DATASCI W261: Machine Learning at Scale

Tuhin Mahmud

tuihinm@ischool.berkeley.edu
W261-1 spring 2016
Week 1: Homework

date:01/21/2016

HW1.0

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

Big data is defined in terms of four **V**s i.e *volume, volicity, variety, veracity*. Any data that is large in all or most of these criteria are considered big data. Data is Big Data when it cannot be stored on a single device e.g our laptop or single workstation machine and/or cannot be handled in a timely manner without large-sized parallel processing.

For volume it big is considered to be peta or zeta byte thats but it changes by context and use and increasing becoming larger.

I work in Verification Simulation tool and our backend result data is growing exponentially with each chip generation .Our daily volume of results are now in the terabyte range with thousands of simulation runs. This volume is becoming increasingly unmanagable with the traditional tools and we are considering using big data tools like Apache hadoop and spark to collect and analyze the results. Current stage of development is to set up the cluster and build the infrastructure needed. We are exploring soft layer offering in this regard as well as builing the infrastructure from hardware machines available.

HW1.1

HW1.0.1:

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

Step 1: Create test, training, and validation datasets

Randomly separate the given dataset ***T*** into a test dataset ***A*** (20% of the data) and a training dataset ***B*** (remaining 80% of the data). Next, create *S* training and validation datasets from ***B*** using the sub-sampled cross-valida tion (ssCV)

Step 2: Build models

For each training set $train_s$, build a model $h_s^p(x) = \beta_0 + \beta_1 x^1 + \ldots + \beta_p x^p$ for p = 1, ... , 5.

Step 3: Estimate Bias

For each validation set $valid_s$, calculate the squared bias of $h_s^p(x)$ for p = 1, ... , 5 using the formula:

$$Bias_s^p = \frac{1}{N}\sum_{i=1}^{N}[h_s^p(x_i) - y_i]^2$$

where $(x_i, y_i) \in valid_s$.

Estimate the overall bias for p = 1, ... , 5 by calculating the mean of the biases above using the formula:

$$Bias^p = \frac{1}{S}\sum_{s=1}^{S} Bias_s^p$$

Step 4: Estimate Variance

Estimate the variances for p = 1, ... , 5 by calculating:

$$Variance^p = \frac{1}{S}\sum_{s=1}^{S}\frac{1}{N_s}\sum_{i=1}^{N_s}[h_s^p(x_{si}) - \frac{1}{S}\sum_{t=1}^{S} h_t^p(x_{si})]^2$$

where $x_{si} \in valid_s$.

Step 5: Estimate Error

Estimate the model's mean squared error for p = 1, ... , 5 using the test dataset ***A*** by calculating:

$$MSE^p = \frac{1}{N}\sum_{i=1}^{N}[\frac{1}{S}\sum_{s=1}^{S} h_s^p(x_i) - y_i]^2$$

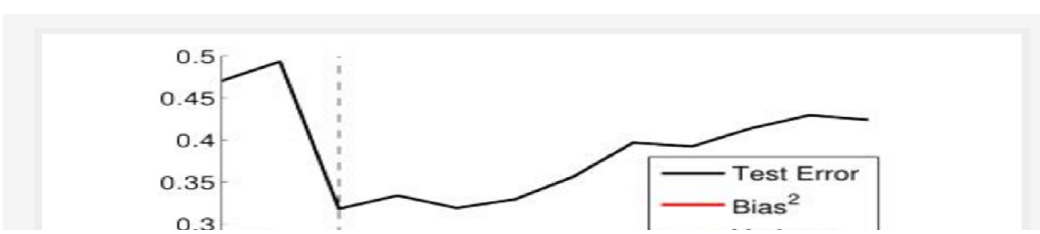
where $(x_i, y_i) \in$ ***A***.

Step 6: Select a model

To select a model, choose the polynomial with the lowest *MSE*.

Overall Bias vs Variance Trade off

Following grarph depicts the bias vs variance trade off that is typical of model selection process as different model complexity is considered. The same overall trade off is considered in the 6 steps algorithm described above.



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 command,
##               and focus on how arguments are supplied to mapper.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 processes
linesinchunk=`echo "$linesindata/$m+1" | bc`
#echo $linesinchunk

#exit
## 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 f
    ####
    ####
   ./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 STDOUT to disk
####
####
./reducer.py $countfiles > $data.output
## pass the number of mapper outfile and wordlist to the reducer
####
####

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

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

In [3]:
```
def hw1_1():
    print 'Done.'
    return

hw1_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.

Mapper

In [4]:
```
%%writefile mapper.py
#!/usr/bin/env python
import sys
import re
from collections import Counter
def mapper(fname,mlist):
    #print mlist
    wordCounter = Counter()
    for line in open(fname):
        line = line.strip()
        words = line.split()
        id =line.split('\t')[0]
        for word in words:
            # remove any trailing punctuations from the word
            nword = word.rstrip('?:!.,;')
            #if the list of words provided is empty consider all
            if not mlist:
                wordCounter.update({nword:1})
            else:
               #find the word match
               if nword in mlist:
                   wordCounter.update({nword:1})
                   #print "id=",id,"word",word,"count",wordCounter[nword]
    #print wordCounter
    for word in wordCounter:
        print word,"\t",wordCounter[word]
def main():
    if len(sys.argv) < 3:
        print "incorrect number of arguments"
        return
    fname=sys.argv[1]
    wordlist=sys.argv[2].lower()
    c = Counter()
    if sys.argv[2] == "*":
        wordlist=[]
    mapper(fname,wordlist.split())

if __name__ == "__main__":
    main()
```

```
Overwriting mapper.py
```

In [6]:
```
!chmod a+x mapper.py
```

Reducer 1.2

In [5]:
```
%%writefile reducer.py
#!/usr/bin/env python
from operator import itemgetter
import sys
from collections import Counter

def reducer(countfiles):
    #print countfiles
    wordCounter = Counter()
    for fname in countfiles:
        current_word = None
        current_count = 0
        word = None
        for line in open(fname):
            line = line.strip()
            word, count = line.split('\t', 1)

            try:
                count = int(count)
            except ValueError:
                continue
            #print word,count
            wordCounter.update({word:count})
    #print wordCounter
    for word in wordCounter:
        print word,"\t",wordCounter[word]
def main():
    if len(sys.argv) < 2:
        print "%s wrong number of arguments" % sys.argv
        return
    countfiles=[]
    countfiles=sys.argv[1:]
    reducer(countfiles)

if __name__ =="__main__":
    main()
```

```
Overwriting reducer.py
```

In [7]:
```
!chmod a+x reducer.py
```

Run MapReduce 1.2

In [8]:
```
def hw1_2():
    !./pNaiveBayes.sh 2 "assistance"

    # print out results
    with open ("enronemail_1h.txt.output", "r") as myfile:
        for line in myfile.readlines():
            print line
    return

hw1_2()
```

```
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. Examine the word "assistance" and report your results. To do so, make sure that (a) mapper.py is the same as in part (2), and (b) reducer.py performs a single word Naive Bayes classification.

pNaiveBayesh.sh 1.3

In [9]:
```
%%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 command,
##               and focus on how arguments are supplied to mapper.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 processes
linesinchunk=`echo "$linesindata/$m+1" | bc`
#echo $linesinchunk

#exit
## 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 f
    ####
    ####
   ./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 STDOUT to disk
####
####
./reducer.py $countfiles > $data.output
## pass the number of mapper outfile and wordlist to the reducer
#num_countfiles=`echo "$linesindata/($linesindata/$m+1)+1" | bc`
#./reducer.py $num_countfiles "$wordlist" $countfiles > $data.output
####
####

## clean up the data chunks and temporary count files
```

In [10]:
```
!chmod a+x pNaiveBayes.sh
```

Mapper 1.3

In [11]:
```
%%writefile mapper.py
#!/usr/bin/python
import sys
import re
from collections import Counter

debugFile=open ("debug.mapper.out", "w")

WORD_RE = re.compile(r"[\w']+")
def mapper(filename,findwords):
    findwords = [item.lower() for item in findwords] # make all findwords lowercase
    findwordsSet = set(findwords) # create a set of unique find words
    with open (filename, "r") as myfile:
        for line in myfile.readlines():
            components = line.split('\t')
            if len(components) < 3:
                continue
            ID = components[0]
            flag = components[1]
            text = " ".join(components[2:])
            wordCounter = Counter()
            for word in WORD_RE.findall(text):
                print >>debugFile,line
                wordCounter.update({word:1})
                #print >>debugFile, word,word_count[word]c
            for word in wordCounter:
                found_word = 0
                if word.lower() in findwordsSet:
                    found_word = 1
                print "%s\t%s\t%s\t%s\t%s" % (ID,str(flag),word,str(wordCounter[word

def main():
    fname=sys.argv[1]
    findwords=sys.argv[2].split()
    mapper(fname,findwords)
if __name__ == "__main__":
    main()
```

Overwriting mapper.py

In [12]:
```
!chmod a+x mapper.py
```

Reducer 1.3

In [13]:
```
%%writefile reducer.py
#!/usr/bin/python
import sys
import string
import re
import numpy as np
from collections import Counter
debugFile =open("debug.reducer.out","w")


def reducer(files,laplace):

    #spam/ham related data structure to keep counts
    vocab = set()

    spamCount = 0 # number of vocab words that are in spam
    hamCount= 0  # number of vocab words that are in spam
    spam =set() # unique set of email IDs that are in spam
    ham=set()    # unique set of email IDs that are in ham
    spamWordCounter = Counter() # counter for each vocab word that are spam
    hamWordCounter = Counter() # counter for each vocab word that are ham
    condProbSpam = Counter()
    condProbHam = Counter()

    for filename in files:
        with open(filename, 'r') as myfile:
            for line in myfile.readlines():
                ID,flag,word,count,findword = line.split('\t')
                print >>debugFile, ID,flag,word,count,findword
                flag =int(flag)
                count=int(count)
                findword=int(findword)
                if findword == 1:
                    print >>debugFile, "findword==1" ,ID,flag,word,count,findword
                    vocab.add(word)
                if flag == 1:
                    spam.add(ID) # save unique IDs of emails in spam
                    spamCount += count
                    if findword == 1:
                        spamWordCounter.update({word:count})
                else:
                    ham.add(ID)  # save unique IDs of emails in ham
                    hamCount += count
                    if findword == 1:
                        hamWordCounter.update({word:count})
                    #print >>debugFile, spamCount,hamCount,word,spamWordCounter[word

    # calculate prior probabilities
    count_ham = len(ham)
    count_spam = len(spam)
    prior_Pr_ham = count_ham*1.0/(count_ham + count_spam)
    prior_Pr_spam = count_spam*1.0/(count_ham + count_spam)


    hamWordCount=len(hamWordCounter)
    spamWordCount=len(spamWordCounter)

    #print >>debugFile,vocab
    B_value = len(vocab)
    for v in vocab:
        if (laplace==1) :
            #laplace smoothing
            value =float(spamWordCounter[v]+1)/float(spamCount + B_value)
            condProbSpam.update({v:value})
```

In [14]:
```
!chmod a+x reducer.py
```

In [15]:
```
# function used to read teh output from reducer.py and find the accuracy of the pred

def find_accuracy(df):
    count = 0
    correct = 0
    for index, row in df.iterrows():
        #print row['actual'],row['predicted']
        try:
            actual_value =int(row['actual'])
        except:
            continue
        try:
            predicted_value =int(row['predicted'])
        except:
            continue

        if actual_value == predicted_value:
            #print "found correct"
            correct += 1
        count += 1
    accuracy = correct*1.0/count
    print "prediction accuracy:", accuracy
```

Run MapReduce 1.3

In [16]:
```
import pandas as pd
from IPython.display import display
def hw1_3():
    !./pNaiveBayes.sh 10 "assistance"

    df = pd.read_csv("enronemail_1h.txt.output",sep='\t')
    find_accuracy(df)
    with pd.option_context('display.max_rows', 999, 'display.max_columns', 3):
        display(df)
    return

hw1_3()
```

```
prediction accuracy: 0.59
```

	id	actual	predicted
0	0001.1999-12-10.farmer	0	0
1	0001.1999-12-10.kaminski	0	0
2	0001.2000-01-17.beck	0	0
3	0001.2000-06-06.lokay	0	0
4	0001.2001-02-07.kitchen	0	0
5	0001.2001-04-02.williams	0	0
6	0002.1999-12-13.farmer	0	0
7	0002.2001-02-07.kitchen	0	0
8	0002.2001-05-25.SA_and_HP	1	0
9	0002.2003-12-18.GP	1	0
10	0002.2004-08-01.BG	1	1
11	0003.1999-12-10.kaminski	0	0
12	0003.1999-12-14.farmer	0	0
13	0003.2000-01-17.beck	0	0
14	0003.2001-02-08.kitchen	0	0
15	0003.2003-12-18.GP	1	0
16	0003.2004-08-01.BG	1	0
17	0004.1999-12-10.kaminski	0	1
18	0004.1999-12-14.farmer	0	0
19	0004.2001-04-02.williams	0	0
20	0004.2001-06-12.SA_and_HP	1	0
21	0004.2004-08-01.BG	1	0
22	0005.1999-12-12.kaminski	0	1
23	0005.1999-12-14.farmer	0	0
24	0005.2000-06-06.lokay	0	0
25	0005.2001-02-08.kitchen	0	0
26	0005.2001-06-23.SA_and_HP	1	0
27	0005.2003-12-18.GP	1	0

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

Reducer 1.4

- Uses Laplace smoothing

Run MapReduce 1.4

- Improved accuracy from 1.3 from use of multiple words
- uses Laplace smoothing

In [17]:
```
import pandas as pd
from IPython.display import display
def hw1_4():
    !./pNaiveBayes.sh 2 'assistance valium enlargementWithATypo'

    df = pd.read_csv("enronemail_1h.txt.output",sep='\t')
    find_accuracy(df)
    with pd.option_context('display.max_rows', 999, 'display.max_columns', 3):
        display(df)
    return

hw1_4()
```

```
prediction accuracy: 0.6
```

	id	actual	predicted
0	0001.1999-12-10.farmer	0	0
1	0001.1999-12-10.kaminski	0	0
2	0001.2000-01-17.beck	0	0
3	0001.2000-06-06.lokay	0	0
4	0001.2001-02-07.kitchen	0	0
5	0001.2001-04-02.williams	0	0
6	0002.1999-12-13.farmer	0	0
7	0002.2001-02-07.kitchen	0	0
8	0002.2001-05-25.SA_and_HP	1	0
9	0002.2003-12-18.GP	1	0
10	0002.2004-08-01.BG	1	0
11	0003.1999-12-10.kaminski	0	0
12	0003.1999-12-14.farmer	0	0
13	0003.2000-01-17.beck	0	0
14	0003.2001-02-08.kitchen	0	0
15	0003.2003-12-18.GP	1	0
16	0003.2004-08-01.BG	1	0
17	0004.1999-12-10.kaminski	0	0
18	0004.1999-12-14.farmer	0	0
19	0004.2001-04-02.williams	0	0
20	0004.2001-06-12.SA_and_HP	1	0
21	0004.2004-08-01.BG	1	0
22	0005.1999-12-12.kaminski	0	0
23	0005.1999-12-14.farmer	0	0
24	0005.2000-06-06.lokay	0	0
25	0005.2001-02-08.kitchen	0	0
26	0005.2001-06-23.SA_and_HP	1	0
27	0005.2003-12-18.GP	1	0

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

Design Summary and Testing

1. updated pNaiveBayesh.sh to pass the wordlist and count
2. mapper.py outputs data in the following format for each of the chunk created
    A. email ID, true class, and word counts

    exmaple :

    - 0001.1999-12-10.farmer 0 farm 1 pictures 1 ....
    - 0001.1999-12-10.kaminski 0 re 1 you 1 thank 1 ....
3. email "subject" and "text" are combined and searched for matching words for NaiveBayes algorithm
4. Used Llaplace smoothing(add one) to avoid divide by zero error . Testing the non laplace results they were not as good.
5. Reducer uses log based implementation of NB as given in 13.4 of the book
6. Initial testing was done on the test.tst data similar to that of table 13.1 and matched with the Exercise 13.1 example conditional probabilities

```
P(Chinese|C) = 3/7
P(Japan|~C) =   2/9
```

In [18]:

```
%%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 command,
##               and focus on how arguments are supplied to mapper.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 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 f
    ####
    ####
    ./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 $data.chunk.*.counts | perl -pe 's/\n/ /'`
## feed the list of countfiles to the python reducer and redirect STDOUT to disk
####
####
num_countfiles=`echo "$linesindata/($linesindata/$m+1)+1" | bc`
./reducer.py $num_countfiles "$wordlist" $countfiles > $data.output
####
####

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

```
Overwriting pNaiveBayes.sh
```

Mapper 1.5

In [19]:
```
%%writefile mapper.py
#!/usr/bin/python
import sys
import string
import re

filename = sys.argv[1]

# read in the data and create an output list with email ID, true class, and word cou
output = []
with open (filename, "r") as myfile:                 # for the given file
    for line in myfile.readlines():                  # for each line in the file
        line_output = []                             # create an empty list for the
        split_line = string.split(line, maxsplit = 2)  # split the line into three p
#         split_line = string.split(line, sep='\t')    # split the line into four pa
        line_output.append(split_line[0])              # first part is the email ID
        line_output.append(split_line[1])              # second part is the true cla
        text = split_line[2]                           # third part is the email tex
        word_count = {}                              # create an empty dictionary fo
        for word in text.lower().split():            # for each word in the email te
            clean_word = re.sub(r'[^\w\s]','',word)  # remove punctuation
            if clean_word != "":                     # if the cleaned word is not em
                if clean_word not in word_count:     # if the cleaned word is not in
                    word_count[clean_word] = 1       # set its count to 1
                else:                                # otherwise if it is already in
                    word_count[clean_word] += 1      # add 1 to its count

        line_output.append(word_count)               # add the word counts dictionar
        output.append(line_output)                   # add the line output to the fi

# print out tab-delimited data of email ID, true class, and word counts
for example in output:
    counts = "\t".join(["%s\t%s" % (word, count) for word, count in example[2].items
    print example[0], "\t", example[1], "\t", counts
```

```
Overwriting mapper.py
```

Reducer 1.5

- Uses Laplace smoothing

In [20]:
```
%%writefile reducer.py
#!/usr/bin/python
import sys
import string
import re
import numpy as np

# read in the data from the mapper output files and create a list
# data = [email_ID, true_class, {word: count}]
data = []
for i in range(int(sys.argv[1])):         # for each chunk
    filename = sys.argv[i+3]              # get the filename for that chunk
    with open (filename, "r") as myfile:  # open the file
        for line in myfile.readlines():   # for each line in the file
            line_output = []              # create a list for that line
            split_line = string.split(line, maxsplit=2)  # split the line into three
            line_output.append(split_line[0])  # append the email id
            line_output.append(split_line[1])  # append the true class
            text = split_line[2].split()       # split the email text
            word_count = {}                    # create a dictionary for the word co
            for j in range(len(text)/2):       # for every other token in the email
                word_count[text[2*j]] = int(text[2*j+1])  # assign the count to the
            line_output.append(word_count)     # add the wordcount dictionary to the
            data.append(line_output)           # add the line to the data

# create a vocabulary of words from all examples and count occurrences in ham and sp
# count total number of hams, total number of spams
# vocab = {word:[number of occurrences in ham, number of occurrences in spam]}
vocab = {}
count_ham = 0
count_spam = 0

for example in data:
    if example[1] == '0':
        count_ham += 1
    else:
        count_spam += 1
    for word in example[2]:
        if word not in vocab:
            if example[1] == '0':
                vocab[word] = [example[2][word], 0]
            else:
                vocab[word] = [0, example[2][word]]
        else:
            if example[1] == '0':
                vocab[word][0] += example[2][word]
            else:
                vocab[word][1] += example[2][word]

# count total number of words in vocab, ham, and spam
words_vocab = len(vocab)
words_ham = sum(h for h, s in vocab.itervalues())
words_spam = sum(s for h, s in vocab.itervalues())

# calculate prior probabilities of ham and spam
prior_Pr_ham = count_ham*1.0/(count_ham + count_spam)
prior_Pr_spam = count_spam*1.0/(count_ham + count_spam)

# calculate conditional probabilities for each word in vocab for ham and spam using
#cond_prob{} = {word:[conditional probability ham, conditional probability spam]}
cond_prob = {}
for word in vocab:
    cond_prob[word] = [(vocab[word][0] + 1)*1.0/(words_ham + words_vocab),
                       (vocab[word][1] + 1)*1.0/(words_spam + words_vocab)]
```

In [21]:
```
!chmod a+x reducer.py
```

Run MapReduce 1.5

In [22]:
```
import pandas as pd
from IPython.display import display
def hw1_5():
    !./pNaiveBayes.sh 2 '*'

    df = pd.read_csv("enronemail_1h.txt.output",sep='\t')
    find_accuracy(df)
    display(df)
    return

hw1_5()
```

```
prediction accuracy: 1.0
```

	id	actual	predicted
0	0001.1999-12-10.farmer	0	0
1	0001.1999-12-10.kaminski	0	0
2	0001.2000-01-17.beck	0	0
3	0001.2000-06-06.lokay	0	0
4	0001.2001-02-07.kitchen	0	0
5	0001.2001-04-02.williams	0	0
6	0002.1999-12-13.farmer	0	0
7	0002.2001-02-07.kitchen	0	0
8	0002.2001-05-25.SA_and_HP	1	1
9	0002.2003-12-18.GP	1	1
10	0002.2004-08-01.BG	1	1
11	0003.1999-12-10.kaminski	0	0
12	0003.1999-12-14.farmer	0	0
13	0003.2000-01-17.beck	0	0
14	0003.2001-02-08.kitchen	0	0
15	0003.2003-12-18.GP	1	1
16	0003.2004-08-01.BG	1	1
17	0004.1999-12-10.kaminski	0	0
18	0004.1999-12-14.farmer	0	0
19	0004.2001-04-02.williams	0	0
20	0004.2001-06-12.SA_and_HP	1	1
21	0004.2004-08-01.BG	1	1
22	0005.1999-12-12.kaminski	0	0
23	0005.1999-12-14.farmer	0	0
24	0005.2000-06-06.lokay	0	0
25	0005.2001-02-08.kitchen	0	0
26	0005.2001-06-23.SA_and_HP	1	1
27	0005.2003-12-18.GP	1	1
28	0006.1999-12-13.kaminski	0	0

HW1.6

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

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) more

Lets define 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:

1.6.1 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)

1.6.2 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

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

1.6.4 Please prepare a table to present your results

1.6.5 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)

1.6.6 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

Reducer 1.6

Reducer 1.6 outputs data in the format needed by the Naive Bayes functions in SciKit-Learn

- Reducer output is changed for use in scilearn input . # create a new output file with full vocabulary word counts for each email and first column is the true class used in trianing

Exmaple: 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

In [23]:
```
%%writefile reducer.py
#!/usr/bin/python
import sys
import string
import re
import numpy as np

# read in the data from the mapper output files
data = []
for i in range(int(sys.argv[1])):         # for each chunk
    filename = sys.argv[i+3]              # get the filename for that chunk
    with open (filename, "r") as myfile:  # open the file
        for line in myfile.readlines():   # for each line in the file
            line_output = []              # create a list for that line
            split_line = string.split(line, maxsplit=2)  # split the line into three
            line_output.append(split_line[0])  # append the email id
            line_output.append(split_line[1])  # append the true class
            text = split_line[2].split()       # split the email text
            word_count = {}                    # create a dictionary for the word co
            for j in range(len(text)/2):       # for every other token in the email
                word_count[text[2*j]] = int(text[2*j+1])  # assign the count to the
            line_output.append(word_count)     # add the wordcount dictionary to the
            data.append(line_output)           # add the line to the data

# create a vocabulary of all words that occur in the Enron email file with their cou
vocab = []
for example in data:
    for word in example[2]:
        if word not in vocab:
            vocab.append(word)

# create a new output file with full vocabulary word counts for each example
for example in data:                            # for each example in the data
    word_count = []                             # create a word count list
    for word in vocab:                          # for each word in the vocabulary
        if word in example[2]:                  # if the word is in the example ema
            word_count.append(example[2][word]) # get the count of that word in the
        else:                                   # if the word is not in the example
            word_count.append(0)                # set the word's count to 0

    # print out tab-delimited data of true class and word counts
    counts = "\t".join(["%i" %(count) for count in word_count])
    print example[1], "\t", counts
```

```
Overwriting reducer.py
```

In [24]:
```
!chmod a+x reducer.py
```

Run Reducer 1.6

In [25]:
```
!./pNaiveBayes.sh 20 "assistance"
```

1.6.1-1.6.2: SciKit-Learn Naive Bayes

In [26]:
```
import csv
from sklearn.naive_bayes import BernoulliNB
from sklearn.naive_bayes import MultinomialNB

# import data created by Reducer 1.6
data = []
with open('enronemail_1h.txt.output', 'rb') as csvfile:
    datareader = csv.reader(csvfile, delimiter='\t')
    for line in datareader:
        data.append(line)

# create training data
y = []
X = []
for example in data:
    counts = []
    try:
        int(example[0])
    except:
        continue
    y.append(int(example[0]))
    for i in range(1, len(example),1):
        counts.append(int(example[i]))
    X.append(counts)

# Multinomial Naive Bayes Model
mnb = MultinomialNB(fit_prior = True) # instantiate the Naive Bayes model
mnb.fit(X,y)
mnb_training_error = 1 - mnb.score(X,y)
print mnb_training_error

# Bernoulli Naive Bayes Model
bnb = BernoulliNB(binarize = 0.0, fit_prior = True) # instantiate the Naive Bayes mo
bnb.fit(X,y)
bnb_training_error = 1 - bnb.score(X,y)
print bnb_training_error
```

```
0.0
0.2
```

Rewrite Reducer 1.5

*

In [27]:
```
%%writefile reducer.py
#!/usr/bin/python
import sys
import string
import re
import numpy as np

# read in the data from the mapper output files and create a list
# data = [email_ID, true_class, {word: count}]
data = []
for i in range(int(sys.argv[1])):         # for each chunk
    filename = sys.argv[i+3]              # get the filename for that chunk
    with open (filename, "r") as myfile:  # open the file
        for line in myfile.readlines():   # for each line in the file
            line_output = []              # create a list for that line
            split_line = string.split(line, maxsplit=2)  # split the line into three
            line_output.append(split_line[0])  # append the email id
            line_output.append(split_line[1])  # append the true class
            text = split_line[2].split()       # split the email text
            word_count = {}                    # create a dictionary for the word co
            for j in range(len(text)/2):       # for every other token in the email
                word_count[text[2*j]] = int(text[2*j+1])  # assign the count to the
            line_output.append(word_count)     # add the wordcount dictionary to the
            data.append(line_output)           # add the line to the data

# create a vocabulary of words from all examples and count occurrences in ham and sp
# count total number of hams, total number of spams
# vocab = {word:[number of occurrences in ham, number of occurrences in spam]}
vocab = {}
count_ham = 0
count_spam = 0

for example in data:  # data has rows  in the format [ID,true Class, text]
    if example[1] == '0': #if true class is ham
        count_ham += 1  #increemnt ham count
    else:
        count_spam += 1 # increment spam count
    for word in example[2]: # for each word in the text
        if word not in vocab: # and if the word in the vocabilary
            if example[1] == '0': #if ham
                vocab[word] = [example[2][word], 0] # initialize with count accordin
            else:                                   # swap the count position in the
                vocab[word] = [0, example[2][word]]
        else:
            if example[1] == '0':
                vocab[word][0] += example[2][word] # add to the right counter ,here
            else:
                vocab[word][1] += example[2][word] # spam

# count total number of words in vocab, ham, and spam
words_vocab = len(vocab)
words_ham = sum(h for h, s in vocab.itervalues())
words_spam = sum(s for h, s in vocab.itervalues())

# calculate prior probabilities of ham and spam
prior_Pr_ham = count_ham*1.0/(count_ham + count_spam)
prior_Pr_spam = count_spam*1.0/(count_ham + count_spam)

# calculate conditional probabilities for each word in vocab for ham and spam using
#cond_prob{} = {word:[conditional probability ham, conditional probability spam]}
cond_prob = {}
for word in vocab:
    cond_prob[word] = [(vocab[word][0] + 1)*1.0/(words_ham + words_vocab),
                       (vocab[word][1] + 1)*1.0/(words_spam + words_vocab)]
```

In [28]:
```
!chmod a+x reducer.py
```

1.6.3: Rerun MapReduce 1.5

In [29]:
```
import csv

import pandas as pd
from IPython.display import display

!./pNaiveBayes.sh 20 "assistance"

df = pd.read_csv("enronemail_1h.txt.output",sep='\t',header=0)
display(df)
mr_training_error=round(df['training_error'][0],2)
```

	training_error
0	0.009901

1.6.4: Table of Results

In [30]:
```
import matplotlib.pyplot as plt

data = [[str(mnb_training_error) + '0'], [bnb_training_error], [str(mr_training_erro


import pandas as pd
from IPython.display import display
df =pd.DataFrame(data)
dft=df.transpose()
dft.columns =['Multinomial Naive Bayes', 'Bernoulli Naive Bayes', 'MapReduce 1.5 Nai
display(dft)
```

	Multinomial Naive Bayes	Bernoulli Naive Bayes	MapReduce 1.5 Naive Bayes
0	0.00	0.2	0.010

1.6.5: MapReduce vs. SciKit-Learn

There was *no difference in the training error* between the MapReduce implementation in HW1.5 and the Multinomial Naive Bayes implementation using SciKit-Learn.

I think using following are some of the reasons for getting 100% accuracy

- the laplace smoothing which I used for HW1.5 and Sci-learng uses
- use of the training set as test set gave us the 100% accuracy,

1.6.6: Multinomial vs. Bernoulli Naive Bayes

A key difference between Multinomial(MNB) and Bernoulli Naive Bayes(BNB) is that MNB counts the occurance of a word in the document and disregards the number of times the word appears in a document. Since some information is lost, we can expect BNB to perform less than MNB for our test examples.

A good example may be table 13.1 from our text where the fifth entry is "Chinese Chinese Chinese Tokyo Japan"

In MNB the word Chinese appearing 3 times increases its chance of indentified as class "yes" whereas BNB will consider only one Chinese and will identify it as a Class belonging to "no" (related to Japan)