HW1 DATASCI W261: Machine Learning at Scale

- **Name:** Megan Jasek
- **Email:** meganjasek@ischool.berkeley.edu
- **Class Name:** W261-2
- **Week Number:** 1
- **Date:** 5/16/16

HW0.0: Prepare your bio and include it in this HW submission. Please limit to 100 words. Count the words in your bio and print the length of your bio (in terms of words) in a separate cell.

Bio: I am currently working on my Master of Information and Data Science (MIDS) degree at UC Berkeley (expected graduation August 2016). My courses include Machine Learning at Scale, Statistics, Data Analysis and Data Storage and Retrieval. I have a bachelor's and master's degree in computer science from MIT and 10 years work experience as a technical project manager at Trilogy Software (proprietary software company), Nordstrom.com and Washington Mutual Bank. I live in Seattle, Washington where I enjoy improvisation comedy and biking.

Bio word length: 81 words

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

Big data is a term used in the digital age to describe the large volumes of data that are now be collected by organizations. This data includes click streams, log files and mathematical results. The term is usually described in relation to 3 categories: volume, velocity and variety.

Volume: Every day the world is creating approximately 10^15 bytes of data. Ninety percent of the data that exists today was created in the last 2 years. This data comes in the form of sensor data, log files, videos, images, sales transactions, social media and more.

Velocity: Although batch jobs analyzing massive quantities of data are still run at off-peak times, data collection is all occurring real-time from social media sites and log files and other sources. Organizations want to analyze this data in real-time.

Variety: Data is coming from different sources and can occur in different formats. Data could be structured with a regular schema as is found in a relational database or data could be unstructured like emails and audio and video files.

It is common to think of big data as an amount of data that currently overwhelms the systems in place in terms of storage, processing and cost. As the world deals with the new amount, speed and type of data being generated, new technologies are being created to store and process it.

An example of a company using big data is insurance companies putting sensors in cars and recording their customer's driving habits like speed or the amount of pressure used for braking. Depending on how frequently the sensor output data, the amount of data collected per driver could be very large. They can then process and analyze this data and create new metrics to use for assessing a customer's insurance risk. One such example would be to find out if a customer regularly drives the speed limit. It could be argued that if a person regularly exceeds the speed limit, then they are not as safe of a driver as a person who follows the speed limit. Drivers that are less safe could be a bigger insurance risk and could be charged higher rates.

HW1.0.1. Bias Variance. 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?

Bias, variance and irreducible error are typically estimated over multiple different datasets obtained from the same distribution. Since there is only one dataset here, bootstrapping will be used. I don't know the size of dataset T, but I will assume that it is greater than 100 and has size N. I will take 100 bootstrap samples with replacement. This means that I will create 100 samples with replacement that are each of size N from the original dataset.

For each of the N datasets, I will fit 5 models: M1, M2, M3, M4 and M5. Each model is a regression model. M1, M2, M3, M4 and M5 have degrees 1, 2, 3, 4 and 5, respectively.

For each model I will calculate the bias, variance and irreducible error according to the following formulas, where g(x) is the is the model, f(x) is the true mean and y is the y values in the dataset.

bias = E[g(x)] - f(x)

variance= E[(g(x)- E[g(x)])^2]

irreducible error = E[(y-f(x))^2]

I would select a model that minimizes the expected prediction error which is defined by

Expected prediction error = variance + bias^2 + irreducible error

As model complexity goes up, variance goes up and bias goes down. The best models are the ones where both bias and variance are minimized. I would create a graph of the bias, variance and irreducible error for each of the models. Then I would find the model where the expected prediction error is the lowest. I would expect that model to be around a polynomial order of 3 since this model would have a balance of 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.

In [1]:
```
print('done')
```
```
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.

The results from running the command line code './pNaiveBayes.sh 4 "assistance"' are as follows:

Total number of occurances of "assistance" is: 10

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

# Import sys library to access arguments passed in when the module is called
import sys
# Import re library to manipulate regular expressions
import re
# Create a variable to store the count of the word that is passed in to the function
# Initialize this count to 0.
count = 0
# Define the regular expression used to designate what a 'word' is among a string of
# characters.
WORD_RE = re.compile(r"[\w']+")

## collect user input
# filename is the 1st argument passed in to the module and is the name of the file
# that is to be analyzed.
filename = sys.argv[1]
# findword is the 2nd argument passed in to the module and is the word whose
# occurances will be counted.  Convert the string to lowercase.
findword = sys.argv[2].lower()

# Open the file passed in to the module
with open (filename, "r") as myfile:
    # Loop through each line of the file.
    for line in myfile.readlines():
        # Create a list of words found in each line
        words = re.findall(WORD_RE, line.lower())
        # Count the number of occruances of findword in the words list.
        count += words.count(findword)

# Print the name of the word and the number of occurances of the word.  These values
# will be passed to reducer.py
print('%s %d' % (findword, count))
```
```
Overwriting mapper.py
```

In [3]:

```
%%writefile reducer.py
#!/usr/bin/python
## reducer.py
## Author: Megan Jasek
## Description: reducer code for HW1.2

# Import sys library to access arguments passed in when the module is called
import sys

# Create a variable to store the total number occurances of the word that was given by the user
# Initialize this total to 0.
total = 0

# Loop through each of the files that are passed into the module.
for i in range(1, len(sys.argv)):
    # Each argument that was passed in is a filename.
    filename = sys.argv[i]
    # Open the file
    with open (filename, "r") as myfile:
        # Loop through each line of the file.
        for line in myfile.readlines():
            # Split the line by spaces
            words = line.split(" ")
            # The 1st value in the line is the word that is being analyzed.
            findword = words[0]
            # The 2nd value in the line is the count of the word from each file.  Add the count from
            # this line to the total count variable.
            total += int(words[1])

# Print the word that is being analyzed as well as the total number of occurances of the word.
print('Total number of occurances of %s is: %d' % (findword, total))
```

```
Overwriting reducer.py
```

In [4]:

```
# Change the permissions for mapper.py, reducer.py and pNaiveBayes.sh so that they include
# execute permissions.
!chmod +x mapper.py
!chmod +x reducer.py
!chmod a+x pNaiveBayes.sh
```

In [5]:

```
# Usage: pNaiveBayes.sh m wordlist
!./pNaiveBayes.sh 4 "assistance"
```

In [6]:

```
# Report the results from HW1.2 be examining the output file: 'enronemail_1h.txt.output'
!cat enronemail_1h.txt.output
!rm enronemail_1h.txt.*
```

```
Total number of occurances of assistance is: 10
```

HW1.3. HW1.3. Provide a mapper/reducer pair that, when executed by pNaiveBayes.sh will classify the email messages by a single, user-specified word using the multinomial Naive Bayes Formulation. Examine the word “assistance” and report your results.

The results from running the command line code './pNaiveBayes.sh 4 "assistance"' are that as follows:

The program had an error rate of 40%. It classified 40 documents incorrectly.

In [7]:

```
%%writefile mapper.py
#!/usr/bin/python
## mapper.py
## Author: Megan Jasek
## Description: mapper code for HW1.3

# Import sys library to access arguments passed in when the module is called
import sys
# Import re library to manipulate regular expressions
import re
# Define the regular expression used to designate what a 'word' is among a string of
# characters.
WORD_RE = re.compile(r"[\w']+")

## collect user input
# filename is the 1st argument passed in to the module and is the name of the file
# that is to be analyzed.
filename = sys.argv[1]
# vocab is the 2nd argument passed in to the module and is the word that will be used
# to classify the documents.  Convert the string to lowercase.
vocab = sys.argv[2].lower()

# Initialize counts.  All of the variables are lists storing 2 counts.  The 1st count
# in the list is the count for class 0 which is Ham and the 2nd count is the count
# for class 1 which is Spam.
# Variable count_docs:  stores the total number of documents for each class.
# Variable count_words:  stores the total number of words in the subject and body fields
# for each class.
# Variable count_vocab_terms:  stores the total number of occurances of the word that is
# being used for classification in each of the classes.
count_docs = [0, 0]
count_words = [0, 0]
count_vocab_terms = [0, 0]

# Open the file that was passed in
with open (filename, "r") as myfile:
    # For each line of the file
    for line in myfile.readlines():
        # Split the line tabs and store the result in the 'items' list.
        items = line.split('\t')
        # The 1st value of the line is the ID of the document
        ID = items[0]
        # The 2nd value of the line is the true classification of the document called: TRUTH
        TRUTH = int(items[1])
        # The 3rd value of the line is the subject.  Convert this value to lowercase and then
        # break it up into separate words using the WORD_RE regular expression
        words_subject = re.findall(WORD_RE, items[2].lower())
        # The 4th value of the line is the body.  Convert this value to lowercase and then
        # break it up into separate words using the WORD_RE regular expression
        words_body = re.findall(WORD_RE, items[3].lower())
        # Concatenate the subject and body in to one list called 'words_all'
        words_all = words_subject + words_body
        # Update 'count_docs' by 1 for the class of this document
        count_docs[TRUTH] += 1
        # Update 'count_words' by the length of the 'words_all' list for the class of this document
        count_words[TRUTH] += len(words_all)
        # Store the number of occurances of the vocab word in the document
        vocab_terms_in_doc = words_all.count(vocab)
        # Update 'count_vocab_terms' for the class of this document
        count_vocab_terms[TRUTH] += vocab_terms_in_doc
        # Print the ID, TRUTH and count of the vocab word that get passed to reducer.py
        print('%s\t%s\t%d' % (ID, TRUTH, vocab_terms_in_doc))

# Print the count totals for each class that get passed to reducer.py
print('count_docs\t%d\t%d' % (count_docs[0], count_docs[1]))
print('count_words\t%d\t%d' % (count_words[0], count_words[1]))
print('count_vocab_terms\t%d\t%d' % (count_vocab_terms[0], count_vocab_terms[1]))
```

```
Overwriting mapper.py
```

In [8]:

```
%%writefile reducer.py
#!/usr/bin/python
## reducer.py
## Author: Megan Jasek
## Description: reducer code for HW1.3

# Import sys library to access arguments passed in when the module is called
import sys
# Import the math library in order to use the 'log' method to take logorithms
import math

# Initialize a dictionary 'all_docs' which will store the necessary information for each
# document that will be passed to the reducer.py.
all_docs = {}
# Initialize counts.  All of the variables are lists storing 2 counts.  The 1st count
# in the list is the count for class 0 which is Ham and the 2nd count is the count
# for class 1 which is Spam.
# Variable count_docs:  stores the total number of documents for each class.
# Variable count_words:  stores the total number of words in the subject and body fields
# for each class.
# Variable count_vocab_terms:  stores the total number of occurances of the word that is
# being used for classification in each of the classes.
count_docs = [0, 0]
count_words = [0, 0]
count_vocab_terms = [0, 0]

# Loop through each of the files that are passed into the module.
for i in range(1, len(sys.argv)):
    # Each argument that was passed in is a filename.
    filename = sys.argv[i]
    # Open the file
    with open (filename, "r") as myfile:
        # For each line in the file
        for line in myfile.readlines():
            # Split the line by tabs
            elements = line.split("\t")
            # The 1st value in the line is the ID of the document
            ID = elements[0]
            # If the ID is one of the labels that reports the counts of an entire file, the
            # store that value appropriately.
            if ID[:6] == 'count_':
                # Update the 'count_docs' variable with the numbers from the file for each class
                if ID == 'count_docs':
                    count_docs[0] += int(elements[1])
                    count_docs[1] += int(elements[2])
                # Update the 'count_words' variable with the numbers from the file for each class
                elif ID == 'count_words':
                    count_words[0] += int(elements[1])
                    count_words[1] += int(elements[2])
                # Update the 'count_vocab_terms' variable with the numbers from the file for each class
                else:
                    count_vocab_terms[0] += int(elements[1])
                    count_vocab_terms[1] += int(elements[2])
            else:
                # Save the document in the all_docs dictionary for later use.  Use the ID as the key
                # for the dictionary.  Store the TRUTH value and the # of occurances of the vocab
                # word in a list.
                TRUTH = int(elements[1])
                vocab_terms_in_doc = int(elements[2])
                all_docs[ID] = [TRUTH, vocab_terms_in_doc]

# Create a list to store the 2 classes in this problem:  0 for Ham and 1 for Spam.
classes = [0, 1]
# Calculate the total number of documents by adding the total counts from each class.
total_docs = sum(count_docs)
# Initialize the 'prior' varialbe to store the prior probability for each class.
prior = [0, 0]
# Initialize the 'condprob' varialbe to store the conditional probilitity of the vocab word for
# each class.
condprob = [0, 0]

# Train the MultinomialNB classifer using the algorithm in the book: An Introduction to Information Retrieval
# By Christopher D. Manning, Prabhakar Raghavan & Hinrich Schutzepage page 260, Figure 13.2.
# For each of the classes:
# 1. Calculate the prior probability (prior) by dividing the total # of documents in that class by the total # of documents.
# 2. Calculate the conditional probability (condprob) by dividing the total # of terms in that class by the
# total number of terms in all document in that class
# Do not use smoothing.
for cls in classes:
    prior[cls] = count_docs[cls] / float(total_docs)
    condprob[cls] = count_vocab_terms[cls] / float(count_words[cls])
```

```
Overwriting reducer.py
```

```
# Initialize variables to track the total correct predictions and the total predictions
count_correct = 0
```

```
count_total = 0
# Apply the MultinomialNB classifier to each document and make a prediction for each.
# For each document:
# 1. Initialize a score of 0 for each class
# 2. Store the true value for that document in the 'TRUTH' variable
# 3. Store the # of times the vocab term occurs in the document
# 4. For each class: calculate the score as a sum of the log of the prior probability +
# (# of times the vocab term occurs in the document) * log of the conditional probability for the
# vocab term.
# 5. Make a prediction:  Choose the class with the highest score.
# 6. Print the values to the output file:  ID, true value, prediction
for key, value in all_docs.iteritems():
    score = [0, 0]
    TRUTH = value[0]
    vocab_terms_in_doc = value[1]
    for cls in classes:
        score[cls] = math.log(prior[cls])
        score[cls] += vocab_terms_in_doc * math.log(condprob[cls])
    prediction = 1 if (score[1] > score[0]) else 0
    print('%s\t%d\t%d' % (key, TRUTH, prediction))
    # Update the totals used for accuracy
    count_total += 1
    if TRUTH == prediction:
        count_correct += 1
print('Training Error: %f' % ((count_total - count_correct) / float(count_total)))
```

In [9]:

```
# Usage: pNaiveBayes.sh m wordList
!./pNaiveBayes.sh 4 "assistance"
```

In [10]:

```
# Report the results from HW1.3 be examining the output file: 'enronemail_1h.txt.output'
!cat enronemail_1h.txt.output
!rm enronemail_1h.txt.*
```

```
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
Training Error: 0.400000
```

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).

The results from running the command line code './pNaiveBayes.sh 4 "assistance valium enlargementWithATypo"' are that as follows:

- The program had an accuracy of 60%. It classified 60 documents correctly.
- The program had an error rate of 40%. It classified 40 documents incorrectly.

In [11]:

```
%%writefile mapper.py
#!/usr/bin/python
## mapper.py
## Author: Megan Jasek
## Description: mapper code for HW1.4

# Import pring function from python 3
from __future__ import print_function
# Import sys library to access arguments passed in when the module is called
import sys
# Import re library to manipulate regular expressions
import re
# Define the regular expression used to designate what a 'word' is among a string of
# characters.
WORD_RE = re.compile(r"[\w']+")

## collect user input
# filename is the 1st argument passed in to the module and is the name of the file
# that is to be analyzed.
filename = sys.argv[1]
# vocab is the 2nd argument passed in to the module and is the list of words that will be used
# to classify the documents.  Convert the string to lowercase and create a list called vocab
# that stores each word passed in.
vocab = re.split(" ",sys.argv[2].lower())

# Initialize counts.  All of the variables are lists storing 2 counts.  The 1st count
# in the list is the count for class 0 which is Ham and the 2nd count is the count
# for class 1 which is Spam.
# Variable count_docs:  stores the total number of documents for each class.
# Variable count_words:  stores the total number of words in the subject and body fields
# for each class.
# Variable count_vocab_terms:  stores the total number of occurances of the words in the
# vocab that is being used for classification in each of the classes.
count_docs = [0, 0]
count_words = [0, 0]
count_vocab_terms = {}
for term in vocab:
    count_vocab_terms[term] = [0, 0]

# Open the file that was passed in
with open (filename, "r") as myfile:
    # For each line of the file
    for line in myfile.readlines():
        # Split the line tabs and store the result in the 'items' list.
        items = line.split('\t')
        # The 1st value of the line is the ID of the document
        ID = items[0]
        # The 2nd value of the line is the true classification of the document called: TRUTH
        TRUTH = int(items[1])
        # The 3rd value of the line is the subject.  Convert this value to lowercase and then
        # break it up into separate words using the WORD_RE regular expression
        words_subject = re.findall(WORD_RE, items[2].lower())
        # The 4th value of the line is the body.  Convert this value to lowercase and then
        # break it up into separate words using the WORD_RE regular expression
        words_body = re.findall(WORD_RE, items[3].lower())
        # Concatenate the subject and body in to one list called 'words_all'
        words_all = words_subject + words_body
        # Update 'count_docs' by 1 for the class of this document
        count_docs[TRUTH] += 1
        # Update 'count_words' by the length of the 'words_all' list for the class of this document
        count_words[TRUTH] += len(words_all)
        # Print the ID and true classification to send reducer.py
        print('%s\t%s' % (ID, TRUTH),end="")
        # For each of the words in the vocabulary print out the 'term' and the '# of occurances
        # of the term' for reducer.py
        for term in vocab:
            # Store the number of occurances of the vocab word in the document
            count_term = words_all.count(term)
            # Update 'count_vocab_terms' for the class of this document
            count_vocab_terms[term][TRUTH] += count_term
            print('\t%s\t%d' % (term, count_term),end="")
        print()

# Print the count totals for each class that get passed to reducer.py
print('count_docs\t%d\t%d' % (count_docs[0], count_docs[1]))
print('count_words\t%d\t%d' % (count_words[0], count_words[1]))
for term in vocab:
    print('count_vocab_term\t%s\t%d\t%d' % (term, count_vocab_terms[term][0], count_vocab_terms[term][1]))
```

```
Overwriting mapper.py
```

In [12]:

```
%%writefile reducer.py
#!/usr/bin/python
## reducer.py
## Author: Megan Jasek
## Description: reducer code for HW1.4

# Import sys library to access arguments passed in when the module is called
import sys
# Import the math library in order to use the 'log' method to take logorithms
import math

# Initialize a dictionary 'all_docs' which will store the necessary information for each
# document that will be passed to the reducer.py.
all_docs = {}
# Initialize counts.  All of the variables are lists storing 2 counts.  The 1st count
# in the list is the count for class 0 which is Ham and the 2nd count is the count
# for class 1 which is Spam.
# Variable count_docs:  stores the total number of documents for each class.
# Variable count_words:  stores the total number of words in the subject and body fields
# for each class.
# Variable count_vocab_terms:  stores the total number of occurances of the each of the
# words in the vocab that is being used for classification in each of the classes.
count_docs = [0, 0]
count_words = [0, 0]
count_vocab_terms = {}
vocab = []
# Keep track of the first file in order to just store once the vocab that is being analyzed
FIRST_FILE = True

# Loop through each of the files that are passed into the module.
for i in range(1, len(sys.argv)):
    # Each argument that was passed in is a filename.
    filename = sys.argv[i]
    # Open the file
    with open (filename, "r") as myfile:
        # For each line in the file
        for line in myfile.readlines():
            # Split the line by tabs
            elements = line.split("\t")
            # The 1st value in the line is the ID of the document
            ID = elements[0]
            # If the ID is one of the labels that reports the counts of an entire file, the
            # store that value appropriately.
            if ID[:6] == 'count_':
                # Update the 'count_docs' variable with the numbers from the file for each class
                if ID == 'count_docs':
                    count_docs[0] += int(elements[1])
                    count_docs[1] += int(elements[2])
                # Update the 'count_words' variable with the numbers from the file for each class
                elif ID == 'count_words':
                    count_words[0] += int(elements[1])
                    count_words[1] += int(elements[2])
                # Update the 'count_vocab_terms' variable with the numbers from the file for each class
                else:
                    term = elements[1]
                    if term not in count_vocab_terms:
                        count_vocab_terms[term] = [int(elements[2]), int(elements[3])]
                    else:
                        count_vocab_terms[term][0] += int(elements[2])
                        count_vocab_terms[term][1] += int(elements[3])
                    # if it's the first file, save the vocab
                    if FIRST_FILE:
                        vocab.append(term)
            else:
                # Save the document in the all_docs dictionary for later use.  Use the ID as the key
                # for the dictionary.  Store the TRUTH value and a dictionary consisting of the each
                # term in the vocab followed by # of occurances of that term
                TRUTH = int(elements[1])
                terms = {}
                for j in range(2, len(elements),2):
                    term = elements[j]
                    terms[term] = int(elements[j+1])
                all_docs[ID] = [TRUTH, terms]
    # Update the FIRST_FILE variable after the first file is read.
    FIRST_FILE = False

# Create a list to store the 2 classes in this problem:  0 for Ham and 1 for Spam.
classes = [0, 1]
# Calculate the total number of documents by adding the total counts from each class.
total_docs = sum(count_docs)
# Initialize the 'prior' varialbe to store the prior probability for each class.
prior = [0, 0]
# Initialize the 'condprob' varialbe to store the conditional probilitity of each of the
# terms in the vocab each class.
```

```
condprob = {}
for term in vocab:
    condprob[term] = [0, 0]

# Train the MultinomialNB classifer using the algorithm in the book: An Introduction to Information Retrieval
# By Christopher D. Manning, Prabhakar Raghavan & Hinrich Schutzepage page 260, Figure 13.2.
# For each of the classes:
# 1. Calculate the prior probability (prior) by dividing the total # of documents in that class by the total # of documents.
# 2. Calculate the conditional probability (condprob) for each term in the vocab by dividing the
# total # of that term in that class by the total number of terms in all documents in that class
# Do not use smoothing.
for cls in classes:
    prior[cls] = count_docs[cls] / float(total_docs)
    for term in vocab:
        condprob[term][cls] = count_vocab_terms[term][cls] / float(count_words[cls])

# Initialize variables to track the total correct predictions and the total predictions
count_correct = 0
count_total = 0
# Apply the MultinomialNB classifier to each document and make a prediction for each.
# For each document:
# 1. Initialize a score of 0 for each class
# 2. Store the true value for that document in the 'TRUTH' variable
# 3. Store the # of times the vocab term occurs in the document
# 4. For each class: calculate the score as a sum of the log of the prior probability +
# for each term in the vocab: (# of times the vocab term occurs in the document) * log of the
# conditional probability for the vocab term.
# 5. Make a prediction:  Choose the class with the highest score.
# 6. Print the values to the output file:  ID, true value, prediction
for ID, value in all_docs.iteritems():
    score = [0, 0]
    TRUTH = value[0]
    vocab_terms_in_doc = value[1]
    for cls in classes:
        score[cls] = math.log(prior[cls])
        for term, count in vocab_terms_in_doc.iteritems():
            if condprob[term][cls] > 0.0:
                score[cls] += (count * math.log(condprob[term][cls]))
    prediction = 1 if (score[1] > score[0]) else 0
    print('%s\t%d\t%d' % (ID, TRUTH, prediction))
    # Update the totals used for accuracy
    count_total += 1
    if TRUTH == prediction:
        count_correct += 1
print('Accuracy: %f' % (count_correct / float(count_total)))
print('Training Error: %f' % ((count_total - count_correct) / float(count_total)))
```

```
Overwriting reducer.py
```

In [13]:

```
# Usage: pNaiveBayes.sh m wordlist
!./pNaiveBayes.sh 4 "assistance valium enlargementWithATypo"
```

In [14]:

```
# Report the results from HW1.4 be examining the output file: 'enronemail_1h.txt.output'
!cat enronemail_1h.txt.output
!rm enronemail_1h.txt.*
```

```
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: 0.600000
Training Error: 0.400000
```

HW1.5. Provide a mapper/reducer pair that, when executed by pNaiveBayes.sh will classify the email messages by all words present. Report the performance of your learnt classifier in terms of misclassifcation error rate of your multinomial Naive Bayes Classifier with smoothing and without smoothing using all the vocabulary on the training set. Please comment on what you see. Explain!

ANSWER: The results from running the command line code './pNaiveBayes.sh 4' are that as follows:

Without smoothing: The program had an error rate of 0%. It classified 0 documents incorrectly. (Run the program with the SMOOTHING flag set to False.)

With smoothing: The program had an error rate of 0%. It classified 0 documents incorrectly. (Run the program with the SMOOTHING flag set to True.)

The reason they are 100% accurate is as follows. There are 4 categories of terms. Category A contains terms that are only contained in Ham documents. Category B contains terms that are only contained in Spam documents. Category C contains terms that are in both Ham and Spam documents. Category D contains terms that are in neither Ham nor Spam documents. Since we are not classifying any new documents, we do not have any terms in Category D.

In this problem, we do not have any documents that contain terms from ONLY Category C either. When the classifier is run without smoothing it counts how many documents encounter 0 probabilities while doing the classification. The count comes out to 100 or all of them. Said another way, each document contains at least one term that exists in exactly one class: the class that that document belongs to. This means that we have 4 types of documents.

- Type 1: Documents that contain terms from Category A only
- Type 2: Documents that contain terms from Category B only
- Type 3: Documents that contain terms from Category A and C
- Type 4: Documents that contain terms from Category B and C

When the classifier is handling documents that only contain terms from Category A (Type 1), then the classification will be Ham because the document does not contain any terms that are found in Spam documents, so even with smoothing the conditional probabilities for the terms in the Ham class will always be higher than the conditional probabilities for the terms in the Spam class.

And for the same reason, when the classifier is handling documents that only contain terms from Category B (Type 2), then the classification will be Spam because the document does not contain any terms that are found in Ham documents, so even with smoothing the conditional probabilities for the terms in the Spam class will always be higher than the conditional probabilities for the terms in the Ham class.

For documents that contain terms from Category A and C (Type 3), several things can happen. Case 1: It could be the case that the number of words in Category C is small compared to the number of words in Category A and the conditional probabilities from Category A provide more weight to the total posterior probability, so essentially Category C words are not making a difference. OR Case 2: It could be the case that the number of words in Category C is not trivial compared to the number of words in Category A in which case the conditional probabilities from words in Category C contribute a non-trivial amount to the total posterior probability. If this is the case then it must be the case that there is enough of a difference between the conditional probabilities of the words in Category C for Spam and Ham that the classifier correctly classifies the Spam and Ham documents. If too many of the terms have similar probabilities then the classifier will not be able to classify the documents properly. And if this is true, a human classifier would not be able to classify the documents correctly either because what is happening is that too many of the terms appear together in Spam and Ham documents and there are not enough distinguishing terms for machine or human to be able to tell the difference. Some actual difference must exist.

The same logic goes for Type 4 documents except reverse the terms Ham and Spam.

In the case of this problem, the conditional probabilities are such that there is enough difference between the Spam and Ham conditional probabilities that the classifier can tell the difference between the documents.

In simpler terms, the reason that the model is 100% accurate is because the set of words, H, that occur frequently in Ham documents does not have a lot of overlap with the set of words, S, that occur frequently in Spam documents. Enough of a difference between which terms appear in Ham or Spam for the classifier (or a human) to tell the two classes apart.

In [15]:

```
%%writefile mapper.py
#!/usr/bin/python
## mapper.py
## Author: Megan Jasek
## Description: mapper code for HW1.5

# Import pring function from python 3
from __future__ import print_function
# Import sys library to access arguments passed in when the module is called
import sys
# Import re library to manipulate regular expressions
import re
# Define the regular expression used to designate what a 'word' is among a string of
# characters.
WORD_RE = re.compile(r"[\w']+")

## collect user input
# filename is the 1st argument passed in to the module and is the name of the file
# that is to be analyzed.
filename = sys.argv[1]

# Initialize counts.  All of the variables are lists storing 2 counts.  The 1st count
# in the list is the count for class 0 which is Ham and the 2nd count is the count
# for class 1 which is Spam.
# Variable count_docs:  stores the total number of documents for each class.
# Variable count_words:  stores the total number of words in the subject and body fields
# for each class.
# Variable count_vocab_terms:  stores the total number of occurances of the words in the
# vocab that is being used for classification in each of the classes.
# Variable vocab:  stores the unique terms used in this file
count_docs = [0, 0]
count_words = [0, 0]
count_vocab_terms = {}
vocab = []

# Open the file that was passed in
with open (filename, "r") as myfile:
    # For each line of the file
    for line in myfile.readlines():
        # Split the line tabs and store the result in the 'items' list.
        items = line.split('\t')
        # The 1st value of the line is the ID of the document
        ID = items[0]
        # The 2nd value of the line is the true classification of the document called: TRUTH
        TRUTH = int(items[1])
        # The 3rd value of the line is the subject.  Convert this value to lowercase and then
        # break it up into separate words using the WORD_RE regular expression
        words_subject = re.findall(WORD_RE, items[2].lower())
        # The 4th value of the line is the body.  Convert this value to lowercase and then
        # break it up into separate words using the WORD_RE regular expression
        words_body = re.findall(WORD_RE, items[3].lower())
        # Concatenate the subject and body in to one list called 'words_all'
        words_all = words_subject + words_body
        # Update 'count_docs' by 1 for the class of this document
        count_docs[TRUTH] += 1
        # Update 'count_words' by the length of the 'words_all' list for the class of this document
        count_words[TRUTH] += len(words_all)
        # Create a variable to store the unique words from this line.
        vocab_line = []
        # Loop through each of the terms in the words of this line
        for term in words_all:
            # Update the vocab for this file
            if term not in vocab:
                vocab.append(term)
            # Update the vocab for this line
            if term not in vocab_line:
                vocab_line.append(term)
            # Initialize a list to store the count of this term
            if term not in count_vocab_terms:
                count_vocab_terms[term] = [0,0]
        # Print the ID and true classification to send reducer.py
        print('%s\t%s' % (ID, TRUTH),end="")
        # For each of the words in the vocabulary print out the 'term' and the '# of occurances
        # of the term' for reducer.py
        for term in vocab_line:
            # Store the number of occurances of the vocab word in the document
            count_term = words_all.count(term)
            # Update 'count_vocab_terms' for the class of this document
            count_vocab_terms[term][TRUTH] += count_term
            print('\t%s\t%d' % (term, count_term),end="")
        print()

# Print the count totals for each class that get passed to reducer.py
print('count_docs\t%d\t%d' % (count_docs[0], count_docs[1]))
print('count_words\t%d\t%d' % (count_words[0], count_words[1]))
for term in vocab:
    print('count_vocab_term\t%s\t%d\t%d' % (term, count_vocab_terms[term][0], count_vocab_terms[term][1]))
```

```
Overwriting mapper.py
```

In [16]:

```
%%writefile reducer.py
#!/usr/bin/python
## reducer.py
## Author: Megan Jasek
## Description: reducer code for HW1.5

# Import sys library to access arguments passed in when the module is called
import sys
# Import the math library in order to use the 'log' method to take logorithms
import math

# Initialize a dictionary 'all_docs' which will store the necessary information for each
# document that will be passed to the reducer.py.
all_docs = {}
# Initialize counts.  All of the variables are lists storing 2 counts.  The 1st count
# in the list is the count for class 0 which is Ham and the 2nd count is the count
# for class 1 which is Spam.
# Variable count_docs:  stores the total number of documents for each class.
# Variable count_words:  stores the total number of words in the subject and body fields
# for each class.
# Variable count_vocab_terms:  stores the total number of occurances of the each of the
# words in the vocab that is being used for classification in each of the classes.
count_docs = [0, 0]
count_words = [0, 0]
count_vocab_terms = {}

# Loop through each of the files that are passed into the module.
for i in range(1, len(sys.argv)):
    # Each argument that was passed in is a filename.
    filename = sys.argv[i]
    # Open the file
    with open (filename, "r") as myfile:
        # For each line in the file
        for line in myfile.readlines():
            # Split the line by tabs
            elements = line.split("\t")
            # The 1st value in the line is the ID of the document
            ID = elements[0]
            # If the ID is one of the labels that reports the counts of an entire file, the
            # store that value appropriately.
            if ID[:6] == 'count_':
                # Update the 'count_docs' variable with the numbers from the file for each class
                if ID == 'count_docs':
                    count_docs[0] += int(elements[1])
                    count_docs[1] += int(elements[2])
                # Update the 'count_words' variable with the numbers from the file for each class
                elif ID == 'count_words':
                    count_words[0] += int(elements[1])
                    count_words[1] += int(elements[2])
                # Update the 'count_vocab_terms' variable with the numbers from the file for each class
                else:
                    term = elements[1]
                    if term not in count_vocab_terms:
                        count_vocab_terms[term] = [int(elements[2]), int(elements[3])]
                    else:
                        count_vocab_terms[term][0] += int(elements[2])
                        count_vocab_terms[term][1] += int(elements[3])
            else:
                # Save the document in the all_docs dictionary for later use.  Use the ID as the key
                # for the dictionary.  Store the TRUTH value and a dictionary consisting of each
                # term in the vocab followed by # of occurances of that term
                TRUTH = int(elements[1])
                terms = {}
                for j in range(2, len(elements),2):
                    term = elements[j]
                    terms[term] = int(elements[j+1])
                all_docs[ID] = [TRUTH, terms]


# Create a list to store the 2 classes in this problem:  0 for Ham and 1 for Spam.
classes = [0, 1]
# Calculate the total number of documents by adding the total counts from each class.
total_docs = sum(count_docs)
# Initialize the 'prior' varialbe to store the prior probability for each class.
prior = [0, 0]
# Initialize the 'condprob' varialbe to store the conditional probilitity of each of the
# terms in the vocab each class.
condprob = {}
for term in count_vocab_terms:
    condprob[term] = [0, 0]

# Train the MultinomialNB classifer using the algorithm in the book: An Introduction to Information Retrieval
# By Christopher D. Manning, Prabhakar Raghavan & Hinrich Schutzepage page 260, Figure 13.2.
# For each of the classes:
```

```
# 1. Calculate the prior probability (prior) by dividing the total # of documents in that class by the total # of documents.
# 2. Calculate the conditional probability (condprob) for each term in the vocab by dividing the
# total # of that term in that class by the total number of terms in all documents in that class
# Use smoothing by adding 1.0 to the numerator and the denominator in the condprob equation.
# Create a variable to control if smoothing is used.
SMOOTHING = True
for cls in classes:
    prior[cls] = count_docs[cls] / float(total_docs)
    for term in count_vocab_terms:
        # If using smoothing, then add 1.0 to the numerator and the denominator in the condprob equation.
        if SMOOTHING:
            condprob[term][cls] = (count_vocab_terms[term][cls]+1.0) / (float(count_words[cls])+1.0)
        # If NOT using smoothing, then leave the numerator and the denominator alone.
        else:
            condprob[term][cls] = (count_vocab_terms[term][cls]) / (float(count_words[cls]))

# Initialize variables to track the total correct predictions and the total predictions
count_correct = 0
count_total = 0
# Apply the MultinomialNB classifier to each document and make a prediction for each.
# For each document:
# 1. Initialize a score of 0 for each class
# 2. Store the true value for that document in the 'TRUTH' variable
# 3. Store the # of times the vocab term occurs in the document
# 4. For each class: calculate the score as a sum of the log of the prior probability +
# for each term in the vocab: (# of times the vocab term occurs in the document) * log of the
# conditional probability for the vocab term.
# 5. Make a prediction:  Choose the class with the highest score.
# 6. Print the values to the output file:  ID, true value, prediction
count_docs_zero_prob = 0
for ID, value in all_docs.iteritems():
    score = [0, 0]
    TRUTH = value[0]
    vocab_terms_in_doc = value[1]
    for cls in classes:
        score[cls] = math.log(prior[cls])
        for term, count in vocab_terms_in_doc.iteritems():
            # If any of the conditional probabilities for the document are 0, then set the score
            # to -100000000 because, in actuality the probabilities are being multiplied, so if one is 0
            # then the whole score will be 0.  If this happens, set the score to a very low number so that
            # this class will have a very low score and won't be selected and break out of the loop
            if condprob[term][cls] == 0.0:
                score[cls] = -1000000000
                count_docs_zero_prob += 1
                #print('%d, term: %s, class: %d' % (count_docs_zero_prob, term, cls))
                break
            else:
                score[cls] += (count * math.log(condprob[term][cls]))
    prediction = 1 if (score[1] > score[0]) else 0
    i += 1
    print('%s\t%d\t%d' % (ID, TRUTH, prediction))
    # Update the totals used for accuracy
    count_total += 1
    if TRUTH == prediction:
        count_correct += 1
print('Accuracy: %f' % (count_correct / float(count_total)))
print('Training Error: %f' % ((count_total - count_correct) / float(count_total)))
if not SMOOTHING:
    print('With NO smoothing, number of documents that contain at least one term with zero probability')
    print('in its non-true class: %d' % (count_docs_zero_prob))
```

```
Overwriting reducer.py
```

In [17]:

```
# Usage: pNaiveBayes.sh m wordlist
!./pNaiveBayes.sh 4
```

In [18]:

```
# Report the results from HW1.5 be examining the output file: 'enronemail_1h.txt.output'
!cat enronemail_1h.txt.output
!rm enronemail_1h.txt.*
```

```
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: 1.000000
Training Error: 0.000000
```

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

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)

ANSWER: MultinomialNB - Scikit Learn Training Error: 0.000000

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

ANSWER: BernoullNB - Scikit Learn Training Error: 0.160000

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

ANSWER: MultinomialNB - Command Line Training Error: 0.000000

1.6.4. Please prepare a table to present your results

ANSWER:

Algorithm	**Training Error**
MultinomialNB - SciKit-Learn	0%
BernoulliNB - SciKit-Learn	16%
MultinomialNB - Command Line	0%

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).

ANSWER: There is no difference between the training error rate with the HW1.5 solution and the SciKit-Learn Multinomial Naive Bayes implemention. The training error rates are the same. This makes sense for the following reasons: 1. Add-1 smoothing was used in both cases. 2. The words fed in to each algorithm were the same. The text was first parsed in the same way and a regular expression was used to break down the text in to words. The same set of words were passed to both the algorithms.

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.

ANSWER: The training error rate for the SciKit-Learn MultinomialNB was 0% but for the SciKit-Learn Bernoulli algorithm it was 16%. In the MultinomialNB model, the number of occurances of a word in a document is used to make the classification and there is no penalty assigned if a word from the vocabulary is not in the document. In the BernouliNB model, the existence of a word in a document (either yes or no) is used to make the classification and there is a penalty assigned if a word from the vocabulary is not in the document. Given these definitions, since the training error rate is higher for the Bernoulli model, it means that the frequency of a word in a document makes a difference in it's classification. And it also means that the words that are present in the document make more of a difference in the classification that the words that are not present.

In [19]:

```
# Import SK-learn libraries for learning.
from sklearn.naive_bayes import BernoulliNB
from sklearn.naive_bayes import MultinomialNB
# Import SK-learn libraries for feature extraction from text.
from sklearn.feature_extraction.text import *

# Import re library to manipulate regular expressions
import re
# Define the regular expression used to designate what a 'word' is among a string of
# characters.
WORD_RE = re.compile(r"[\w']+")

# Transform the training data into a format that the sklearn libraries can use.
# train_ids:  stores the IDs from the training data.
train_ids = []
# train_data:  stores the text from each document
train_data = []
# train_labels:  stores the true labels for each document
train_labels = []
# set filename to the file that is being analyzed
filename = "enronemail_1h.txt"
# Open the file
with open (filename, "r") as myfile:
    # Loop through each line in the file
    for line in myfile.readlines():
        # Split the lines by tab
        items = line.split('\t')
        # Add each ID to the 'train_ids' list
        train_ids.append(items[0])
        # Add each label to the 'train_labels' list
        train_labels.append(int(items[1]))
        # The 3rd value of the line is the subject.  Convert this value to lowercase and then
        # break it up into separate words using the WORD_RE regular expression
        words_subject = re.findall(WORD_RE, items[2].lower())
        # The 4th value of the line is the body.  Convert this value to lowercase and then
        # break it up into separate words using the WORD_RE regular expression
        words_body = re.findall(WORD_RE, items[3].lower())
        # Concatenate the subject and body in to one list called 'words_all'
        words_all = words_subject + words_body
        # Create one string containing all of the words in the document separated by spaces
        # and then add that string to the 'train_data' list
        train_data.append(' '.join(words_all))

# Create a CountVectorizer object that will transform the text data into a feature vector
cv = CountVectorizer()
# Transform the train_data into a feature vector
train_fv = cv.fit_transform(train_data)

##### PART 1.6.1 #######
# Create a MultinomialNB model using the default parameters
multiNB = MultinomialNB()
# Fit the model with the training data
multiNB.fit(train_fv, train_labels)
# Create predictions from the model using the training data
predict = multiNB.predict(train_fv)

# Initialize variables to track the total incorrect predictions and the total predictions
count_total = 0
count_incorrect = 0
# Calculate the training error by dividing the count_incorrect by the total count
for i in range(len(predict)):
    if predict[i] != train_labels[i]:
        count_incorrect += 1
    count_total += 1
print('PART 1.6.1')
print('MultinomialNB - SciKit-Learn with smoothing Training Error: %f' % (count_incorrect / float(count_total)))
print('')

##### PART 1.6.2 #######
# Create a MultinomialNB model
BernoulliNB = BernoulliNB()
# Fit the model with the training data
BernoulliNB.fit(train_fv, train_labels)
# Create predictions from the model using the training data
predict = BernoulliNB.predict(train_fv)

# Initialize variables to track the total incorrect predictions and the total predictions
count_total = 0
count_incorrect = 0
# Calculate the training error by dividing the count_incorrect by the total count
for i in range(len(predict)):
    if predict[i] != train_labels[i]:
        count_incorrect += 1
    count_total += 1
```

```
print('PART 1.6.2')
print('BernoulliNB - SciKit-Learn Training Error: %f' % (count_incorrect / float(count_total)))
print('')

##### PART 1.6.3 #######
print('PART 1.6.3')
print('MultinomialNB - Command Line Training Error: 0.000000')
print('')

##### PART 1.6.4 #######
# Import tabulate to create a table
from tabulate import tabulate
# Create a variable 'headers' to store the headers of the table
headers = ['   ', 'Training Error']
# Initialize a list called data to hold the data for the rows of the table
data = [['MultinomialNB - SciKit-Learn', '0%'],
        ['BernoulliNB - SciKit-Learn', '16%'],
        ['MultinomialNB - Command Line', '0%']]

# Print the table
print('PART 1.6.4')
print(tabulate(data, headers=headers))
print('')

##### PART 1.6.7 #######
# Create a MultinomialNB model using no smoothing (alpha = 0)
multiNB = MultinomialNB(alpha=0)
# Fit the model with the training data
multiNB.fit(train_fv, train_labels)
# Create predictions from the model using the training data
predict = multiNB.predict(train_fv)

# Initialize variables to track the total incorrect predictions and the total predictions
count_total = 0
count_incorrect = 0
# Calculate the training error by dividing the count_incorrect by the total count
for i in range(len(predict)):
    if predict[i] != train_labels[i]:
        count_incorrect += 1
    count_total += 1
print('PART 1.6.7')
print('MultinomialNB - SciKit-Learn with NO smoothing Training Error: %f' % (count_incorrect / float(count_total)))
print('')
```

```
PART 1.6.1
MultinomialNB - SciKit-Learn with smoothing Training Error: 0.000000

PART 1.6.2
BernoulliNB - SciKit-Learn Training Error: 0.160000

PART 1.6.3
MultinomialNB - Command Line Training Error: 0.000000

PART 1.6.4
                              Training Error
----------------------------  ----------------
MultinomialNB - SciKit-Learn  0%
BernoulliNB - SciKit-Learn    16%
MultinomialNB - Command Line  0%

PART 1.6.7
MultinomialNB - SciKit-Learn with NO smoothing Training Error: 0.000000


/usr/local/anaconda2/lib/python2.7/site-packages/sklearn/naive_bayes.py:664: RuntimeWarning: divide by zero encount
  self.feature_log_prob_ = (np.log(smoothed_fc)
```