarely', 'seldom', 'neither', 'nor') and then I used the classifier to classify the polarity of a new sentence.

```
In [29]: 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
```

```
In [30]: NOT_featuresets = [(notFeatures(d, wordFeatures, negationwords), c) for (d, c) in documents]
NOT_featuresets[0][0]['contains(NOTlike)']
NOT_featuresets[0][0]['contains(always)']

trainSet, testSet = NOT_featuresets[1000:], NOT_featuresets[:1000]
naiveBayesClassifier = nltk.NaiveBayesClassifier.train(trainSet)
print(nltk.classify.accuracy(naiveBayesClassifier, testSet))

naiveBayesClassifier.show_most_informative_features(30)
```

```
0.785
Most Informative Features
contains(engrossing) = True      pos : neg = 17.9 : 1.0
contains(routine) = True        neg : pos = 15.5 : 1.0
contains(mediocre) = True       neg : pos = 14.8 : 1.0
contains(boring) = True         neg : pos = 14.0 : 1.0
contains(refreshing) = True     pos : neg = 13.2 : 1.0
contains(90) = True            neg : pos = 12.8 : 1.0
contains(wonderful) = True     pos : neg = 12.3 : 1.0
contains(flat) = True          neg : pos = 12.1 : 1.0
contains(warm) = True          pos : neg = 11.9 : 1.0
contains(urban) = True         pos : neg = 11.1 : 1.0
contains(stale) = True         neg : pos = 10.2 : 1.0
contains(mindless) = True      neg : pos = 10.2 : 1.0
contains(NOTenough) = True     neg : pos = 10.2 : 1.0
contains(powerful) = True      pos : neg = 10.0 : 1.0
contains(provides) = True     pos : neg = 9.9 : 1.0
contains(dull) = True         neg : pos = 9.8 : 1.0
contains(mesmerizing) = True  pos : neg = 9.8 : 1.0
contains(realistic) = True    pos : neg = 9.8 : 1.0
contains(challenging) = True  pos : neg = 9.8 : 1.0
contains(ages) = True         pos : neg = 9.8 : 1.0
contains(stupid) = True       neg : pos = 9.7 : 1.0
contains(unless) = True       neg : pos = 9.6 : 1.0
contains(bears) = True        neg : pos = 9.6 : 1.0
contains(offensive) = True    neg : pos = 9.6 : 1.0
```

## Cross Validation

### Stopword Filtering

I defined a function that removes the stop words from our word features. Then I create a sentiment analyzer object exclusive for this feature.

```

stopset = set(stopwords.words('english'))

def stopword_filtered_word_feats(words):
    return dict([(word, True) for word in words if word not in stopset])

sentim_analyzer_stop = SentimentAnalyzer()
sentim_analyzer_stop.add_feat_extractor(stopword_filtered_word_feats)
training_set = sentim_analyzer_stop.apply_features(training_docs)
test_set = sentim_analyzer_stop.apply_features(testing_docs)
trainer = NaiveBayesClassifier.train
classifier = sentim_analyzer_stop.train(trainer, training_set)

for key,value in sorted(sentim_analyzer_stop.evaluate(test_set).items()):
    print('{0}: {1}'.format(key, value))

/Users/rathod14/anaconda3/lib/python3.7/site-packages/nltk/twitter/__init__.py:20: UserWarning: The twython library has not been installed. Some functionality from the twitter package will not be available.
  warnings.warn("The twython library has not been installed. ")

Training classifier
Evaluating NaiveBayesClassifier results...
Accuracy: 0.668702132063755
F-measure [neg]: 0.6634423299337608
F-measure [pos]: 0.6738000611433813
Precision [neg]: 0.6748663101604279
Precision [pos]: 0.6629236013635452
Recall [neg]: 0.652398676592225
Recall [pos]: 0.6850393700787402

```

## Comparison

Features	Accuracy
Bag of Words	0.752
Subjectivity	0.761
Negation	0.785
Stop Word	0.668

As we see clearly from the table that highest accuracy is achieved when we use negation features

## Other Evaluation Measures

First, I built the reference and the test list from the classifier on the test set and than use nltk function to define the confusion matrix. Finally using the confusion matrix module I calculated the TP, FP, TN, FN

```

In [44]: refflist = []
         testlist = []
         for (features, label) in testSet:
             refflist.append(label)
             testlist.append(naiveBayesClassifier.classify(features))

In [48]: ConfMat = ConfusionMatrix(refflist, testlist)
         print(ConfMat)

      |   n   p   |
      |   e   o   |
      |   g   s   |
      +-----+
neg |<350>121 |
pos | 118<411>|
      +-----+
(row = reference; col = test)

```

```
In [51]: from sklearn.metrics import confusion_matrix  
         tn, fp, fn, tp = confusion_matrix(reflist, testlist).ravel()  
         (tn, fp, fn, tp)
```

```
Out[51]: (350, 121, 118, 411)
```