DATASCI W261: Machine Learning at Scale

Nick Hamlin
nickhamlin@gmail.com
Time of Initial Submission: 9:21 PM EST, Monday, January 18, 2016
Time of **Resubmission**: 8:38 AM EST, Friday, January 22, 2016 W261-3, Spring 2016
Week 1 Homework

Submission Notes:

- For each problem, I've included a summary of the question as posed in the instructions. In many cases, I have not included the full text to keep the final submission as uncluttered as possible. For reference, I've included a link to the original instructions in the "Useful Reference" below.
- Problem statements are listed in *italics*, while my responses are shown in plain text.
- I have written driver functions for each problem where a solution is provided in pure Python. For simplicity, I have omitted them for the sections that use Bash commands either directly or to create files.

Useful References:

- **Original Assignment Instructions (https://www.dropbox.com/sh/jylzkmauxkostck/AAA2pH0cTvb0zDrbbbze3zf-a/hw1_instructions.txt?dl=0)**
- Wikipedia explaination of Naive Bayes document classification (https://en.wikipedia.org/wiki/Naive_Bayes_classifier#Document_classification)
- Original paper describing the background of the Enron email corpus (http://www.aueb.gr/users/ion/docs/ceas2006_paper.pdf)
- Documentation for Scikit-Learn implementation of Naive Bayes (http://scikit-learn.org/stable/modules/naive_bayes.html)
- Stanford NLP Group's explaination of Naive Bayes algorithm (http://nlp.stanford.edu/IR-book/html/htmledition/properties-of-naive-bayes-1.html)

HW1.0.0.

Define big data. Provide an example of a big data problem in your domain of expertise.

Big data is data with high volumn, velocity, or variety. This data typically represents terabytes or petabytes worth of storage, and is often too much for a single computer in terms of both processing and throughput. For example, a personal laptop with 1TB of storage space is typically only able to effectively process 3-4GB of data at once, orders of magnitude smaller than many "big datasets". As a result, parallel solutions become essential tools for extracting meaning at scale. A big data problem I encounter in my role is in aggregating information about all the individual visitors and their daily activity on the website that my organization maintains. Logging every click, page view, email, call, etc. creates a large, diverse set of data that must be stored and processed effectively at scale for us to be able to derive insights from it.

HW1.0.1.

In 500 words (English or pseudo code or a combination) describe how to estimate the bias, the variance, the irreducible error for a test dataset T when using polynomial regression models of degree 1, 2,3, 4,5 are considered. How would you select a model?

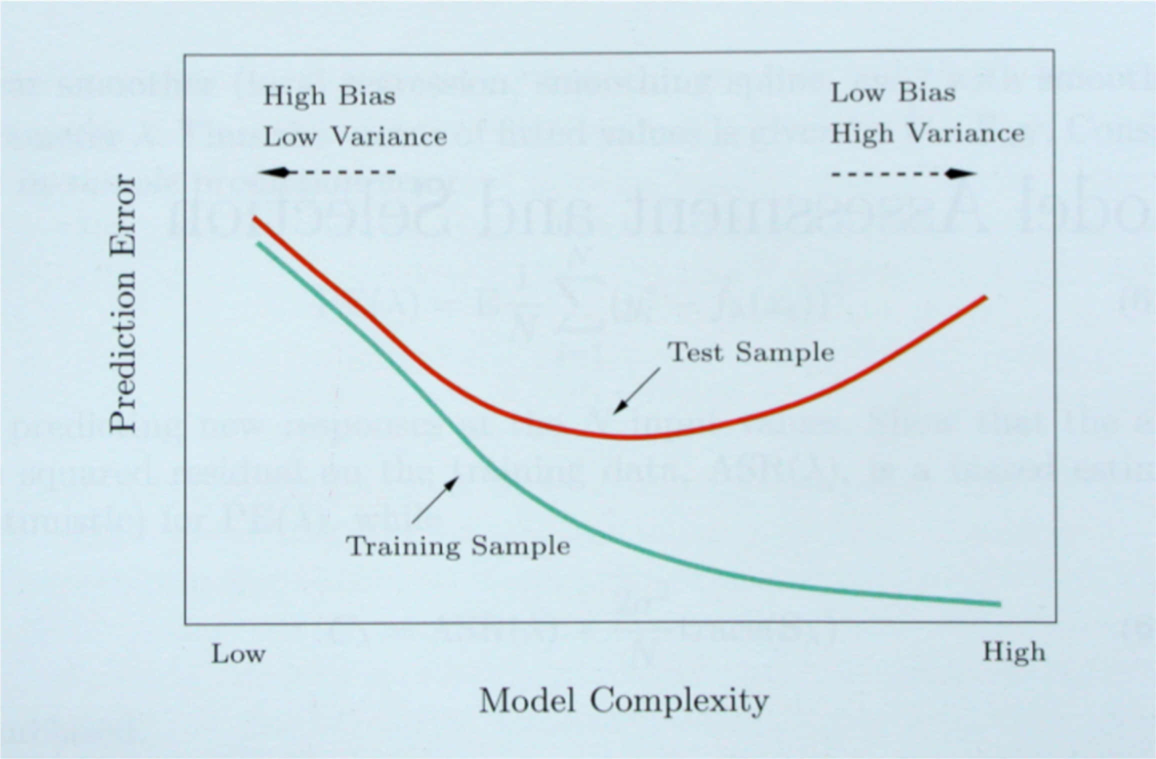
First, we should define the overall error in terms of the squared bias, variance, and irreducible error:

$$Err = (E[\hat{y}] - y)^2 + E[\hat{y} - E[\hat{y}]]^2 + E[(y_{true} - y)^2]$$

The first term is the squared bias, which measures the average error of the model. The second term is the variance, which measures how much our model's predictions vary from one training set to another. The final term represent the irreducible error; the variation between our model and reality that we are unable to do anything about (thus the name "irreducible error").

We'd like to choose the model that minimizes both bias and variance. However, we don't know the true function from which T was derived, so calculating this directly doesn't work. In practice, we will have (or can create by segmenting our training data) a test set of data on which to evaluate our model. Therefore, we can create regression models for each increasing degree of polynomial and use them to attempt to classify our test data. The model that has the lowest variance AND the lowest bias will be the one we choose. In the event that multiple models produce the minimum combination of bias and variance, we should choose the simplest (in this case, the model using the lowest degree polynomial).

Generally, as we increase the degree of the polynomial we use, our bias will drop because we'll fit the points in our training data more closely. However, this will likely come at a cost of increased variance, as the higher degree polynomials will also mean that our model will be comes less likely to generalize well to new, unseen data points. Consequently, our ability to successfully classify our training data will fall as our model complexity rises, but our performance on our test sample will begin to suffer as we overfit to the training data. This is clearly shown in the schematic below.



Hastie, Tibshirani, Friedman “Elements of Statistical Learning” 2001

We can use the following pseudocode for these calculations:

In [ ]:

```
#HW1.0.1 Pseudocode
for model in models:
    #this is the bagging step needed to calculate variance
    #where n is some constant (like 50)
    for iteration from 1:n
        Split training data randomly into train_data and test_data
        Train model using train_data
        h_star=predict results for test_data
    h_bar=calculate average prediction across all iterations
    bias=h_bar-y_true #y_true is the vector of true classes in the test_
    variance=sum((h_bar-h_star)^2)/n #in practice, one would need to go
    noise=mean((y_true-h_star)^2) #As with variance, this needs to be ca
choose model that minimizes (bias^2+variance)
```

Summary of Instructions for Rest of Assignment

In the remainder of this assignment you will produce a spam filter that is backed by a multinomial naive Bayes classifier (http://nlp.stanford.edu/IR-book/html/htmledition/properties-of-naive-bayes-1.html),*which counts words in parallel via a unix, poor-man's map-reduce framework.*

For the sake of this assignment we will focus on the basic construction of the parallelized classifier and not consider its validation or calibration, and so you will have the classifier operate on its own training data (unlike a field application where one would use non-overlapping subsets for training, validation and testing).

The data you will use is a curated subset of the Enron email corpus (whose details you may find in the file enronemail_README.txt in the directory surrounding these instructions).

HW1.1.

Read through the provided control script (pNaiveBayes.sh) and all of its comments. When you are comfortable with their purpose and function, respond to the remaining homework questions below. A simple cell in the notebook with a print statmement with a "done" string will suffice here.

In [66]:
```
## HW 1.1 Code

#Display contents of pNaiveBayes.sh (it's convenient to keep everythin
!cat pNaiveBayes.sh
!echo ""
!echo "Question 1.1: DONE"
```

```
## pNaiveBayes.sh
## Author: Jake Ryland Williams
## Usage: pNaiveBayes.sh m wordlist
## Input:
##       m = number of processes (maps), e.g., 4
##       wordlist = a space-separated list of words in quotes, e.g.,
"the and of"
##
## Instructions: Read this script and its comments closely.
##               Do your best to understand the purpose of each comm
and,
##               and focus on how arguments are supplied to mapper.p
y/reducer.py,
##               as this will determine how the python scripts take
input.
##               When you are comfortable with the unix code below,
##               answer the questions on the LMS for HW1 about the s
tarter code.

## collect user input
m=$1 ## the number of parallel processes (maps) to run
wordlist=$2 ## if set to "*", then all words are used

## a test set data of 100 messages
data="enronemail_1h.txt"

## the full set of data (33746 messages)
# data="enronemail.txt"

## 'wc' determines the number of lines in the data
## 'perl -pe' regex strips the piped wc output to a number
linesindata=`wc -l $data | perl -pe 's/^.*?(\d+).*?$/$1/'`

## determine the lines per chunk for the desired number of processes
linesinchunk=`echo "$linesindata/$m+1" | bc`

## split the original file into chunks by line
split -l $linesinchunk $data $data.chunk.

## assign python mappers (mapper.py) to the chunks of data
## and emit their output to temporary files
for datachunk in $data.chunk.*; do
    ## feed word list to the python mapper here and redirect STDOUT
to a temporary file on disk
    ####
    ####
    ./mapper.py $datachunk "$wordlist" > $datachunk.counts &
    ####
    ####
done
## wait for the mappers to finish their work
wait

## 'ls' makes a list of the temporary count files
## 'perl -pe' regex replaces line breaks with spaces
countfiles=`\ls $data.chunk.*.counts | perl -pe 's/\n/ /'`

## feed the list of countfiles to the python reducer and redirect ST
DOUT to disk
####
####
./reducer.py $countfiles > $data.output
####
####

## clean up the data chunks and temporary count files
\rm $data.chunk.*
```

```
Question 1.1: DONE
```

HW1.2.

Provide a mapper/reducer pair that, when executed by pNaiveBayes.sh will determine the number of occurrences of a single, user-specified word. Examine the word “assistance” and report your results. To do so, make sure that

- *mapper.py* ***counts all occurrences*** *of a single word*
- *reducer.py* ***collates the counts*** *of the single word*

CROSSCHECK: >grep assistance enronemail_1h.txt|cut -d$'\t' -f4| grep assistance|wc -l 8

HW 1.2 - Mapper Function

Our mapper function is pretty simple. It simply counts the number of instances of each word in the chunk and passes the result to the reducer.

```
In [1]: %%writefile mapper.py
        #!/usr/bin/python

        #HW 1.2 - Mapper Function Code
        import sys
        count = 0 #Running total of occurrances for the chosen word
        filename = sys.argv[1]
        findword = sys.argv[2].lower() #we probably don't need this extra case
        with open (filename, "r") as myfile:
            for line in myfile.readlines():
                subject_and_body=" ".join(line.split('\t')[-2:])#parse the sub
                count+=subject_and_body.count(findword) #Python's str.count()
        print findword+'\t'+str(count)
```

```
Overwriting mapper.py
```

HW 1.2 - Reducer Function

Since the mapper file does most of the work in this instance, the reducer can be very simple. Here, all we need to do it extract the intermediate total for each chunk and add it to our overall running total.

```
In [2]: %%writefile reducer.py
        #!/usr/bin/python

        #HW 1.2 - Reducer Function Code
        import sys
        sum = 0 #Running total of occurrances for the chosen word
        for chunk in sys.argv[1:]:
            with open (chunk, "r") as myfile:
                for i in myfile.readlines():
                    line=i.split('\t') #Parse line into a list of fields
                    sum+=int(line[1]) #Extract chunk count from the second fie
        print line[0]+'\t'+str(sum)
```

```
Overwriting reducer.py
```

HW 1.2 - Run Files and Check Output

```
In [69]: #Run our HW 1.2 code and check the results in the output file
         !chmod a+x mapper.py reducer.py
         !./pNaiveBayes.sh 5 "assistance"
         !echo "HW 1.2 - Results"
         !cat enronemail_1h.txt.output
```

```
HW 1.2 - Results
assistance      10
```

```
In [70]: #Run our crosscheck command as a sanity check
         !echo "HW 1.2 - Crosscheck Results"
         !grep assistance enronemail_1h.txt|cut -d$'\t' -f4| grep assistance|wc
```

```
HW 1.2 - Crosscheck Results
       8
```

Our crosschecker gives us a close sanity check, but since it's only looking at the body of each message and it's only counting the number of lines containing 'assistance' not the overall number of times that the word occurs, its result will be a bit off.

HW1.3.

Provide a mapper/reducer pair that, when executed by pNaiveBayes.sh will classify the email messages by a single, user-specified word using the multinomial Naive Bayes Formulation. Examine the word “assistance” and report your results. To do so, make sure that mapper.py and reducer.py perform a single word Naive Bayes classification.

For multinomial Naive Bayes, the $Pr(X = \text{“}assistance\text{”}|Y = SPAM)$ *is calculated as follows: the number of times “assistance” occurs in SPAM labeled documents / the number of words in documents labeled SPAM.*

NOTE: If “assistance” occurs 5 times in all of the documents labeled SPAM, and the length in terms of the number of words in all documents labeled as SPAM (when concatenated) is 1,000, then $Pr(X = \text{“}assistance\text{”}|Y = SPAM) = 5/1000$*. This is a multinomial estimation of the class conditional for a Naive Bayes Classifier.*

HW 1.3 - Define training error function

It's useful to define this function early on, so we can recycle it throughout the rest of the problems.

In [1]:

```
#HW 1.3-1.6 Training Error Function

from __future__ import division

def calculate_training_error(pred, true):
    """Calculates the training error given a vector
    of predictions and a vector of true classes"""

    num_wrong=0
    for i in zip(pred,true):
        if i[0]!=i[1]: #If predicted value doesn't equal true value, i
            num_wrong+=1

    #Divide number of incorrect examples by total number of examples i
    print "Training error: "+str(num_wrong/len(pred))
```

HW 1.3 - Mapper function

This mapper function will send one line for every instance of every word to the reducer. This approach, while easier to write and debug, is unlikely to be the best choice for a larger scale implementation because of the large volume of data that would have to be sent to the reducers. A potentially more-streamlined alternative would be to add a "combiner" step at the end of the mapper that would send one line for each word-email combination (E.G. Key:email-word-flag, Value:count).

In addition, if we only care about generating the conditional probabilities for each word and don't need to classify all the training emails, we could simplify the implementation even more and not send the email contents themselves to the reducer. Not only would this decrease throughput, but it would also dramatically reduce the amount of information that the reducer would need to store in memory. In this situation, our mapper could simply emit words along with their conditional class counts. I have not implemented this in this homework, but we'll want to keep this in mind for the future.

In [2]:

```
%%writefile mapper.py
#!/usr/bin/python

#HW 1.3 - Mapper Function Code
import sys
import re
WORD_RE = re.compile(r"[\w']+") #Compile regex to easily parse complet
filename = sys.argv[1]
findwords = sys.argv[2].lower()
with open (filename, "r") as myfile:
    for num,line in enumerate(myfile.readlines()):
        fields=line.split('\t') #parse line into separate fields
        subject_and_body=" ".join(fields[-2:]).strip()#parse the subje
        words=re.findall(WORD_RE,subject_and_body) #create list of wor
        for word in words:
            #This flag indicates to the reducer that a given word shou
            #by the reducer when calculating the conditional probabili
            flag=0
            if word in findwords:
                flag=1

            #This will send one row for every word instance to the red
            print fields[0]+'\t'+fields[1]+'\t'+word+'\t1\t'+str(flag)
```

```
Overwriting mapper.py
```

HW 1.3 - Reducer function

The reducer maintains two associative arrays. The first stores information about each word, including how many times it appears in spam and ham messages, as well as if it's been flagged in the mapper. The second stores information about emails, including whether it is marked as spam, as well as a list of words it contains. As described above, a more scalable solution that does not need to maintain all the contents of the emails in memory for classification could simply calculate conditional probabilities "lazily" and only store the running probability values rather than the words themselves.

Once all the data has arrived from the mappers, the array containing words is updated with the calculated conditional probabilities of spam and ham. At this point, the model is "trained". Finally, these conditional probabilities are reapplied to the word lists associated with each email to make the final spam/ham classification.

Note: I have also included (in the comments) an alternative calculation for conditional probabilities that applies a Laplace smoothing approach, since I'd already implemented it before the assignment instructions were updated. I've done the same thing with the log probability calculations.

In [3]:

```
%%writefile reducer.py
#!/usr/bin/python

#HW 1.3 - Reducer Function Code
from __future__ import division #Python 3-style division syntax is muc
import sys
from math import log

words={}
emails={}
spam_email_count=0 #number of emails marked as spam
spam_word_count=0 #number of total (not unique) words in spam emails
ham_word_count=0 #number of total (not unique) words in ham emails
flagged_words=[]
for chunk in sys.argv[1:]:
    with open (chunk, "r") as myfile:
        for i in myfile.readlines():

            #parse the incoming line
            result=i.split("\t")
            email=result[0]
            spam=int(result[1])
            word=result[2]
            flag=int(result[4])

            #initialize storage for word/email data
            if word not in words.keys():
                words[word]={'ham_count':0,'spam_count':0,'flag':flag}
            if email not in emails.keys():
                emails[email]={'spam':spam,'word_count':0,'words':[]}
                if spam==1:
                    spam_email_count+=1

            #store word data
            if spam==1:
                words[word]['spam_count']+=1
                spam_word_count+=1
            else:
                words[word]['ham_count']+=1
                ham_word_count+=1

            if flag==1 and word not in flagged_words:
                flagged_words.append(word)

            #store email data
            emails[email]['words'].append(word)
            emails[email]['word_count']+=1

#Calculate stats for entire corpus
prior_spam=spam_email_count/len(emails)
prior_ham=1-prior_spam
vocab_count=len(words)#number of unique words in the total vocabulary

for k,word in words.iteritems():
```

```
for k,word in words.iteritems():
    #These versions calculate conditional probabilities WITH Laplace s
    #word['p_spam']=(word['spam_count']+1)/(spam_word_count+vocab_coun
    #word['p_ham']=(word['ham_count']+1)/(ham_word_count+vocab_count)

    #Compute conditional probabilities WITHOUT Laplace smoothing
    word['p_spam']=(word['spam_count'])/(spam_word_count)
    word['p_ham']=(word['ham_count'])/(ham_word_count)

#At this point the model is now trained, and we can use it to make our
for j,email in emails.iteritems():

    #Log versions - previously used, but removed for now
    #p_spam=log(prior_spam)
    #p_ham=log(prior_ham)

    p_spam=prior_spam
    p_ham=prior_ham
    for word in email['words']:
        if word in flagged_words:
            try:
                #p_spam+=log(words[word]['p_spam']) #Log version - No
                p_spam*=(words[word]['p_spam'])
            except ValueError:
                continue #This means that words that do not appear in
            try:
                #p_ham+=log(words[word]['p_ham']) #Log version - No lo
                p_ham*=(words[word]['p_ham'])
            except ValueError:
                continue
    if p_spam>p_ham:
        spam_pred=1
    else:
        spam_pred=0

    print j+'\t'+str(email['spam'])+'\t'+str(spam_pred)
```

```
Overwriting reducer.py
```

HW 1.3 - Running code and evaluating results

In [4]:
```
#Run our HW 1.3 code and check the results in the output file
!chmod a+x mapper.py reducer.py
!./pNaiveBayes.sh 5 "assistance"
!echo "HW 1.3 - Results"
!cat enronemail_1h.txt.output
```

```
HW 1.3 - Results
0010.2003-12-18.GP      1       0
0010.2001-06-28.SA_and_HP       1       1
0001.2000-01-17.beck    0       1
0018.1999-12-14.kaminski        0       0
0005.1999-12-12.kaminski        0       1
0011.2001-06-29.SA_and_HP       1       1
0008.2004-08-01.BG      1       1
0009.1999-12-14.farmer  0       0
0017.2003-12-18.GP      1       0
0011.2001-06-28.SA_and_HP       1       1
0015.2001-07-05.SA_and_HP       1       1
0015.2001-02-12.kitchen 0       1
0009.2001-06-26.SA_and_HP       1       1
0017.1999-12-14.kaminski        0       0
0012.2000-01-17.beck    0       0
0003.2000-01-17.beck    0       0
0004.2001-06-12.SA_and_HP       1       0
0008.2001-06-12.SA_and_HP       1       0
0007.2001-02-09.kitchen 0       1
0016.2004-08-01.BG      1       0
0015.2000-06-09.lokay   0       0
0005.1999-12-14.farmer  0       1
0016.1999-12-15.farmer  0       0
0013.2004-08-01.BG      1       1
0005.2003-12-18.GP      1       1
0012.2001-02-09.kitchen 0       0
0003.2001-02-08.kitchen 0       1
0009.2001-02-09.kitchen 0       0
0006.2001-02-08.kitchen 0       1
0014.2003-12-19.GP      1       0
0010.1999-12-14.farmer  0       0
0010.2004-08-01.BG      1       0
0014.1999-12-14.kaminski        0       0
0006.1999-12-13.kaminski        0       0
0011.1999-12-14.farmer  0       1
0013.1999-12-14.kaminski        0       1
0001.2001-02-07.kitchen 0       1
0008.2001-02-09.kitchen 0       0
0007.2003-12-18.GP      1       0
0017.2004-08-02.BG      1       1
0014.2004-08-01.BG      1       0
0006.2003-12-18.GP      1       0
0016.2001-07-05.SA_and_HP       1       1
0008.2003-12-18.GP      1       0
0014.2001-07-04.SA_and_HP       1       1
0001.2001-04-02.williams        0       0
```

```
0012.2000-06-08.lokay   0       1
0014.1999-12-15.farmer  0       0
0009.2000-06-07.lokay   0       0
0001.1999-12-10.farmer  0       0
0008.2001-06-25.SA_and_HP       1       1
0017.2001-04-03.williams        0       0
0014.2001-02-12.kitchen 0       0
0016.2001-07-06.SA_and_HP       1       1
0015.1999-12-15.farmer  0       1
0009.1999-12-13.kaminski        0       1
0001.2000-06-06.lokay   0       1
0011.2004-08-01.BG      1       0
0004.2004-08-01.BG      1       1
0018.2003-12-18.GP      1       1
0002.1999-12-13.farmer  0       1
0016.2003-12-19.GP      1       1
0004.1999-12-14.farmer  0       0
0015.2003-12-19.GP      1       1
0006.2004-08-01.BG      1       1
0009.2003-12-18.GP      1       1
0007.1999-12-14.farmer  0       0
0005.2000-06-06.lokay   0       1
0010.1999-12-14.kaminski        0       0
0007.2000-01-17.beck    0       0
0003.1999-12-14.farmer  0       0
0003.2004-08-01.BG      1       1
0017.2004-08-01.BG      1       0
0013.2001-06-30.SA_and_HP       1       0
0003.1999-12-10.kaminski        0       0
0012.1999-12-14.farmer  0       0
0004.1999-12-10.kaminski        0       1
0018.2001-07-13.SA_and_HP       1       1
0002.2001-02-07.kitchen 0       0
0007.2004-08-01.BG      1       0
0012.1999-12-14.kaminski        0       1
0005.2001-06-23.SA_and_HP       1       0
0007.1999-12-13.kaminski        0       0
0017.2000-01-17.beck    0       0
0006.2001-06-25.SA_and_HP       1       0
0006.2001-04-03.williams        0       0
0005.2001-02-08.kitchen 0       0
0002.2003-12-18.GP      1       1
0003.2003-12-18.GP      1       0
0013.2001-04-03.williams        0       0
0004.2001-04-02.williams        0       0
0010.2001-02-09.kitchen 0       0
0001.1999-12-10.kaminski        0       0
0013.1999-12-14.farmer  0       0
0015.1999-12-14.kaminski        0       0
0012.2003-12-19.GP      1       0
0016.2001-02-12.kitchen 0       0
0002.2004-08-01.BG      1       1
0002.2001-05-25.SA_and_HP       1       1
0011.2003-12-18.GP      1       0
```

In [5]:

```
#HW 1.3 Evaluation Code
import pandas as pd #Use pandas to quickly read results from our outpu

def eval_1_3():
    with open('enronemail_1h.txt.output','rb') as f:
        mr_data=pd.read_csv(f, sep='\t', header=None)
    print "Multinomial NB Results via Poor-Man's MapReduce Implementat
    calculate_training_error(mr_data[1],mr_data[2])

eval_1_3()
```

```
Multinomial NB Results via Poor-Man's MapReduce Implementation using
'Assistance' only
Training error: 0.38
```

Unsurprisingly, just using "assistance" as an indicator of spam isn't a particularly effective classification approach.

HW1.4.

Provide a mapper/reducer pair that, when executed by pNaiveBayes.sh will classify the email messages by a list of one or more user-specified words. Examine the words “assistance”, “valium”, and “enlargementWithATypo” and report your results. To do so, make sure that

- *mapper.py* ***counts all occurrences*** *of a list of words, and*
- *reducer.py* ***performs the multiple-word Naive Bayes classification*** *via the chosen list.*

HW 1.4 - Mapper function

This mapper function works very similarly to the implementation in 1.3. The only difference is that it enables iteration through a list of words (provided as arguments) for flagging for inclusion in the conditional probability calculation.

In [14]:

```
%%writefile mapper.py
#!/usr/bin/python

#HW 1.4 - Mapper Function
import sys
import re
WORD_RE = re.compile(r"[\w']+")
filename = sys.argv[1]
findwords = sys.argv[2].lower().split()
with open (filename, "r") as myfile:
    for num,line in enumerate(myfile.readlines()):
        fields=line.split('\t') #parse line into separate fields
        subject_and_body=" ".join(fields[-2:]).strip()#parse the subje
        words=re.findall(WORD_RE,subject_and_body)
        for word in words:
            flag=0
            if word in findwords:
                flag=1
            print fields[0]+'\t'+fields[1]+'\t'+word+'\t1\t'+str(flag)
```

```
Overwriting mapper.py
```

HW 1.4 - Reducer function

This reducer is almost exactly the same as in Problem 1.3. The only difference is not in the code itself, but in the fact that it receives more than one flagged word from the mapper. Because the flagged words are tracked via a list, the reducer doesn't care how many flagged words it receives. It will incorporate all of them into the conditional probability calculation.

Note: Again, the code for Laplace smoothing and log probability is included as comments, but is not used in the final implementation.

```
In [16]:
```

```
%%writefile reducer.py
#!/usr/bin/python

#HW 1.4 - Reducer Function
from __future__ import division
import sys
from math import log

#
emails={} #Associative array to hold email data
words={} #Associative array for word data

spam_email_count=0 #number of emails marked as spam
spam_word_count=0 #number of total (not unique) words in spam emails
ham_word_count=0 #number of total (not unique) words in ham emails
flagged_words=[] #list of flagged words to include in conditional prob
for chunk in sys.argv[1:]:
    with open (chunk, "r") as myfile:
        for i in myfile.readlines():

            #parse the line
            result=i.split("\t")
            email=result[0]
            spam=int(result[1])
            word=result[2]
            flag=int(result[4])

            #initialize storage for word/email data
            if word not in words.keys():
                words[word]={'ham_count':0,'spam_count':0,'flag':flag}
            if email not in emails.keys():
                emails[email]={'spam':spam,'word_count':0,'words':[]}
                if spam==1:
                    spam_email_count+=1

            #store word data
            if spam==1:
                words[word]['spam_count']+=1
                spam_word_count+=1
            else:
                words[word]['ham_count']+=1
                ham_word_count+=1

            if flag==1 and word not in flagged_words:
                flagged_words.append(word)

            #store email data
            emails[email]['words'].append(word)
            emails[email]['word_count']+=1

#Calculate stats for entire corpus
prior_spam=spam_email_count/len(emails)
prior_ham=1-prior_spam
vocab_count=len(words)#number of unique words in the total vocabulary
```

```
vocab_count=len(words)#number of unique words in the total vocabulary

for k,word in words.iteritems():
    #These versions calculate conditional probabilities WITH Laplace s
    #word['p_spam']=(word['spam_count']+1)/(spam_word_count+vocab_coun
    #word['p_ham']=(word['ham_count']+1)/(ham_word_count+vocab_count)

    #Compute conditional probabilities WITHOUT Laplace smoothing
    word['p_spam']=(word['spam_count'])/(spam_word_count)
    word['p_ham']=(word['ham_count'])/(ham_word_count)

#At this point the model is now trained, and we can use it to make our
for j,email in emails.iteritems():

    #Log versions - no longer used
    #p_spam=log(prior_spam)
    #p_ham=log(prior_ham)

    p_spam=prior_spam
    p_ham=prior_ham

    for word in email['words']:
        if word in flagged_words:
            try:
                #p_spam+=log(words[word]['p_spam']) #Log version - no
                p_spam*=words[word]['p_spam']
            except ValueError:
                pass #This means that words that do not appear in a cl
            try:
                #p_ham+=log(words[word]['p_ham']) #Log version - no lo
                p_ham*=words[word]['p_ham']
            except ValueError:
                pass
    if p_spam>p_ham:
        spam_pred=1
    else:
        spam_pred=0

    print j+'\t'+str(email['spam'])+'\t'+str(spam_pred)
```

```
Overwriting reducer.py
```

HW 1.4 - Running code and evaluating results

In [17]:

```
#Run our HW 1.4 code and check the results in the output file
!chmod a+x mapper.py reducer.py
!./pNaiveBayes.sh 5 "assistance valium enlargementWithATypo"
!echo "HW 1.4 - Results"
!cat enronemail_1h.txt.output
```

```
HW 1.4 - Results
0010.2003-12-18.GP      1       0
0010.2001-06-28.SA_and_HP       1       1
0001.2000-01-17.beck    0       0
0018.1999-12-14.kaminski        0       0
0005.1999-12-12.kaminski        0       1
0011.2001-06-29.SA_and_HP       1       0
0008.2004-08-01.BG      1       0
0009.1999-12-14.farmer  0       0
0017.2003-12-18.GP      1       0
0011.2001-06-28.SA_and_HP       1       1
0015.2001-07-05.SA_and_HP       1       0
0015.2001-02-12.kitchen 0       0
0009.2001-06-26.SA_and_HP       1       0
0017.1999-12-14.kaminski        0       0
0012.2000-01-17.beck    0       0
0003.2000-01-17.beck    0       0
0004.2001-06-12.SA_and_HP       1       0
0008.2001-06-12.SA_and_HP       1       0
0007.2001-02-09.kitchen 0       0
0016.2004-08-01.BG      1       0
0015.2000-06-09.lokay   0       0
0005.1999-12-14.farmer  0       0
0016.1999-12-15.farmer  0       0
0013.2004-08-01.BG      1       1
0005.2003-12-18.GP      1       0
0012.2001-02-09.kitchen 0       0
0003.2001-02-08.kitchen 0       0
0009.2001-02-09.kitchen 0       0
0006.2001-02-08.kitchen 0       0
0014.2003-12-19.GP      1       0
0010.1999-12-14.farmer  0       0
0010.2004-08-01.BG      1       0
0014.1999-12-14.kaminski        0       0
0006.1999-12-13.kaminski        0       0
0011.1999-12-14.farmer  0       0
0013.1999-12-14.kaminski        0       0
0001.2001-02-07.kitchen 0       0
0008.2001-02-09.kitchen 0       0
0007.2003-12-18.GP      1       0
0017.2004-08-02.BG      1       0
0014.2004-08-01.BG      1       0
0006.2003-12-18.GP      1       0
0016.2001-07-05.SA_and_HP       1       0
0008.2003-12-18.GP      1       0
0014.2001-07-04.SA_and_HP       1       0
0001.2001-04-02.williams        0       0
```

```
0012.2000-06-08.lokay   0       0
0014.1999-12-15.farmer  0       0
0009.2000-06-07.lokay   0       0
0001.1999-12-10.farmer  0       0
0008.2001-06-25.SA_and_HP       1       0
0017.2001-04-03.williams        0       0
0014.2001-02-12.kitchen 0       0
0016.2001-07-06.SA_and_HP       1       0
0015.1999-12-15.farmer  0       0
0009.1999-12-13.kaminski        0       0
0001.2000-06-06.lokay   0       0
0011.2004-08-01.BG      1       0
0004.2004-08-01.BG      1       0
0018.2003-12-18.GP      1       1
0002.1999-12-13.farmer  0       0
0016.2003-12-19.GP      1       1
0004.1999-12-14.farmer  0       0
0015.2003-12-19.GP      1       0
0006.2004-08-01.BG      1       0
0009.2003-12-18.GP      1       1
0007.1999-12-14.farmer  0       0
0005.2000-06-06.lokay   0       0
0010.1999-12-14.kaminski        0       0
0007.2000-01-17.beck    0       0
0003.1999-12-14.farmer  0       0
0003.2004-08-01.BG      1       0
0017.2004-08-01.BG      1       1
0013.2001-06-30.SA_and_HP       1       0
0003.1999-12-10.kaminski        0       0
0012.1999-12-14.farmer  0       0
0004.1999-12-10.kaminski        0       1
0018.2001-07-13.SA_and_HP       1       1
0002.2001-02-07.kitchen 0       0
0007.2004-08-01.BG      1       0
0012.1999-12-14.kaminski        0       0
0005.2001-06-23.SA_and_HP       1       0
0007.1999-12-13.kaminski        0       0
0017.2000-01-17.beck    0       0
0006.2001-06-25.SA_and_HP       1       0
0006.2001-04-03.williams        0       0
0005.2001-02-08.kitchen 0       0
0002.2003-12-18.GP      1       0
0003.2003-12-18.GP      1       0
0013.2001-04-03.williams        0       0
0004.2001-04-02.williams        0       0
0010.2001-02-09.kitchen 0       0
0001.1999-12-10.kaminski        0       0
0013.1999-12-14.farmer  0       0
0015.1999-12-14.kaminski        0       0
0012.2003-12-19.GP      1       0
0016.2001-02-12.kitchen 0       0
0002.2004-08-01.BG      1       1
0002.2001-05-25.SA_and_HP       1       0
0011.2003-12-18.GP      1       0
```

In [18]:

```
#HW 1.4 - Evaluation code
def eval_1_4():
    with open('enronemail_1h.txt.output','rb') as f:
        mr_data=pd.read_csv(f, sep='\t', header=None)
    print "Multinomial NB Results via Poor-Man's MapReduce Implementat
    calculate_training_error(mr_data[1],mr_data[2])

eval_1_4()
```

```
Multinomial NB Results via Poor-Man's MapReduce Implementation using
'Assistance valium EnlargementWithATypo'
Training error: 0.37
```

Unsurprisingly, the addition of a few more words improves performance slightly, but not enough to make this an effective model for real spam classification.

APPENDIX

Since I'd already completed the rest of the assignment as of Monday afternoon when problems 1.5 and 1.6 were removed, I've left both in here for posterity.

HW1.5.

Provide a mapper/reducer pair that, when executed by pNaiveBayes.sh will classify the email messages by all words present. To do so, make sure that:

- *mapper.py **counts all occurrences of all words**, and*
- *reducer.py performs a **word-distribution-wide Naive Bayes classification** .*

HW 1.5 - Mapper function

Again, this mapper function works very similarly to the implementation in 1.3 and 1.4. The difference here is that this mapper removes the "flagging" feature present in the other two, because we will care about including all words in our conditional probability calculation.

In [10]:

```
%%writefile mapper.py
#!/usr/bin/python

#HW 1.5 - Mapper Function
import sys
import re
WORD_RE = re.compile(r"[\w']+")
filename = sys.argv[1]
with open (filename, "r") as myfile:
    for num,line in enumerate(myfile.readlines()):
        fields=line.split('\t') #parse line into separate fields
        subject_and_body=" ".join(fields[-2:]).strip()#parse the subje
        words=re.findall(WORD_RE,subject_and_body)
        for word in words:
            print fields[0]+'\t'+fields[1]+'\t'+word+'\t1'
```

```
Overwriting mapper.py
```

HW 1.5 - Reducer function

This reducer is similar to those used in 1.3 and 1.4, but removes the check for flagged words because it's no longer necessary and the flags aren't even passed from the mapper anymore. All words are accounted for the the conditional probability logic.

That said, in situations where a word only appears in one class or the other, we would be forced to calculate a Log(0), which doesn't work. In these situations, we have three choices. We can simply skip over these situations and use the prior probability for the class, we can use Laplace smoothing, or we can go back to implementing the basic MLE approach and multiply the non-log-transformed probabilities (meaning that words not appearing in a class will create a zero probability for that class - not a great classification approach). I've implemented all three versions, with the last as the only one not commented out.

In [11]:

```
%%writefile reducer.py
#!/usr/bin/python

#HW 1.5 - Reducer Function
from __future__ import division
import sys
from math import log
emails={}
words={}
spam_email_count=0 #number of emails marked as spam
spam_word_count=0 #number of total (not unique) words in spam emails
ham_word_count=0 #number of total (not unique) words in ham emails

for chunk in sys.argv[1:]:
    with open (chunk, "r") as myfile:
        for i in myfile.readlines():

            #parse the line
            result=i.split("\t")
            email=result[0]
            spam=int(result[1])
            word=result[2]

            #initialize storage for word/email data
            if word not in words.keys():
                words[word]={'ham_count':0,'spam_count':0}
            if email not in emails.keys():
                emails[email]={'spam':spam,'word_count':0,'words':[]}
                if spam==1:
                    spam_email_count+=1

            #store word data
            if spam==1:
                words[word]['spam_count']+=1
                spam_word_count+=1
            else:
                words[word]['ham_count']+=1
                ham_word_count+=1

            #store email data
            emails[email]['words'].append(word)
            emails[email]['word_count']+=1

#Calculate stats for entire corpus
prior_spam=spam_email_count/len(emails)
prior_ham=1-prior_spam
vocab_count=len(words)#number of unique words in the total vocabulary

for k,word in words.iteritems():
    #These versions calculate conditional probabilities WITH Laplace s
    #word['p_spam']=(word['spam_count']+1)/(spam_word_count+vocab_coun
    #word['p_ham']=(word['ham_count']+1)/(ham_word_count+vocab_count)

    #Compute conditional probabilities WITHOUT Laplace smoothing
    #compute conditional probabilities without laplace smoothing
    word['p_spam']=(word['spam_count'])/(spam_word_count)
    word['p_ham']=(word['ham_count'])/(ham_word_count)

#At this point the model is now trained, and we can use it to make our
for j,email in emails.iteritems():

    #Log Version - not used
    p_spam=log(prior_spam)
    p_ham=log(prior_ham)

    p_spam=prior_spam
    p_ham=prior_ham
    for word in email['words']:
        try:
            #p_spam+=log(words[word]['p_spam']) #Log Version - not use
            p_spam*=(words[word]['p_spam'])
        except ValueError:
            continue #This means that words that do not appear in a cl
        try:
            #p_ham+=log(words[word]['p_ham']) #Log Version - not used
            p_ham*=(words[word]['p_ham'])
        except ValueError:
            continue
    if p_spam>p_ham:
        spam_pred=1
    else:
        spam_pred=0

    #print spam_pred, email['spam'],p_spam,p_ham,j
    print j+'\t'+str(email['spam'])+'\t'+str(spam_pred)
```

```
Overwriting reducer.py
```

In [12]:

```
#Run our HW 1.5 code and check the results in the output file
!chmod a+x mapper.py reducer.py
!./pNaiveBayes.sh 5 *
!echo "HW 1.5 - Results"
!cat enronemail_1h.txt.output
```

```
HW 1.5 - Results
0010.2003-12-18.GP      1       1
0010.2001-06-28.SA_and_HP       1       0
0001.2000-01-17.beck    0       0
0018.1999-12-14.kaminski        0       0
0005.1999-12-12.kaminski        0       0
0011.2001-06-29.SA_and_HP       1       0
0008.2004-08-01.BG      1       0
0009.1999-12-14.farmer  0       0
0017.2003-12-18.GP      1       1
0011.2001-06-28.SA_and_HP       1       0
0015.2001-07-05.SA_and_HP       1       0
0015.2001-02-12.kitchen 0       0
0009.2001-06-26.SA_and_HP       1       0
0017.1999-12-14.kaminski        0       0
0012.2000-01-17.beck    0       0
0003.2000-01-17.beck    0       0
0004.2001-06-12.SA_and_HP       1       0
0008.2001-06-12.SA_and_HP       1       0
0007.2001-02-09.kitchen 0       0
0016.2004-08-01.BG      1       1
0015.2000-06-09.lokay   0       0
0005.1999-12-14.farmer  0       0
0016.1999-12-15.farmer  0       0
0013.2004-08-01.BG      1       0
0005.2003-12-18.GP      1       0
0012.2001-02-09.kitchen 0       0
0003.2001-02-08.kitchen 0       0
0009.2001-02-09.kitchen 0       0
0006.2001-02-08.kitchen 0       0
0014.2003-12-19.GP      1       1
0010.1999-12-14.farmer  0       0
0010.2004-08-01.BG      1       0
0014.1999-12-14.kaminski        0       0
0006.1999-12-13.kaminski        0       0
0011.1999-12-14.farmer  0       0
0013.1999-12-14.kaminski        0       0
0001.2001-02-07.kitchen 0       0
0008.2001-02-09.kitchen 0       0
0007.2003-12-18.GP      1       0
0017.2004-08-02.BG      1       0
0014.2004-08-01.BG      1       0
0006.2003-12-18.GP      1       0
0016.2001-07-05.SA_and_HP       1       0
0008.2003-12-18.GP      1       0
0014.2001-07-04.SA_and_HP       1       0
0001.2001-04-02.williams        0       0
```

```
0012.2000-06-08.lokay   0       0
0014.1999-12-15.farmer  0       0
0009.2000-06-07.lokay   0       0
0001.1999-12-10.farmer  0       0
0008.2001-06-25.SA_and_HP       1       0
0017.2001-04-03.williams        0       0
0014.2001-02-12.kitchen 0       0
0016.2001-07-06.SA_and_HP       1       0
0015.1999-12-15.farmer  0       0
0009.1999-12-13.kaminski        0       0
0001.2000-06-06.lokay   0       0
0011.2004-08-01.BG      1       1
0004.2004-08-01.BG      1       0
0018.2003-12-18.GP      1       0
0002.1999-12-13.farmer  0       0
0016.2003-12-19.GP      1       0
0004.1999-12-14.farmer  0       0
0015.2003-12-19.GP      1       0
0006.2004-08-01.BG      1       0
0009.2003-12-18.GP      1       0
0007.1999-12-14.farmer  0       0
0005.2000-06-06.lokay   0       0
0010.1999-12-14.kaminski        0       0
0007.2000-01-17.beck    0       0
0003.1999-12-14.farmer  0       0
0003.2004-08-01.BG      1       0
0017.2004-08-01.BG      1       0
0013.2001-06-30.SA_and_HP       1       0
0003.1999-12-10.kaminski        0       0
0012.1999-12-14.farmer  0       0
0004.1999-12-10.kaminski        0       0
0018.2001-07-13.SA_and_HP       1       0
0002.2001-02-07.kitchen 0       0
0007.2004-08-01.BG      1       0
0012.1999-12-14.kaminski        0       0
0005.2001-06-23.SA_and_HP       1       1
0007.1999-12-13.kaminski        0       0
0017.2000-01-17.beck    0       0
0006.2001-06-25.SA_and_HP       1       1
0006.2001-04-03.williams        0       0
0005.2001-02-08.kitchen 0       0
0002.2003-12-18.GP      1       0
0003.2003-12-18.GP      1       0
0013.2001-04-03.williams        0       0
0004.2001-04-02.williams        0       0
0010.2001-02-09.kitchen 0       0
0001.1999-12-10.kaminski        0       0
0013.1999-12-14.farmer  0       0
0015.1999-12-14.kaminski        0       0
0012.2003-12-19.GP      1       1
0016.2001-02-12.kitchen 0       0
0002.2004-08-01.BG      1       0
0002.2001-05-25.SA_and_HP       1       1
0011.2003-12-18.GP      1       1
```

In [13]:

```
#HW 1.5 - Evaluation code
def eval_1_5():
    with open('enronemail_1h.txt.output','rb') as f:
        mr_data=pd.read_csv(f, sep='\t', header=None)
    print "Multinomial NB Results via Poor-Man's MapReduce Implementat
    calculate_training_error(mr_data[1],mr_data[2])

eval_1_5()
```

```
Multinomial NB Results via Poor-Man's MapReduce Implementation
Training error: 0.34
```

HW1.6

Benchmark your code with the Python SciKit-Learn implementation of Naive Bayes

- *Run the **Multinomial Naive Bayes algorithm (using default settings) from SciKit-Learn** over the same training data used in HW1.5 and report the Training error (please note some data preparation might be needed to get the Multinomial Naive Bayes algorithm from SkiKit-Learn to run over this dataset*
- *Run the **Bernoulli Naive Bayes algorithm from SciKit-Learn** (using default settings) over the same training data used in HW1.5 and report the Training error*
- *Run the **Multinomial Naive Bayes algorithm you developed for HW1.5** over the same data used HW1.5 and report the Training error*
- *Please **prepare a table** to present your results*
- *Explain/justify any differences in terms of training error rates over the dataset in HW1.5 **between your Multinomial Naive Bayes implementation (in Map Reduce) versus the Multinomial Naive Bayes implementation in SciKit-Learn***
- *Discuss the performance differences in terms of training error rates over the dataset in HW1.5 **between the Multinomial Naive Bayes implementation in SciKit-Learn with the Bernoulli Naive Bayes implementation in SciKit-Learn***

In [84]:

```
#HW 1.6 - Model comparison code

#Load required packages
from sklearn.naive_bayes import MultinomialNB, BernoulliNB
from sklearn.feature_extraction.text import CountVectorizer
import pandas as pd

def run_1_6():

    #Load data and preprocess for easy scikit-learn use
    with open('enronemail_1h.txt','rb') as f:
        data=pd.read_csv(f, sep='\t', header=None)
    columns=['id','spam','subject','body']
    data.columns=columns #change column headers for easier reference
    data = data.fillna('') #remove nulls
    data['text']=data['subject']+data['body'] #combine subject and bod

    #Break data into vocabulary
    vec=CountVectorizer(analyzer='word')
    vocab=vec.fit_transform(data['text'])

    #Run Sklearn implementation of Multinomial NB
    mnb = MultinomialNB()
    mnb.fit(vocab,data['spam'])
    m_results=mnb.predict(vocab)
    print "Multinomial NB Results via Scikit-Learn Implementation"
    calculate_training_error(m_results,data['spam'])

    #Run Sklearn implementation of Bernoulli NB
    bnb = BernoulliNB()
    bnb.fit(vocab,data['spam'])
    b_results=bnb.predict(vocab)
    print "Bernoulli NB Results via Scikit-Learn Implementation"
    calculate_training_error(b_results,data['spam'])

    #Recalculate training error results for MapReduce implementation i
    with open('enronemail_1h.txt.output','rb') as f:
        mr_data=pd.read_csv(f, sep='\t', header=None)
    print "Multinomial NB Results via Poor-Man's MapReduce Implementat
    calculate_training_error(mr_data[1],mr_data[2])

run_1_6()
```

```
Multinomial NB Results via Scikit-Learn Implementation
Training error: 0.0
Bernoulli NB Results via Scikit-Learn Implementation
Training error: 0.16
Multinomial NB Results via Poor-Man's MapReduce Implementation
Training error: 0.51
```

HW 1.6 - Summary of Results

Model	Training Error
Multinomial NB, Scikit-Learn Implementation	0.0
Bernoulli NB, Scikit-Learn Implementation	0.16
Multinomial NB, MapReduce implementation	0.34
Multinomial NB, MapReduce Implementation (with smoothing, not shown above)	0.0

HW 1.6 - Comparing implementations of Multinomial NB

The scikit-learn version of Multinomial NB does significantly better than our MapReduce implementation. This is because, by default, scikit-learn implements Laplace smoothing (alpha=1.0). Adding smoothing makes a major difference when we have words that do not appear in a class. Instead of simply using the class prior, Laplace smoothing allows us to incorporate these words into our model (albeit with a low class conditional probability).

To confirm, I reran the code for HW 1.5 using the (now commented out) code to implement this smoothing (these results are not shown above for brevity). When I do this, I'm able to reproduce the 0.0 training error generated by the scikit-learn implementation. In either case, it's not surprising that we should see no training error, because we are evaluating our model on the same dataset on which we trained it.

HW 1.6 - Comparing Multinomial NB and Bernoulli NB in Scikit-Learn

When running the different flavors of Naive Bayes in scikit-learn, we see that the Bernoulli implementation has a slightly higher error rate than the Multinomial version, which correctly classifies all the emails. The difference here derives from the assumptions required for each model. In the Bernoulli NB implementation, features are assumed to come from a bernoulli distribution, that is, each feature is assumed to be binary. In contrast, a multinomial NB model assumes features come from a discrete distribution (each feature is a categorical variable, rather than binary). Since our source data is in terms of word counts (in both the MapReduce and Scikit-Learn implementations), we should expect the Multinomial NB to perform better than the Bernoulli version.

End of Submission