Problem set 1, Applications of Natural Language Processing, Fall 2021

Collab with: Zeng Chang

This is due on September 24th at 11:59pm. Please see detailed submission instructions below. 110 points total.

How to do this problem set:

- Use Python version 3. We strongly suggest installing Python from the Anaconda Individual Edition (https://docs.anaconda.com/anaconda/) software package.
- Download `large_movie_review_dataset.zip` (https://people.cs.umass.edu/~brenocon/inlp2017/hw1/large_movie_review_dataset.zi and `lotr_script.txt` (https://people.cs.umass.edu/~brenocon/cs490a_f21/hw1/lotr_script.txt). We will use these two datasets in this homework.
- Most of these questions require writing Python code or Unix commands and computing results, while the remainder have textual answers. To complete this assignment, you will need to fill out the supporting files, `hw1.py` (https://people.cs.umass.edu/~brenocon/cs490a_f21/hw1/hw1.py) and `hw1.sh` (https://people.cs.umass.edu/~brenocon/cs490a_f21/hw1/hw1.sh).
- For all of the textual answers, replace the placeholder text ("Answer in one or two sentences here.") with your answer.
- 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 a mistake in your code.

How to submit this problem set:

- Write all the answers in this ipython notebook. Once you are finished, (1) Generate a PDF via (File -> Download As -> PDF) (2) Upload your pdf file to Gradescope. (3) Compress `hw1.py` , `hw1.sh` , and `hw1.ipynb` into one zip file and upload to Gradescope.
- **Important:** Check your PDF before you turn it in to gradescope to make sure it exported correctly. If ipython 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 on the commandline `jupyter nbconvert --to pdf hw1.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.

- If you are having trouble with PDF export, you can always paste screenshots into a word processor then turn that into PDF.

Academic honesty:

- Like always, check the course website's collaboration policy (https://people.cs.umass.edu/~brenocon/cs490a_f21/grading.html): all of the content you submit, both code and text, needs to be produced independently; do not share code or written materials, and list anyone you worked with for discussions. We will check code for plagiarism and we will follow up collaboration policy violations with the university's Academic Honesty Policy and Procedures.

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

# This code makes plots appear inline in this document rather than 
import matplotlib.pyplot as plt

# This code imports your work from hw1.py
from hw1 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 
# see http://stackoverflow.com/questions/1907993/autoreload-of-modu
%load_ext autoreload
%autoreload 2
```

In [2]:
```
# download the IMDB large movie review corpus to a file location on

PATH_TO_DATA = 'large_movie_review_dataset'  # set this variable to
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
    else:
        print("Oh no! Something is wrong. Check your code which loa
```

```
Great! You have 12500 pos reviews in large_movie_review_dataset/tr
ain/pos
Great! You have 12500 neg reviews in large_movie_review_dataset/tr
ain/neg
```

In [3]:
```
# Actually reading the data you are working with is an important pa

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

```
Right away, this film was ridiculous. Not that it didn't have rede
eming aspects 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 have beautiful mountains like those
found in the West. I knew it was Utah right off the bat, but perha
ps Dalton couldn't suppress his English accent, so they had to exc
use it by saying this was a southern town. Subverting his accent i
nto 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 n
ot like Utah doesn't have meth labs... so why the writers thought
it necessary to pretend it was in the South is beyond me. <br /><b
r />One other thing in action pictures always puzzles me. Why do t
hey always make the "cocking" sound effect when the character pull
s out an automatic handgun? It seemed every other sound effect in
this movie was a "chuk-chich" signifying a 9mm was loaded and read
y to fire. Of course, the weapons already had rounds chambered so
this was unnecessary. <br /><br />Lastly, the pyrotechnics were WA
Y over the top. But hey, this film was targeted to a certain 'mark
et 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 Unix commands

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 `hw1.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 hw1.py. Here is a sho

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 abstrac
for b in bow2:
    print(b)
```

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

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
import codecs
from collections import defaultdict, Counter
word_counts = Counter() # Counters are often useful for NLP in pyth

for label in [POS_LABEL, NEG_LABEL]:
    for directory in [TRAIN_DIR, TEST_DIR]:
        for fn in glob.glob(directory + "/" + label + "/*txt"):
            doc = open(fn, 'r', encoding='utf8') # Open the file wi
            # IMPLEMENT ME
            words = tokenize_doc(doc.read())
            for word in words:
                if word not in word_counts:
                    word_counts[word] = words[word]
                else:
                    word_counts[word] += words[word]
```

In [6]:
```
if word_counts["movie"] == 61492:
    print ("yay! there are {} total instances of the word type movi
else:
    print ("hmm. Something seems off. Double check your code")
```

```
yay! there are 61492.0 total instances of the word type movie in t
he corpus
```

Question 1.2 (5 points)

Fill out the functions `n_word_types` and `n_word_tokens` in `hw1.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(
print ("there are {} word tokens in the corpus".format(n_word_token
```

```
there are 390931 word types in the corpus
there are 11557847.0 word tokens in the corpus
```

What is the difference between word types and tokens? Why are the number of tokens much higher than the number of types?

Answer in one or two sentences here. word types are like how many type of word are in the document, for example movie appear 61492 times in the document, but it's only count one type of word. And word tokens are the total word count of the document

Question 1.3 (5 points)

Using `word_counts` dictionary you just created, make a new list of (word,count) pairs called `sorted_list` where tuples are sorted according to counts, in decending order. Then print the first 30 values from `sorted_list`.

In [8]:
```
# IMPLEMENT ME!
sorted_list = []
for word in word_counts:
    sorted_list.append((word, word_counts[word]))
sorted_list = sorted(sorted_list,key = lambda word:word[1],reverse=
print(sorted_list[:30])
```

```
[('the', 638861.0), ('a', 316615.0), ('and', 313637.0), ('of', 286
661.0), ('to', 264573.0), ('is', 204876.0), ('in', 179807.0), ('i'
, 141587.0), ('this', 138483.0), ('that', 130140.0), ('it', 129614
.0), ('/><br', 100974.0), ('was', 93258.0), ('as', 88242.0), ('wit
h', 84590.0), ('for', 84510.0), ('but', 77864.0), ('on', 62890.0),
('movie', 61492.0), ('are', 57009.0), ('his', 56870.0), ('not', 56
765.0), ('you', 55600.0), ('film', 55086.0), ('have', 54423.0), ('
he', 51062.0), ('be', 50901.0), ('at', 45259.0), ('one', 44983.0),
('by', 43359.0)]
```

Unix Text Processing

In this part, you will practice extracting and processing information from text with Unix commands. Download `lotr_script.txt` on the course website to a file location on your computer. This text file corresponds to the movie script of *The Fellowship of the Rings* (2001). This script comes from a larger corpus of movie scripts, the ScriptBase-J (https://github.com/EdinburghNLP/scriptbase) corpus.

First, let's open and examine `lotr_script.txt`.

Question 1.4 (5 points)

Describe the structure of this script. How are roles, scene directions, and dialogue organized?

Answer in one or two sentences here. roles followed by 25 white space, scene directions did not follow by any white space, dialogue organized followed by 17 white spaces

Now that we've identified this file's structure, let's use Unix commands to process & analyze its contents.

You may want to take revisit the optional reading Ken Church, "Unix for Poets" (https://people.cs.umass.edu/~brenocon/cs490a_f21/kwc-unix-for-poets.pdf).

Question 1.5 (5 points)

Use Unix commands to print the name of each character with dialogue in the script, one name per line. This script's text isn't perfect, so expect a few additional names.

Implement this in `hw1.sh`. Then, copy your implementation and its resulting output into the following two cells.

Copy Unix commands here grep -o '^[[:space:]]{25}[A-Z][A-Z][A-Z][A-Z]*[[:space:]]?[A-Z]{0,9}[[:space:]]?$' lotr_script.txt | sort -u

```
ARAGORN
ARWEN
BILBO
BLACK RIDER
BOROMIR
BUTTERBUR
CELEBORN
ELROND
FARMER MAGGOT
FRO DO
FRODO
GALADRIEL
GANDALF
GATEKEEPER
GIMLI
GOLLUM
HALDIR
HOBBIT BOUNDER
ISILDUR
LEGOLAS
LURTZ
MERRY
ODO PROUDFOOT
ORC OVERSEER
PIPPIN
SAM
SARUMAN
STRIDER
WITCH KING
```

Question 1.6 (5 points)

Now, let's extract and analyze the *dialogue* of this script using Unix commands

First, extract all lines of dialogue in this script. Then, normalize and tokenize this text such that all alphabetic characters are converted to lowercase and words are sequences of alphabetic characers. Finally, print the top-20 most frequent word types and their corresponding counts.

Hint: Ignore parantheticals. These contain short stage directions.

Implement this in `hw1.sh`. Then, copy your implementation and its resulting output into the following two cells.

Copy Unix commands here export double='"' export num='0-9' grep -o "^[[:space:]]{17}['$.double]\backslash?[A-Za-z$num]+.*$" lotr_script.txt | tr "[A-Z]" "[a-z]" |tr -sc 'A-Za-z' '\012' | sort | uniq -c | sort -r | head -20

Copy output here 497 the 274 you 261 of 247 i 225 to 202 it 166 and 155 a 147 is 138 s 131 frodo 100 in 98 we 85 ring 85 not 84 my 81 will 81 that 79 what 79 have

Question 1.7 (5 points)

If we instead tokenized *all* text in the script, how might the results from Question 1.6 to change? Are there specific word types that might become more frequent?

Answer in one or two sentences here. Yes, as a script or in general english writing the most common used non-alphabet symbol is white space the number of white space is extramly high. There are specific word types become more frequent.

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 and `tokenize_and_update_model` function in `hw1.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` and `tokenize_and_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) == 251637:
    print("Great! The vocabulary size is {}".format(251637))
else:
    print("Oh no! Something seems off. Double check your code befor
```

```
REPORTING CORPUS STATISTICS
NUMBER OF DOCUMENTS IN POSITIVE CLASS: 12500.0
NUMBER OF DOCUMENTS IN NEGATIVE CLASS: 12500.0
NUMBER OF TOKENS IN POSITIVE CLASS: 2958832.0
NUMBER OF TOKENS IN NEGATIVE CLASS: 2885848.0
VOCABULARY SIZE: NUMBER OF UNIQUE WORDTYPES IN TRAINING CORPUS: 25
1637
Great! The vocabulary size is 251637
```

Exploratory analysis

Let's begin to explore the count statistics stored by the update model function. Implement `top_n` function in the Naive Bayes Block to find the top 10 most common words in the positive class and top 10 most common words in the negative class.

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 165805.0
 and 87029.0
 a 82055.0
 of 76155.0
 to 65869.0
 is 55785.0
 in 48422.0
 i 33143.0
 it 32802.0
 that 32705.0

TOP 10 WORDS FOR CLASS neg:
 the 156393.0
 a 77898.0
 and 71543.0
 of 68307.0
 to 68098.0
 is 48386.0
 in 42105.0
 i 37337.0
 this 37301.0
 that 33587.0
```

Question 2.2 (5 points)

What is the first thing that you notice when you look at the top 10 words for the 2 classes? Are these words helpful for discriminating between the two classes? Do you imagine that processing other English text will result in a similar phenomenon? What about other languages?

Answer in one or two lines here.

All the 10 words in both classes are natural words (like we speaking with it). Those wards is not helpful for discriminating between the two classes. I think that processing other English text will result in a similar phenomenon. I think it's going to happened to Chinese too(which is the only other language I know of)

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 “amazing” given each sentiment label. Repeat the computation for the word “dull.”

In [11]:
```
print("P('amazing'|pos):",  nb.p_word_given_label("amazing", POS_LA
print("P('amazing'|neg):",  nb.p_word_given_label("amazing", NEG_LA
print("P('dull'|pos):",  nb.p_word_given_label("dull", POS_LABEL))
print("P('dull'|neg):",  nb.p_word_given_label("dull", NEG_LABEL))
```

```
P('amazing'|pos): 0.00026158970837141145
P('amazing'|neg): 7.207586816769282e-05
P('dull'|pos): 3.278320634628799e-05
P('dull'|neg): 0.00014311218054450546
```

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 in one or two lines here. fatastic should have a higher probability in positive class. boring should have a higher probility in negative class. I think the output was I expected

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-thiever
print("P('car-thievery'|neg):",  nb.p_word_given_label("car-thiever
```

```
P('car-thievery'|pos): 3.37971199446268e-07
P('car-thievery'|neg): 0.0
```

What is unusual about P('car-thievery'|neg)? 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('dull'|neg)"? Why might these operations cause problems for a Naive Bayes classifier?

Answer in one or two lines here.

car-thievery never appear in negative class. since P('car-thievery'|neg) is 0 log of P('car-thievery'|neg) would be so close to -infinite. if we multiplied "P('car-thievery'|neg)" by "P('dull'|neg)" the result still going to be 0 since P('car-thievery'|neg) is 0

Question 2.5 (5 pts)

We can address the issues from question 2.4 with add-α smoothing (like add-1 smoothing except instead of adding 1 we add α). Implement `p_word_given_label_and_alpha` in the `Naive Bayes Block` and then run the next cell. Hint: look at the slides from the lecture on add-1 smoothing.

In [13]:
```
print("P('stop-sign.'|pos):",  nb.p_word_given_label_and_alpha("sto
```

```
P('stop-sign.'|pos): 6.646374399441918e-08
```

Question 2.6 (5 pts) (getting ready for question 2.11)

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 alpha 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...

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 UMass) 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

Answer in one or two lines here. $P(w_d) = \sum_{d=0}^{n} P(w_d|y_d)P(y_d)$

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 in one or two lines here. Because the value of both class are the same, therefore, its would not change the value of argmax

Question 2.9 (10 pts) As we saw earlier, the top 10 words from each class do not seem to tell us much about the classes. A much more informative metric, which in some ways the model actually directly uses, is the likelihood ratio, which is defined as

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

A word with LR=3 is 3 times more likely to appear in the positive class than in the negative. A word with LR 0.33 is one-third as likely to appear in the positive class as opposed to the negative class.

In [14]:
```
# Implement the nb.likelihood_ratio function and use it to investig
print ("LIKELIHOOD RATIO OF 'amazing':", nb.likelihood_ratio('amazi
print ("LIKELIHOOD RATIO OF 'dull':", nb.likelihood_ratio('dull', 0
print ("LIKELIHOOD RATIO OF 'and':", nb.likelihood_ratio('and', 0.2
print ("LIKELIHOOD RATIO OF 'to':", nb.likelihood_ratio('to', 0.2))
```

```
LIKELIHOOD RATIO OF 'amazing': 3.628350587556548
LIKELIHOOD RATIO OF 'dull': 0.22953174277018223
LIKELIHOOD RATIO OF 'and': 1.1869527527674362
LIKELIHOOD RATIO OF 'to': 0.9438077915764572
```

What is the minimum and maximum possible values the likelihood ratio can take? Does it make sense that $LR('amazing') > LR('to')$?

Answer in one or two lines here. The maximum value of likelihood ratio is unbound and the minimum value of likelihood is 0. It's make sence that $LR('amazing') > LR('to')$. Becuase that 'amazing' is more like a positive word, 'amazing' appear in positive document more than 'to', which 'to' is more like a natural word(' i want to ', 'I'm going to', 'im heading to',etc) can appears close number is both positive document and negative document

Question 2.10 (5 pts)

Find the word in the vocabulary with the highest likelihood ratio below.

In [15]:
```
# Implement me!
toReturn = ''
max = 0.0
for word in nb.vocab:
    cur = nb.likelihood_ratio(word,0.2)
    if cur > max:
        max = cur
        toReturn = word
print (toReturn)
print(max)
```

```
edie
401.0316267725798
```

Question 2.11 (5 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 [16]:
```
print(nb.evaluate_classifier_accuracy(0.2))
```

```
81.668
```

Question 2.12 (5 pts)

Try evaluating your model again with a smoothing parameter of 1000.

In [17]:
```
print(nb.evaluate_classifier_accuracy(1000.0))
```

```
75.9
```

Does the accuracy go up or down when alpha is raised to 1000? Why do you think this is?

Answer in one or two lines here.

The accuracy go down when alpha raised. Because the it's over smoothed. For example some work appeared 100 times in 20000 counts with 1000 size vocab words the probility was 0.5% after alpha its become 1100/(20000+1000000) => 0.0011% which would effect the overall accuracy.

Question 2.13 (5 pts)

Find a review that your classifier got wrong.

In [18]:
```
# in this cell, print out a review that your classifier got wrong.
pos_path = os.path.join(nb.test_dir, POS_LABEL)
neg_path = os.path.join(nb.test_dir, NEG_LABEL)
for (p, label) in [(pos_path, POS_LABEL), (neg_path, NEG_LABEL)]:
    for f in os.listdir(p):
        with open(os.path.join(p, f), 'r') as doc:
            content = doc.read()
            bow = nb.tokenize_doc(content)
            if nb.classify(bow, 0.2) != label:
                print(content,label)
                break
```

```
I really like this show. It has drama, romance, and comedy all rol
led into one. I am 28 and I am a married mother, so I can identify
both with Lorelei's and Rory's experiences in the show. I have bee
n watching mostly the repeats on the Family Channel lately, so I a
m not up-to-date on what is going on now. I think females would li
ke this show more than males, but I know some men out there would
enjoy it! I really like that is an hour long and not a half hour,
as th hour seems to fly by when I am watching it! Give it a chance
if you have never seen the show! I think Lorelei and Luke are my f
avorite characters on the show though, mainly because of the way t
hey are with one another. How could you not see something was ther
e (or take that long to see it I guess I should say)? <br /><br />
Happy viewing! pos
In Los Angeles, the alcoholic and lazy Hank Chinaski (Matt Dillon)
performs a wide range of non-qualified functions just to get enoug
h money to drink and gamble in horse races. His primary and only o
bjective is writing and having sexy with dirty women.<br /><br />"
Factotum" is an uninteresting, pointless and extremely boring movi
e about an irresponsible drunken vagrant that works a couple of da
ys or weeks just to get enough money to buy spirits and gamble, be
ing immediately fired due to his reckless behavior. In accordance
with IMDb, this character would be the fictional alter-ego of the
author Charles Bukowski, and based on this story, I will certainly
never read any of his novels. Honestly, if the viewer likes this t
heme of alcoholic couples, better off watching the touching and he
artbreaking Hector Babenco's "Ironweed" or Marco Ferreri's "Storie
di Ordinaria Follia" that is based on the life of the same writer.
My vote is four.<br /><br />Title (Brazil): "Factotum  Sem Destino
" ("Factotum  Without Destiny") neg
```

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?

Answer in one or two lines here. The very first reason I could think of is the train data size is too small. for example words 'drink' 'gamble' 'heartbreaking'are negative words at most of the time which could bring the negative rate higher. I think larger training set is more accurate it's going to get

Question 2.14 (5 pts)

Often times we care about multi-class classification rather than binary classification.

How many counts would we need to keep track of if the model were modified to support 5-class classification?

Answer in one or two lines here. for class_total_doc_counts since we have 5 class here is 5 numbers for class_total_word_counts since we have 5 class here is 5 numbers for self.class_word_counts since we have 5 class here going to be 5 dict 5+5+5*len(vocab)

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 cells below.

Roughly speaking, the larger performance improvement, the more extra credit. We will also give points for the effort in the evaluation and analysis process. For example, you can split the training data into training and validation set to prevent overfitting, and report results from trying different versions of features. You can also provide some qualitative examples you found in the dataset to support your choices on preprocessing steps. Whatever you choose to try, make sure to describe your method and the reasons that you hypothesize for why the method works. You can use this ipython notebook to show your work. Be sure to explain what your code is doing in the notebook.

In [19]:
```
# from nltk.corpus import stopwords
from nltk.tokenize import RegexpTokenizer
from nltk.stem.snowball import SnowballStemmer
from nltk.util import ngrams
stemmer = SnowballStemmer('english')
# stopset = set(stopwords.words('english'))
def tokenize_doc_and_more(doc):
    """
    Return some representation of a document.
    At a minimum, you need to perform tokenization, the rest is up to
    """
    # Implement me!
    # your code goes here
    bow = defaultdict(float)
    tokenizer = RegexpTokenizer('\w+|\$[\d\.]+|\S+') # make a tokeniz
    bow = list(ngrams(tokenizer.tokenize(doc), 2)) # create bi-grams
    bow = Counter(bow)
    return bow
```

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: 12500.0
NUMBER OF DOCUMENTS IN NEGATIVE CLASS: 12500.0
NUMBER OF TOKENS IN POSITIVE CLASS: 3335586.0
NUMBER OF TOKENS IN NEGATIVE CLASS: 3264080.0
VOCABULARY SIZE: NUMBER OF UNIQUE WORDTYPES IN TRAINING CORPUS: 14
36772
```

Out[22]:
```
87.436
```

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. Doing a good job examining, explaining and justifying your work with small experiments and comments is as important as making the accuracy number go up!

In [21]:
```
# Your experiments and explanations go here
# At the first I tokenlized all the word by using regexptokenizer.
# reviews. At last I used counter to get the count of each word in
# which I count it as a win. By using bigrams most of the words in
# word have doubled the count which increament the size of the trai
```