

SUVRAJIT BHATTACHARJEE

SUID:866572756

PRODUCT CATEGORY: BABY PRODUCTS REVIEWS

DATASET:

I chose the baby dataset to run my experiment.

I downloaded the dataset and then load the dataset into “baby” dataframe and continue to experiment from that.

```
def parse_gz(path):
```

```
    g = gzip.open(path, 'rb')
```

```
    for l in g:
```

```
        yield eval(l)
```

```
def convert_to_DF(path):
```

```
    i = 0
```

```
    df = { }
```

```
    for d in parse_gz(path):
```

```
        df[i] = d
```

```
        i += 1
```

```
    return pd.DataFrame.from_dict(df, orient='index')
```

```
baby= convert_to_DF('reviews_Baby_5.json.gz')
```

```
print('Dataset size: {:,} words'.format(len(baby)))
```

This code only give us the REVIEWS from the baby dataframe.

```
reviews = baby['reviewText']
```

Preprocessing:

Now we do some preprocessing. I ran some pre processing on my data like removing some of the stop words ,lemmatization.

```
stops = stopwords.words('english')
```

```
def tokenize(text):
```

```
    tokenized = word_tokenize(text)
```

```
    no_punc = []
```

```
    for review in tokenized:
```

```
        line = "".join(char for char in review if char not in string.punctuation)
```

```
        no_punc.append(line)
```

```
    tokens = lemmatize(no_punc)
```

```
    return tokens
```

```
#lemmatization
```

```
def lemmatize(tokens):
```

```
    lmtzr = WordNetLemmatizer()
```

```
    lemma = [lmtzr.lemmatize(t) for t in tokens]
```

```
    return lemma
```

```
reviews = reviews.apply(lambda x: tokenize(x))
```

BEFORE PREPROCESSING

```
Anaconda Prompt - python
>>> print(baby_text)
0      Perfect for new parents. We were able to keep ...
1      This book is such a life saver. It has been s...
2      Helps me know exactly how my babies day has go...
3      I bought this a few times for my older son and...
4      I wanted an alternative to printing out daily ...
5      This is great for basics, but I wish the space...
6      My 3 month old son spend half of his days with...
7      This book is perfect! I'm a first time new mo...
8      I wanted to love this, but it was pretty expen...
9      The Baby Tracker brand books are the absolute ...
10     During your postpartum stay at the hospital th...
11     I use this so that our babysitter (grandma) ca...
12     This book is a great way for keeping track of ...
13     Has columns for all the info I need at a glanc...
14     I like this log, but think it would work bette...
15     My wife and I have a six month old baby boy an...
16     I thought keeping a simple handwritten journal...
17     Easy to use, simple! I got this when my baby w...
18     We used this to help us keep track of pees and...
19     This item was extremely helpful, especially wi...
20     I've been using the baby tracker since the day...
21     Of course this has been a great/easy way for m...
22     I've been using this since the day my baby was...
23     I didn't think I would really use this and I w...
24     I have used this book since my son was born. ...
25     I am 4 weeks from delivery of my first baby an...
26     I have used this tracker for all 3 of my child...
27     We found this book at a rummage sale and found...
28     GREAT journal. We use this everyday to communi...
29     I have been using this baby tracker for three ...
...
168762    My daughter is so excited to finally be facing...
168763    Short story, I was very disappointed with the ...
168764    We have been using Graco car seats for years a...
168765    The Argos 3-in-1 seat is pretty much identical...
168766    This is a very sturdy and well-made forward-fa...
168767    These shrink in a wash terribly which is expec...
168768    Love the Best Bottom cloth diaper system (cove...
168769    These are now my go to inserts! I have used th...
168770    .. I received 3 sets of this 3 pack (9 total)....
168771    I have the stay dry cotton in large for my 24 ...
168772    I've been using Best Bottoms for a few months ...
168773    They are super absorbent and incredibly soft, ...
168774    I love these inserts. Our son is a very heavy...
168775    I'm pretty new to cloth diapers, but the Best ...
168776    These inserts work alright but with my son bei...
168777    I am cheap and therefor prefer the econobums. ...
168778    I ordered these after having great success wit...
168779    These are great inserts. I'm always impressed...
```

AFTER PREPROCESSING

```

Anaconda Prompt - python
>>> stops = stopwords.words('english')
>>> def tokenize(text):
...     tokenized = word_tokenize(text)
...     no_punc = []
...     for review in tokenized:
...         line = ""
...         no_punc.append(line)
...     tokens = lemmatize(no_punc)
...     return tokens
...
>>>
>>> def lemmatize(tokens):
...     lemmatizer = WordNetLemmatizer()
...     lemma = [lemmatizer.lemmatize(t) for t in tokens]
...     return lemma
...
>>> reviews = reviews.apply(lambda x: tokenize(x))
>>> reviews[10]
[During, your, postpartum, stay, at, the, hospital, the, nurse, will, ask, you, to, keep, a, log, of, your, baby, s, feeding, urination, and, bowel, movement,
However, when, you, get, home, you, do, not, have, a, nurse, to, remind, you, or, ask, when, your, baby, s, last, bowel, movement, wa, and, you, are,
so, overwhelmed, with, the, new, schedule, you, ca, nt, even, remember, what, day, it, is, The, daily, tracker, helped, u, so, much, because, once, we, g
ot, home, our, baby, cried, all, the, time, could, not, sleep, and, we, were, having, major, problem, breastfeeding, It, wa, so, bad, we, had, to, contac
t, our, pediatrician, s, after, hour, oncall, doctor, looking, at, the, tracker, we, realized, a, trend, the, baby, wa, not, feeding, for, a, long, time,
AND, we, had, very, few, wet, diaper, and, bowel, movement, This, wa, an, indication, our, baby, wa, dehydrated, and, the, doctor, suggested, we, start, pumpi
ng, breast, milk, to, see, how, much, we, get, and, supplement, with, formula, until, sufficient, breast, milk, finally, came, in, The, feeding, column,
we, wrote, how, many, ounce, of, breast, milk, or, formula, we, gave, and, how, long, he, breast, fed, for, In, the, comment, column, we, use, it, for, t
racking, how, much, water, I, drink, because, I, wa, not, drinking, enough, to, make, an, adequate, supply, of, breastmilk, For, a, while, I, wa, nt, sure,
what, wa, giving, him, horrible, gas, so, I, started, to, track, what, I, ate, in, case, it, wa, a, food, allergy, we, have, a, peanut, allergy, in,
the, family, We, even, write, note, about, his, first, smile, who, came, to, visit, ect, to, keep, in, his, memory, box, ]
>>> reviews[:10]
0 [Perfect, for, new, parent, , We, were, able, ...
1 [This, book, is, such, a, life, saver, , It, h...
2 [Helps, me, know, exactly, how, my, baby, day,...
3 [I, bought, this, a, few, time, for, my, older...
4 [I, wanted, an, alternative, to, printing, out...
5 [This, is, great, for, basic, , but, I, wish, ...
6 [My, 3, month, old, son, spend, half, of, his,...
7 [This, book, is, perfect, , I, m, a, first, ti...
8 [I, wanted, to, love, this, , but, it, wa, pre...
9 [The, Baby, Tracker, brand, book, are, the, ab...
Name: reviewText, dtype: object
>>>
```

RESULTS:

- 0 [Perfect, for, new, parent, , We, were, able, ...
- 1 [This, book, is, such, a, life, saver, , It, h...
- 2 [Helps, me, know, exactly, how, my, baby, day,...
- 3 [I, bought, this, a, few, time, for, my, older...
- 4 [I, wanted, an, alternative, to, printing, out...
- 5 [This, is, great, for, basic, , but, I, wish, ...
- 6 [My, 3, month, old, son, spend, half, of, his,...
- 7 [This, book, is, perfect, , I, m, a, first, ti...
- 8 [I, wanted, to, love, this, , but, it, wa, pre...
- 9 [The, Baby, Tracker, brand, book, are, the, ab...
- 10 [During, your, postpartum, stay, at, the, hosp...

Name: reviewText, dtype: object

Experiment:

Now we explore the sentiment of the comments at the 3 sentence level. This includes how to process the words and how to conduct the sentiment analysis using classifiers.

We collect all the words in the sentence_polarity corpus and select some number of most frequent words to be the word features.

```
reviews = reviews.apply(lambda x: tokenize(x))

from nltk.corpus import sentence_polarity
import random

sentences = sentence_polarity.sents()

documents = [(sent, reviews) for reviews in sentence_polarity.categories()
              for sent in sentence_polarity.sents(categories=reviews)]

random.shuffle(documents)

all_words_list = [word for (sent, reviews) in documents for word in sent]

all_words = nltk.FreqDist(all_words_list)

import nltk

all_words = nltk.FreqDist(all_words_list)

word_items = all_words.most_common(100)

word_features = [word for (word, freq) in word_items]

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

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

train_set, test_set = featuresets[50:], featuresets[:50]
```

```
classifier = nltk.NaiveBayesClassifier.train(train_set)
```

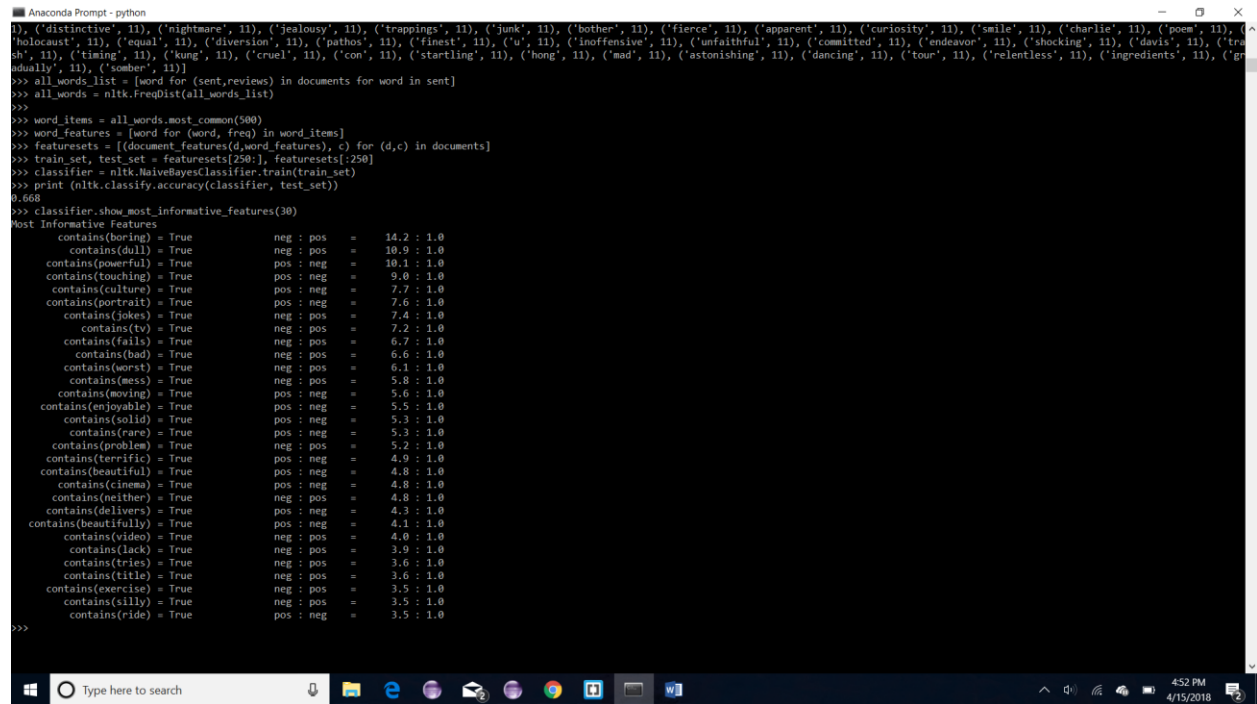
```
print (nltk.classify.accuracy(classifier, test_set))
```

0.668

```
classifier.show_most_informative_features(20)
```

Before implementing ANY lexicons features the accuracy rate is coming as 0.66.

Here is the sentence list with Negative and Positive features.



```
1), ('distinctive', 11), ('nightmare', 11), ('jealousy', 11), ('trappings', 11), ('junk', 11), ('bother', 11), ('fierce', 11), ('apparent', 11), ('curiosity', 11), ('smile', 11), ('charlie', 11), ('poem', 11), ('holocaust', 11), ('equal', 11), ('diversion', 11), ('pathos', 11), ('finest', 11), ('u', 11), ('inoffensive', 11), ('unfaithful', 11), ('committed', 11), ('endeavor', 11), ('shocking', 11), ('davis', 11), ('trashed', 11), ('tizing', 11), ('kung', 11), ('cruel', 11), ('con', 11), ('startling', 11), ('hong', 11), ('mad', 11), ('astonishing', 11), ('dancing', 11), ('tour', 11), ('relentless', 11), ('ingredients', 11), ('gradually', 11), ('somber', 11)]
>>> all_words_list = [word for (sent, reviews) in documents for word in sent]
>>> all_words = nltk.FreqDist(all_words_list)
>>>
>>> word_items = all_words.most_common(500)
>>> word_features = [word for (word, freq) in word_items]
>>> featuresets = [(document_features(d, word_features), c) for (d, c) in documents]
>>> train_set, test_set = featuresets[250:], featuresets[:250]
>>> classifier = nltk.NaiveBayesClassifier.train(train_set)
>>> print (nltk.classify.accuracy(classifier, test_set))
0.668
>>> classifier.show_most_informative_features(30)
Most Informative Features
contains(boring) = True          neg : pos = 14.2 : 1.0
contains(dull) = True           neg : pos = 18.9 : 1.0
contains(powerful) = True       pos : neg = 18.1 : 1.0
contains(touching) = True       pos : neg = 9.0 : 1.0
contains(culture) = True        pos : neg = 7.7 : 1.0
contains(portrait) = True       pos : neg = 7.6 : 1.0
contains(jokes) = True          neg : pos = 7.4 : 1.0
contains(to) = True             neg : pos = 7.2 : 1.0
contains(fails) = True          neg : pos = 6.7 : 1.0
contains(bad) = True            neg : pos = 6.6 : 1.0
contains(worst) = True          neg : pos = 6.1 : 1.0
contains(mess) = True           neg : pos = 5.8 : 1.0
contains(moving) = True         pos : neg = 5.6 : 1.0
contains(enjoyable) = True      pos : neg = 5.5 : 1.0
contains(solid) = True          pos : neg = 5.3 : 1.0
contains(rare) = True           pos : neg = 5.3 : 1.0
contains(problem) = True        pos : neg = 5.2 : 1.0
contains(terrific) = True       pos : neg = 4.9 : 1.0
contains(beautiful) = True      pos : neg = 4.8 : 1.0
contains(cinema) = True         pos : neg = 4.8 : 1.0
contains(neither) = True        neg : pos = 4.8 : 1.0
contains(delivers) = True       pos : neg = 4.3 : 1.0
contains(beautifully) = True    pos : neg = 4.1 : 1.0
contains(video) = True          neg : pos = 4.0 : 1.0
contains(lack) = True           neg : pos = 3.9 : 1.0
contains(tries) = True          neg : pos = 3.6 : 1.0
contains(title) = True          neg : pos = 3.6 : 1.0
contains(exercise) = True       neg : pos = 3.5 : 1.0
contains(silly) = True          neg : pos = 3.5 : 1.0
contains(ride) = True           pos : neg = 3.5 : 1.0
>>>
```

Now we implement Subjectivity Lexicon and run the same classifier.

```
def SL_features(document, word_features, reviews):
```

```
...     document_words = set(document)
```

```
...     features = { }
```

```
...     for word in word_features:
```

```
...         features['contains({ })'.format(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 reviews:
...             strength, posTag, isStemmed, polarity = reviews[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 = [(SL_features(d, word_features, reviews), c) for (d,c) in documents]
>>> train_set, test_set = SL_featuresets[100:], SL_featuresets[:100]
>>> classifier = nltk.NaiveBayesClassifier.train(train_set)
>>> print (nltk.classify.accuracy(classifier, test_set))

```

0.61

RESULTS:

Most Informative Features

contains(bad) = True	neg : pos = 6.3 : 1.0
contains(too) = True	neg : pos = 3.6 : 1.0
contains(?) = True	neg : pos = 2.7 : 1.0
contains(best) = True	pos : neg = 2.4 : 1.0
contains(no) = True	neg : pos = 2.2 : 1.0

contains(doesn't) = True	neg : pos = 2.1 : 1.0
contains(was) = True	neg : pos = 2.0 : 1.0
contains(only) = True	neg : pos = 1.9 : 1.0
contains(just) = True	neg : pos = 1.9 : 1.0
contains(would) = True	neg : pos = 1.9 : 1.0
contains(there's) = True	neg : pos = 1.8 : 1.0
contains(love) = True	pos : neg = 1.8 : 1.0
contains(us) = True	pos : neg = 1.8 : 1.0
contains(life) = True	pos : neg = 1.8 : 1.0
contains(been) = True	neg : pos = 1.8 : 1.0
contains(funny) = True	pos : neg = 1.8 : 1.0
contains(or) = True	neg : pos = 1.6 : 1.0
contains(so) = True	neg : pos = 1.6 : 1.0
contains(much) = True	neg : pos = 1.6 : 1.0
contains(i) = True	neg : pos = 1.5 : 1.0

We see improvement in accuracy.

The other strategy with negation words is to negate the word following the negation word. Now we try the negation strategy. We build a negation word features on top of SL word features to improve the accuracy.

```
for sent in list(sentences)[:50]:
    for word in sent:
        if (word.endswith("n't")):
            print(sent)
```

['there', 'is', 'a', 'difference', 'between', 'movies', 'with', 'the', 'courage', 'to', 'go', 'over', 'the', 'top',
'and', 'movies', 'that', "don't", 'care', 'about', 'being', 'stupid']

['a', 'farce', 'of', 'a', 'parody', 'of', 'a', 'comedy', 'of', 'a', 'premise', ',', 'it', "isn't", 'a', 'comparison', 'to',
'reality', 'so', 'much', 'as', 'it', 'is', 'a', 'commentary', 'about', 'our', 'knowledge', 'of', 'films', ',']

['i', "didn't", 'laugh', ',', 'i', "didn't", 'smile', ',', 'i', 'survived', ',']

['i', "didn't", 'laugh', ',', 'i', "didn't", 'smile', ',', 'i', 'survived', ',']

['most', 'of', 'the', 'problems', 'with', 'the', 'film', "don't", 'derive', 'from', 'the', 'screenplay', ',', 'but',
'rather', 'the', 'mediocre', 'performances', 'by', 'most', 'of', 'the', 'actors', 'involved']

['the', 'lack', 'of', 'naturalness', 'makes', 'everything', 'seem', 'self-consciously', 'poetic', 'and',
'forced', ',', ',', ',', "it's", 'a', 'pity', 'that', "[nelson's]", 'achievement', "doesn't", 'match', 'his',
'ambition', ',']

negationwords = ['no', 'not', 'never', 'none', 'nowhere', 'nothing', 'noone', 'rather', 'hardly',
'scarcely', 'rarely', 'seldom', 'neither', 'nor']

```
def NOT_features(document, word_features, negationwords):
```

```
    features = { }
```

```
    for word in SL_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 SL_features)
```

```
        else:
```

```
            features['contains({ })'.format(word)] = (word in SL_features)
```

```
    return features
```

```
NOT_featuresets = [(NOT_features(d, SL_features, negationwords), c) for (d, c) in documents]
```



```
train_set, test_set = NOT_featuresets[50:], NOT_featuresets[:50]
```

```
classifier = nltk.NaiveBayesClassifier.train(train_set)
```

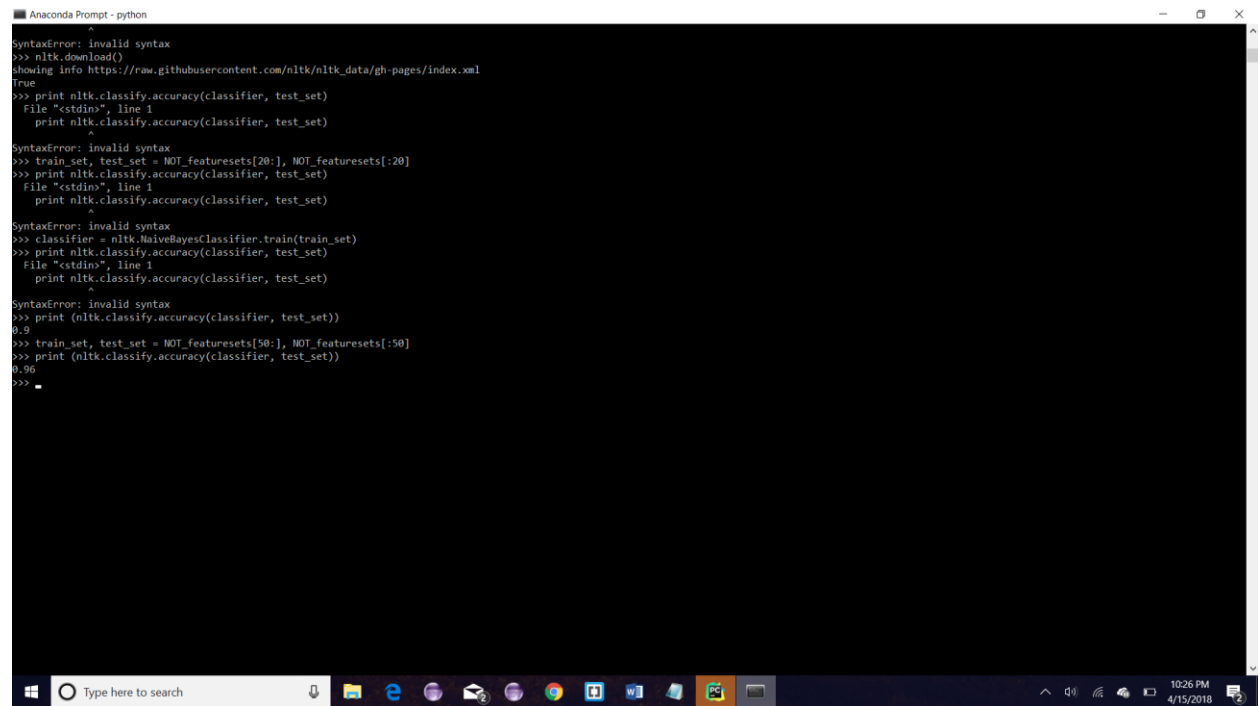
We define the negation words and then convert the double negation into positives.

```
print (nltk.classify.accuracy(classifier, test_set))
```

0.78

```
classifier.show_most_informative_features(20)
```

Now our accuracy is increased a lot.



```
Anaconda Prompt - python
>
SyntaxError: invalid syntax
>>> nltk.download()
showing info https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml
True
>>> print nltk.classify.accuracy(classifier, test_set)
File "<stdin>", line 1
    print nltk.classify.accuracy(classifier, test_set)
    ^
SyntaxError: invalid syntax
>>> train_set, test_set = NOT_featuresets[20:], NOT_featuresets[:20]
>>> print nltk.classify.accuracy(classifier, test_set)
File "<stdin>", line 1
    print nltk.classify.accuracy(classifier, test_set)
    ^
SyntaxError: invalid syntax
>>> classifier = nltk.NaiveBayesClassifier.train(train_set)
>>> print nltk.classify.accuracy(classifier, test_set)
File "<stdin>", line 1
    print nltk.classify.accuracy(classifier, test_set)
    ^
SyntaxError: invalid syntax
>>> print (nltk.classify.accuracy(classifier, test_set))
0.9
>>> train_set, test_set = NOT_featuresets[50:], NOT_featuresets[:50]
>>> print (nltk.classify.accuracy(classifier, test_set))
0.96
>>>
```

```
Select Anaconda Prompt - python
ModuleNotFoundError: No module named 'nltk'
>>> import nltk
>>> all_words = nltk.FreqDist(all_words_list)
>>> word_items = all_words.most_common(100)
>>> word_features = [word for (word, freq) in word_items]
>>> 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
>>> featuresets = [(document_features(d, word_features), c) for (d, c) in documents]
>>> train_set, test_set = featuresets[90:], featuresets[:90]
>>> classifier = nltk.NaiveBayesClassifier.train(train_set)
>>> print(nltk.classify.accuracy(classifier, test_set))
0.6
>>> classifier.show_most_informative_features(30)
Most Informative Features
contains(bad) = True          neg : pos = 6.5 : 1.0
contains(too) = True          neg : pos = 3.5 : 1.0
contains(?) = True           neg : pos = 2.4 : 1.0
contains(best) = True         pos : neg = 2.3 : 1.0
contains(no) = True           neg : pos = 2.2 : 1.0
contains(only) = True         neg : pos = 2.0 : 1.0
contains(was) = True          neg : pos = 2.0 : 1.0
contains(would) = True        neg : pos = 2.0 : 1.0
contains(doesn't) = True      neg : pos = 1.9 : 1.0
contains(love) = True         pos : neg = 1.9 : 1.0
contains(just) = True         neg : pos = 1.9 : 1.0
contains(there's) = True      neg : pos = 1.9 : 1.0
contains(funny) = True        pos : neg = 1.9 : 1.0
contains(life) = True         pos : neg = 1.8 : 1.0
contains(us) = True           pos : neg = 1.7 : 1.0
contains(or) = True           neg : pos = 1.7 : 1.0
contains(been) = True         neg : pos = 1.7 : 1.0
contains(so) = True           neg : pos = 1.6 : 1.0
contains(like) = True         neg : pos = 1.5 : 1.0
contains(i) = True            neg : pos = 1.5 : 1.0
contains(movie) = True        neg : pos = 1.5 : 1.0
contains(their) = True         pos : neg = 1.5 : 1.0
contains(much) = True         neg : pos = 1.5 : 1.0
contains(most) = True         pos : neg = 1.4 : 1.0
contains(up) = True           neg : pos = 1.4 : 1.0
contains(work) = True         pos : neg = 1.4 : 1.0
contains(he) = True           neg : pos = 1.4 : 1.0
contains(have) = True         neg : pos = 1.4 : 1.0
contains(when) = True         neg : pos = 1.4 : 1.0
contains(little) = True       neg : pos = 1.4 : 1.0
>>> for sent in list(sentences)[:50]:
```

AFTER SUBJECTIVITY:

```
Anaconda Prompt - python
>>> import nltk
>>> print(nltk.classify.accuracy(classifier, test_set))
0.61
>>> train_set, test_set = SL_featuresets[1000:], SL_featuresets[:1000]
>>> classifier = nltk.NaiveBayesClassifier.train(train_set)
>>> print(nltk.classify.accuracy(classifier, test_set))
0.616
>>> classifier.show_most_informative_features(20)
Most Informative Features
contains(bad) = True          neg : pos = 6.3 : 1.0
contains(too) = True          neg : pos = 3.6 : 1.0
contains(?) = True           neg : pos = 2.7 : 1.0
contains(best) = True         pos : neg = 2.4 : 1.0
contains(no) = True           neg : pos = 2.2 : 1.0
contains(doesn't) = True      neg : pos = 2.1 : 1.0
contains(was) = True          neg : pos = 2.0 : 1.0
contains(only) = True         neg : pos = 1.9 : 1.0
contains(just) = True         neg : pos = 1.9 : 1.0
contains(would) = True        neg : pos = 1.9 : 1.0
contains(there's) = True      pos : neg = 1.8 : 1.0
contains(love) = True         pos : neg = 1.8 : 1.0
contains(us) = True           pos : neg = 1.8 : 1.0
contains(life) = True         pos : neg = 1.8 : 1.0
contains(been) = True         neg : pos = 1.8 : 1.0
contains(funny) = True        pos : neg = 1.8 : 1.0
contains(or) = True           neg : pos = 1.6 : 1.0
contains(so) = True           neg : pos = 1.6 : 1.0
contains(much) = True         neg : pos = 1.6 : 1.0
contains(i) = True            neg : pos = 1.5 : 1.0
>>>
```

AFTER NEGATION:

```

Anaconda Prompt - python
...
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 = [(NOT_features(d, word_features, negationwords), c) for (d, c) in documents]
>>> train_set, test_set = NOT_featuresets[1000:], NOT_featuresets[:1000]
>>> classifier = nltk.NaiveBayesClassifier.train(train_set)
>>> nltk.classify.accuracy(classifier, test_set)
0.782
>>> classifier.show_most_informative_features(30)
Most Informative Features
contains(engrossing) = False      pos : neg   = 20.2 : 1.0
contains(generic) = False        neg : pos   = 17.1 : 1.0
contains(inventive) = False      pos : neg   = 14.9 : 1.0
contains(routine) = False        neg : pos   = 14.4 : 1.0
contains(flat) = False           neg : pos   = 14.3 : 1.0
contains(boring) = False         neg : pos   = 13.7 : 1.0
contains(unique) = False         pos : neg   = 13.6 : 1.0
contains(haunting) = False       pos : neg   = 12.9 : 1.0
contains(portrait) = False       pos : neg   = 12.9 : 1.0
contains(dull) = False           neg : pos   = 12.0 : 1.0
contains(warm) = False           pos : neg   = 11.7 : 1.0
contains(refreshingly) = False   pos : neg   = 11.6 : 1.0
contains(intimate) = False       pos : neg   = 11.6 : 1.0
contains(stupid) = False         neg : pos   = 11.5 : 1.0
contains(wonderful) = False      pos : neg   = 11.3 : 1.0
contains(stale) = False          neg : pos   = 11.1 : 1.0
contains(realistic) = False      pos : neg   = 10.9 : 1.0
contains(provides) = False       pos : neg   = 10.5 : 1.0
contains(beauty) = False        pos : neg   = 10.5 : 1.0
contains(NOTenough) = True       neg : pos   = 10.4 : 1.0
contains(loud) = False           neg : pos   = 9.9 : 1.0
contains(suffers) = False        neg : pos   = 9.9 : 1.0
contains(captures) = False       pos : neg   = 9.7 : 1.0
contains(offensive) = False      neg : pos   = 9.7 : 1.0
contains(mesmerizing) = False    pos : neg   = 9.6 : 1.0
contains(ages) = False           pos : neg   = 9.6 : 1.0
contains(flaws) = False          pos : neg   = 9.3 : 1.0
contains(powerful) = False       pos : neg   = 9.2 : 1.0
contains(annoying) = False       neg : pos   = 9.1 : 1.0
contains(harvard) = False        neg : pos   = 9.1 : 1.0
>>>

```

COMPARISON: We saw that after implementing negation on top of subjectivity gave us a good performance and accuracy.

RESULTS:

Most Informative Features

contains(engrossing) = False	pos : neg = 20.2 : 1.0
contains(generic) = False	neg : pos = 17.1 : 1.0
contains(inventive) = False	pos : neg = 14.9 : 1.0
contains(routine) = False	neg : pos = 14.4 : 1.0
contains(flat) = False	neg : pos = 14.3 : 1.0
contains(boring) = False	neg : pos = 13.7 : 1.0
contains(unique) = False	pos : neg = 13.6 : 1.0
contains(haunting) = False	pos : neg = 12.9 : 1.0
contains(portrait) = False	pos : neg = 12.9 : 1.0
contains(dull) = False	neg : pos = 12.0 : 1.0
contains(warm) = False	pos : neg = 11.7 : 1.0
contains(refreshingly) = False	pos : neg = 11.6 : 1.0

contains(intimate) = False	pos : neg = 11.6 : 1.0
contains(stupid) = False	neg : pos = 11.5 : 1.0
contains(wonderful) = False	pos : neg = 11.3 : 1.0
contains(stale) = False	neg : pos = 11.1 : 1.0
contains(realistic) = False	pos : neg = 10.9 : 1.0
contains(provides) = False	pos : neg = 10.5 : 1.0
contains(beauty) = False	pos : neg = 10.5 : 1.0
contains(NOTenough) = True	neg : pos = 10.4 : 1.0
contains(loud) = False	neg : pos = 9.9 : 1.0
contains(suffers) = False	neg : pos = 9.9 : 1.0
contains(captures) = False	pos : neg = 9.7 : 1.0
contains(offensive) = False	neg : pos = 9.7 : 1.0
contains(mesmerizing) = False	pos : neg = 9.6 : 1.0
contains(ages) = False	pos : neg = 9.6 : 1.0
contains(flaws) = False	pos : neg = 9.3 : 1.0
contains(powerful) = False	pos : neg = 9.2 : 1.0
contains(annoying) = False	neg : pos = 9.1 : 1.0

ADDITIONAL LEXICON:

I took the bigram word features as a baseline and see if the features you designed improve the accuracy of the classification.

```
import collections
```

```
import nltk.classify.util, nltk.metrics
```

```
from nltk.classify import NaiveBayesClassifier
```

```
def evaluate_classifier(feats):
```

```
    negids = reviews.fileids('neg')
```

```
    posids = reviews.fileids('pos')
```

```
negfeats = [(featx(reviews.words(fileids=[f])), 'neg') for f in negids]
posfeats = [(featx(reviews.words(fileids=[f])), 'pos') for f in posids]
```

```
negcutoff = len(negfeats)*3/4
poscutoff = len(posfeats)*3/4
```

```
trainfeats = negfeats[:negcutoff] + posfeats[:poscutoff]
testfeats = negfeats[negcutoff:] + posfeats[poscutoff:]
```

```
classifier = NaiveBayesClassifier.train(trainfeats)
refsets = collections.defaultdict(set)
testsets = collections.defaultdict(set)
```

```
for i, (feats, label) in enumerate(testfeats):
    refsets[label].add(i)
    observed = classifier.classify(feats)
    testsets[observed].add(i)
```

```
print 'accuracy:', nltk.classify.util.accuracy(classifier, testfeats)
print 'pos precision:', nltk.metrics.precision(refsets['pos'], testsets['pos'])
print 'pos recall:', nltk.metrics.recall(refsets['pos'], testsets['pos'])
print 'neg precision:', nltk.metrics.precision(refsets['neg'], testsets['neg'])
print 'neg recall:', nltk.metrics.recall(refsets['neg'], testsets['neg'])
classifier.show_most_informative_features()
```

For reviews:

```
def word_feats(reviewss):
```

```
return dict([(reviews, True) for word in words])
```

```
evaluate_classifier(word_feats)
```

```
accuracy: 0.66
```

```
pos precision: 0.651595744681
```

```
pos recall: 0.98
```

```
neg precision: 0.959677419355
```

```
neg recall: 0.476
```

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

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

Stopword Filtering

```
def stopword_filtered_word_feats(reviews):
```

```
    return dict([(reviews, True) for reviews in words if reviews not in stopset])
```

```
evaluate_classifier(stopword_filtered_word_feats)
```

```
accuracy: 0.726
```

```
pos precision: 0.649867374005
```

```
pos recall: 0.98
```

```
neg precision: 0.959349593496
```

```
neg recall: 0.472
```

ADDITIONAL FEATURES:

I used LDA AND NMP for topic modelling to see which are most discussed topics in Baby products reviews.

```
>>> review_text = baby["reviewText"]
```

```
>>> tfidf = tfidf_vectorizer.fit_transform(review_text)
```

```
>>> tf_vectorizer = CountVectorizer(stop_words=stops)
```

```
>>> tf = tf_vectorizer.fit_transform(review_text)
```

```
>>> tfidf_feature_names = tfidf_vectorizer.get_feature_names()
>>> print("Number of total features: {}".format(len(tfidf_feature_names)))
```

Number of total features: 63483

I created a TERM FREQUENCY DOCUMENTS MATRIX and loaded the vectorized text into my TFIDF.

Then We build clustering model Using **Nonnegative Matrix Factorization**.

```
nmf_tf = nmf.fit(tf)
```

```
nmf_ = nmf_tf.transform(tf)
```

```
Counter([np.argmax(i) for i in nmf_])
```

```
retrieve_top_words(nmf_tf, tfidf_feature_names, num_top_words)
```

Topic #0:

would great use old son little months get easy love time well still daughter put

Topic #1:

seat car seats britax facing child back rear straps easy infant use get base fit

Topic #2:

baby carrier use put also time back babies months ergo used much around first bjorn

Topic #3:

stroller easy strollers wheels basket fold city canopy great back handle love also easily bag

Topic #4:

bottles bottle nipple nipples use milk avent water clean formula cup flow dr brush breast

Topic #5:

bag diaper diapers use cloth bags wipes changing size fit pad pocket pockets pail small

Topic #6:

one little two bought another gate side first get new hand buy got second different

Topic #7:

monitor camera night room video unit sound battery see would good light feature screen turn

Topic #8:

like really would also much think get nice seems good better feel little thing well

Topic #9:

pump milk medela pumping use get work breast time suction parts much would used pumps.

```
Anaconda Prompt - python
>>> tfidf = TfidfVectorizer.fit_transform(review_text)
>>> tf_vectorizer = CountVectorizer(stop_words=stops)
>>> tf = tf_vectorizer.fit_transform(review_text)
>>> tfidf_feature_names = tfidf_vectorizer.get_feature_names()
>>> print("Number of total features: {}".format(len(tfidf_feature_names)))
Number of total features: 63483
>>> nmf = NMF(n_components=10, random_state=1,
...          alpha=1, l1_ratio=.5)
>>> lda = LatentDirichletAllocation(n_topics=10, max_iter=5,
...                               learning_method='online',
...                               learning_offset=50.,
...                               random_state=0)
>>> num_top_words = 15
>>> def retrieve_top_words(model, feature_names, num_top_words):
...     for idx, topic in enumerate(model.components_):
...         print("Topic #{}:".format(idx), end='\n')
...         print(" ".join([feature_names[i]
...                           for i in topic.argsort()[::-num_top_words - 1:-1]]), end='\n\n')
...     print()
>>> nmf_tf = nmf.fit(tf)
>>> nmf = nmf_tf.transform(tf)
>>> Counter([np.argmax(i) for i in nmf_])
Counter({0: 46872, 2: 25170, 8: 22100, 6: 19517, 5: 12623, 4: 9570, 1: 8948, 3: 5408, 7: 5340, 9: 5244})
>>> retrieve_top_words(nmf_tf, tfidf_feature_names, num_top_words)
Topic #0:
would great use old son little months get easy love time well still daughter put

Topic #1:
seat car seats britax facing child back rear straps easy infant use get base fit

Topic #2:
baby carrier use put also time back babies months ergo used much around first bjorn

Topic #3:
stroller easy strollers wheels basket fold city canopy great back handle love also easily bag

Topic #4:
bottles bottle nipple nipples use milk avent water clean formula cup flow dr brush breast

Topic #5:
bag diaper diapers use cloth bags wipes changing size fit pad pocket pockets pail small

Topic #6:
one little two bought another gate side first get new hand buy got second different

Topic #7:
monitor camera night room video unit sound battery see would good light feature screen turn

Topic #8:
```

LDA:

We do the same thing for LDA modelling.

```
lda_tf = lda.fit(tf)
```

In [145]:

```
lda_ = lda_tf.transform(tf)
```

```
Counter([np.argmax(i) for i in lda_])
```

```
retrieve_top_words(lda_tf, tfidf_feature_names, num_top_words)
```

Topic #0:

baby monitor night sleep room carrier looks back bed light video sound sleeping crib see

Topic #1:

food chair tray high table clean bibs bib eat eating chairs prefolds highchair plate dishwasher

Topic #2:

diaper soft diapers baby cover great well use cloth love crib wash pad changing mattress

Topic #3:

water tub bath open gate door wall bottom top temperature lock install plastic place wood

Topic #4:

seat car straps back seats britax strap child potty kid front carseat rear infant easy

Topic #5:

pump pillow medela milk nursing pumping breast work use back time day service customer freezer

Topic #6:

bottles bottle cup cups nipple sippy clean nipples milk straw leak avent brush baby formula

Topic #7:

one would like get baby use really little much well time good old great still

Topic #8:

stroller bag easy use great love one baby also clean small easily diaper well put

Topic #9:

baby loves old toy toys little great love son play cute months daughter one like

```
Anaconda Prompt - python
>>>
>>> lda_ = lda_tf.transform(tf)
>>> Counter([np.argmax(i) for i in lda_])
Counter((7: 89458, 9: 24441, 2: 19823, 8: 16762, 6: 3468, 5: 2337, 0: 2055, 4: 1146, 3: 1095, 1: 207))
>>> retrieve_top_words(lda_tf, tfidf_feature_names, num_top_words)
Topic #0:
baby monitor night sleep room carrier looks back bed light video sound sleeping crib see

Topic #1:
food chair tray high table clean bibs bib eat eating chairs prefolds highchair plate dishwasher

Topic #2:
diaper soft diapers baby cover great well use cloth love crib wash pad changing mattress

Topic #3:
water tub bath open gate door wall bottom top temperature lock install plastic place wood

Topic #4:
seat car straps back seats britax strap child potty kid front carseat rear infant easy

Topic #5:
pump pillow medela milk nursing pumping breast work use back time day service customer freezer

Topic #6:
bottles bottle cup cups nipple sippy clean nipples milk straw leak avent brush baby formula

Topic #7:
one would like get baby use really little much well time good old great still

Topic #8:
stroller bag easy use great love one baby also clean small easily diaper well put

Topic #9:
baby loves old toy toys little great love son play cute months daughter one like

>>>
```