261 Homework 1 - Big Data and Naive Bayes

Name: Glenn (Ted) Dunmire

Email: glenn.dunmire.iv@gmail.com

Class: W261-4

Week: 1

Date: 1/19/2016

HW1.0.0. Define big data. Provide an example of a big data problem in your domain of expertise.

Generally, "Big data" is a term for a data set that is so large or so complex that traditional data processing techniques are inadequate. For example, the Twitter firehose provides an enormous volume of data but it is also being generated extremely quickly, capturing the velocity aspect of big data. Typically we would define big data as any set of data that is too large to operate on using a regular laptop with a few GB of memory and 1 TB disk. My domain of expertise is in survey research. Although we do not currently deal with many big data tasks, it is easy to imagine scaling up several of our web surveys. Especially when we use pre-selected internet panels, it would be pretty understandable to see something like the Twitter firehose with large volumes of data responses coming in at a very fast rate.

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?

In plain English, we are establishing how "good" our model is. The mean squared error of the model is usually what we are looking at and can be decomposed into three parts. The MSE is calculated as follows: (NB: predicted means using the model to predict the value, eg f_hat(x))

$$E[(truevalue(x) - predicted(x))^2] = bias^2 + variance + noise$$

The bias is a measure of how much the model prediction would deviate from the true value. We do not know the true function f(x) but we do have the true values for the test dataset. The bias is calculated as:

$$(E[predicted(x)] - truefunction(x))^2$$

However, we do not have the true function. Instead we will approximate it with the true value of the test set so the bias is:

$$(E[predicted(x)] - testvalue(x))^2$$

The variance is a measure of how much the prediction of one training set differs from expected predicted values over different training sets. Essentially this is measuring how consistent predictions are from each other. It is calculated as follows:

$$E[(predicted(x) - E[predicted(x)])^2]$$

Finally, the noise is the irreducible error. This is error that fundamentally cannot be captured in the model, as a result of inherent uncertainty. Essentially, any system will naturally vary and cannot be captured perfectly in a model.

$$Noise = MSE - (bias^2 + variance)$$

In order to calculate these, I would set up functions that enable me to pass a test set and a model which will then produce the bias^2 term and variance. Then I would use a loop to check the different polynomial degrees and get the various errors.

Ex:

for degree in [1, 2, 3, 4, 5]: model = model_function(degree) bias = bias_squared(test, model) variance = variance(test, model) noise = MSE(model) - (bias + variance)

Then for model selection, I would try to select the model that minimizes the combined error of bias squared and variance. However, there is a tradeoff. Usually, as a model becomes more complex, the bias^2 decreases but the variance increases. So I would find the lowest order polynomial with the lowest combined value of bias^2 and variance.

Question 2: Naive Bayes

Question 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.

In [1]:
```
print "Done!"
```

```
Done!
```

NB Certain lines appeared to be incorrect with some text based errors. The ids are listed below. I have edited the text and stored the original, unedited text as a file "enronemail_1h copy.txt"

0011.2001-06-28.SA_and_HP -> incorrect return made two lines

0001.2000-06-06.lokay -> separated text to appropriate place and made subject NA

0009.2001-06-26.SA_and_HP -> moved text to content section and made subject NA

This was manually checked using the following code:

In [4]:
```
with open("enronemail_1h.txt", "r") as myfile:
    for line in myfile.readlines():
        stuff = line.split('\t')
        if len(stuff) != 4:
            print stuff[0]
            print len(stuff)
```

Question 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, and
- reducer.py collates the counts of the single word.

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

```
8 (should really be 10 if counting occurrences)
```

In [1]:
```
%%writefile mapper.py
#!/usr/bin/python
import sys
import re
count = 0 #keep track of count of found word
WORD_RE = re.compile(r"[\w']+") #regex for real words
filename = sys.argv[1] #parse file name
findword = sys.argv[2] #parse findword, the word we're looking for
with open (filename, "r") as myfile: #read file
    for line in myfile.readlines():
        words = re.findall(WORD_RE, line) #grab all words from the
text
        occ = [i for i,x in enumerate(words) if x.lower() == findwo
rd.lower()] #create list based on logical match
        count += len(occ) #increment count
print count #print count to temporary file
```

```
Overwriting mapper.py
```

In [2]:
```
#Change permissions on mapper
!chmod a+x mapper.py
```

In [3]:
```
%%writefile reducer.py
#!/usr/bin/python
import sys

files = sys.argv[1:] #read file
sum = 0 #total sum
for filename in files:
    with open (filename, "r") as myfile:
        for line in myfile.readlines():
            sum = sum + int(line) #increment sum by occurrence of c
ount
print sum
```

```
Overwriting reducer.py
```

In [4]:
```
#change permissions on reducer
!chmod a+x reducer.py
```

Code below provided in assignment document

In [1]:
```
%%writefile pNaiveBayes.sh
## 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 com
mand,
##               and focus on how arguments are supplied to mappe
r.py/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
starter 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 processe
s
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 S
TDOUT to disk
####
####
./reducer.py $countfiles > $data.output
####
####

## clean up the data chunks and temporary count files
#\rm $data.chunk.*
\rm enronemail_1h.txt.chunk.*
```

```
Overwriting pNaiveBayes.sh
```

In [2]:
```
!chmod a+x pNaiveBayes.sh
```

In [7]:
```
!./pNaiveBayes.sh 4 "assistance"
! echo "Number of times word appears: "
! head enronemail_1h.txt.output
```

```
Number of times word appears:
10
```

Verified that checking for assistance correctly prints out the 10 times the word assistance occurs

Question 3

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

  that performs a single word Naive Bayes classification. For multinomial Naive Bayes, the Pr(X=“assistance”|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 labeld as SPAM (when concatenated) is 1,000. Then Pr(X=“assistance”|Y=SPAM) = 5/1000. Note this is a multinomial estimated of the class conditional for a Naive Bayes Classifier. No smoothing is needed in this HW.

In [3]:
```
%%writefile mapper.py
#!/usr/bin/python
import sys
import re


WORD_RE = re.compile(r"[\w']+") #regex for matching to word
filename = sys.argv[1] #grab filename from command line input
findword = sys.argv[2] #get findword in question from command line
input
with open (filename, "r") as myfile:
    for line in myfile.readlines(): #read all lines in file and rep
eat mapping
        components = line.split('\t') #separate line into ID + spam
flag + subject + text
        ID = components[0]
        flag = components[1]
        text = components[2] + components[3] #combine subject and c
ontent into one string
        word_count = {} #dictionary to hold the number of times eac
h word is found. The word is the key, count is value
        for word in WORD_RE.findall(text): #check all words in the
text
            if word in word_count:
                word_count[word] += 1 #increment value by 1
            else:
                word_count[word] = 1 #set word to key in dictionary
and give value of 1
        for word, count in word_count.iteritems(): #scan through di
ctionary for all words and counts
            found_word = 0 #flag for if word is the findword
            if word.lower() == findword.lower(): #check and set fla
g if the word is the findword
                found_word = 1
            #map the following string to the output files
            #ID + spam flag + word + count of word + logical flag i
f word is foundword (1 = True, 0 False)
            print ID + '\t' + str(flag) + '\t' + word + '\t' + st
r(count) + '\t' + str(found_word)
```

```
Overwriting mapper.py
```

In [4]:
```
#change permission for mapper
!chmod a+x mapper.py
```

In [5]:
```
%%writefile reducer.py
#!/usr/bin/python
import sys
import re
from math import log

#in this script I will be using a lot of sets. The point here is to
prevent duplicates. Ex. the email
#IDs occur multiple times because we are parsing by the individual
word (see mapper above). However, we are only
#interested in the ID once, therefore we will use a set to hold it.
emailID = set() #set to hold unique email IDs
spam = set() #set to hold unique email IDs if spam
vocab = set() #set to hold all unique words in text
spamCount = 0 #how many words are in the spam documents in total
spamWordCount = 0 #how many times the findword occurs in the spam t
ext
hamCount= 0 #how many words appear in all ham documents
hamWordCount = 0 #how many times does the findword appear in the ha
m text
foundWord = set() #set of unique findwords (adapted from 1.4)

files = sys.argv[1:]
for filename in files:
    with open(filename, 'r') as myfile:
        for line in myfile.readlines():
            components = line.split('\t') #split file into ID + spa
m flag + word + word count + find word logical flag
            ID = components[0]
            flag = int(components[1])
            word = components[2]
            count = int(components[3])
            findword = int(components[4])
            if findword == 1: #if the findword flag is present, add
the word to the set of foundwords
                foundWord.add(word)
            emailID.add(ID) #add the email's ID to the set of IDs.
            if flag == 1: #if the mapped word came from a spam emai
l
                spam.add(ID) #add ID to spam
                spamCount += count #increase spam word count by cou
nt of this word
                if findword == 1:
                    spamWordCount += count #increase count of findw
ord if necessary
            else: #repeat above procedure for ham
                hamCount += count
                if findword == 1:
                    hamWordCount += count


#Calculate prior probability for spam and ham
priorSpam = float(len(spam))/len(emailID)
```

```
priorHam = float(len(emailID) - len(spam))/len(emailID)

#conditional probability in this case, as given by instructions in
the homework text
#Conditional probability of a document being a class given a word i
s the
#(count of the word in the class)/(total words in class)
condProbSpam = float(spamWordCount)/spamCount
condProbHam = float(hamWordCount)/hamCount

#display priors and conditional probabilities
print "Prior Spam: " + str(priorSpam)
print "Cond prob of spam " + str(condProbSpam)
print "Prior Ham " + str(priorHam)
print "Cond prob of Ham " + str(condProbHam)

#classify new emails
#in this case I decided to read the original file, rather than reas
sembling the emails from the reducer step.
#This increases modularity because in practice we would report our
results on a validation or test set as opposed
#to the training set.
#Note also I am setting a condition that if a word does not appear
in the text of a class, ex a word does not appear
#in ham at all, then the probability does not get updated. Normally
a smoother would take care of this but the
#instructions say not use a smoother for this problem.
WORD_RE = re.compile(r"[\w']+")
with open('enronemail_1h.txt', 'r') as myfile:
    for line in myfile.readlines():
        components = line.split('\t')
        ID = components[0]
        trueLabel = components[1]
        text = components[2] + ' ' + components[3] #combine email s
ubject and content
        spamScore = log(priorSpam) #take log
        hamScore = log(priorHam)
        for word in WORD_RE.findall(text):
            if word in foundWord and condProbSpam != 0: #check if f
indword and if word has appeared in spam
                spamScore += log(condProbSpam) #increment by log
            if word in foundWord and condProbHam != 0: #ditto for h
am
                hamScore += log(condProbHam)
        predicted = 0
        if spamScore > hamScore: #classify as spam only if spamscor
e is higher than hamscore
            predicted = 1
        print ID + '\t' + str(trueLabel) + '\t' + str(predicted) #p
rint ID + true flag + predicted flag
```

```
Overwriting reducer.py
```

In [6]:
```
#Change permissions on reducer
!chmod a+x reducer.py
```

In [7]:
```
!./pNaiveBayes.sh 4 "assistance"
```

In [8]:
```
with open('enronemail_1h.txt.output', 'r') as myfile:
    count = 0 #count of how many emails are misclassified
    for line in myfile.readlines():
        print line
        components = line.split('\t') #display contents ID + spam f
lag + predicted flag (0 = ham, 1 = spam)
        if len(components) > 2:
            if int(components[1]) != int(components[2]):
                count += 1
    print "Misclassified " + str(count)
```

```
Prior Spam: 0.44

Cond prob of spam 0.000428036383093

Prior Ham 0.56

Cond prob of Ham 0.000140637086

0001.1999-12-10.farmer  0       0

0001.1999-12-10.kaminski        0       0

0001.2000-01-17.beck    0       0

0001.2000-06-06.lokay   0       0

0001.2001-02-07.kitchen 0       0

0001.2001-04-02.williams        0       0

0002.1999-12-13.farmer  0       0

0002.2001-02-07.kitchen 0       0

0002.2001-05-25.SA_and_HP       1       0

0002.2003-12-18.GP      1       0

0002.2004-08-01.BG      1       1

0003.1999-12-10.kaminski        0       0

0003.1999-12-14.farmer  0       0

0003.2000-01-17.beck    0       0

0003.2001-02-08.kitchen 0       0

0003.2003-12-18.GP      1       0

0003.2004-08-01.BG      1       0

0004.1999-12-10.kaminski        0       1

0004.1999-12-14.farmer  0       0

0004.2001-04-02.williams        0       0

0004.2001-06-12.SA_and_HP       1       0

0004.2004-08-01.BG      1       0

0005.1999-12-12.kaminski        0       1
```

```
0005.1999-12-14.farmer  0       0

0005.2000-06-06.lokay   0       0

0005.2001-02-08.kitchen 0       0

0005.2001-06-23.SA_and_HP       1       0

0005.2003-12-18.GP      1       0

0006.1999-12-13.kaminski        0       0

0006.2001-02-08.kitchen 0       0

0006.2001-04-03.williams        0       0

0006.2001-06-25.SA_and_HP       1       0

0006.2003-12-18.GP      1       0

0006.2004-08-01.BG      1       0

0007.1999-12-13.kaminski        0       0

0007.1999-12-14.farmer  0       0

0007.2000-01-17.beck    0       0

0007.2001-02-09.kitchen 0       0

0007.2003-12-18.GP      1       0

0007.2004-08-01.BG      1       0

0008.2001-02-09.kitchen 0       0

0008.2001-06-12.SA_and_HP       1       0

0008.2001-06-25.SA_and_HP       1       0

0008.2003-12-18.GP      1       0

0008.2004-08-01.BG      1       0

0009.1999-12-13.kaminski        0       0

0009.1999-12-14.farmer  0       0

0009.2000-06-07.lokay   0       0

0009.2001-02-09.kitchen 0       0

0009.2001-06-26.SA_and_HP       1       0
```

```
0009.2003-12-18.GP      1       0

0010.1999-12-14.farmer  0       0

0010.1999-12-14.kaminski        0       0

0010.2001-02-09.kitchen 0       0

0010.2001-06-28.SA_and_HP       1       1

0010.2003-12-18.GP      1       0

0010.2004-08-01.BG      1       0

0011.1999-12-14.farmer  0       0

0011.2001-06-28.SA_and_HP       1       1

0011.2001-06-29.SA_and_HP       1       0

0011.2003-12-18.GP      1       0

0011.2004-08-01.BG      1       0

0012.1999-12-14.farmer  0       0

0012.1999-12-14.kaminski        0       0

0012.2000-01-17.beck    0       0

0012.2000-06-08.lokay   0       0

0012.2001-02-09.kitchen 0       0

0012.2003-12-19.GP      1       0

0013.1999-12-14.farmer  0       0

0013.1999-12-14.kaminski        0       0

0013.2001-04-03.williams        0       0

0013.2001-06-30.SA_and_HP       1       0

0013.2004-08-01.BG      1       1

0014.1999-12-14.kaminski        0       0

0014.1999-12-15.farmer  0       0

0014.2001-02-12.kitchen 0       0

0014.2001-07-04.SA_and_HP       1       0
```

```
0014.2003-12-19.GP      1       0

0014.2004-08-01.BG      1       0

0015.1999-12-14.kaminski        0       0

0015.1999-12-15.farmer  0       0

0015.2000-06-09.lokay   0       0

0015.2001-02-12.kitchen 0       0

0015.2001-07-05.SA_and_HP       1       0

0015.2003-12-19.GP      1       0

0016.1999-12-15.farmer  0       0

0016.2001-02-12.kitchen 0       0

0016.2001-07-05.SA_and_HP       1       0

0016.2001-07-06.SA_and_HP       1       0

0016.2003-12-19.GP      1       0

0016.2004-08-01.BG      1       0

0017.1999-12-14.kaminski        0       0

0017.2000-01-17.beck    0       0

0017.2001-04-03.williams        0       0

0017.2003-12-18.GP      1       0

0017.2004-08-01.BG      1       0

0017.2004-08-02.BG      1       0

0018.1999-12-14.kaminski        0       0

0018.2001-07-13.SA_and_HP       1       1

0018.2003-12-18.GP      1       1

Misclassified 40
```

Notice that we're getting 60% accuracy here. That is, 40 emails are misclassified while 60 are classified correctly. A lot of the emails are actually spam and are being classified as ham. This is due to the fact that the prior probability of an email being ham is quite a bit larger than the prior probability of an email being spam. So despite the conditional probability of "assistance" being higher for the class spam, in a lot of cases there aren't enough occurrences to classify as spam.

Question 4

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 multinomial Naive Bayes classification via the chosen list. No smoothing is needed in this HW.

In [9]:
```
%%writefile mapper.py
#!/usr/bin/python
import sys
import re


WORD_RE = re.compile(r"[\w']+") #regex for word
filename = sys.argv[1] #get file name
findwords = re.split(" ",sys.argv[2].lower()) #parse input string o
f words into list
with open (filename, "r") as myfile:
    for line in myfile.readlines():
        components = line.split('\t') #split line into components I
D + flag + text
        ID = components[0]
        flag = components[1]
        text = components[2] + components[3] #combine subject and c
ontent into one text list
        word_count = {} #keep track of all words and their counts
        for word in WORD_RE.findall(text):
            if word in word_count:
                word_count[word] += 1
            else:
                word_count[word] = 1
        for word, count in word_count.iteritems():
            found_word = 0
            if word.lower() in findwords: #check if each word is in
the findword list
                found_word = 1
            #print a string with ID + spam flag + word + count + if
word is a findword
            print ID + '\t' + str(flag) + '\t' + word + '\t' + st
r(count) + '\t' + str(found_word)
```

```
Overwriting mapper.py
```

In [10]:
```
#Overwrite mapper.py
!chmod a+x mapper.py
```

In [18]:
```
%%writefile reducer.py
#!/usr/bin/python
import sys
import re
from math import log

emailID = set() #set to hold email IDs
spam = set() #set to hold spam IDs
vocab = set() #set to hold vocab of unique words in corpus of all t
ext
spamCount = 0 #count of how many words in spam documents
spamWordCount = {} #dictionary for findwords in spam documents
hamCount= 0 #count of how many words in ham documents
hamWordCount = {} #dictionary for findwords in ham documents
foundWord = set() #set for findwords

files = sys.argv[1:] #read file name
for filename in files:
    with open(filename, 'r') as myfile:
        for line in myfile.readlines():
            components = line.split('\t') #split line by tab and as
sign components
            ID = components[0]
            flag = int(components[1])
            word = components[2]
            count = int(components[3])
            findword = int(components[4])
            if findword == 1:
                foundWord.add(word) #add findword to set of existin
g foundwords
            emailID.add(ID)
            if flag == 1: #if email is spam
                spam.add(ID) #add ID to set
                spamCount += count #increment counter
                if findword == 1: #if findword present add to dicti
onary or increment counter
                    if word not in spamWordCount:
                        spamWordCount[word] = count
                    else:
                        spamWordCount[word] += count
            else: #repeat above steps for ham
                hamCount += count
                if findword == 1:
                    if word not in hamWordCount:
                        hamWordCount[word] = count
                    else:
                        hamWordCount[word] += count


#Calculate priors for spam and ham
priorSpam = float(len(spam))/len(emailID)
priorHam = float(len(emailID) - len(spam))/len(emailID)
```

```
#dictionary to hold conditional probabilities for spam and ham word
s
condProbSpam = {}
condProbHam = {}

#assign conditional probabilities for each word in the dictionaries
for spam and ham respectively
#probability is (count of word in class)/(total words in class)
for word in foundWord:
    if word in spamWordCount:
        condProbSpam[word] = float(spamWordCount[word])/spamCount
    if word in hamWordCount:
        condProbHam[word] = float(hamWordCount[word])/hamCount

#print priors of spam and ham and conditional probabilities
print "Prior Spam: " + str(priorSpam)
print "Cond prob of spam " + str(condProbSpam)
print "Prior Ham " + str(priorHam)
print "Cond prob of Ham " + str(condProbHam)

#classify new emails
#in this case I decided to read the original file, rather than reas
sembling the emails from the reducer step.
#This increases modularity because in practice we would report our
results on a validation or test set as opposed
#to the training set.
#Note also I am setting a condition that if a word does not appear
in the text of a class, ex a word does not appear
#in ham at all, then the probability does not get updated. Normally
a smoother would take care of this but the
#instructions say not use a smoother for this problem.
WORD_RE = re.compile(r"[\w']+")
with open('enronemail_1h.txt', 'r') as myfile:
    for line in myfile.readlines():
        components = line.split('\t') #split lines
        ID = components[0] #assign components
        trueLabel = components[1]
        text = components[2] + ' ' + components[3] #combine subject
and content into one text blob
        spamScore = log(priorSpam)
        hamScore = log(priorHam)
        for word in WORD_RE.findall(text): #increase score for each
word found that is present in a class
            if word in foundWord and word in condProbSpam:
                spamScore += log(condProbSpam[word])
            if word in foundWord and word in condProbHam:
                hamScore += log(condProbHam[word])
        predicted = 0 #assign class based on score
        if spamScore > hamScore:
            predicted = 1
        print ID + '\t' + str(trueLabel) + '\t' + str(predicted) #p
rint word + spam flag + predicted class
```

```
Overwriting reducer.py
```

In [19]:
```
#overwrite reducer.py
!chmod a+x reducer.py
```

In [20]:
```
!./pNaiveBayes.sh 4 'assistance valium enlargementWithATypo'
```

In [21]:
```
#examine results
with open('enronemail_1h.txt.output', 'r') as myfile:
    count = 0 #count of how many emails are misclassified
    for line in myfile.readlines():
        print line
        components = line.split('\t') #display contents ID + spam f
lag + predicted flag (0 = ham, 1 = spam)
        if len(components) > 2:
            if int(components[1]) != int(components[2]): #increment
counter if true and predicted class don't match
                count += 1
    print "Misclassified " + str(count)
```

```
Prior Spam: 0.44

Cond prob of spam {'assistance': 0.00042803638309256285, 'valium':
0.00016051364365971107}

Prior Ham 0.56

Cond prob of Ham {'assistance': 0.00014063708599957808}

0001.1999-12-10.farmer  0       0

0001.1999-12-10.kaminski        0       0

0001.2000-01-17.beck    0       0

0001.2000-06-06.lokay   0       0

0001.2001-02-07.kitchen 0       0

0001.2001-04-02.williams        0       0

0002.1999-12-13.farmer  0       0

0002.2001-02-07.kitchen 0       0

0002.2001-05-25.SA_and_HP       1       0

0002.2003-12-18.GP      1       0

0002.2004-08-01.BG      1       1

0003.1999-12-10.kaminski        0       0

0003.1999-12-14.farmer  0       0

0003.2000-01-17.beck    0       0

0003.2001-02-08.kitchen 0       0

0003.2003-12-18.GP      1       0

0003.2004-08-01.BG      1       0

0004.1999-12-10.kaminski        0       1

0004.1999-12-14.farmer  0       0

0004.2001-04-02.williams        0       0

0004.2001-06-12.SA_and_HP       1       0

0004.2004-08-01.BG      1       0
```

```
0005.1999-12-12.kaminski        0       1
0005.1999-12-14.farmer  0       0
0005.2000-06-06.lokay   0       0
0005.2001-02-08.kitchen 0       0
0005.2001-06-23.SA_and_HP       1       0
0005.2003-12-18.GP      1       0
0006.1999-12-13.kaminski        0       0
0006.2001-02-08.kitchen 0       0
0006.2001-04-03.williams        0       0
0006.2001-06-25.SA_and_HP       1       0
0006.2003-12-18.GP      1       0
0006.2004-08-01.BG      1       0
0007.1999-12-13.kaminski        0       0
0007.1999-12-14.farmer  0       0
0007.2000-01-17.beck    0       0
0007.2001-02-09.kitchen 0       0
0007.2003-12-18.GP      1       0
0007.2004-08-01.BG      1       0
0008.2001-02-09.kitchen 0       0
0008.2001-06-12.SA_and_HP       1       0
0008.2001-06-25.SA_and_HP       1       0
0008.2003-12-18.GP      1       0
0008.2004-08-01.BG      1       0
0009.1999-12-13.kaminski        0       0
0009.1999-12-14.farmer  0       0
0009.2000-06-07.lokay   0       0
0009.2001-02-09.kitchen 0       0
```

```
0009.2001-06-26.SA_and_HP       1       0
0009.2003-12-18.GP      1       0
0010.1999-12-14.farmer  0       0
0010.1999-12-14.kaminski        0       0
0010.2001-02-09.kitchen 0       0
0010.2001-06-28.SA_and_HP       1       1
0010.2003-12-18.GP      1       0
0010.2004-08-01.BG      1       0
0011.1999-12-14.farmer  0       0
0011.2001-06-28.SA_and_HP       1       1
0011.2001-06-29.SA_and_HP       1       0
0011.2003-12-18.GP      1       0
0011.2004-08-01.BG      1       0
0012.1999-12-14.farmer  0       0
0012.1999-12-14.kaminski        0       0
0012.2000-01-17.beck    0       0
0012.2000-06-08.lokay   0       0
0012.2001-02-09.kitchen 0       0
0012.2003-12-19.GP      1       0
0013.1999-12-14.farmer  0       0
0013.1999-12-14.kaminski        0       0
0013.2001-04-03.williams        0       0
0013.2001-06-30.SA_and_HP       1       0
0013.2004-08-01.BG      1       1
0014.1999-12-14.kaminski        0       0
0014.1999-12-15.farmer  0       0
0014.2001-02-12.kitchen 0       0
```

```
0014.2001-07-04.SA_and_HP       1       0

0014.2003-12-19.GP      1       0

0014.2004-08-01.BG      1       0

0015.1999-12-14.kaminski        0       0

0015.1999-12-15.farmer  0       0

0015.2000-06-09.lokay   0       0

0015.2001-02-12.kitchen 0       0

0015.2001-07-05.SA_and_HP       1       0

0015.2003-12-19.GP      1       0

0016.1999-12-15.farmer  0       0

0016.2001-02-12.kitchen 0       0

0016.2001-07-05.SA_and_HP       1       0

0016.2001-07-06.SA_and_HP       1       0

0016.2003-12-19.GP      1       0

0016.2004-08-01.BG      1       0

0017.1999-12-14.kaminski        0       0

0017.2000-01-17.beck    0       0

0017.2001-04-03.williams        0       0

0017.2003-12-18.GP      1       0

0017.2004-08-01.BG      1       0

0017.2004-08-02.BG      1       0

0018.1999-12-14.kaminski        0       0

0018.2001-07-13.SA_and_HP       1       1

0018.2003-12-18.GP      1       1

Misclassified 40
```

Notice here we have the same thing as above. An accuracy of 60%, meaning we misclassified 40 emails. This is probably due to the same reasons above, the large prior probability of ham and the relatively small conditional probabilities. Also valium only appears in the spam case so it will only add to spam.

Question 5 - Had previously done before revised homework

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.

In [3]:
```
%%writefile mapper.py
#!/usr/bin/python
import sys
import re


WORD_RE = re.compile(r"[\w']+")
filename = sys.argv[1]
with open (filename, "r") as myfile:
    for line in myfile.readlines():
        components = line.split('\t')
        ID = components[0]
        flag = components[1]
        text = components[2] + components[3]
        word_count = {}
        for word in WORD_RE.findall(text):
            if word in word_count:
                word_count[word] += 1
            else:
                word_count[word] = 1
        for word, count in word_count.iteritems():
            print ID + '\t' + str(flag) + '\t' + word + '\t' + st
r(count)
```

```
Overwriting mapper.py
```

In [4]:
```
#change permissions on mapper
!chmod a+x mapper.py
```

In [51]:
```
%%writefile reducer.py
#!/usr/bin/python
import sys
import re
from math import log

emails = set()
spams = set()
words_in_corpus = set()
spam_counts = {}
ham_counts = {}

files = sys.argv[1:]
for filename in files:
    with open (filename, "r") as myfile:
        for line in myfile.readlines():
            components = line.split('\t')
            ID = components[0]
            flag = int(components[1])
            word = components[2]
            count = int(components[3])
            emails.add(ID)
            words_in_corpus.add(word)
            if flag == 1:
                spams.add(ID)
                if word not in spam_counts:
                    spam_counts[word] = count
                else:
                    spam_counts[word] += count
            else:
                if word not in ham_counts:
                    ham_counts[word] = count
                else:
                    ham_counts[word] += count

priorSpam = float(len(spams))/len(emails)
priorHam = float(len(emails) - len(spams))/len(emails)

condProbSpam = {}
condProbHam = {}

for word in words_in_corpus:
    if word in spam_counts:
        spam_tokens = spam_counts[word] + 1
    else:
        spam_tokens = 1
    condProbSpam[word] = float(spam_tokens)/(sum(spam_counts.value
s()) + len(words_in_corpus))
    if word in ham_counts:
        ham_tokens = ham_counts[word] + 1
    else:
        ham_tokens = 1
    condProbHam[word] = float(ham_tokens)/(sum(ham_counts.values())
```

```
+ len(words_in_corpus))

#classify new emails
#in this case I decided to read the original file, rather than reas
sembling the emails from the reducer step.
#This increases modularity because in practice we would report our
results on a validation or test set as opposed
#to the training set.
WORD_RE = re.compile(r"[\w']+")
with open('enronemail_1h.txt', 'r') as myfile:
    for line in myfile.readlines():
        components = line.split('\t')
        ID = components[0]
        trueLabel = components[1]
        text = components[2] + ' ' + components[3]
        spamScore = log(priorSpam)
        hamScore = log(priorHam)
        for word in WORD_RE.findall(text):
            if word in condProbSpam:
                spamScore += log(condProbSpam[word])
            if word in condProbHam:
                hamScore += log(condProbHam[word])
        predicted = 0
        if spamScore > hamScore:
            predicted = 1
        print ID + '\t' + str(trueLabel) + '\t' + str(predicted)
```

```
Overwriting reducer.py
```

In [52]:
```
#change permissions on reducer
!chmod a+x reducer.py
```

In [53]:
```
!./pNaiveBayes.sh 4
```

In [57]:
```
with open('enronemail_1h.txt.output', 'r') as myfile:
    count = 0
    for line in myfile.readlines():
        components = line.split('\t')
        true = int(components[1])
        pred = int(components[2])
        count += (true - pred)
    print count
```

```
0
```

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


WORD_RE = re.compile(r"[\w']+")
filename = sys.argv[1]
with open (filename, "r") as myfile:
    for line in myfile.readlines():
        components = line.split('\t')
        ID = components[0]
        flag = components[1]
        text = components[2] + components[3]
        word_count = {}
        for word in WORD_RE.findall(text):
            if word in word_count:
                word_count[word] += 1
            else:
                word_count[word] = 1
        for word, count in word_count.iteritems():
            print ID + '\t' + str(flag) + '\t' + word + '\t' + st
r(count)
```

Question 6 - Had not done when received revised homework

In [ ]:
```
```