```
# MIDS UC Berkeley - Machine Learning at Scale
## DATSCIW261 ASSIGNMENT #1

[James Gray](https://github.com/jamesgray007)
jamesgray@ischool.berkeley.edu
Time of Initial Submission: 3:30 PM Central, Monday, May 16, 2016
Time of **Resubmission**: 9:30 PM Central, Wednesday, May 18, 2016
(added HW1.0.1)
W261-3, Spring 2016
Week 1 Homework
```

References for this Assignment

- **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)

HW 0.0 Bio

Prepare your bio and include it in this HW submission. Please limit to 100 words. Count the words in your bio and print the length of your bio (in terms of words) in a separate cell.

In [70]:
```
bio = "Over the last five months I have worked as a data scientist in t
focused on optimizing customer support experiences.  Prior to that I le
platform and analytics across the sales, marketing, customer support an
I relocated to Austin about a year ago after residing in Seattle for 15
BS in Electrical Engineering from Union College.  My goal for W261 is t
experience both in theory and hands-on programming."

wordcount = len(bio.split())
print ("Bio wordcount = " + str(wordcount))
```

```
Bio wordcount = 97
```

HW1.0.0. (Graded)

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

```
Conceptually, Big Data is defined as data that are so large, complex
and high velocity that traditional techniques are not adequate to
process and deliver insights. Big Data is often described by three or
four V's.  **Volume** often distinquishes Big Data as datasets that
reach the petabyte (10^15) or zettbyte (10^21) scale when compared to
traditional data processing applications in the terabytes (10^12).
Data these large would not fit on a typical high performance laptop
with 1TB disk storage. Processing and reading even 1TB data on a
laptop would take approximately 3 hours which is not satisfactory
given the need to delivery timely insights. **Variety** includes both
structured and unstructed data such as video, text, JSON, images
unlike traditional systems (relational databases) that handle well-
formed data in rows and columns. **Velocity** includes the ability of
Big Data solutions to handle very fast streaming data from sources
such as the social web or machine-to-machine scenarios in the
[Internet of Things]
(https://en.wikipedia.org/wiki/Internet_of_Things) realm. A typical
laptop with even large memory would not be able to handle and persist
the velocity of these data flows.  Traditional data processing
systems such as a data warehouses operate in batch mode that
generally refresh 1-2 times a day. **Veracity** is the uncertainty of
data and ability to manage and enforce data quality.  All of these
attributes make traditional data processing systems such as databases
completely inadequate.

![bigdata](img/bigdata.png)

An example of a Big Data scenario at Microsoft is the collection of
telemetry from hundreds of millions of Windows devices across the
world on a daily basis. This results in high velocity of incoming
data from user-driven actions (100's of millions of users) and
application health monitoring events. A large distributed system is
required to store and process the petabyte scale dataset that is
collected each day.

References:
```

HW1.0.1. (Graded)

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?

Model selection is guided by the model with the lowest overall prediction error. Prediction error is defined as the squared difference between the observed (true) value and the model prediction. Through mathematical decomposition of this error (The elements of statistical learning: Data mining, inference and prediction - Chapter 7) we see that it's composed of *reproducible error* (squared bias, variance) and *irreproducible error* that is inherent to the natural variability of the system. We will use regression to explain how to estimate bias, variance and irreducible error when using polynomial models.

Expected prediction error = squared estimator bias + estimator variance + noise

- **estimator variance** = the error by which the prediction over one training set differs from the expected (average) predictor over the training data.
- **squared estimator bias** = the error by which the expected model prediction (average) differs from the observed value over the training data.
- **noise** = the variance by how much the observations vary from the true function. Note that in the real world this cannot be measured as we do not know the true function.

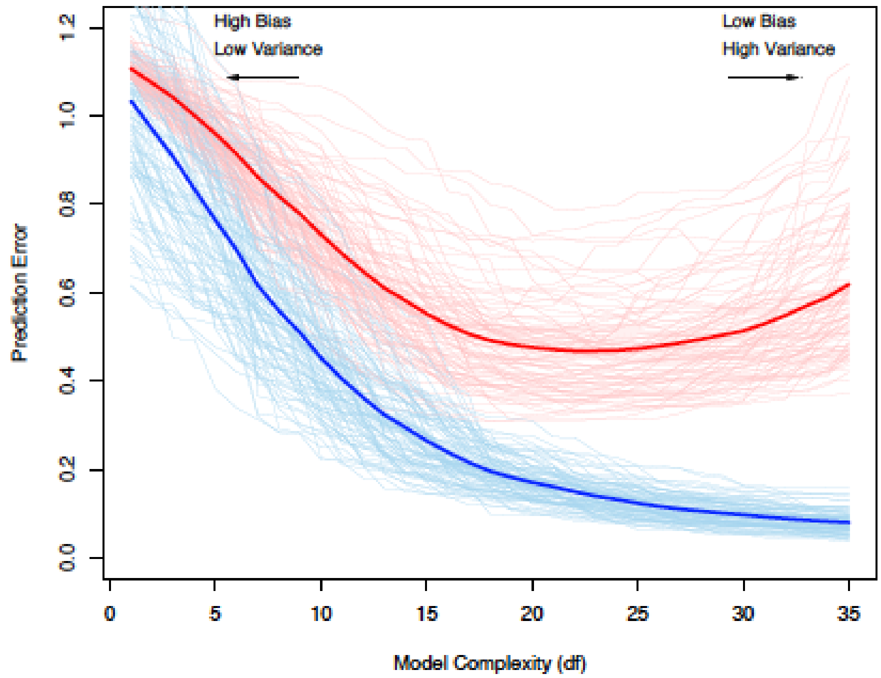
For a given estimator $g(x)$ that approximates the true function $y = f(x)$, and new data pair (x, y) we represent error as follows:

$$E[(g(x) - y)^2) = E[g(x) - y_{true}]^2 + E[g[(x) - E[g(x)]^2] + E[(y - y_{true})^2]$$

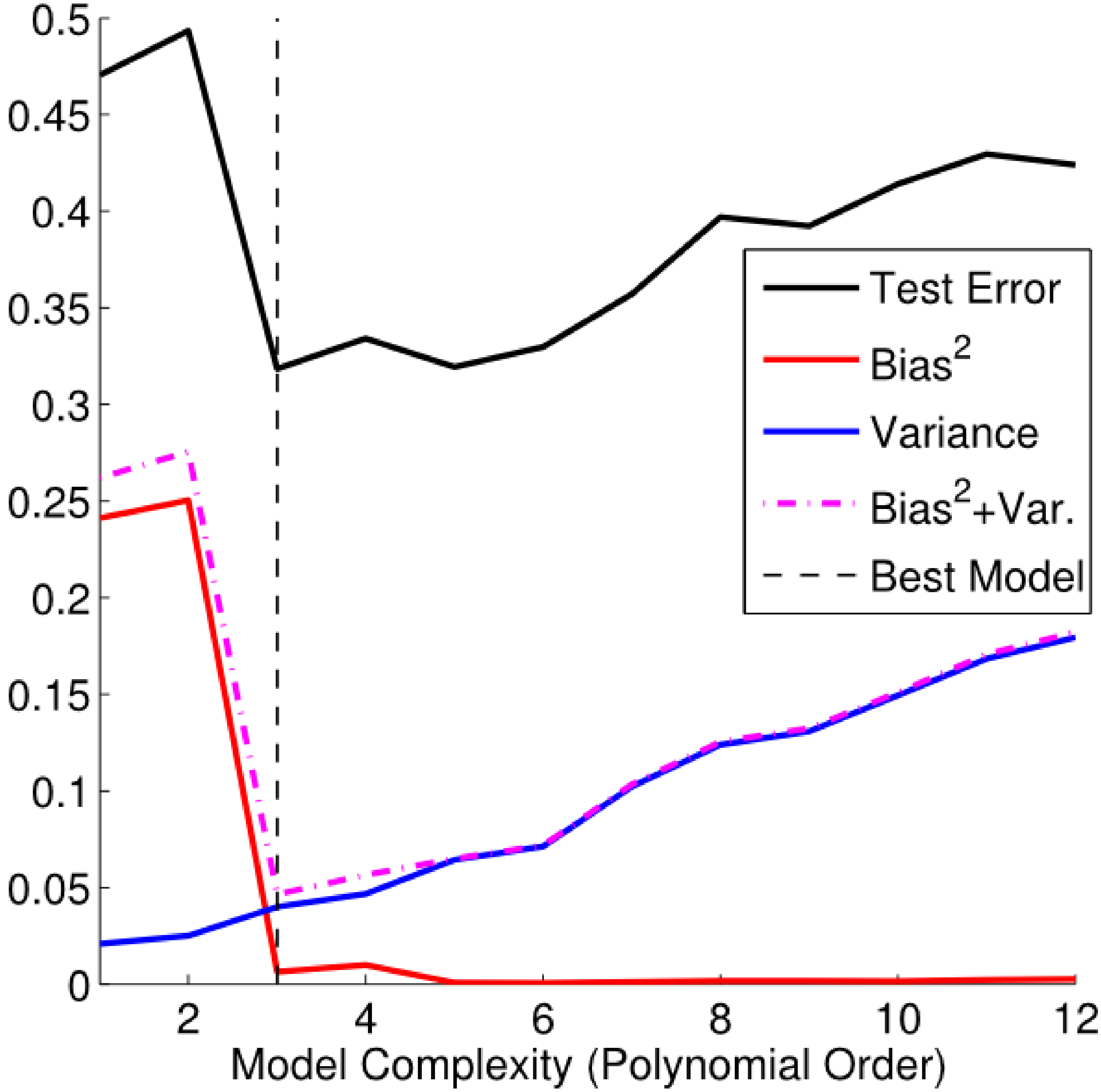
$$Error = Bias^2 + Variance + Noise$$

Ultimately we would choose the model that minimizes bias and variance. In the real world we do not know the true function from which the dataset T was derived. Therefore we must calculate the Expected Error through bootstrap sampling of the training data. To properly evaluate model performance we need to repeat the model fitting process using multiple (N) bootstrap samples for each degree of the polynomial models. Test data from the bootstrap sample is then used to make predictions. These predictions are then used to calculate the bias-squared and variance terms for each hyperparameter model setting (polynomial degree 1-5). The "noise" or irreducible error cannot be calculated in practice since we do not know the true function $f(x)$. We would then select the model with the lowest combined variance and bias-squared and choose the simpler model if multiple models produce the same overall error.

The figure below shows the relationship between prediction error and model complexity and its impact on both bias and variance. We see that the average training error (blue line) is reduced as higher order polynomials are able to fit the data more precisely resulting in low bias. The penalty is that these models do not generalize well (overfitting) to new data resulting in high variance (red line). Conversely, simple models do not fit the training data closely resulting in high bias. These less flexible models result in low variance. The ideal model is somewhere in between these two extremes where there is low bias and low variance.



The figure below represents the tradeoff between bias and variance for models with increasing complexity. In this example, the best estimator is the model that produces the lowest error to test data (black line) when the model is a third order polynomial. This also corresponds to the dashed magenta line where the squared bias and variance is at its lowest point.



We can also represent this process in pseudocode for models of degree 1,2,3,4, shown below.

References used:

- Model Selection: Underfitting, Overfitting and the Bias-Variance Tradeoff (https://theclevermachine.wordpress.com/tag/bias-variance-tradeoff/)
- Ask a Data Scientist: The Bias vs. Variance Tradeoff (http://insidebigdata.com/2014/10/22/ask-data-scientist-bias-vs-variance-tradeoff/)
- The elements of statistical learning: Data mining, inference and prediction (Chapter 7)

In [1]:
```
# Real world where we do know the true function f(x) that produces y_tr

b = 50 # number of bootstamp samples
# x_train, y_train -> bootstrap training data
# x_test, y_train -> bootstrap testing data
# y_hat -> predicted output value from test data
# y_bar -> average predicted value
# error -> holds the overall prediction error for each model hyperparam

# iterate through the degrees of polymnomial regression models
model in models:
    # execute model fitting and predictions for each boostrap sample
    for sample in range(b):
        # generate training and test data from the the base data set
        x_train, y_train, x_test, y_test
        # train model using the bootstramp training data
        mymodel = model.fit(x_train, y_train)
        # make predictions using the test data
        y_hat = mymodel.predict(x_test)
        variance_sum += (y_bar - y_hat)^2
    # calculate metrics for each model
    y_bar = calculate average prediction across the b iterations
    bias2 = (y_bar-y_true)^2
    variance = variance_sum/b
    noise = 0 # set to 0 since we cannot estimate it or control it.
    error[model] = bias2 + variance + noise

# model selection
select the model with smallest error
```

Instructions for Spam Filter using Naive Bayes Classifier

```
In the remainder of this assignment you will produce a spam filter
that is backed by a multinomial naive Bayes classifier b (see
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.

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).

In this directory you will also find starter code (pNaiveBayes.sh),
(similar to the pGrepCount.sh code that was presented in this weeks
lectures), which will be used as control script to a python mapper
and reducer that you will supply at several stages. Doing some
exploratory data analysis you will see (with this very small dataset)
the following
```

Exploratory Data Analysis of Enron Email corpus

In [71]:
```
! wc -l enronemail_1h.txt  # count the number of lines in the subset of
```

```
      99 enronemail_1h.txt
```

In [72]:
```
! cut -f2 -d$'\t' enronemail_1h.txt|wc  #extract second field which is
```

```
     100     100     200
```

In [73]:
```
! cut -f2 -d$'\t' enronemail_1h.txt|head
```

```
0
0
0
0
0
0
0
0
1
1
```

In [ ]:
```
! head -n 100 enronemail_1h.txt|tail -1|less #an example SPAM email rec
```

```
0018.2003-12-18.GP      1       await your response    " dear partn
er,  we are  a team of government officials that belong to an eight-
man committee in the pres idential cabinet as well as the senate.  a
t the moment, we will be requiring you r assistance in a matter that
involves investment of monies, which we intend to  transfer to your
account, upon clarification and a workable agreement reached in con
summating the project with you. based on a recommendation from an as
sociate  concerning your integrity, loyalty and understanding, we de
emed it necessary to  contact you accordingly. all arrangements in r
elation to this investment initiat ive, as well as the initial capit
al for its take off has been tactically set asi de to commence whate
ver business you deemed fit, that will turn around profit fa vourabl
y. we request you immediately contact us if you will be favorably di
spose d to act as a partner in this venture, and possibly will affor
d us the opportuni ty to discuss whatever proposal you may come up w
ith. also  bear in mind that th e initial capital that we shall send
across will not exceed$ 13,731, 000,00 usd  (thirteen million seven
hundred and thirty one thousand united states dollars) s o whatever
areas of investment your proposal shall cover, please it should be w
i thin the set aside capital. in this regard, the proposal you may w
ish to discuss with us should be comprehensive enough for our bette
r understanding; with speci al emphasis on the following:  1. the ta
x obligationin your country  2. the init ial capital base required i
n your proposed  investment area, as well as;  3. the legal technic
alities in setting up a  business in your country with foreigners  a
s share-holders  4. the most convenient and secured mode of receivin
g the funds without our direct involvement.  5. your ability to pro
vide a beneficiary/partn e
```

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 statement with a "done" string will suffice here.

In [9]:
```
#Display contents of pNaiveBayes.sh (it's convenient to keep everything
!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, and
- reducer.py collates the counts of the single word.

1.2 Mapper to process Enron email corpus (100 emails)

All messages are collated to a tab-delimited format:

ID \t SPAM \t SUBJECT \t CONTENT \n

where:

ID = string; unique message identifier
SPAM = binary; with 1 indicating a spam message
SUBJECT = string; title of the message
CONTENT = string; content of the message

Note that either of SUBJECT or CONTENT may be "NA", and that all tab (\t) and newline (\n) characters have been removed from both of the SUBJECT and CONTENT columns.

In [1]:
```
%%writefile mapper.py
#!/usr/bin/python
## mapper.py
## Author: James Gray
## Description: mapper code for HW1.2

import sys
import re
count = 0
WORD_RE = re.compile(r"[\w']+")
filename = sys.argv[1]
findword = sys.argv[2].lower()
with open (filename, "r") as myfile:
    # examine each line of the list of lines
    for line in myfile.readlines( ):
        text = ""
        # parse each line to pull out the subject and body and concaten
        enronEmail = text.join(line.split('\t')[-2:]) # take the last t
        for word in WORD_RE.findall(enronEmail):
            if word == findword:
                count+=1
print count
```

```
Writing mapper.py
```

In [2]:
```
# set file priveleges to execute script
!chmod a+x mapper.py
```

1.2 Reducer to total the occurences of a single word

In [3]:
```
%%writefile reducer.py
#!/usr/bin/python
import sys
sum = 0
for file in sys.argv[1:]: # argument passed in is a list of chunk count
    # open each chunk count file
    with open(file, "r") as chunkfilecount:
        for line in chunkfilecount.readlines():
            tokens=line.split('\t') # the only token in this file is th
            sum+=int(tokens[0])
```

```
Writing reducer.py
```

In [4]:
```
# set file priveleges to execute script
!chmod a+x reducer.py
```

1.2 Execute Unix control script

In [6]:
```
# set priveleges to execute Unix script
!chmod a+x pNaiveBayes.sh

# parameter #1 -> # of mappers to run
# parameter #2 -> wordlist (the word(s) to search for and count)

!./pNaiveBayes.sh 2 assistance
```

1.2 View Output and Cross check against Grep command

In [16]:
```
# read the output file created from the Unix control script
print ("The number of occurences of \"assistance\" in the subject and b
!cat enronemail_1h.txt.output

# check against grep
print ("The number of occurences of \"assistance\" in body using grep:"
!grep assistance enronemail_1h.txt|cut -d$'\t' -f4| grep assistance|wc
```

```
The number of occurences of "assistance" in the subject and body:
10
The number of occurences of "assistance" in body using grep:
       8
```

This is a slight descrepency between the Unix control script and Grep command since the Grep command only evaluates the body.

HW1.3. (Graded)

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

performs 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 $Pr(X =\text{“}assistance\text{”} | Y = SPAM) = 5/1000$. Note this is a multinomial estimated of the class conditional for a Naive Bayes Classifier. No smoothing is needed in this HW.

The Naive Bayes classifier will predict the probability that an email is SPAM given the evidence of a predefined word (in this case the word "assistance"). The formula for Bayes theorem (from the book "Making Sense of Data II")

$$P(H|E) = \frac{P(E|H) * P(H)}{P(E)}$$

- P(H|E) = probability of a hypothesis(H) given some evidence(E) -> probability of SPAM given evidence of "assistance"
- P(E|H) = probability of evidence(E) given some hypothesis(H) -> probability of "assistance" given SPAM
- P(H) = probability of the hypothesis (SPAM)
- P(E) = probability of the evidence (the word "assistance")

1.3 SPAM Classifier Mapper

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 [21]:
```
%%writefile mapper.py
#!/usr/bin/python
## mapper.py
## Author: James Gray
## Description: mapper code for HW1.3

import sys
import re
WORD_RE = re.compile(r"[\w']+") #Compile regex to easily parse complete
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 subjec
        words=re.findall(WORD_RE,subject_and_body) #create list of word
        for word in words:
            #This flag indicates to the reducer that a given word shoul
            #by the reducer when calculating the conditional probabilit
            flag=0
            if word in findwords:
                flag=1

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

```
Overwriting mapper.py
```

1.3 SPAM Classifier Reducer

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 [22]:

```
%%writefile reducer.py
#!/usr/bin/python
#HW 1.3 - Reducer function

from __future__ import division #Python 3-style division syntax is much
import sys
from math import log

words={} # holds all words across the corpus
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:]: #iterate through all of the output files gen
    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])  # spam/ham flag
            word=result[2]       # word of interest
            flag=int(result[4])  # flag from mapper denoting this is a

            #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 probability of SPAM
prior_ham=1-prior_spam # prior probability of HAM
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 sm
    #word['p_spam']=(word['spam_count']+1)/(spam_word_count+vocab_count
    #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 l
                p_spam*=(words[word]['p_spam'])
            except ValueError:
                continue #This means that words that do not appear in a
            try:
                #p_ham+=log(words[word]['p_ham']) #Log version - No lon
                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
```

In [23]:

```
# set execute privilies on mapper.py and reducer.py

!chmod a+x mapper.py reducer.py
```

1.3 Execute Naive Bayes Classifier

The output file produced by the reducer has the following structure:

ID \t SPAM \t SPAM_PREDICTION

In [24]:
```
!./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
```

1.3 Training Error Calculation Function

This function will be used to calculate the training error for the Naive Bayes models

In [26]:
```
from __future__ import division

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

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

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

1.3 Calculate Training Error for Multinomial Naive Bayes model

Interpretation of training error result: 38% of the predictions for SPAM are incorrect using the word "assistance" as an indicator for SPAM

In [28]:
```
import pandas as pd

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
```

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 (accuracy). To do so, make sure that:

- mapper.py -> counts all occurrences of a list of words
- reducer.py -> performs the multiple-word multinomial Naive Bayes classification via the chosen list. No smoothing is needed in this HW.

1.4 SPAM Classifier Mapper

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 this example, the list of words is “assistance”, “valium”, and “enlargementWithATypo”.

In [31]:
```
%%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
        #parse the subject and body fields from the line, and combine i
        subject_and_body=" ".join(fields[-2:]).strip()
        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
```

1.4 SPAM Classifier Reducer

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.

In [32]:

```
%%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 proba
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
```

```
for k,word in words.iteritems():
    #These versions calculate conditional probabilities WITH Laplace sm
    #word['p_spam']=(word['spam_count']+1)/(spam_word_count+vocab_count
    #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 l
                p_spam*=words[word]['p_spam']
            except ValueError:
                pass #This means that words that do not appear in a cla
            try:
                #p_ham+=log(words[word]['p_ham']) #Log version - no lon
                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
```

In [33]:
```
# set execute privileges on mapper.py and reducer.py

!chmod a+x mapper.py reducer.py
```

1.4 Execute Naive Bayes Classifier

In [34]:
```
!./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
```

1.4 Calculate Training Error for Multinomial Naive Bayes model

The training error was reduced by 1% (38% to 37%) by adding a few words to identify SPAM emails.

In [36]:
```
#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
```

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.

1.5 SPAM Classifier Mapper

This mapper will need to consider all words in the email instead of specific pre-defined words. The change from the other mappers is that the check for the word and flag for the reducer was removed.

In [38]:

```
%%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
        # parse the subject and body fields from the line, and combine
        subject_and_body=" ".join(fields[-2:]).strip()
        words=re.findall(WORD_RE,subject_and_body)
        for word in words:
            # ID \t SPAM_flag \t word
            print fields[0]+'\t'+fields[1]+'\t'+word+'\t1'
```

```
Overwriting mapper.py
```

1.5 SPAM Classifier Reducer

The reducer is similar to the above reducers although the check for flagged words was removed as we need to calculate the conditional probabilities of all words

In [39]:

```
%%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 sm
    #word['p_spam']=(word['spam_count']+1)/(spam_word_count+vocab_count
    #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 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 usec
            p_spam*=(words[word]['p_spam'])
        except ValueError:
            continue #This means that words that do not appear in a cla
        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 [40]:
```
# set execute priviles on mapper.py and reducer.py

!chmod a+x mapper.py reducer.py
```

1.5 Execute Naive Bayes Classifier

In [42]:
```
!./pNaiveBayes.sh 4 "*";
! cat enronemail_1h.txt.output
```

```
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
```

1.5 Calculate Training Error for Multinomial Naive Bayes model

By considering all of the words the training error was reduced an additional 3%

In [44]:
```
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 multinomial Naive Bayes.

It always a good idea to test your solutions against publicly available libraries such as SciKit-Learn, The Machine Learning toolkit available in Python. In this exercise, we benchmark ourselves against the SciKit-Learn implementation of multinomial Naive Bayes. For more information on this implementation see: [http://scikit-learn.org/stable/modules/naive_bayes.html (http://scikit-learn.org/stable/modules/naive_bayes.html)](http://scikit-learn.org/stable/modules/naive_bayes.html) more

Training error = misclassification rate with respect to a training set. It is more formally defined here:

Let DF represent the training set in the following: Err(Model, DF) = |{(X, c(X)) ∈ DF : c(X) != Model(x)}| / |DF|

Where || denotes set cardinality; c(X) denotes the class of the tuple X in DF; and Model(X) denotes the class inferred by the Model “Model”

In this exercise, please complete the following:

1. 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)
2. Please prepare a table to present your results
3. 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 (Hint: smoothing, which we will discuss in next lecture)
4. 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 [52]:
```
import re
import numpy as np
import pandas as pd
from sklearn.naive_bayes import MultinomialNB, BernoulliNB
from sklearn.feature_extraction.text import CountVectorizer
```

1.6 Data read and cleaning for SK Learn Multinomial NB Algorithm

There are some lines with "NA" as the body and these should be removed to properly train the model

In [55]:
```
with open('enronemail_1h.txt','rb') as f:
    data=pd.read_csv(f, sep='\t', header=None, na_filter=True)

    # add column names to DataFrame
df_columns = ["id", "spamflag", "subject", "body"]
data.columns = df_columns
# concatenate subject and body to create a text string to be evaluated
data['subject_body'] = data["subject"] + data["body"]
dataClean = data.dropna()
dataClean
```

Out[55]:

	id	spamflag	subject	body	subject_body
1	0001.1999-12-10.kaminski	0	re: rankings	thank you.	re: rankings thank you.
2	0001.2000-01-17.beck	0	leadership development pilot	sally: what timing, ask and you shall receiv...	leadership development pilot sally: what tim...
4	0001.2001-02-07.kitchen	0	key hr issues going forward	a) year end reviews-report needs generating l...	key hr issues going forward a) year end revie...
5	0001.2001-04-02.williams	0	re: quasi	good morning, i'd love to go get some coffee...	re: quasi good morning, i'd love to go get s...

1.6 Create text features using CountVectorizer

The [CountVectorizer (http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html)](http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html) converts a collection of text documents to a matrix of token counts. Here it will process the 'subject_body' field of the DataFrame to create a vector of words and their counts. The fit_transform method creates a document-term matrix on the number of times a specific word (feature) appears in the document. In this scenario we have 100 or less records given that we filtered out the records where an "NA" appears in the row.

In [58]:
```
# create matrix for a single word, convert to lowercase first, filter w
vectorizer = CountVectorizer(analyzer='word', lowercase=True)
# create a document-term matrix
vocabulary = vectorizer.fit_transform(dataClean['subject_body'])
print (vocabulary)
```

```
(0, 3886)     1
(0, 3871)     1
(0, 4731)     1
(0, 5255)     1
(1, 4731)     1
(1, 5255)     5
(1, 2838)     7
(1, 1524)     3
(1, 3606)     6
(1, 4167)     1
(1, 5136)     1
(1, 4785)     1
(1, 625)      1
(1, 526)      11
(1, 4301)     1
(1, 3914)     2
(1, 619)      3
(1, 3552)     1
(1, 3448)     2
(1, 1581)     1
(1, 2910)     3
(1, 785)      3
(1, 2651)     4
(1, 518)      2
(1, 4967)     2
:      :
(93, 2883)    1
(93, 1233)    1
(93, 4854)    2
(93, 565)     1
(93, 3824)    1
(93, 3678)    1
(93, 607)     1
(93, 4633)    1
(93, 3840)    1
(93, 1652)    1
(93, 4717)    1
(93, 2081)    1
(93, 2633)    1
(93, 3553)    1
(93, 1261)    1
(93, 395)     1
(93, 1190)    1
(93, 4506)    1
(93, 1399)    1
(93, 382)     1
(93, 729)     1
(93, 1392)    1
(93, 103)     1
(93, 271)     1
(93, 280)     1
```

1.6 Run SK Learn Multinomial Naive Bayer Classifier

The multinomial Naive Bayes (http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html) algorithm uses word counts for classification.

In [64]:
```
multiNB = MultinomialNB()
# fit the model to the training data using document-term matrix and cla
multiNB.fit(vocabulary, dataClean['spamflag'])
# make predictions using the training data
mnbclf_results = multiNB.predict(vocabulary)
# calculate accuracy using metrics libary
from sklearn.metrics import accuracy_score
print ("SK Learn Multinomial NB training error: " + str(1-accuracy_scor
```

```
SK Learn Multinomial NB training error: 0.0
```

1.6 Run SK Learn Bernoulli Naive Bayer Classifier

The bernoulli Naives Bayes (http://nlp.stanford.edu/IR-book/html/htmledition/the-bernoulli-model-1.html) algorithm generates either a 1 for the presence of the term in a document or 0 for an absence.

In [65]:
```
# instantiate a bernoulli model
bernNB = BernoulliNB()
# fit the model to the training data using
bernNB.fit(vocabulary, dataClean['spamflag'])
bernclf_results = bernNB.predict(vocabulary)
# calculate accuracy using metrics libary
print ("SK Learn Bernoulli NB training error: " + str(1-accuracy_score(
```

```
SK Learn Bernoulli NB training error: 0.234042553191
```

1.6 Summary of Error Results and Conclusions

This section compares and contrasts errors produces by map-reduce and scikit-learn

Model	Training Error
Multinomial NB, Scikit-Learn Implementation	0.00
Bernoulli NB, Scikit-Learn Implementation	0.23
Multinomial NB HW1.5, MapReduce implementation	0.34

Analysis of Multinomial Naive Bayes models (MapReduce and scikit-learn)

The map-reduce Multinomial Naive Bayes model accuracy was significantly higher than the scikit-learn implementation. A few considerations come to mind why there was a difference. In the map-reduce implementation we did not design for Laplace smoothing and therefore terms that do not exist in the vocabulary produce a zero probability. Laplace smoothing is set by default in the scikit-learn implementation. Feature extraction may have also played a role. In the map-reduce implementation we used regex to parse for words while scikit-learn's countvectorizer includes digits in words while ignoring hyphenations and contractions. This can result in larger and richer vocabulary resulting in lower error.

Analysis of scikit-learn Bernoulli and Multinomial Naive Bayes models (scikit-learn)

The error of the multinomial and bernoulli naive bayes models implemented in scikit-learn differ significantly. The multinomial model error was 0 (not realistic in the real world because in this case we tested the model with training data) while the bernoulli model was 0.23. The primary reason is the multinomial model uses word counts to estimate probabilities while the bernoulli model is binary if the word appears in the document or not. This enables the multinomial model to discriminate at a deeper level and we should expect to see a lower error when compared to the Bernoulli model.

In [ ]:

```
```