=====DATSCIW261 ASSIGNMENT #1=====

MIDS UC Berkeley, Machine Learning at Scale DATSCIW261 ASSIGNMENT #1 (version 2016-01-14)

Hetal Chandaria (hetalchandaria@berkeley.edu)

W261 - 2 , Assignment 01

Submission Date : 01/18/2016

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

Answer :

Big data is a term for data sets that are so large that traditional computing systems are not sufficient for processing them. They are usually characterised by the 4 V's :

1. Volume : Amount of data generated
2. Velocity: How frequently is the data being generated
3. Variety : Different forms of data
4. Veracity : How certain are we about the data

Example of big data:

Consider eBay as an example. eBay has millions of users browsing through its website searching for, buying and selling products in 100s of categories. Trying to understand users and what their interests can be considered a big data challenge.

One can consider tracking the activity of all unique users across all pages/categories/product pages of eBay on all devices. These events would need to be tracked, aggregated on a per user basis and historically tracked to try and create some form of a profile of a user in terms of which categories/products the user is interested in. Additionally, present/future behaviour can be compared to models learnt on past user behaviour.

HW1.0.1.In 500 words (English or pseudo code or a combination) describe how to estimate the bias, the variance, the irreduciable 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?

Answer:

Consider f(x) to be the true relationship function and g(x) be an estimator of f(x). Assume g(x) estimator to be polynomial regression model of degree 1,2,3,4,5 which is fit over the test dataset T.

Based on this

1. Variance of the estimator is defined as E[(g(x) - E[g(x)])^2]. Variance basically helps us understand how much the preictions for a given data point vary between different realizations of the same model.
2. Bias of the estimator is defined as E[g(x)]-f(x). Bias measures the difference between the avergaes of the estimated value and the true values.
3. Irreducible Error is defined as variance in the data itself

As model complexity increases (i.e high order polynomials) we tend to find that the estimator fits the data well which means that the estimator variance increases while bias decreases. Models that are simple, have low variance but hgh bias. While selecting a model we should pick a model that will have both low bias and variance.

HW1.1.

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

Note

enronemail_1h.txt file had to be fixed to remove mac-style line endings for correct processing. Below are the commands that were run to fix the issue.

cat enronemail_1h.txt | tr '^M' '\n' > ./foo

mv foo enronemail_1h.txt

In [1]:
```
print 'done'
```

done

In [34]:
```
%%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 comman
d,
##               and focus on how arguments are supplied to mapper.py/
reducer.py,
##               as this will determine how the python scripts take in
put.
##               When you are comfortable with the unix code below,
##               answer the questions on the LMS for HW1 about the sta
rter 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 STDO
UT to disk
####
####
./reducer.py $countfiles > $data.output
####
####

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

```
Overwriting pNaiveBayes.sh
```

In [3]:

```
!chmod a+x pNaiveBayes.sh
```

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.

In [39]:
```
%%writefile mapper.py
#!/usr/bin/python
import sys
import re
count = 0
WORD_RE = re.compile(r"[\w']+")
filename = sys.argv[1]
findword = sys.argv[2]
with open (filename, "r") as myfile:
    for line in myfile.readlines():
        #Tokenize each line
        # Split the line by <TAB> delimiter
        content = re.split(r'\t+', line)
        # verify correct content structure else ignore bad data
        if len(content) <> 4:
            continue
        text = content[2] + ' ' + content[3]
        result = re.findall(WORD_RE,text)
        #Now find index of each matching instance of the word for that
email
        #lower is used to do case insensitive search
        indices = [i for i,x in enumerate(result) if x.lower() == find
word.lower()]
        # Correct approach is to increment the count based on the numb
er of occurences found.
        # but shell script example provided only increments once per l
ine matched.
        count += len(indices)
output = findword+"\t"+str(count)
print output
```

```
Overwriting mapper.py
```

In [43]:
```
%%writefile reducer.py
#!/usr/bin/python
import sys
import re
sum = 0
for x in range(1,len(sys.argv)):
    with open (sys.argv[x], "r") as myfile:
        for line in myfile.readlines():
            #Please insert your code
            #convert to int and increment the sum
            content = re.split(r'\t+', line)
            sum += int(content[1])
print content[0]+"\t"+str(sum)
```

```
Overwriting reducer.py
```

In [44]:
```
!chmod a+x mapper.py; chmod a+x reducer.py
```

In [45]:
```
!./pNaiveBayes.sh 2 assistance
```

```
assistance      10
```

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.

In [8]:

```
%%writefile mapper.py
#!/usr/bin/python
import sys
import re
count = 0
WORD_RE = re.compile(r"[\w']+")
filename = sys.argv[1]
findword = sys.argv[2]
with open (filename, "r") as myfile:
    for line in myfile.readlines():
        #Tokenize each line
        # Split the line by <TAB> delimiter
        content = re.split(r'\t+', line)
        # verify correct content structure else ignore bad data
        if len(content) <> 4:
            continue
        text = content[2] + ' ' + content[3]
        result = re.findall(WORD_RE,text)
        #Now find index of each matching instance of the word for that
email
        #lower is used to do case insensitive search
        indices = [i for i,x in enumerate(result) if x.lower() == find
word.lower()]
        # Correct approach is to increment the count based on the numb
er of occurences found.
        # but shell script example provided only increments once per l
ine matched.
        findword_count = len(indices)
        total_doc_count = len(result)
        output =content[0]+ "\t" + content[1]+"\t"+ findword
        output += "\t" + str(findword_count) + "\t" + str(total_doc_co
unt)

        print output
```

Overwriting mapper.py

In [12]:

```
%%writefile reducer.py
#!/usr/bin/python
import sys
import re
import math
sum = 0
# Total count of spam emails
spam_email_cnt  = 0
# Total count of non spam emails
non_spam_email_cnt = 0

#Total count of words in all spam emails
```

```
total_spam_words = 0
# Total count of words in all non spam emails
total_nonspam_words = 0
total_spam_findword = 0
total_nonspam_findword = 0

for x in range(1,len(sys.argv)):
    with open (sys.argv[x], "r") as myfile:
        for line in myfile.readlines():
            # Split the line by <TAB> delimiter
            content = re.split(r'\t+', line)
            docId = content[0]
            true_class = int(content[1])
            findword = content[2]
            findword_freq = int(content[3])
            total_doc_word_cnt = int(content[4])
            if (true_class == 1):
                spam_email_cnt += 1
                total_spam_findword += findword_freq
                total_spam_words += total_doc_word_cnt

            else:
                non_spam_email_cnt += 1;
                total_nonspam_findword += findword_freq
                total_nonspam_words += total_doc_word_cnt

prior_spam = math.log((1.0)*spam_email_cnt / (spam_email_cnt + non_spa
m_email_cnt ))
prior_ham = math.log((1.0)*non_spam_email_cnt / (spam_email_cnt + non_
spam_email_cnt ))
# Probability of word given email class spam
pr_findword_spam = math.log((1.0)*(total_spam_findword)/total_spam_wor
ds)
pr_findword_ham = math.log((1.0)*(total_nonspam_findword)/total_nonspa
m_words)

correct_match_cnt = 0
total_match = 0
###### Classification #########
for x in range(1,len(sys.argv)):
    with open (sys.argv[x], "r") as myfile:
        for line in myfile.readlines():
            # Split the line by <TAB> delimiter
            content = re.split(r'\t+', line)
            docId = content[0]
            true_class = content[1]
            findword_freq = int(content[3])
            # calculate prob for spam , ham
            pr_spam_doc = prior_spam + (pr_findword_spam*findword_freq
)
```

```
            pr_ham_doc = prior_ham + (pr_findword_ham*findword_freq)
            output =  docId+"\t"+true_class+"\t"
            predicted_class =0
            if(pr_spam_doc > pr_ham_doc) :
                predicted_class = 1
                output += "1"
            else:
                output += "0"
            if(int(true_class)==predicted_class):
                correct_match_cnt += 1
            total_match += 1
            print output
print "Accuracy of the model: %3.2f" %(correct_match_cnt*100.0/total_m
atch)
```

```
Overwriting reducer.py
```

In [13]:

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

In [14]:

```
!./pNaiveBayes.sh 2 assistance
```

```
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
Accuracy of the model: 60.00
```

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

In [35]:

```
%%writefile mapper.py
#!/usr/bin/python
import sys
import re
count = 0
WORD_RE = re.compile(r"[\w']+")
filename = sys.argv[1]
findwords ={}
local_findwords = re.split(" ",sys.argv[2].lower())
vocab_len = len(sys.argv[2:])
for word in local_findwords:
    findwords[word] = 1
with open (filename, "r") as myfile:
    for line in myfile.readlines():
        #Tokenize each line
        # Split the line by <TAB> delimiter
        content = re.split(r'\t+', line)
        # verify correct content structure else ignore bad data
        if len(content) <> 4:
            continue
        text = content[2] + ' ' + content[3]
        result = re.findall(WORD_RE,text)
        #build a vocabluary of words
        vocab ={}
        for word in local_findwords:
            vocab[word] = 0
        for key in result:
            if key not in findwords:
                continue
            if key in vocab:
                vocab[key] += 1
            else:
                vocab[key] = 1
        output =content[0]+ "\t" + content[1]+"\t"+str(len(result))+"\
t"+str(vocab_len)
        for key, value in vocab.iteritems():
            output += "\t" + key + "\t" + str(value)
        print output
```

```
Overwriting mapper.py
```

In [36]:

```
%%writefile reducer.py
#!/usr/bin/python
import sys
import re
import math
sum = 0
# Dictionary to store overall frequency of words for spam emails
spam_words_freq = {}
```

```
# Dictionary to store overall frequency of words for non spam emails
not_spam_words_freq ={}
# Total count of spam emails
spam_email_cnt  = 0
# Total count of non spam emails
non_spam_email_cnt = 0
# Unique vocab length
unique_word_cnt = 0
#Total count of words in all spam emails
total_spam_words = 0
# Total count of words in all non spam emails
total_nonspam_words = 0
for x in range(1,len(sys.argv)):
    with open (sys.argv[x], "r") as myfile:
        for line in myfile.readlines():
            # Split the line by <TAB> delimiter
            content = re.split(r'\t+', line)
            docId = content[0]
            true_class = int(content[1])
            doc_word_cnt = int(content[2])
            unique_word_cnt = int(content[3])
            if (true_class == 1):
                spam_email_cnt += 1
                total_spam_words += doc_word_cnt
            else:
                non_spam_email_cnt += 1
                total_nonspam_words += doc_word_cnt
            if(len(content) > 4 ):
                for x in range(4,len(content),2):
                    word = content[x]
                    freq = int(content[x+1])
                    if (true_class == 1):
                        if word in spam_words_freq:
                            spam_words_freq[word] += freq
                        else:
                            spam_words_freq[word] = freq
                    else:
                        if word in not_spam_words_freq:
                            not_spam_words_freq[word] += freq
                        else:
                            not_spam_words_freq[word] = freq

prior_spam = math.log((1.0)*spam_email_cnt / (spam_email_cnt + non_spa
m_email_cnt ))
prior_ham = math.log((1.0)*non_spam_email_cnt / (spam_email_cnt + non_
spam_email_cnt ))
# Probability of word given email class spam
pr_word_spam = {}
pr_word_ham = {}
for word in spam_words_freq:
```

```
    if(spam_words_freq[word] > 0 ):
        pr_word_spam[word] = math.log((1.0)*(spam_words_freq[word])/ (
total_spam_words))
    else:
        pr_word_spam[word] = float('-inf')
for word in not_spam_words_freq:
    if(not_spam_words_freq[word] > 0):
        pr_word_ham[word] = math.log((1.0)*(not_spam_words_freq[word])
/(total_nonspam_words))
    else:
        pr_word_ham[word] = float('-inf')

correct_match_cnt = 0
total_match = 0
###### Classification ##########
for x in range(1,len(sys.argv)):
    with open (sys.argv[x], "r") as myfile:
        for line in myfile.readlines():
            # Split the line by <TAB> delimiter
            content = re.split(r'\t+', line)
            docId = content[0]
            true_class = content[1]
            doc_vocab = {}
            if(len(content) > 4 ):
                for x in range(4,len(content),2):
                    word = content[x]
                    freq = int(content[x+1])
                    doc_vocab[word] = freq
            # calculate prob for spam , ham
            pr_spam_doc = 0.0
            pr_ham_doc = 0.0
            for key,value in doc_vocab.iteritems():
                if (pr_word_spam[key] == float('-inf')):
                    if(value !=0):
                        pr_spam_doc += float('-inf')
                    else :
                        pr_spam_doc += 0
                else:
                    pr_spam_doc += (pr_word_spam[key]*value)
                if(pr_word_ham[key] == float('-inf')):
                    if(value !=0):
                        pr_ham_doc += float('-inf')
                    else :
                        pr_ham_doc += 0
                else :
                    pr_ham_doc += (pr_word_ham[key]*value)
            pr_spam_doc = prior_spam + pr_spam_doc
            pr_ham_doc = prior_ham + pr_ham_doc
            output =  docId + "\t" + true_class + "\t"
            predicted_class = 0
```

```
            if(pr_spam_doc > pr_ham_doc) :
                predicted_class = 1
                output += "1"
            else:
                output += "0"
            if(int(true_class) == predicted_class):
                correct_match_cnt += 1
            total_match += 1
            print output
print "Accuracy of the model: %3.2f" %(correct_match_cnt*100.0/total_m
atch)
```

```
Overwriting reducer.py
```

In [37]:
```
!chmod a+x mapper.py; chmod a+x reducer.py
```

In [38]:
```
!./pNaiveBayes.sh 2 "assistance valium enlargementWithATypo"
```

```
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       1
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       1
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       1
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
Accuracy of the model: 63.00
```

HW1.5.

Provide a mapper/reducer pair that, when executed by pNaiveBayes.sh will classify the email messages by all words present.

This question was already solved before the email for ignoring it came.

In [ ]:
```
%%writefile mapper.py
#!/usr/bin/python
import sys
import re
count = 0
WORD_RE = re.compile(r"[\w']+")
filename = sys.argv[1]
with open (filename, "r") as myfile:
    for line in myfile.readlines():
        #Tokenize each line
        # Split the line by <TAB> delimiter
        content = re.split(r'\t+', line)
        # verify correct content structure else ignore bad data
        if len(content) <> 4:
            continue
        text = content[2] + ' ' + content[3]
        result = re.findall(WORD_RE,text)
        #build a vocabluary of words
        vocab ={}
        for key in result:
            if key in vocab:
                vocab[key] += 1
            else:
                vocab[key] = 1
        output =content[0]+ "\t" + content[1]
        for key, value in vocab.iteritems():
            output += "\t" + key + "\t" + str(value)

        print output
```

In [ ]:
```
%%writefile reducer.py
#!/usr/bin/python
import sys
import re
import math
sum = 0
# Dictionary to store overall frequency of words for spam emails
spam_words_freq = {}
# Dictionary to store overall frequency of words for non spam emails
not_spam_words_freq ={}
# Total count of spam emails
spam_email_cnt  = 0
# Total count of non spam emails
non_spam_email_cnt = 0
# Unique vocab length
unique_word_cnt = 0
#Total count of words in all spam emails
total_spam_words = 0
# Total count of words in all non spam emails
```

```
total_nonspam_words = 0
for x in range(1,len(sys.argv)):
    with open (sys.argv[x], "r") as myfile:
        for line in myfile.readlines():
            # Split the line by <TAB> delimiter
            content = re.split(r'\t+', line)
            docId = content[0]
            true_class = int(content[1])
            if (true_class == 1):
                spam_email_cnt += 1
            else:
                non_spam_email_cnt += 1;
            if(len(content) > 2 ):
                for x in range(2,len(content),2):
                    word = content[x]
                    freq = int(content[x+1])
                    if (not(word in spam_words_freq or word in not_spa
m_words_freq)):
                        unique_word_cnt += 1;
                    if (true_class == 1):
                        total_spam_words += freq;
                        if word in spam_words_freq:
                            spam_words_freq[word] += freq
                        else:
                            spam_words_freq[word] = freq
                        if word not in not_spam_words_freq:
                            not_spam_words_freq[word] = 0
                    else:
                        total_nonspam_words += freq;
                        if word in not_spam_words_freq:
                            not_spam_words_freq[word] += freq
                        else:
                            not_spam_words_freq[word] = freq
                        if word not in spam_words_freq:
                            spam_words_freq[word] = 0

prior_spam = math.log((1.0)*spam_email_cnt / (spam_email_cnt + non_spa
m_email_cnt ))
prior_ham = math.log((1.0)*non_spam_email_cnt / (spam_email_cnt + non_
spam_email_cnt ))
# Probability of word given email class spam
pr_word_spam = {}
pr_word_ham = {}
for word in spam_words_freq:
    # 1 is added for laplace smoothing
    pr = (1.0)*(spam_words_freq[word]+1)/ (total_spam_words + unique_w
ord_cnt)
    pr_word_spam[word] = pr
for word in not_spam_words_freq:
     # 1 is added for laplace smoothing
```

```
    pr = (1.0)*(not_spam_words_freq[word]+1)/ (total_nonspam_words + u
nique_word_cnt)
    pr_word_ham[word] = pr

correct_match_cnt = 0
total_match = 0

###### Classification ##########
for x in range(1,len(sys.argv)):
    with open (sys.argv[x], "r") as myfile:
        for line in myfile.readlines():
            # Split the line by <TAB> delimiter
            content = re.split(r'\t+', line)
            docId = content[0]
            true_class = content[1]
            doc_vocab = {}
            if(len(content) > 2 ):
                for x in range(2,len(content),2):
                    word = content[x]
                    freq = int(content[x+1])
                    doc_vocab[word] = freq
            # calculate prob for spam , ham
            pr_spam_doc = 0.0
            pr_ham_doc = 0.0
            for key,value in doc_vocab.iteritems():
                pr_spam_doc +=  math.log(pr_word_spam[key])*value
                pr_ham_doc += math.log(pr_word_ham[key])*value
            pr_spam_doc = prior_spam + pr_spam_doc
            pr_ham_doc = prior_ham + pr_ham_doc
            output =  docId+"\t"+true_class+"\t"
            predicted_class = 0
            if(pr_spam_doc > pr_ham_doc) :
                predicted_class = 1
                output += "1"
            else:
                output += "0"
            if(int(true_class) == predicted_class):
                correct_match_cnt += 1
            total_match += 1
            print output
print "Accuracy of the model: %3.2f" %(correct_match_cnt*100.0/total_m
atch)
```

In [ ]:
```
!chmod a+x mapper.py; chmod a+x reducer.py
```

In [ ]:
```
!./pNaiveBayes.sh 2
```

HW1.6 Benchmark your code with the Python SciKit-Learn implementation of multinomial Naive Bayes

Parts of this question were already solved before the email for ignoring it came

In [46]:
```
import sys
import re
from sklearn.naive_bayes import BernoulliNB
from sklearn.naive_bayes import MultinomialNB
import numpy
from sklearn.feature_extraction.text import CountVectorizer

alpha = 1
WORD_RE = re.compile(r"[\w']+")
filename = "enronemail_1h.txt"
train_data = []
train_label =[]

with open (filename, "r") as myfile:
    for line in myfile.readlines():
        #Tokenize each line
        # Split the line by <TAB> delimiter
        content = re.split(r'\t+', line)
        # verify correct content structure else ignore bad data
        if len(content) <> 4:
            continue
        true_class = int(content[1])
        text = content[2] + ' ' + content[3]
        text = re.sub(r'[\W]+', ' ', text)
#        text = re.sub('[^0-9a-zA-Z]+', ' ', text)
        train_data.append(text)
        train_label.append(true_class)

count_vectorizer = CountVectorizer()
text_matrix = count_vectorizer.fit_transform(train_data)
feature_names = count_vectorizer.get_feature_names()
```

— 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)

In [51]:
```
# nb =  MultinomialNB(alpha=alpha)
nb =  MultinomialNB()
nb.fit(text_matrix, train_label)

# Compute accuracy on the test data.
print "Using our Multinomial classifier"
accuracy = nb.score(text_matrix, train_label)*100
tr_error = 100-accuracy
print 'sklearn accuracy: %3.2f' %accuracy
print 'sk learn training error %3.2f' %tr_error
```

```
Using our Multinomial classifier
sklearn accuracy: 100.00
sk learn training error 0.00
```

— Run the Bernoulli Naive Bayes algorithm from SciKit-Learn (using default settings) over the same training data used in HW1.5 and report the Training error

In [50]:
```
# Compare to sklearn's implementation.
print "Using sklearn's NB classifier"
# clf = BernoulliNB(alpha=alpha)
clf = BernoulliNB()
clf.fit(text_matrix, train_label)
accuracy = clf.score(text_matrix, train_label)*100
tr_error = 100 - accuracy
print 'sklearn accuracy: %3.2f' %accuracy
print 'sk learn training error %3.2f' %tr_error
```

```
Using sklearn's NB classifier
sklearn accuracy: 84.00
sk learn training error 16.00
```

— Run the Multinomial Naive Bayes algorithm you developed for HW1.5 over the same data used HW1.5 and report the Training error

In [ ]:

— 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)

In [ ]:

- 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 [ ]:

```

```