# Problem set 1, Intro to NLP, 2017

### This is due on September 22nd at 11PM. Please see detailed submission instructions below. 100 points total.

#### *How to do this problem set:*

- What version of Python should I use? 2.7
- Most of these questions require writing Python code and computing results, and the rest of them have textual answers. To generate the answers, you will have to fill out a supporting file, `hw_1.py`.
- For all of the textual answers you have to fill out have placeholder text which says "Answer in one or two sentences here." For each question, you need to replace "Answer in one or two sentences here" with your answer.
- Write all the answers in this ipython notebook. Once you are finished (1) Generate a PDF via (File -> Download As -> PDF) and upload to Gradescope (2)Turn in `hw_1.py` and `hw_1.ipynb` on Moodle.
- **Important:** Check your PDF before you turn it in to gradescope to make sure it exported correctly. If ipyhton notebook gets confused about your syntax it will sometimes terminate the PDF creation routine early. You are responsible for checking for these errors. If your whole PDF does not print, try running `$jupyter nbconvert --to pdf hw_1.ipynb` to identify and fix any syntax errors that might be causing problems.
- **Important:** When creating your final version of the PDF to hand in, please do a fresh restart and execute every cell in order. Then you'll be sure it's actually right. One convenient way to do this is by clicking `Cell -> Run All` in the notebook menu.
- This assignment is designed so that you can run all cells in a few minutes of computation time. If it is taking longer than that, you probably have made a mistake in your code.

#### *Academic honesty*

- We will audit the Moodle code from a few dozen students, chosen at random. The audits will check that the code you wrote and turned on Moodle generates the answers you turn in on your Gradescope PDF. If you turn in correct answers on your PDF without code that actually generates those answers, we will consider this a potential case of cheating. See the course page for honesty policies.
- We will also run automatic checks of code on Moodle for plagiarism. Copying code from others is considered a serious case of cheating.

In [1]:
```
# Run this cell! It sets some things up for you.

# This code makes plots appear inline in this document rather than in a new wi
ndow.
import matplotlib.pyplot as plt
from __future__ import division  # this line is important to avoid unexpected
 behavior from division

# This code imports your work from hw_1.py
from hw_1 import *

%matplotlib inline
plt.rcParams['figure.figsize'] = (5, 4) # set default size of plots

# Some more magic so that the notebook will reload external python modules;
# see http://stackoverflow.com/questions/1907993/autoreload-of-modules-in-ipyt
hon
%load_ext autoreload
%autoreload 2
```

In [2]:
```
# download the IMDB large movie review corpus from the class webpage to a file
 location on your computer

PATH_TO_DATA = 'large_movie_review_dataset'  # set this variable to point to t
he location of the IMDB corpus on your computer
POS_LABEL = 'pos'
NEG_LABEL = 'neg'
TRAIN_DIR = os.path.join(PATH_TO_DATA, "train")
TEST_DIR = os.path.join(PATH_TO_DATA, "test")

for label in [POS_LABEL, NEG_LABEL]:
    if len(os.listdir(TRAIN_DIR + "/" + label)) == 12500:
        print "Great! You have 12500 {} reviews in {}".format(label, TRAIN_DIR
 + "/" + label)
    else:
        print "Oh no! Something is wrong. Check your code which loads the revi
ews"
```

```
Great! You have 12500 pos reviews in large_movie_review_dataset\train/pos
Great! You have 12500 neg reviews in large_movie_review_dataset\train/neg
```

In [3]:
```
# Actually reading the data you are working with is an important part of NLP!
 Let's look at one of these reviews

print open(TRAIN_DIR + "/neg/3740_2.txt").read()
```

```
Right away, this film was ridiculous. Not that it didn't have redeeming aspec
ts For example, the best thing about this film was the beautiful background
 scenery. Anyone not living on the East Coast should know the South doesn't h
ave beautiful mountains like those found in the West. I knew it was Utah righ
t off the bat, but perhaps Dalton couldn't suppress his English accent, so th
ey had to excuse it by saying this was a southern town. Subverting his accent
into a Southern one was easier. Sure the film has plot twists, but its phony
 sense of place was something I couldn't get past. It's not like Utah doesn't
have meth labs... so why the writers thought it necessary to pretend it was i
n the South is beyond me. <br /><br />One other thing in action pictures alwa
ys puzzles me. Why do they always make the "cocking" sound effect when the ch
aracter pulls out an automatic handgun? It seemed every other sound effect in
this movie was a "chuk-chich" signifying a 9mm was loaded and ready to fire.
 Of course, the weapons already had rounds chambered so this was unnecessary.
<br /><br />Lastly, the pyrotechnics were WAY over the top. But hey, this fil
m was targeted to a certain 'market segment' I suppose... It's too bad. Each
 of the actors can act, but this film was lame.
```

# Part One: Intro to NLP in Python: types, tokens and Zipf's law

#### Types and tokens

One major part of any NLP project is word tokenization. Word tokenization is the task of segmenting text into individual words, called tokens. In this assignment, we will use simple whitespace tokenization. You will have a chance to improve this for extra credit at the end of the assigment. Take a look at the `tokenize_doc` function in `hw_1.py`. **You should not modify tokenize_doc** but make sure you understand what it is doing.

In [4]:
```
# We have provided a tokenize_doc function in hw_1.py. Here is a short demo of
 how it works

d1 = "This SAMPLE doc has   words tHat  repeat repeat"
bow = tokenize_doc(d1)

assert bow['this'] == 1
assert bow['sample'] == 1
assert bow['doc'] == 1
assert bow['has'] == 1
assert bow['words'] == 1
assert bow['that'] == 1
assert bow['repeat'] == 2

bow2 = tokenize_doc("Computer science is both practical and abstract.")
for b in bow2:
    print b
```

```
and
both
computer
abstract.
science
is
practical
```

**Question 1.1 (5 points)**

Now we are going to count the word types and word tokens in the corpus. In the cell below, use the `word_counts` dictionary variable to store the count of each word in the corpus. Use the `tokenize_doc` function to break documents into tokens.

`word_counts` keeps track of how many times a word type appears across the corpus. For instance, `word_counts["dog"]` should store the number 990 -- the count of how many times the word dog appears in the corpus.

In [5]:
```
import glob
word_counts = defaultdict(float)  # you might want to use a defaultdict instea
d https://docs.python.org/2/library/collections.html
                  # defaultdicts are often useful for NLP in python

for label in [POS_LABEL, NEG_LABEL]:
    for directory in [TRAIN_DIR, TEST_DIR]:
        for fn in glob.glob(directory + "/" + label + "/*txt"):
            ## Implement me!
            temp = tokenize_doc(open(fn).read())
            for key, item in temp.items():
                word_counts[key] += item
```

In [6]:
```
# you should see 990 instances of the word type "dog" in the corpus. (updated
 9/13)
if word_counts["dog"] == 990:
    print "yay! there are {} total instances of the word type dog in the corpu
s".format(word_counts["dog"])
else:
    print "hrm. Something seems off. Double check your code"
```

```
yay! there are 990.0 total instances of the word type dog in the corpus
```

### Question 1.2 (5 points)

Fill out the functions `n_word_types` and `n_word_tokens` in `hw_1.py`. These functions return the total number of word types and tokens in the corpus. **important** The autoreload "magic" that you setup early in the assignment should automatically reload functions as you make changes and save. If you run into trouble you can always restart the notebook and clear any .pyc files.

In [7]:
```
print "there are {} word types in the
corpus".format(n_word_types(word_counts))
print "there are {} word tokens in the corpus".format(n_word_tokens(word_count
s))
```

```
there are 391997 word types in the corpus
there are 11557403.0 word tokens in the corpus
```

### Question 1.3 (5 points)

You should see a much higher number of tokens than types. Why is that?

Answer: Because types are unique words in the corpus. But, tokens are sum of number of times each word type appeared in the corpus.

**Zipf's Law**

**Question 1.4 (5 points)**

In this section, you will verify a key statistical properties of text: Zipf's Law (https://en.wikipedia.org/wiki/Zipf%27s_law).

Zipf's Law describes the relations between the frequency rank of words and frequency value of words. For a word $w$, its frequency is inversely proportional to its rank:

$$count_w = K \frac{1}{rank_w}$$

or in other words

$$\log(count_w) = K - \log(rank_w)$$

for some constant $K$, specific to the corpus and how words are being defined.

Therefore, if Zipf's Law holds, after sorting the words descending on frequency, word frequency decreases in an approximately linear fashion under a log-log scale.

Please make such a log-log plot by ploting the rank versus frequency. Use a scatter plot where the x-axis is the *log(rank)*, and y-axis is *log(frequency)*. You should get this information from `word_counts`; for example, you can take the individual word counts and sort them. dict methods `.items()` and/or `values()` may be useful. (Note that it doesn't really matter whether ranks start at 1 or 0 in terms of how the plot comes out.) You can check your results by comparing your plots to ones on Wikipedia; they should look qualitatively similar.

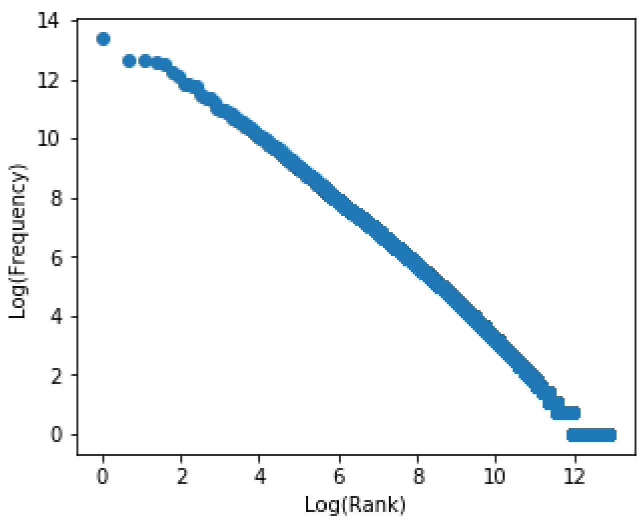
*Please remember to label the meaning of the x-axis and y-axis.*

In [8]:
```
import math
y = []
for w in sorted(word_counts, key= word_counts.get, reverse=True):
    y.append(word_counts[w])
x = [math.log(X) for X in range(1, len(word_counts) + 1)]
y = [math.log(Y) for Y in y]
X_LABEL = "Log(Rank)"
Y_LABEL = "Log(Frequency)"

# implement me! you should fill the x and y arrays. Add your code here

plt.scatter(x, y)
plt.xlabel(X_LABEL)
plt.ylabel(Y_LABEL)
```

Out[8]: <matplotlib.text.Text at 0x79fe198>




**Question 1.5 (5 points)**

You should see some discountinuities on the left and right sides of this figure. Why are we seeing them on the left? Why are we seeing them on the right? On the right, what are those "ledges"?

Answer: On the left side we are seeing some discountinuities because there are very common words like "the" which appears in the corpus quite a lot of times. On the right side, we are seeing discountinuities because of words like "'thrice-cursed', 'shakespearean-child'" which only once in the larger corpus like one we are working on.

# Part Two: Naive Bayes

This section of the homework will walk you through coding a Naive Bayes classifier that can distinguish between postive and negative reviews (at some level of accuracy).

**Question 2.1 (5 pts)** To start, implement the `update_model` function in `hw_1.py`. Make sure to read the function comments so you know what to update. Also review the NaiveBayes class variables in the `def __init__` method of the NaiveBayes class to get a sense of which statistics are important to keep track of. Once you have implemented `update_model`, run the train model function using the code below. You'll need to provide the path to the dataset you downloaded to run the code.

In [9]:
```
nb = NaiveBayes(PATH_TO_DATA, tokenizer=tokenize_doc)
nb.train_model()

if len(nb.vocab) == 252165:
    print "Great! The vocabulary size is {}".format(252165)
else:
    print "Oh no! Something seems off. Double check your code before continuin
g. Maybe a mistake in update_model?"
```

```
REPORTING CORPUS STATISTICS
NUMBER OF DOCUMENTS IN POSITIVE CLASS: 1875650.0
NUMBER OF DOCUMENTS IN NEGATIVE CLASS: 1861010.0
NUMBER OF TOKENS IN POSITIVE CLASS: 2958730.0
NUMBER OF TOKENS IN NEGATIVE CLASS: 2885734.0
VOCABULARY SIZE: NUMBER OF UNIQUE WORDTYPES IN TRAINING CORPUS: 252165
Great! The vocabulary size is 252165
```

### Exploratory analysis

Let's begin to explore the count statistics stored by the update model function. Use the provided `top_n` function to find the top 10 most common words in the positive class and top 10 most common words in the negative class. You don't have to code anything to do this.

In [10]:
```
print "TOP 10 WORDS FOR CLASS " + POS_LABEL + ":"
for tok, count in nb.top_n(POS_LABEL, 10):
    print '', tok, count
print ''

print "TOP 10 WORDS FOR CLASS " + NEG_LABEL + ":"
for tok, count in nb.top_n(NEG_LABEL, 10):
    print '', tok, count
print ''
```

```
TOP 10 WORDS FOR CLASS pos:
 the 165803.0
 and 87022.0
 a 82054.0
 of 76155.0
 to 65869.0
 is 55785.0
 in 48420.0
 i 33143.0
 it 32795.0
 that 32702.0

TOP 10 WORDS FOR CLASS neg:
 the 156385.0
 a 77895.0
 and 71534.0
 of 68304.0
 to 68097.0
 is 48385.0
 in 42103.0
 i 37335.0
 this 37301.0
 that 33585.0
```

### Question 2.2 (5 points)

Will the top 10 words of the positive/negative classes help discriminate between the two classes? Do you imagine that processing other English text will result in a similar phenomenon?

Answer: The positive and negative classes cannot be discriminated using the top ten words. Because top 10 words are same in both the classes. Yes other english text will result in the similar phenomenon because words like "the" , 'a' are really common in the language.

### Question 2.3 (5 pts)

The Naive Bayes model assumes that all features are conditionally independent given the class label. For our purposes, this means that the probability of seeing a particular word in a document with class label $y$ is independent of the rest of the words in that document. Implement the `p_word_given_label` function. This function calculates P (w|y) (i.e., the probability of seeing word w in a document given the label of that document is y).

Use your `p_word_given_label` function to compute the probability of seeing the word “fantastic” given each sentiment label. Repeat the computation for the word “boring.”

In [11]:
```
print "P('fantastic'|pos):",  nb.p_word_given_label("fantastic", POS_LABEL)
print "P('fantastic'|neg):",  nb.p_word_given_label("fantastic", NEG_LABEL)
print "P('boring'|pos):",  nb.p_word_given_label("boring", POS_LABEL)
print "P('boring'|neg):",  nb.p_word_given_label("boring", NEG_LABEL)
```

```
P('fantastic'|pos): 0.000154458162793
P('fantastic'|neg): 3.77720191813e-05
P('boring'|pos): 6.18508616873e-05
P('boring'|neg): 0.000287275265149
```

Which word has a higher probability given the positive class, fantastic or boring? Which word has a higher probability given the negative class? Is this what you would expect?

Answer:- Fantastic have higher probability in the postive label class, the word boring have higher probability in the negative class, and yes that is expected word.

**Question 2.4 (5 pts)**

In the next cell, compute the probability of the word "car-thievery" in the positive training data and negative training data.

In [12]:
```
print "P('car-thievery'|pos):",  nb.p_word_given_label("car-thievery", POS_LAB
EL)
print "P('car-thievery'|neg):",  nb.p_word_given_label("car-thievery", NEG_LAB
EL)
```

```
P('car-thievery'|pos): 3.3798285075e-07
P('car-thievery'|neg): 0.0
```

What do you notice about "P('car-thievery'|neg)"? Why do you see this number? What would happen if we took the log of "P('car-thievery'|neg)"? What would happen if we multiplied "P('car-thievery'|neg)" by "P('cliche'|neg)"? Why might these operations cause problems for a Naive Bayes classifier?

Answer:- There is no word like "car-thievery" in the negative class thats why we are seeing 0 probability of P('car-thievery'|neg). The log of 0 is undefined or infinity, if we multiple the P('car-thievery'|neg) = 0 to P('cliche'|neg) whis make the whole term zero.

**Question 2.5 (5 pts)**

We can address the issues from question 2.4 with psuedocounts. A psuedocount is a fixed amount added to the count of each word stored in our model. Psuedocounts are used to help smooth calculations involving words for which there is little data. Implement `p_word_given_label_and_psuedocount` and then run the next cell. Hint: look at the slides from the lecture on pseudocounts.

In [13]:
```
print "P('car-thievery'|neg):",  nb.p_word_given_label_and_pseudocount("car-th
ievery", NEG_LABEL, 1)
```

```
P('car-thievery'|neg): 3.18684572066e-07
```

**Question 2.6 (getting ready for question 2.10)**

*Prior and Likelihood*

As noted before, the Naive Bayes model assumes that all words in a document are independent of one another given the document's label. Because of this we can write the likelihood of a document as:

$$P(w_{d1}, \cdots, w_{dn}|y_d) = \prod_{i=1}^{n} P(w_{di}|y_d)$$

However, if a document has a lot of words, the likelihood will become extremely small and we'll encounter numerical underflow. Underflow is a common problem when dealing with prob- abilistic models; if you are unfamiliar with it, you can get a brief overview on Wikipedia (https:/en.wikipedia.org/wiki/Arithmetic_underflow). To deal with underflow, a common transformation is to work in log-space.

$$\log[P(w_{d1}, \cdots, w_{dn}|y_d)] = \sum_{i=1}^{n} \log[P(w_{di}|y_d)]$$

Implement the `log_likelihood` function (Hint: it should make calls to the p word given label and psuedocount function). Implement the `log_prior` function. This function takes a class label and returns the log of the fraction of the training documents that are of that label.

There is nothing to print out for this question. But you will use these functions in a moment...

In [14]:
```
nb.log_prior(NEG_LABEL)
```

Out[14]:
```
-0.6970728131922387
```

**Question 2.7 (5 pts)**

Naive Bayes is a model that tells us how to compute the posterior probability of a document being of some label (i.e., $P(y_d|\mathbf{w_d})$). Specifically, we do so using bayes rule:

$$P(y_d|\mathbf{w_d}) = \frac{P(y_d)P(\mathbf{w_d}|y_d)}{P(\mathbf{w_d})}$$

In the previous section you implemented functions to compute both the log prior ($\log[P(y_d)]$) and the log likelihood ($\log[P(\mathbf{w_d}|y_d)]$ ). Now, all your missing is the *normalizer*, $P(\mathbf{w_d})$.

Derive the normalizer by expanding $P(\mathbf{w_d})$. You will have to use "MathJax" to write out the equations. MathJax is very similar to LaTeX. 99% of the MathJax you will need to write for this course (and others at U Mass) is included in the first answer of [this (https://math.meta.stackexchange.com/questions/5020/mathjax-basic-tutorial-and-quick-reference)](https://math.meta.stackexchange.com/questions/5020/mathjax-basic-tutorial-and-quick-reference) tutorial. MathJax and LaTeX can be annoying first, but once you get a little practice, using these tools will feel like second nature.

Derive the normalizer by expanding $P(\mathbf{w_d})$. Fill out the answer with MathJax here

$P(\mathbf{w_d})$ = $\sum P(\mathbf{w_d}, \mathbf{y_c})$ (Where C represents classes)

Applying bayes rule in $P(\mathbf{w_d}, \mathbf{y_c})$

$P(\mathbf{w_d}, \mathbf{y_c})$ = $\sum P(\mathbf{w_d}/\mathbf{y_c}) * P(\mathbf{y_c})$

So,

$P(\mathbf{w_d})$ = $\sum P(\mathbf{w_d}/\mathbf{y_c}) * P(\mathbf{y_c})$

**Question 2.8 (5 pts)**

One way to classify a document is to compute the unnormalized log posterior for both labels and take the argmax (i.e., the label that yields the higher unnormalized log posterior). The unnormalized log posterior is the sum of the log prior and the log likelihood of the document. Why don't we need to compute the log normalizer here?

Answer:-

$C_{map}$= $\mathrm{argmax}_{\mathrm{c}}(-log(P(w_d) + log(P(c))$ + $\sum_{i=1}^{n} log(P(w_{di}/c))$

$log(P(w_d)$ This is Constant

$C_{map}$ = $\mathrm{argmax}_{\mathrm{c}}(log(P(c))$ + $\sum_{i=1}^{n} log(P(w_{di}/c))$

**Question 2.9 (15 pts)**

Implement the `unnormalized_log_posterior` function and the `classify` function. The `classify` function should use the unnormalized log posteriors but should not compute the normalizer. Once you implement the `classify` function, we'd like to evaluate its accuracy. `evaluate_classifier_accuracy` is implemented for you so you don't need to change that method.

In [15]:
```
print nb.evaluate_classifier_accuracy(1.0)
```

```
82.956
```

**Question 2.10 (5 pts)**

Try evaluating your model again with a pseudocount parameter of 500.

In [16]:
```
print nb.evaluate_classifier_accuracy(500.0)
```

```
79.868
```

Does the accuracy go up or down when the pseudo count parameter is raised to 500? Why do you think this is?

Answer:- The Accuracy went down when increasing the hyperparameter alpha to 500.0, The reason for probability going down is, we are shifting or transfering more weight to the words which appeared quite less in the corpus from the word which appeared more in the corpus.

**Question 2.11 (5 pts)**

Our trained model can be queried to do exploratory data analysis. We saw that the top 10 most common words for each class were not very discriminative. Often times, a more descriminative statistic is a word's likelihood ratio. A word's likelihood ratio is defined as

$$LR(w) = \frac{P(w|y{=}\text{pos})}{P(w|y{=}\text{neg})}$$

A word with $LR = 5$ is five times more likely to appear in a positive review than it is in a negative review; a word with $LR = 0.33$ is one third as likely to appear in a positive review than a negative review

In [17]:
```
# Implement the nb.likelihod_ratio function and use it to investigate the like
lihood ratio of "fantastic" and "boring"
print "LIKEHOOD RATIO OF 'fantastic':", nb.likelihood_ratio('fantastic', 1.0)
print "LIKEHOOD RATIO OF 'boring':", nb.likelihood_ratio('boring', 1.0)
print "LIKEHOOD RATIO OF 'the':", nb.likelihood_ratio('the', 1.0)
print "LIKEHOOD RATIO OF 'to':", nb.likelihood_ratio('to', 1.0)
```

```
LIKEHOOD RATIO OF 'fantastic': 4.06898088596
LIKEHOOD RATIO OF 'boring': 0.216646954101
LIKEHOOD RATIO OF 'the': 1.03611983814
LIKEHOOD RATIO OF 'to': 0.94529239345
```

Does it make sense that $LR$('fantastic') $> LR$('to')?

Answer:- Yes, The $LR$(Fantastic) $> LR$(to) make sense, because the word fantastic appears more in the positive class then negative class that means the denominator P(fantastic|neg) is less and so $LR$('fantastic') is high, but in the case of $LR$('to') both the numerator and denominator have equal probability and so $LR$("to") have more chance of being close to 1 as seen above.

**Question 2.12 (15 pts)**

Find a review that your classifier got wrong.

In [18]:
```
alpha = 1.0
pos_path = os.path.join(TEST_DIR, POS_LABEL)
for (p, label) in [ (pos_path, POS_LABEL)]:
    for f in os.listdir(p):
        with open(os.path.join(p,f),'r') as doc:
                content = doc.read()
                bow = tokenize_doc(content)
                if nb.classify(bow, alpha) != label:
                    print "File name from the positive label:- ",
os.path.join(p,f), "\n"
                    print "Classified Label:- ", nb.classify(bow, alpha), "\n"

                    print content
                    break
```

```
File name from the positive label:-  large_movie_review_dataset\test\pos\1000
9_10.txt

Classified Label:-  neg

I loved this movie from beginning to end.I am a musician and i let drugs get
 in the way of my some of the things i used to love(skateboarding,drawing) bu
t my friends were always there for me.Music was like my rehab,life support,an
d my drug.It changed my life.I can totally relate to this movie and i wish th
ere was more i could say.This movie left me speechless to be honest.I just sa
w it on the Ifc channel.I usually hate having satellite but this was a perk o
f having satellite.The ifc channel shows some really great movies and without
it I never would have found this movie.Im not a big fan of the international
 films because i find that a lot of the don't do a very good job on translati
ng lines.I mean the obvious language barrier leaves you to just believe thats
what they are saying but its not that big of a deal i guess.I almost never go
t to see this AMAZING movie.Good thing i stayed up for it instead of going to
bed..well earlier than usual.lol.I hope you all enjoy the hell of this movie
 and Love this movie just as much as i did.I wish i could type this all in ca
ps but its again the rules i guess thats shouting but it would really show my
excitement for the film.I Give It Three Thumbs Way Up!<br /><br />This Movie
 Blew ME AWAY!
```

What are two reasons your system might have misclassified this example? What improvements could you make that may help your system classify this example correctly?

Fill out reason nubmer 1 here | Fill out a possible improvement here 1st Reason: After reading this review its clear this review is a positive review, but the classifier classified this review as negative because this contains lots of positive words (love, liked.. etc) but also contain lots of negative words like (don't, not, hate.. etc) thats because its hard for bag of words apporach.

Fill out reason nubmer 2 here | Fill out a possible improvement here

2nd Reason:- N - grams apporach like bigram or trigrams can help improve this model, data cleaning and removing stop words can also help.

**Extra credit (up to 10 points)**

If you don't want to do the extra credit, you can stop here! Otherwise... keep reading...

In this assignment, we use whitespace tokenization to create a bag-of-unigrams representation for the movie reviews. It is possible to improve this represetation to improve your classifier's performance. Use your own code or an external library such as nltk to perform tokenization, text normalization, word filtering, etc. Fill out your work in `def tokenize_doc_and_more` (below) and then show improvement by running the following.

```
nb = NaiveBayes(PATH_TO_DATA, tokenizer=tokenize_doc_and_more)
nb.train_model()
nb.evaluate_classifier_accuracy(1.0)
```

Roughly speaking, the larger performance improvement, the more extra credit. However, doing a good job investigating, explaining and justifying your work with small experiments and comments is also extremely important. Make sure to describe what you did and analyze why your method works. Use this ipython notebook to show your work.

In [19]:
```
from nltk.corpus import stopwords
import re
from collections import Counter
def tokenize_doc_and_more(doc):
    bow = defaultdict(float)

    # Converting into lower case text
    doc_lower = doc.lower()

    # removing pucntuations ("..", ".", ",")
    doc_wo_punc = re.sub(r'(\.+$|\?+$|\,|\'|\.{2,}|<br *(/>)?)',"",doc_lower)

    # removing two spaces
    text_with_one_space = re.sub(r'[ ]{2,}'," ", doc_wo_punc).split()

    # removing stop words like "and", "the"
    stop_words = set(stopwords.words('english'))
    filtered_sentence = [w for w in text_with_one_space if not w in
stop_words]

    bow = Counter(filtered_sentence)
    return dict(bow)
```

In [20]:
```
# the better tokenizer test example
tokenize_doc_and_more("I loved this movie from beginning to end.I am a musicia
n and i let drugs get in the way of my some of the things i used to love(skate
boarding,drawing) ")
```

Out[20]:
```
{'beginning': 1,
 'drugs': 1,
 'end.i': 1,
 'get': 1,
 'let': 1,
 'love(skateboardingdrawing)': 1,
 'loved': 1,
 'movie': 1,
 'musician': 1,
 'things': 1,
 'used': 1,
 'way': 1}
```

In [21]:
```
# Previous tokinizer test example
tokenize_doc("I loved this movie from beginning to end.I am a musician and i l
et drugs get in the way of my some of the things i used to love(skateboarding,
drawing) ")
```

Out[21]:
```
{'a': 1.0,
 'am': 1.0,
 'and': 1.0,
 'beginning': 1.0,
 'drugs': 1.0,
 'end.i': 1.0,
 'from': 1.0,
 'get': 1.0,
 'i': 3.0,
 'in': 1.0,
 'let': 1.0,
 'love(skateboarding,drawing)': 1.0,
 'loved': 1.0,
 'movie': 1.0,
 'musician': 1.0,
 'my': 1.0,
 'of': 2.0,
 'some': 1.0,
 'the': 2.0,
 'things': 1.0,
 'this': 1.0,
 'to': 2.0,
 'used': 1.0,
 'way': 1.0}
```

In [22]:
```
nb = NaiveBayes(PATH_TO_DATA, tokenizer=tokenize_doc_and_more)
nb.train_model()
nb.evaluate_classifier_accuracy(1.0)
```

```
REPORTING CORPUS STATISTICS
NUMBER OF DOCUMENTS IN POSITIVE CLASS: 1343920.0
NUMBER OF DOCUMENTS IN NEGATIVE CLASS: 1313726.0
NUMBER OF TOKENS IN POSITIVE CLASS: 1578999.0
NUMBER OF TOKENS IN NEGATIVE CLASS: 1531854.0
VOCABULARY SIZE: NUMBER OF UNIQUE WORDTYPES IN TRAINING CORPUS: 224314
```

Out[22]: 83.2

In [23]:
```
#Better accuracy with the alpha   = 16
nb.evaluate_classifier_accuracy(16.0)
```

Out[23]: 84.8

Use cells at the bottom of this notebook to explain what you did in `better_tokenize_doc`. Include any experiments or explanations that you used to decide what goes in your function.

In [ ]:
```
# Your experiments and explanations go here
"""Tokinizer_doc_and_more is sanatizing text more, if we see the example of to
kinized text from the old tokenizer and new tokinizer
its clear that new tokinizer is having all the important words from the corpus
 and removing stop words, also the new tokinizer
is removing puntuations and two spaces.

The Accuracy is going up from 82.8 to 84.8 for alpha == 16.0

"""
```

In [24]:
```
bow = "I loved this movie from beginning to end.I am a musician and i let drug
s get in the way of my some of the things i used to love(skateboarding,drawin
g) but my friends were always there for me.Music was like my rehab,life suppor
t,and my drug.It changed my life.I can totally relate to this movie and i wish
 there was more i could say.This movie left me speechless to be honest.I just
 saw it on the Ifc channel.I usually hate having satellite but this was a perk
 of having satellite.The ifc channel shows some really great movies and withou
t it I never would have found this movie.Im not a big fan of the international
 films because i find that a lot of the don't do a very good job on translatin
g lines.I mean the obvious language barrier leaves you to just believe thats w
hat they are saying but its not that big of a deal i guess.I almost never got
 to see this AMAZING movie.Good thing i stayed up for it instead of going to b
ed..well earlier than usual.lol.I hope you all enjoy the hell of this movie an
d Love this movie just as much as i did.I wish i could type this all in caps b
ut its again the rules i guess thats shouting but it would really show my exci
tement for the film.I Give It Three Thumbs Way Up!<br /><br />This Movie Blew
 ME AWAY!"
```

In [25]:
```
bow = tokenize_doc(bow)
print(nb.classify(bow, 1))
```

neg

In [26]:
```
#The accuracy did not increase significantly, there is still lot of room for i
mprovements.
```

In [ ]: