

Semantic analysis of amazon product reviews

NLP HOMEWORK 4



April 22, 2018

KOMAL GUJARATHI, 211778351

SYRACUSE UNIVERSITY

Table of Contents

[Semantic Analysis of Amazon Product Reviews 2](#_Toc512202427)

[Data Pre-Processing Step (20%) 2](#_Toc512202428)

[Write the Python Code that extracts only review texts. 2](#_Toc512202429)

[Sentiment Analysis (80%) 5](#_Toc512202430)

[Detailed Technique Description 6](#_Toc512202431)

[Sentiment Analysis using Bag of words approach 6](#_Toc512202432)

[Additional Changes in the word\_feature definition 8](#_Toc512202433)

[Subjective Count Features 11](#_Toc512202434)

[Negation Features 12](#_Toc512202435)

[Classifier defined based by making the use of POS tagging and other word features 14](#_Toc512202436)

[Bonus Credit (10%) 18](#_Toc512202437)

[Appendix 19](#_Toc512202438)

[Screen Shots of code and python processing screen shots 19](#_Toc512202439)

# Semantic Analysis of Amazon Product Reviews

It is increasingly common that Internet users engage in various of online reviews. The availability of these review content offers researchers opportunities to better understand and model online social behavior. In this homework, you will conduct sentiment analysis to gain some understanding about the Amazon product reviews.

I have chosen the ‘Clothing, Shoes and Jewelry’ dataset for analysis.

This File ‘clothing\_shoes\_jewelry.txt' is mainly divided into number of reviews from the users from different years

Each review is of the format –

-  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)

# Data Pre-Processing Step (20%)

## Write the Python Code that extracts only review texts.

I have first used the regex pattern matching to match with each review in the given file. Upon processing, I could find that there are around 278225 reviews in this given text, with each review being the combination of one or more sentences by the customer of that product. Since it takes a lot of processing time to run the classifier on each of these reviews, I have mainly decided to work with a small part of this dataset. I will be working with around 5500 reviews from the year 2014.

From the reviews text, I could see that there are many reviews spanning from years including 2012, 2013, 2014. I have decided to work with 2014 reviews mainly because it is the latest year, and it will be more useful for the users of this classifier to get the reviews of the recent years, than the old set of reviews.

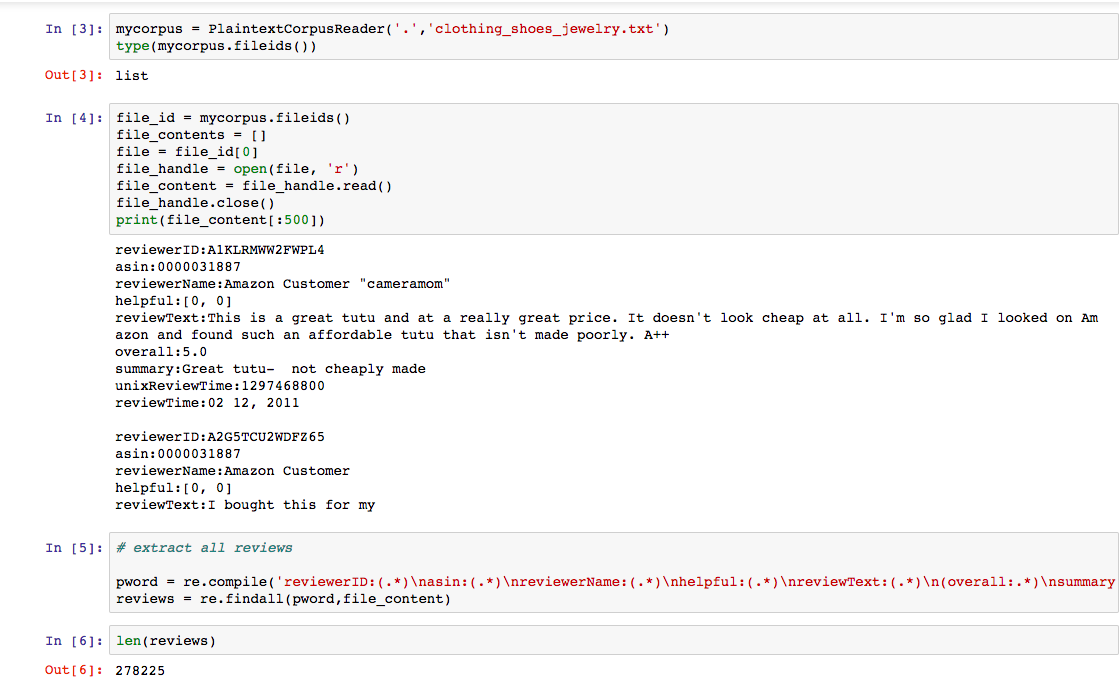
In the year 2014, there are total 67617 reviews. I have run the classifier on 5500 of these reviews as my device was taking more than 10 hours to process even small set of inputs. Please find the screen shot below.

**Code** -

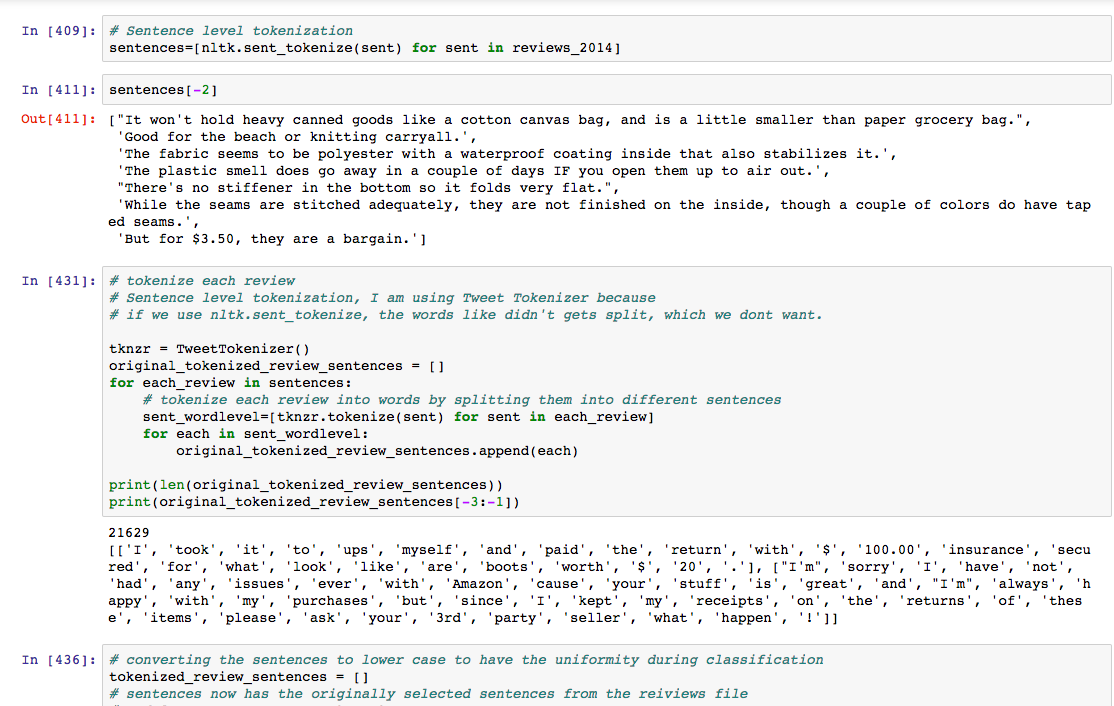
pword=re.compile('reviewerID:(.\*)\nasin:(.\*)\nreviewerName:(.\*)\nhelpful:(.\*)\nreviewText:(.\*)\n(overall:.\*)\nsummary:(.\*)\nunixReviewTime:(.\*)\nreviewTime:(.\*)\n')

reviews = re.findall(pword,file\_content)

I have used regex pattern matching to find out the reviewText of each of the chosen review.

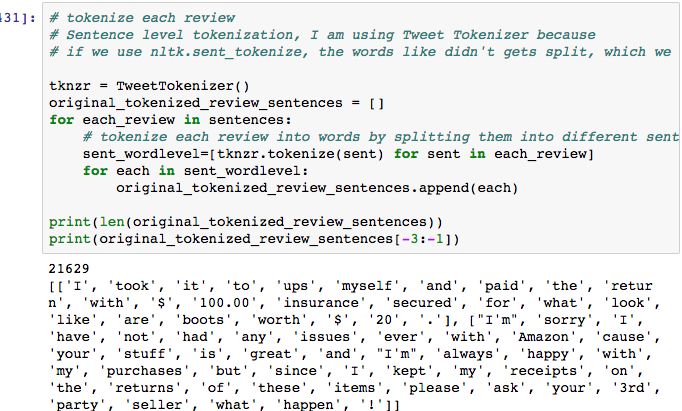






# Sentiment Analysis (80%)

Now, after getting the reviews, we had to split the reviews in the sentence level. I have used nltk.sent\_tokenizer() to get the sentences in each review. Also, during the preprocessing step, I have converted all the text in these sentences into the lower case to make the classification more accurate. Also, while word\_tokenizing the sentences, I have used tweetTokenizer. Doing this, I could preserve the n’t type words. It can be clearly seen in the following screen shot. If we use the nltk.sent\_tokenize, the words like didn’t gets identified as 2 different tokens which we don’t wish to happen for better classification accuracy.



After this step, my data is ready to be run against the classifier.

In this exercise, I have used the “bag of words” features where I collected all the words in the sentence\_polarity corpus and selected some most frequent words to be the word features. I have described the detailed processing steps in the later sections of this report.

Also, I have tried different types of experiments in order to complete my assignment.

Filter by stop words and other pre-processing methods

Using a sentiment lexicon with scores or counts: Subjectivity

Representing Negation

Using the modified word\_features inorder to give the preference to the adjectives! Since that is more important in sentiment classification. I have described different classifiers in the later sections.

I found out that the Subjectivity Classifier gave me the best accuracy amoung the rest of the classifiers, It came around **78.8%**

I have used NaiveBayes classifier to train and test the classifier on my feature sets. I have also tried to use the validate my results and found out the number of false positives, false negatives, true positives, true negatives in based on the result of my classifier.

# Detailed Technique Description

## Sentiment Analysis using Bag of words approach

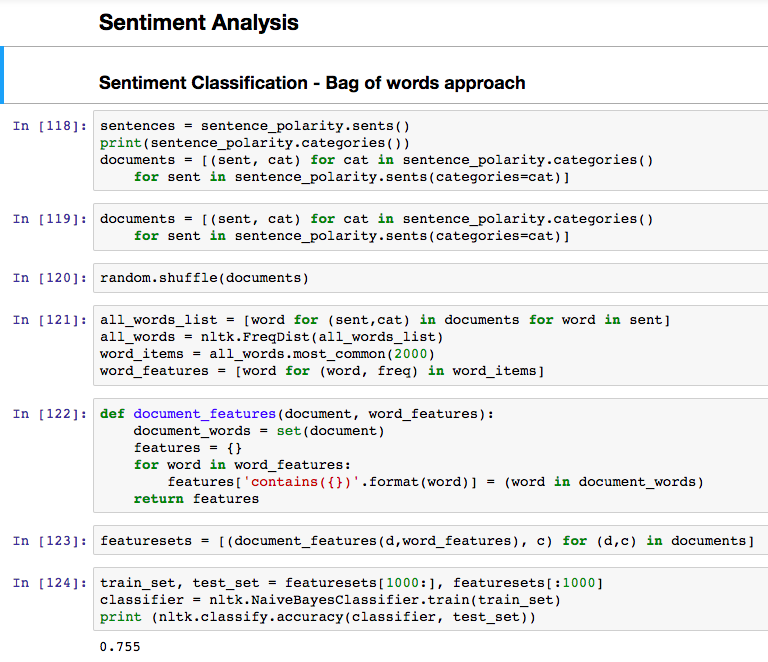
While writing this classifier, I have started by loading the sentence\_polarity corpus and created a list of documents where each document represents a single sentence with the words and its labels. First I created a list of documents where each document(sentence) is paired with it’s label. Each item is a pair(sent, cat) where sent is a list of words from the sentence\_polarity document and cat is its label, either ‘pos’ or ‘neg’.

Since the documents are in order by label, I have used the random.shuffle() method to suffle the documents, for later separation into training and test sets.

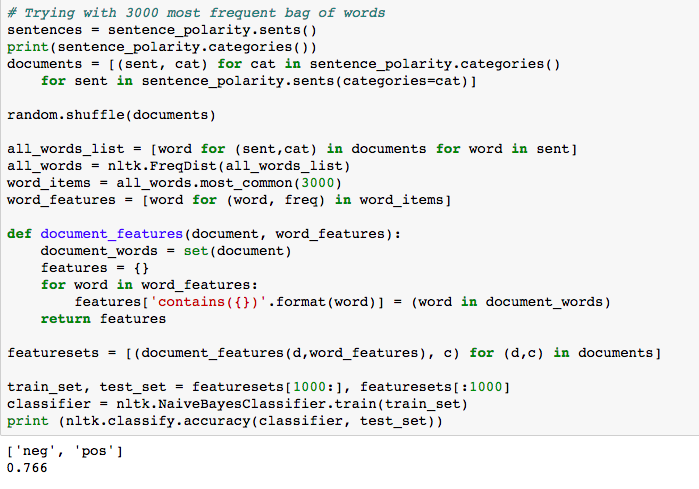
After this, I have defined the set of words that will be used for features. This is all the words in the entire document collection. I have used top 2000 most frequent words from this set of words.

After this step, I have defined the features for each document, using just the words, (Bag Of Words)/ Unigrams, and the value of the feature is a Boolean, according to whether the word is contained in that document.

Then I created a training and test sets, train a Naïve Bayes Classifier, and looked at the accuracy. And at this time, I have divided the documents to 90/10 split for test set and train set. This gave the the classifier accuracy of around 75.5%.



After this step, I have tried to experiment little more with different set of words while defining the word\_features. I have chosen to take 3000 most frequent words this time. And tried to find the accuracy again. This time the Classifier accuracy improved to 76.6%. This was better than previously achieved accuracy.

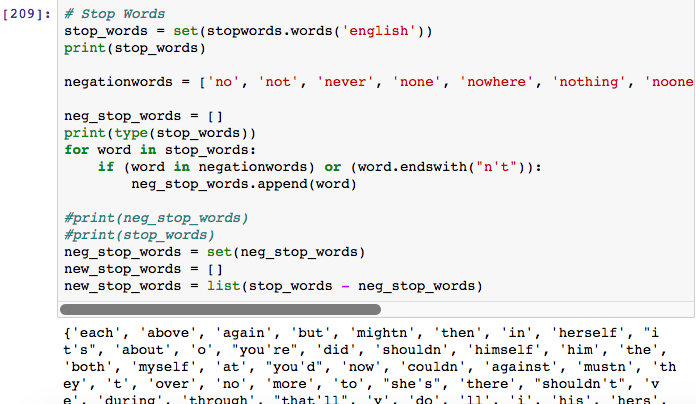


## Additional Changes in the word\_feature definition

I then tried to rerun the training and test sets and recalculated the document features by removing the stop words. This further improved my accuracy.

While removing the stop words, I simply didn’t remove all the stop words. Because, if we closely review the stop words list, then we notice that stop words also has the important negation conveying words, like not, neither, etc. Which should be included in out word\_features list.

For this, I have defined the separate list of negating\_words and removed those words from the list of the stop words. Also, doing this is not enough. There are also few stop words ending with n’t e.g.‘aren’t’. These words also are important wrt word\_features, and should not be removed from the word\_features. I have demonstrated this in the following code snippet. New\_stop\_words is the list I used while finding the word\_features, instead of using the original stop words list provided by nltk.



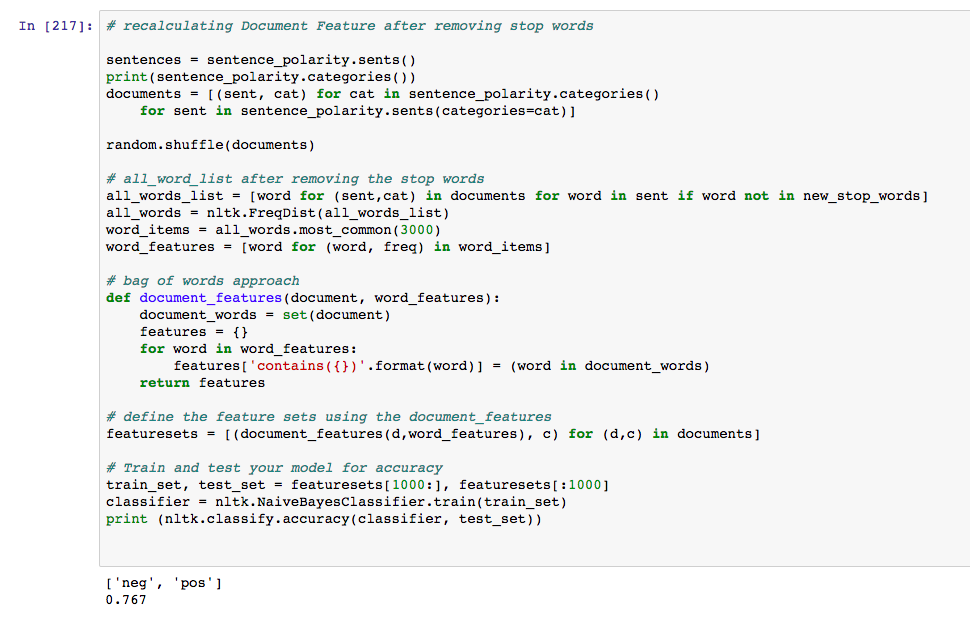
Also, in order to test that the remaining stop words in the new\_stop\_words list are not of prime importance wrt classification, I carried out further analysis of words like ‘very’, ‘too’, ‘again’.

I tried to check the pos\_scores, neg\_scores, and obj\_scores of each of these words. I noticed that for each of these words, the obj\_score is greater than pos\_score or neg\_score. Thus it is ok to remove these set of stop words while defining the word\_features.

Please find the screen shot attached.

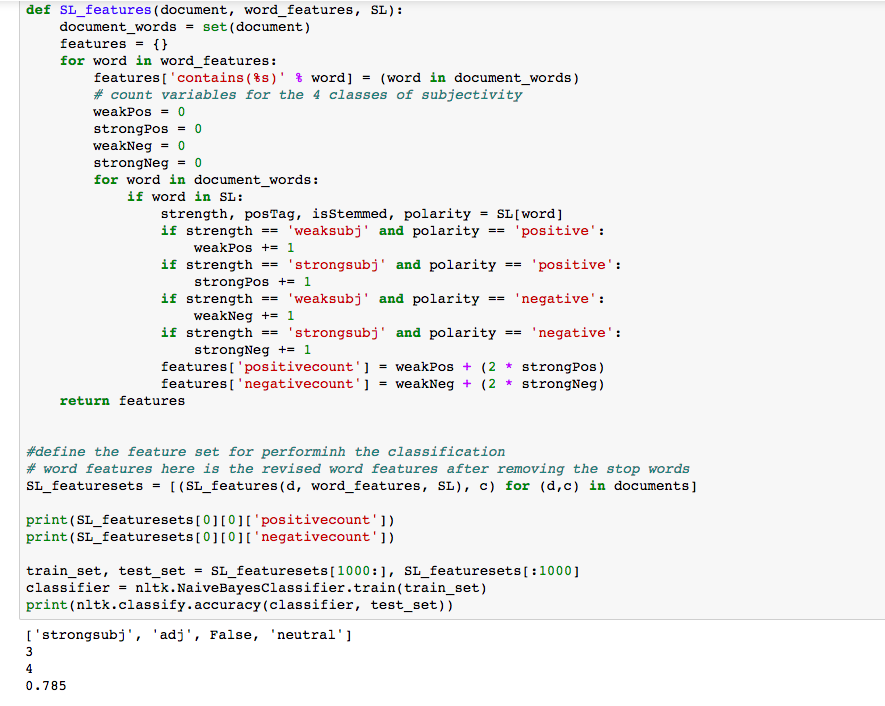


Now, after successfully identifying the stop words to be removed, I have modified the word\_features to be defined based on the removal of the words from the new\_stop\_words list. Doing this, I found out that the accuracy of the classifier was improved to 76.7%, which was better than the previous results. Please find the processing screen shot below.



## Subjective Count Features

In this approach, I first started with reading the subjectivity words from the subjectivity lexicon file. These words are often used as a feature themselves or in conjunction with some other information, I created two more features that involve counting the positive and negative subjectivity words present in the document. 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! Now it turned out to be **78.5%,** which was higher than the previously defined classifier accuracy. Please find the code in the following screen shot. Note that the word\_features used while using this classifier was one after removing the revised stop words list.



## Negation Features

The next classification technique I followed is 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. I have tried using this technique here. I went through the document words in order adding the word\_features, but if the word follows a negation words, then change the feature of to negated word.

Here is one list of negation words, including some adverbs called “approximate negators”:

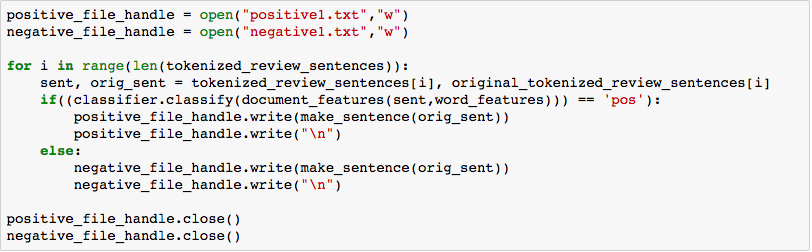
no, not, never, none, rather, hardly, scarcely, rarely, seldom, neither, nor, couldn't, wasn't, didn't, wouldn't, shouldn't, weren't, don't, doesn't, haven't, hasn't, won't, hadn't

I rerun the classifier by defining the word\_features considering the negation words. Accuracy on randomly split sentences this time gave the accuracy of 78.3% again. Which is better again.



Since this was one of the highest accuracy I gained with the different classification techniques I have followed, I decided to categorize the reviews in the positive reviews and negative reviews based on this classifier.

I have created 2 files positive.txt and negative.txt, and I have run this classifier on the set of Amazon reviews I collected.



## Classifier defined based by making the use of POS tagging and other word features

While defining the word features for this classifier, I used following techniques. First I removed the revised stop words (as discussed in the previous sections), then I reviewed the word\_features list and noticed that there are many words in this frequent words list, which are just the special characters. These characters make no sense in the most frequent word features. So I tried to remove those tuples from the list which have the special characters in them. Also, I tried to remove the words less than 4 character length to make the classifier more accurate.

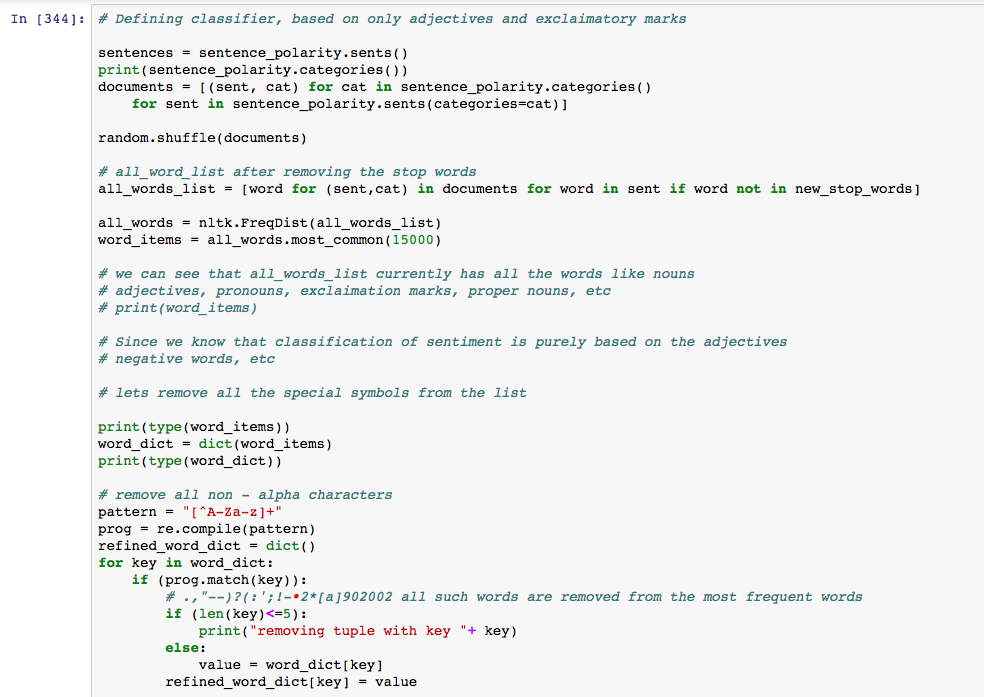
I then realized that the word features in order to perform sentiment analysis must be dominated by the adjectives. E.g. ‘Larry has a good dress.’ We will identify this sentence as a positive sentiment based on the word good. How did our brain identify this as a positive sentiment? It was by looking at the adjective ‘good’. Words like ‘good’, ‘pleasant’, play a very important role in sentiment analysis. We can also note that ‘Larry’ in this case is not really helping us to get the sentiment of this sentence.

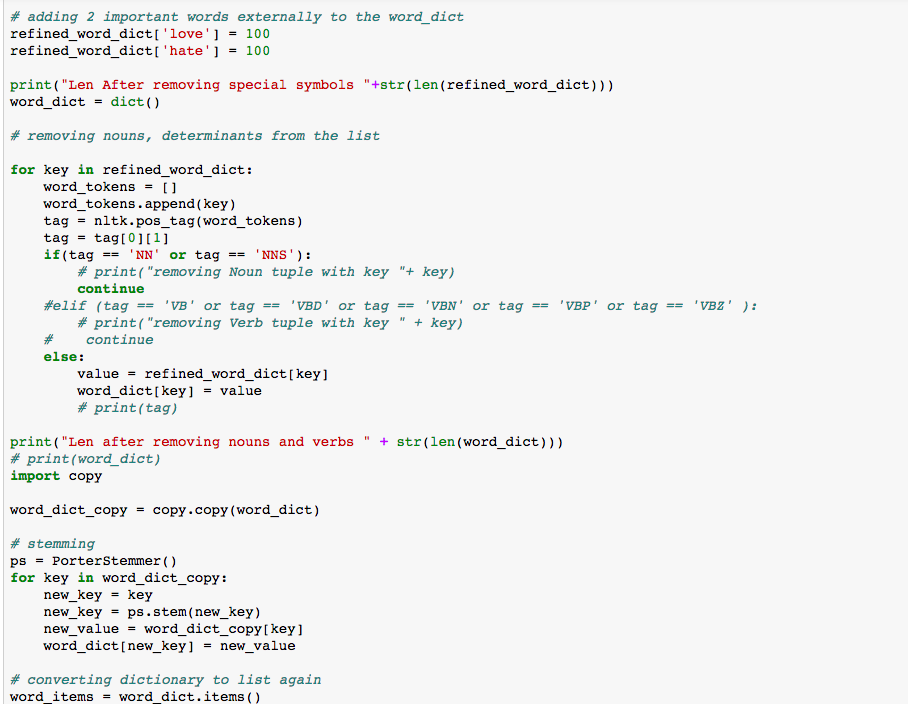
Considering both these above points, I decided to remove the nouns, noun phrases from the word\_features, so that we could focus on more adjectives than repeating nouns while classification. We can also try with verbs, adverbs, determinant removal, etc and check removal of which of these or not removal of these features yields better results.

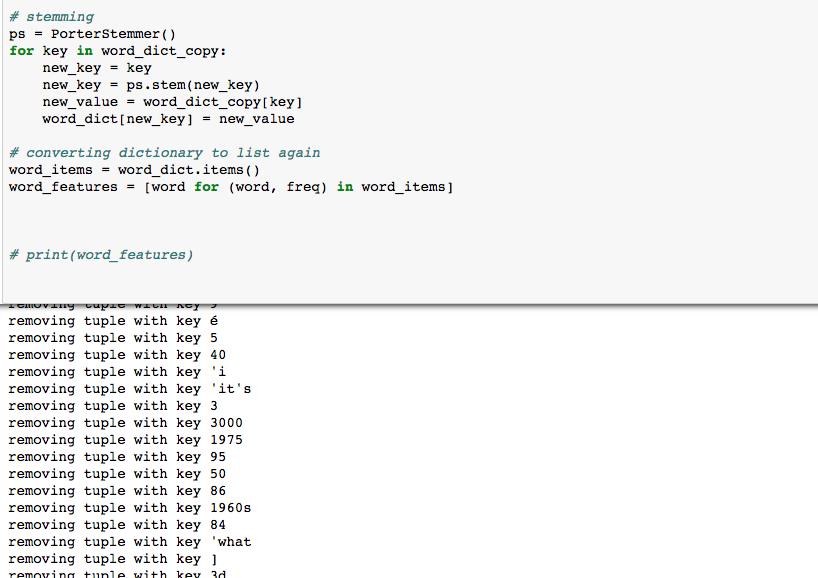
Also, I tried to add few more important words like ‘love’, ‘hate’ which play important role in semantic analysis to this word\_features list.

I also did stemming, because the words like working, worked, work should not be considered different while defining the classifier. Please find it in following screen shots.

Due to randomness of the split of the documents, I couldn’t get better accuracy. I could get 73% accuracy when I performed this type of word\_feature classification. This accuracy can be further improved by closely looking at the word\_features, and identifying the word\_features which reduces the classifier accuracy, or by adding new features to this list which are of more importance.



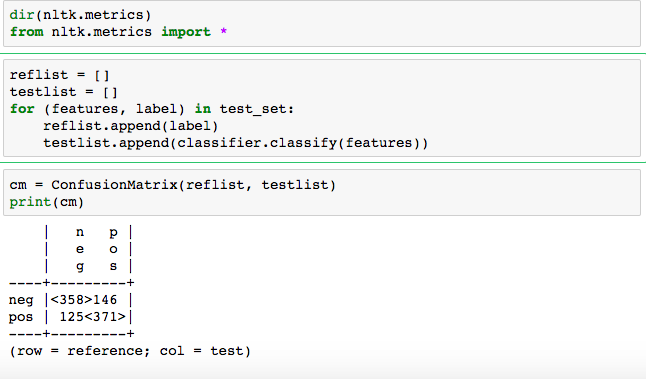




After trying different classifiers, I have tried to use the evaluation measure of the predictive capability of the model that was learned from the training data. We can learn more by looking at the predictions for each of the labels in our classifier.

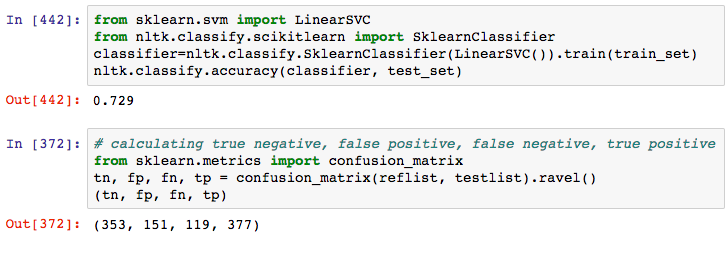
In nltk the confusion matrix is given by a function that takes two lists of labels for the test sets. First list called as a reference list which is all the correct/gold labels for the test set. Second list is the test list, which are the predicted labels in the test set.

Using this, I found out the number of true positives, true negatives, false positives and false positives, respectively.



# Bonus Credit (10%)

I have chosen an additional, more advanced type of the task from the list. I have tried to use the SciKit Learn Classifier with the features produced in nltk. By doing this, I observed that I could get 73% accuracy. I also tried to find out the true negative, false positive, false negative, false positive using the Confusion\_Matrix provided in the sklearn.metrics package. Please find the screen shot attached.



# Appendix

## Screen Shots of code and python processing screen shots

