MIDS-W261-2016-HW1-Week01-Lane

May 20, 2016

1 DATASCI W261: Machine Learning at Scale

1.1 Assignment Week 1

Jackson Lane (jelane@berkeley.edu) W261-3

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

Big data means data that is too big or complex for traditional data analysis tools. Its distinguishing characteristics include the four Vs. Big data exists in the context of a Big data problem, which may involve performing a certain type of analysis on one or more very large and/or multidimensional datasets in a limited period of time. For example, 50 pages of text alone is probably not big data, but if a human has to read 50 pages in just 5 minutes, then that text becomes Big data in the context of the Big data problem of parsing a large corpus of text in semi-realtime: To achieve the desired speed of analysis, you will need to special tools like NLP techniques or parallelization.

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

Note that the expected total error in any model will always be:

$$E[\hat{y} - y]^2 = Bias^2 + IrreducibleError^2 + Variance$$

To estimate variance for each degree, I'd run at least 100 different models of that degree across random subsamples of the test dataset T. Then for each degree, variance is:

$$E[\hat{y} - h(\hat{y})]^2$$

Where h(y) is the mean value of the models' predictions. In other words, variance is simply the standard deviation squared of the predictions of the 100 models. It can be calculated without knowing the true function or value.

Bias is estimated as:

$$f(y) - h(\hat{y})$$

and irreducible error is estimated as:

$$E[y - f(y)]$$

where f(y) reperesents the value of the true function f and y represents the true value as observed in reality. However, it's difficult to estiamte bias and irreducible error because it requires knowledge of the true function ahead of time. While one can observe the true values from experiments and calculate variance from samples, it's mathematically impossible to derive the true function from just data alone.

But one can still estimate the sum of irreducible error squared and bias squared by subtracting the variance from the model squared error.

$$IrreducibleError^2 + Bias^2 = E[\hat{y} - y]^2 - Variance$$

Furthermore, one can minimize bias by using a high degree polynomial (such as 12). So if one subtracts the variance from the model squared error for a model with polynomial 12, one can get a pretty good estimate of the irreducible error.

$$IrreducibleError^2 + Bias^2_{->0} = IrreducibleError^2 = E[\hat{y} - y]^2 - Variance$$

By defintion, irreducible error should remain constant regardless of which degree polynomial we use. Since we now have an estimate for irreducible error, we can estimate bias for all the other degress by subtracting variance and irreducible error squared from the model squared error.

$$Bias^2 = E[\hat{y} - y]^2 - Variance - IrreducibleError^2$$

But in the end however, I'd still choose my model based off just mean squared error alone. Whether that error can be attributed to bias or variance is not really as relevant to me. However, I'd make sure to keep a set of hold out data that I will only use to test my model against after my model is training. If use every record in the data fr both training and testing, then I'll end up with a model that probably doesn't generalize well to the rest of the population.

1.1.3 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.

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

```
done
```

```
In [2]: %%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 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 STDOUT to disk
####
####
./reducer.py $countfiles > $data.output
####
####

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

```
Overwriting pNaiveBayes.sh
```

1.2 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 [3]: %%writefile mapper.py
        #!/usr/bin/python
        # mapper.py
        # Author: Jackson Lane
        # Description: mapper code for HW1.2-1.5

        import sys
        import re
        # collect user input
        filename = sys.argv[1]
        findword = sys.argv[2].lower()
        file = open (filename, "r")
        for line in file.readlines() :
            line = re.sub('\n','',line)
            #Parse each line and get the textual part of the e-mail
            [email,spam,subject,body] = re.split("\t",line)
            data = body.lower() + subject.lower()
            #Split each email into a list of words
            words = re.split('\W+',data)
            for word in words:
                #Emit if a word matches the findword
                if (word == findword): print word , 1
```

```
Overwriting mapper.py
```

```
In [4]: %%writefile reducer.py
        #!/usr/bin/python
        # reducer.py
        # Author: Jackson Lane
        # Description: reducer code for HW1.2

        import sys
        import re

        countfiles = sys.argv[1:len(sys.argv)]
        word = ""
        total = 0

        # loop over the files produced by the mapper
        for filename in countfiles:
            with open (filename, "r") as myfile:
                for line in myfile.readlines():
                    #Get rid of the newline character strangly
                    line = re.sub('\n','',line)
                    [word,count] = line.split()
                    #'count' is the number of times that the mapper said this word appeared
                    #It will always be 1 for this assignment,
                    #but might vary once we start using combiners in week 3
                    count = int(count)
                    #Add 1 to total number of instances of a word
                    total += count

        print "The word \"" , word, "\" appeared " ,total ," times in the dataset."
```

```
Overwriting reducer.py
```

1.2.1 The call for HW1.2:

Note that the word assistance appears only 9 times total in the email's contents, and 1 time in the email's subjects. This implementation and those below focus strictly on the content.

```
In [5]: # change permissions and run the naive bayes shell script
        !chmod +x mapper.py; chmod +x reducer.py
        !chmod +x pNaiveBayes.sh;
        !./pNaiveBayes.sh 4 "assistance"; cat enronemail_1h.txt.output
```

```
The word " assistance " appeared  10  times in the dataset.
```

1.2.2 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 [6]: %%writefile mapper.py
        #!/usr/bin/python
        # mapper.py
        # Author: Jackson Lane
        # Description: mapper code for HW1.3-1.5

        import sys
        import re

        filename = sys.argv[1]
        # get list of user specified words
        findwords = []
        if(len(sys.argv) > 2):
            findwords = re.split(" ",sys.argv[2].lower())

        file = open (filename, "r")
        for line in file.readlines() :
            line = re.sub('\n','',line)
            [email,spam,subject,content] = re.split("\t",line)
            data = content.lower() + subject.lower()
            words = re.split('\W+',data)
            for word in words:
                #Flag if word is in findwords list
                if word in findwords:
                    flag=1
                else: flag =0
                #Emit ID, label, word, and flag
                print [email,spam,word,flag]
```

```
Overwriting mapper.py
```

```
In [7]: %%writefile reducer.py
        #!/usr/bin/python
        #HW 1.3 - Reducer function
        #Description: Reducer code for HW 1.3 - 1.4

        #HW 1.3 - Reducer Function Code
        from __future__ import division #Python 3-style division syntax is much cleaner
        import sys

        words={}
        emails={}
        filenames = sys.argv[1:]
        spam_email_count=0 #number of spam emails
        ham_email_count=0 #number of ham emails
        spam_word_count=0 #number of words in spam emails
        ham_word_count=0 #number of words in ham emails
        for file in filenames:
            # Train classifier with data from mapper.py
            with open(file, "r") as opened:
                for line in opened.readlines():
```

```
            #parse the incoming line
            [email,spam,word,flag]=eval(line)
            spam=int(spam)
            flag=int(flag)

            # If a word is flagged, then record whether it appeared in spam or ham
            if flag==1:
                if spam==1:
                    words.setdefault(word,{'ham_count':0,'spam_count':0})["spam_count"]+=1
                else:
                    words.setdefault(word,{'ham_count':0,'spam_count':0})["ham_count"]+=1

            # Count total number of words in each class
            if spam==1:
                  spam_word_count+=1
            else:
                ham_word_count+=1

            #Count total number of e-mails in each class
            if email not in emails:
                if spam==1:
                    spam_email_count+=1
                else:
                    ham_email_count+=1
                emails[email] = {'spam':spam,'word_count':0,'words':[]}
            emails[email]['words'].append(word)
            emails[email]['word_count']+=1

#Calculate priors
prior_spam=spam_email_count/len(emails)
prior_ham=ham_email_count / len(emails)
for k,word in words.iteritems():
    word['p_spam']=(word['spam_count'])/(spam_word_count)
    word['p_ham']=(word['ham_count'])/(ham_word_count)

#Accuracy here refers to amount gotten right
accuracy =0
for j,email in emails.iteritems():
    p_spam=prior_spam
    p_ham=prior_ham
    for word in email['words']:
        if word in words:
            #Multiply priors by conditional probabilities to get posteriors.
            p_spam*=(words[word]['p_spam'])
            p_ham*=(words[word]['p_ham'])
    if p_spam>p_ham:
        spam_pred=1
    else:
        spam_pred=0
    #Increment accuracy count if made correct predictin
    if (spam_pred == email['spam']): accuracy += 1
    #Print prediction vs actual
    print j,'\t',email['spam'],'\t',spam_pred
```

```
        print "Accuracy: ", accuracy / len(emails)
```

```
Overwriting reducer.py
```

1.2.3 The call for HW1.3:

```
In [8]: #Run our HW 1.3 code and check the results in the output file
        !chmod a+x mapper.py reducer.py
        !./pNaiveBayes.sh 5 "assistance"
        !echo "HW 1.3 - Results"
        !cat enronemail_1h.txt.output
```

```
HW 1.3 - Results
0010.2003-12-18.GP         1         0
0010.2001-06-28.SA_and_HP         1         1
0001.2000-01-17.beck         0         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         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         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
Accuracy:  0.6
```

1.3 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)

1.3.1 The call for HW1.4:

```
In [9]: #Uses same mapper and reducer as previous problem.
        #Run our HW 1.4 code and check the results in the output file
        !chmod a+x mapper.py reducer.py
        !./pNaiveBayes.sh 5 "assistance valium enlargementWithATypo"
        !echo "HW 1.4 - Results"
        !cat enronemail_1h.txt.output
```

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

```
0014.1999-12-14.kaminski         0         0
0006.1999-12-13.kaminski         0         0
0011.1999-12-14.farmer         0         0
0013.1999-12-14.kaminski         0         0
0001.2001-02-07.kitchen         0         0
0008.2001-02-09.kitchen         0         0
0007.2003-12-18.GP         1         0
0017.2004-08-02.BG         1         0
0014.2004-08-01.BG         1         0
0006.2003-12-18.GP         1         0
0016.2001-07-05.SA_and_HP         1         0
0008.2003-12-18.GP         1         0
0014.2001-07-04.SA_and_HP         1         0
0001.2001-04-02.williams         0         0
0012.2000-06-08.lokay         0         0
0014.1999-12-15.farmer         0         0
0009.2000-06-07.lokay         0         0
0001.1999-12-10.farmer         0         0
0008.2001-06-25.SA_and_HP         1         0
0017.2001-04-03.williams         0         0
0014.2001-02-12.kitchen         0         0
0016.2001-07-06.SA_and_HP         1         0
0015.1999-12-15.farmer         0         0
0009.1999-12-13.kaminski         0         0
0001.2000-06-06.lokay         0         0
0011.2004-08-01.BG         1         0
0004.2004-08-01.BG         1         0
0018.2003-12-18.GP         1         1
0002.1999-12-13.farmer         0         0
0016.2003-12-19.GP         1         1
0004.1999-12-14.farmer         0         0
0015.2003-12-19.GP         1         0
0006.2004-08-01.BG         1         0
0009.2003-12-18.GP         1         1
0007.1999-12-14.farmer         0         0
0005.2000-06-06.lokay         0         0
0010.1999-12-14.kaminski         0         0
0007.2000-01-17.beck         0         0
0003.1999-12-14.farmer         0         0
0003.2004-08-01.BG         1         0
0017.2004-08-01.BG         1         1
0013.2001-06-30.SA_and_HP         1         0
0003.1999-12-10.kaminski         0         0
0012.1999-12-14.farmer         0         0
0004.1999-12-10.kaminski         0         1
0018.2001-07-13.SA_and_HP         1         1
0002.2001-02-07.kitchen         0         0
0007.2004-08-01.BG         1         0
0012.1999-12-14.kaminski         0         0
0005.2001-06-23.SA_and_HP         1         0
0007.1999-12-13.kaminski         0         0
0017.2000-01-17.beck         0         0
0006.2001-06-25.SA_and_HP         1         0
0006.2001-04-03.williams         0         0
```

```
0005.2001-02-08.kitchen         0         0
0002.2003-12-18.GP         1         0
0003.2003-12-18.GP         1         0
0013.2001-04-03.williams         0         0
0004.2001-04-02.williams         0         0
0010.2001-02-09.kitchen         0         0
0001.1999-12-10.kaminski         0         0
0013.1999-12-14.farmer         0         0
0015.1999-12-14.kaminski         0         0
0012.2003-12-19.GP         1         0
0016.2001-02-12.kitchen         0         0
0002.2004-08-01.BG         1         1
0002.2001-05-25.SA_and_HP         1         0
0011.2003-12-18.GP         1         0
Accuracy:  0.63
```

1.4 HW1.5. Provide a mapper/reducer pair that, when executed by pNaive-Bayes.sh will classify the email messages by all words present.

```
In [10]: %%writefile reducer.py
         #!/usr/bin/python
         #HW 1.3 - Reducer function
         #Same as reducer for 1.3,1.4
         #Except that this time we don't take into account whether a word is flagged
         from __future__ import division #Python 3-style division syntax is much cleaner
         import sys, math

         words={}
         emails={}
         filenames = sys.argv[1:]
         spam_email_count=0 #number of spam emails
         ham_email_count=0 #number of ham emails
         spam_word_count=0 #number of words in spam emails
         ham_word_count=0 #number of words in ham emails
         for file in filenames:
             # Train classifier with data from mapper.py
             with open(file, "r") as opened:
                 for line in opened.readlines():

                     #parse the incoming line
                     line=eval(line)
                     email=line[0]
                     spam=int(line[1])
                     word=line[2]
                     flag=int(line[3])
                     if spam==1:
                         #using +1 smoothing
                         words.setdefault(word,{'ham_count':1,'spam_count':1})["spam_count"]+=1
                         spam_word_count+=1
                     else:
                         #using +1 smoothing
                         words.setdefault(word,{'ham_count':1,'spam_count':1})["ham_count"]+=1
                         ham_word_count+=1
                     #store email data
```

```
                    if(email not in emails.keys()):
                        if spam==1:
                            spam_email_count+=1
                        else:
                            ham_email_count+=1
                        emails[email] = {'spam':spam,'word_count':0,'words':[]}
                    emails[email]['words'].append(word)
                    emails[email]['word_count']+=1

        #Calculate priors
        prior_spam=spam_email_count/len(emails)
        prior_ham=ham_email_count / len(emails)
        for k,word in words.iteritems():
            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 predictions
        accuracy =0
        for j,email in emails.iteritems():
            p_spam=prior_spam
            p_ham=prior_ham
            for word in email['words']:
                if word in words:
        #Since there are so many words, the posteriors are going to be really low.
        # So we need to use log to compute the posteriors.  Otherwise, we'll get underflow errors
                    try:
                        p_spam+=math.log((words[word]['p_spam']))
                        p_ham+=math.log((words[word]['p_ham']))
                    except ValueError:
                        raise #theoretically, this shouldn't happen since we have smoothing
            if p_spam>p_ham:
                spam_pred=1
            else:
                spam_pred=0
            if (spam_pred == email['spam']): accuracy += 1
            print j,'\t',email['spam'],'\t',spam_pred

        print "Accuracy: ", accuracy / len(emails)
```

```
Overwriting reducer.py
```

1.4.1 The call for HW1.5:

```
In [11]: #Uses same mapper and reducer as previous problem.
         #Run our HW 1.4 code and check the results in the output file
         !chmod a+x mapper.py reducer.py
         !./pNaiveBayes.sh 5
         !echo "HW 1.5 - Results"
         !cat enronemail_1h.txt.output
```

```
HW 1.5 - Results
0010.2003-12-18.GP          1          1
0010.2001-06-28.SA_and_HP          1          1
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         1
0008.2004-08-01.BG         1         1
0009.1999-12-14.farmer         0         0
0017.2003-12-18.GP         1         1
0011.2001-06-28.SA_and_HP         1         1
0015.2001-07-05.SA_and_HP         1         1
0015.2001-02-12.kitchen         0         0
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         1
0008.2001-06-12.SA_and_HP         1         1
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         1
0005.2003-12-18.GP         1         1
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         1
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         1
0017.2004-08-02.BG         1         1
0014.2004-08-01.BG         1         1
0006.2003-12-18.GP         1         1
0016.2001-07-05.SA_and_HP         1         1
0008.2003-12-18.GP         1         1
0014.2001-07-04.SA_and_HP         1         1
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         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         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          1
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          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          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          1
0017.2004-08-01.BG         1          1
0013.2001-06-30.SA_and_HP         1          1
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          1
0002.2001-02-07.kitchen         0          0
0007.2004-08-01.BG         1          1
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          1
0003.2003-12-18.GP         1          1
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          1
0002.2001-05-25.SA_and_HP         1          1
0011.2003-12-18.GP         1          1
Accuracy:  1.0
```

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

```
In [12]: import re
         import pandas as pd
         import numpy as np
         from sklearn.naive_bayes import MultinomialNB, BernoulliNB
         from sklearn.feature_extraction.text import CountVectorizer

         data = np.array([])
```

```
labels = np.array([])
opened = open ("enronemail_1h.txt", "r")
for line in opened.readlines():
        [email,spam,subject,content] = re.split("\t",line)
        text = content.lower() + subject.lower()
        data=np.append(data,[text])
        labels=np.append(labels,[spam])

# Create features for train and dev data
vectorizer = CountVectorizer()
trainingData = vectorizer.fit_transform(data)
#Data frame that I will use to show training errors
d = pd.DataFrame({"Model":[],"Training Error":[]})
classifier = BernoulliNB()
classifier.fit(trainingData, labels)
#Training error is 1- accuracy score
d.loc[1]=["Bernoulli NB", 1-classifier.score(trainingData,labels)]
classifier = MultinomialNB()
classifier.fit(trainingData, labels)
d.loc[2]=["Multinomial NB", 1-classifier.score(trainingData,labels)]
#I'm hardcoding the error rate I got from HW1.5
d.loc[3]=["HW1.5 Model", 0]

print d
```

```
Model  Training Error
1    Bernoulli NB            0.16
2  Multinomial NB            0.00
3     HW1.5 Model            0.00
```

The sklearn Multinomial NB mode and the model developed in HW 1.5 model both perform much better than the sklearn Bernoulli NB classifier. This is likely because they are both Multinomial models while the Bernoulli model is a different algorithm. I think that Multinomial models take into account the frequency of each class in the training data, meaning that it will generally bias more towards ham than spam.

The training error between the sklearn Multinomial and the HW 1.5 model is the same. This makes sense because the two models and feature sets should theoretically be the same. Both parse words based on spaces and then use log addition to calculate posterior probabilities. Both also use +1 smoothing.

```
In [ ]:
```