DATASCI W261: Machine Learning at Scale

Assignment Week 1

Miki Seltzer (miki.seltzer@berkeley.edu)
W261-2, Spring 2016
Submission: 18-Jan-2016 2pm

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

"Big data" is a broad term that has many interpretations. It doesn't seem fair to assign a concrete size barrier above which is considered "big," and below which is not. A definition that resonates with me is that if a data set is too large to fit on or process with one computer in a reasonable amount of time, then it is considered big data. This definition allows for more flexibility as hard drives become larger and computers become more powerful.

I currently work at a home security company, so we are constantly generating IoT-type data. Analysis on any of the data we generate from wireless sensors is usually a big data problem, unless we are looking at a tiny subset of data. One of the most important problems we are currently trying to solve is determining when a home is unoccupied using data from wireless sensors such as motion detectors and door sensors. It is relatively easy to determine when a home is occupied: we can be fairly certain that someone is home when motion is detected. Conversely, the absence of motion does not always indicate that the home is unoccupied. A simple example is at nighttime when people are usually sleeping.

HW1.0.1: In 500 words (English or pseudo code or a combination) describe how to estimate the bias, the variance, the irreducible error for a test dataset T when using polynomial regression models of degree 1, 2, 3, 4, 5 are considered. How would you select a model?

In order to estimate the bias, variance and irreducible error (noise) for a single data that is generated from the unknown true function $f(x)$, we first need to generate multiple sets of data from our original data set T. We can do this by sampling the original data set with replacement (bootstrapping). If we repeat the bootstrap resample 50 times, we will have 50 data sets to work with.

With each of the 50 new data sets, we split the data set into training set and a testing set. We fit polynomials of degree 1, 2, 3, 4 and 5 to the training set. This yields 50 models per polynomial degree. For each polynomial degree, we can then estimate the average variance and bias using the testing set.

Variance estimation

For each observation, x, in the testing set, we now have 50 predictions per polynomial degree, which we denote as $y_1, y_2, ..., y_{50}$. We find the average of these predictions, denoted as $\bar{y}$. We can find the variance of the predictions using the formula: $E((y_i - \bar{y})^2)$. We then average the variance for each testing data set, and then repeat the process for each polynomial degree. Thus, we will have one average variance per polynomial degree.

Bias estimation

For each observation, x, we also have the actual value of y in the testing set. The bias of the observation x is the difference between the average prediction and the actual value: $\bar{y} - y$. If we happened to have the true function $f(x)$ from which the data were generated, we would calculate the bias as: $\bar{y} - f(x)$.

We then average the bias over each data point in each testing set, and repeat for each polynomial degree. This yields one average bias per polynomial degree. It is also useful to calculate the average squared bias, which is calculated by squaring the bias before averaging all of the observations in the testing set.

Noise estimation

If we do not know the true function $f(x)$, we are forced to make the assumption that the noise is zero. If we knew the true function $f(x)$ that the data set T was generated from, we would be able to calculate the irreducible error, which would be the square of the difference between the observed values and the true function: $(y - f(x))^2$.

Model selection

We know that there is a trade-off between bias and variance. As the model gets more complex, bias generally decreases, while variance generally increases. We can plot the sum of the squared bias and the variance, and choose the degree where the sum is minimized.

HW1.1: Read through the provided control script (pNaiveBayes.sh)

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

In [2]:
```
# Using a Linux system, need to modify new line character to work corr
ectly

!perl -pi -e 's/\r/\n/g' enronemail_1h.txt
```

Shell script

We will need to use the pNaiveBayes.sh file multiple times during this homework assignment, so let's make sure that it is written to our working directory.

In [3]:

```
%%writefile pNaiveBayes.sh
## pNaiveBayes.sh
## Author: Jake Ryland Williams
## Usage: pNaiveBayes.sh m wordlist
## Input:
##       m = number of processes (maps), e.g., 4
##       wordlist = a space-separated list of words in quotes, e.g., "
the and of"
##
## Instructions: Read this script and its comments closely.
##               Do your best to understand the purpose of each comman
d,
##               and focus on how arguments are supplied to mapper.py/
reducer.py,
##               as this will determine how the python scripts take in
put.
##               When you are comfortable with the unix code below,
##               answer the questions on the LMS for HW1 about the sta
rter code.

## collect user input
m=$1 ## the number of parallel processes (maps) to run
wordlist=$2 ## if set to "*", then all words are used

## a test set data of 100 messages
data="enronemail_1h.txt"

## the full set of data (33746 messages)
# data="enronemail.txt"

## 'wc' determines the number of lines in the data
## 'perl -pe' regex strips the piped wc output to a number
linesindata=`wc -l $data | perl -pe 's/^.*?(\d+).*?$/$1/'`

## determine the lines per chunk for the desired number of processes
linesinchunk=`echo "$linesindata/$m+1" | bc`

## split the original file into chunks by line
split -l $linesinchunk $data $data.chunk.

## assign python mappers (mapper.py) to the chunks of data
## and emit their output to temporary files
for datachunk in $data.chunk.*; do
    ## feed word list to the python mapper here and redirect STDOUT to
 a temporary file on disk
    ####
    ####
    ./mapper.py $datachunk "$wordlist" > $datachunk.counts &
    ####
    ####
done
## wait for the mappers to finish their work
wait
```

```
## 'ls' makes a list of the temporary count files
## 'perl -pe' regex replaces line breaks with spaces
countfiles=`\ls $data.chunk.*.counts | perl -pe 's/\n/ /'`

## feed the list of countfiles to the python reducer and redirect STDO
UT to disk
####
####
./reducer.py $countfiles > $data.output
####
####

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

```
Overwriting pNaiveBayes.sh
```

In [4]:

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

HW1.2: Provide a mapper/reducer pair that, when executed by pNaiveBayes.sh will determine the number of occurrences of a single, user-specified word.

Examine the word “assistance” and report your results.

To do so, make sure that

- mapper.py counts all occurrences of a single word, and
- reducer.py collates the counts of the single word.

Mapper

In [5]:

```
%%writefile mapper.py
#!/usr/bin/python
## mapper.py
## Author: Miki Seltzer
## Description: mapper code for HW1.2

import sys
import re
import string

# Collect input
filename = sys.argv[1]
findwords = re.split(" ",sys.argv[2].lower())

# Initialize dictionary to empty
wordcount = {}

with open(filename, "rU") as myfile:
    for line in myfile:
        # Format each line, fields separated by \t according to enrone
mail_README.txt
        # Remove \n text at end of each line
        fields = line.split("\t")
        fields[3] = fields[3].replace("\n", "")
        subj = fields[2].translate(string.maketrans("",""), string.pun
ctuation)
        body = fields[3].translate(string.maketrans("",""), string.pun
ctuation)

        # For each word in list provided by user, count occurrences in
 subj and body
        for word in findwords:
            if word not in wordcount:
                wordcount[word] = 0
            wordcount[word] += subj.count(word) + body.count(word)

for word in wordcount:
    print [word, wordcount[word]]
```

```
Overwriting mapper.py
```

Reducer

In [6]:

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

import sys

filenames = sys.argv[1:]

wordcount = {}

# Each mapper outputs a [word, count] pair
# For each file and each word, sum the counts
for file in filenames:
    with open(file, "rU") as myfile:
        for line in myfile:
            pair = eval(line)
            word = pair[0]
            count = pair[1]
            if word not in wordcount:
                wordcount[word] = 0
            wordcount[word] += count

for word in wordcount:
    print word + "\t" + str(wordcount[word])
```

```
Overwriting reducer.py
```

Run shell script

In [7]:

```
# Change file permissions
!chmod a+x mapper.py
!chmod a+x reducer.py

# Run shell script
!./pNaiveBayes.sh 4 "assistance"
```

Function to print formatted output of shell script

In [8]:

```
def print_counts():
    with open("enronemail_1h.txt.output", "r") as myfile:
        print "{:<15s}{:3s}".format("word", "count")
        print "----------------------"
        for line in myfile:
            pair = line.split("\t")
            word = pair[0]
            count = int(pair[1])
            print "{:<15s}{:3d}".format(word, count)
```

Formatted Output of HW1.2

In [9]:
```
print_counts()
```

```
word           count
----------------------
assistance      10
```

HW1.3. Provide a mapper/reducer pair that, when executed by pNaiveBayes.sh will classify the email messages by a single, user-specified word using the multinomial Naive Bayes Formulation.

Examine the word “assistance” and report your results. To do so, make sure that mapper.py and reducer.py perform a single word Naive Bayes classification. For multinomial Naive Bayes, the Pr(X=“assistance”|Y=SPAM) is calculated as follows:

$$\frac{\text{number of times ”assistance” occurs in SPAM labeled documents}}{\text{the number of words in documents labeled SPAM}}$$

Mapper

In [10]:

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

import sys
import re
import string

# Collect input
filename = sys.argv[1]
findwords = re.split(" ",sys.argv[2].lower())

# Initialize dictionary to empty
word_count = {}

with open(filename, "rU") as myfile:
    for line in myfile:
        # Format each line, fields separated by \t according to enrone
mail_README.txt
        fields = line.split("\t")
        fields[3] = fields[3].replace("\n", "")
        subj = fields[2].translate(string.maketrans("",""), string.pun
ctuation)
        body = fields[3].translate(string.maketrans("",""), string.pun
ctuation)

        # For each word in list provided by user, count occurrences in
 subj and body
        for word in findwords:
            my_key = (fields[0], fields[1], word)
            if my_key not in word_count:
                word_count[my_key] = 0
            word_count[my_key] += subj.count(word) + body.count(word)

for key in word_count:
    print [key, word_count[key]]
```

```
Overwriting mapper.py
```

Reducer

In [11]:

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

import sys
import math

filenames = sys.argv[1:]

doc_ids = {}
document_words = {}
class_counts = {'1':0.0, '0':0.0}
vocab = {}
word_counts = {
    '0': {},
    '1': {}
}

for file in filenames:
    # Train classifier with data from mapper.py
    with open(file, "r") as myfile:
        for line in myfile:
            pair = eval(line)
            doc_id = pair[0][0]
            spam = pair[0][1]
            word = pair[0][2]
            count = int(pair[1])

            # We need to aggregate counts on specific levels. We need:
            #    - number of total documents
            #    - number of documents per class
            #    - our entire vocabulary
            #    - word counts per class
            #    - word counts per document (for testing purposes)
            if doc_id not in doc_ids:
                doc_ids[doc_id] = spam
                class_counts[spam] += 1
                document_words[doc_id] = {}
            if word not in vocab: vocab[word] = 0.0
            vocab[word] += count
            if word not in word_counts[spam]: word_counts[spam][word]
= 0.0
            word_counts[spam][word] += count
            if word not in document_words[doc_id]: document_words[doc_
id][word] = 0.0
            document_words[doc_id][word] += count


prior_spam = class_counts['1'] / len(doc_ids)
prior_ham = class_counts['0'] / len(doc_ids)
```

```
p_spam_word = 0.0
p_ham_word = 0.0

for doc in document_words:
    pred = 'none'

    # Test classifier using training data
    for word in doc:
        # Make sure that the word was in the training data
        if word not in vocab: continue

        # Calculate P(word)
        p_word = vocab[word] / sum(vocab.values())

        # Calculate P(word|spam) and P(word|ham)
        # If word does not occur in spam or ham documents, probability
 is 0
        if word not in word_counts['1']: p_word_spam = 0.0
        else: p_word_spam = word_counts['1'][word] / sum(word_counts['
1'].values())
        if word not in word_counts['0']: p_word_ham = 0.0
        else: p_word_ham = word_counts['0'][word] / sum(word_counts['0
'].values())

        # Calculate P(spam|word) and P(ham|word)
        p_spam_word = prior_spam * p_word_spam ** document_words[doc][
word]
        p_ham_word = prior_ham * p_word_ham ** document_words[doc][wor
d]

    if p_spam_word > p_ham_word: pred = '1'
    else: pred = '0'

    print doc + "\t" + doc_ids[doc] + "\t" + pred
```

```
Overwriting reducer.py
```

Run shell script

In [12]:
```
# Change file permissions
!chmod a+x mapper.py
!chmod a+x reducer.py

# Run shell script
!./pNaiveBayes.sh 4 "assistance"
```

Function to print formatted output

In [13]:

```
def print_predictions():

    correct = 0.0
    total = 0.0

    print "{:<35s}{:5s}{:5s}".format("document id", "true", "pred")
    print "---------------------------------------------"

    with open('enronemail_1h.txt.output', 'rU') as myfile:
        for line in myfile:
            fields = line.split("\t")
            fields[-1] = fields[-1].replace("\n", "")
            print "{:<35s}{:5s}{:5s}".format(fields[0], fields[1], fie
lds[2])
            total += 1
            if fields[1] == fields[2]: correct += 1

    print "\nAccuracy: {:.2%}".format(correct/total)
```

Formatted Output of HW1.3

In [14]:
```
print_predictions()
```

```
document id                          true pred
---------------------------------------------
0010.2003-12-18.GP                   1    0
0010.2001-06-28.SA_and_HP            1    0
0001.2000-01-17.beck                 0    0
0018.1999-12-14.kaminski             0    0
0005.1999-12-12.kaminski             0    0
0011.2001-06-29.SA_and_HP            1    0
0008.2004-08-01.BG                   1    0
0009.1999-12-14.farmer               0    0
0017.2003-12-18.GP                   1    0
0011.2001-06-28.SA_and_HP            1    0
0015.2001-07-05.SA_and_HP            1    0
0015.2001-02-12.kitchen              0    0
0009.2001-06-26.SA_and_HP            1    0
0018.2001-07-13.SA_and_HP            1    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
0016.1999-12-15.farmer               0    0
0013.2004-08-01.BG                   1    0
0005.2003-12-18.GP                   1    0
0012.2001-02-09.kitchen              0    0
0011.1999-12-14.farmer               0    0
0013.1999-12-14.kaminski             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
0005.1999-12-14.farmer               0    0
0003.2001-02-08.kitchen              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    0
0002.1999-12-13.farmer          0    0
0016.2003-12-19.GP              1    0
0004.1999-12-14.farmer          0    0
0015.2003-12-19.GP              1    0
0006.2004-08-01.BG              1    0
0009.2003-12-18.GP              1    0
0007.1999-12-14.farmer          0    0
0005.2000-06-06.lokay           0    0
0010.1999-12-14.kaminski        0    0
0007.2000-01-17.beck            0    0
0003.1999-12-14.farmer          0    0
0003.2004-08-01.BG              1    0
0017.2004-08-01.BG              1    0
0013.2001-06-30.SA_and_HP       1    0
0003.1999-12-10.kaminski        0    0
0012.1999-12-14.farmer          0    0
0004.1999-12-10.kaminski        0    0
0017.1999-12-14.kaminski        0    0
0002.2001-02-07.kitchen         0    0
0007.2004-08-01.BG              1    0
0012.1999-12-14.kaminski        0    0
0005.2001-06-23.SA_and_HP       1    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    0
0002.2001-05-25.SA_and_HP       1    0
0011.2003-12-18.GP              1    0

Accuracy: 56.00%
```

HW1.4: Provide a mapper/reducer pair that, when executed by pNaiveBayes.sh will classify the email messages by a list of one or more user-specified words.

Examine the words “assistance”, “valium”, and “enlargementWithATypo” and report your results. To do so, make sure that mapper.py counts all occurrences of a list of words, and reducer.py performs the multiple-word multinomial Naive Bayes classification via the chosen list.

Mapper

In [15]:

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

import sys
import re
import string

# Collect input
filename = sys.argv[1]
findwords = re.split(" ",sys.argv[2].lower())

# Initialize dictionary to empty
word_count = {}

with open(filename, "rU") as myfile:
    for line in myfile:
        # Format each line, fields separated by \t according to enrone
mail_README.txt
        fields = line.split("\t")
        fields[3] = fields[3].replace("\n", "")
        subj = fields[2].translate(string.maketrans("",""), string.pun
ctuation)
        body = fields[3].translate(string.maketrans("",""), string.pun
ctuation)

        # For each word in list provided by user, count occurrences in
 subj and body
        for word in findwords:
            my_key = (fields[0], fields[1], word)
            if my_key not in word_count:
                word_count[my_key] = 0
            word_count[my_key] += subj.count(word) + body.count(word)

for key in word_count:
    print [key, word_count[key]]
```

```
Overwriting mapper.py
```

Reducer

In [16]:

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

import sys
import math

filenames = sys.argv[1:]

doc_ids = {}
document_words = {}
class_counts = {'1':0.0, '0':0.0}
vocab = {}
word_counts = {
    '0': {},
    '1': {}
}

for file in filenames:
    # Train classifier with data from mapper.py
    with open(file, "r") as myfile:
        for line in myfile:
            pair = eval(line)
            doc_id = pair[0][0]
            spam = pair[0][1]
            word = pair[0][2]
            count = int(pair[1])

            # We need to aggregate counts on specific levels. We need:
            #    - number of total documents
            #    - number of documents per class
            #    - our entire vocabulary
            #    - word counts per class
            #    - word counts per document (for testing purposes)
            if doc_id not in doc_ids:
                doc_ids[doc_id] = spam
                class_counts[spam] += 1
                document_words[doc_id] = {}
            if word not in vocab: vocab[word] = 0.0
            vocab[word] += count
            if word not in word_counts[spam]: word_counts[spam][word]
= 0.0
            word_counts[spam][word] += count
            if word not in document_words[doc_id]: document_words[doc_
id][word] = 0.0
            document_words[doc_id][word] += count


prior_spam = class_counts['1'] / len(doc_ids)
prior_ham = class_counts['0'] / len(doc_ids)
```

```
for doc in document_words:
    pred = 'none'

    # Test classifier using training data
    for word in document_words[doc]:
        prob_spam_word = prior_spam
        prob_ham_word = prior_ham

        # Make sure that the word was in the training data
        if word not in vocab: continue

        # Calculate P(word)
        p_word = vocab[word] / sum(vocab.values())

        # Calculate P(word|spam) and P(word|ham)
        # If word does not occur in spam or ham documents, probability
 is 0
        if word not in word_counts['1']: p_word_spam = 0.0
        else: p_word_spam = word_counts['1'][word] / sum(word_counts['
1'].values())
        if word not in word_counts['0']: p_word_ham = 0.0
        else: p_word_ham = word_counts['0'][word] / sum(word_counts['0
'].values())

        # Update probabilities
        prob_spam_word *= p_word_spam ** document_words[doc][word]
        prob_ham_word *= p_word_ham ** document_words[doc][word]

    if prob_spam_word > prob_ham_word: pred = '1'
    else: pred = '0'

    print doc + "\t" + doc_ids[doc] + "\t" + pred
```

```
Overwriting reducer.py
```

Run shell script

In [17]:

```
# Change file permissions
!chmod a+x mapper.py
!chmod a+x reducer.py

# Run shell script
!./pNaiveBayes.sh 4 "assistance valium enlargementWithATypo"
```

Formatted Output of HW1.4

```
In [18]:  print_predictions()
```

```
document id                            true pred
---------------------------------------------
0010.2003-12-18.GP                     1    0
0010.2001-06-28.SA_and_HP              1    0
0001.2000-01-17.beck                   0    0
0018.1999-12-14.kaminski               0    0
0005.1999-12-12.kaminski               0    0
0011.2001-06-29.SA_and_HP              1    0
0008.2004-08-01.BG                     1    0
0009.1999-12-14.farmer                 0    0
0017.2003-12-18.GP                     1    0
0011.2001-06-28.SA_and_HP              1    0
0015.2001-07-05.SA_and_HP              1    0
0015.2001-02-12.kitchen                0    0
0009.2001-06-26.SA_and_HP              1    0
0017.1999-12-14.kaminski               0    0
0012.2000-01-17.beck                   0    0
0003.2000-01-17.beck                   0    0
0004.2001-06-12.SA_and_HP              1    0
0008.2001-06-12.SA_and_HP              1    0
0007.2001-02-09.kitchen                0    0
0016.2004-08-01.BG                     1    0
0015.2000-06-09.lokay                  0    0
0016.1999-12-15.farmer                 0    0
0013.2004-08-01.BG                     1    0
0005.2003-12-18.GP                     1    0
0012.2001-02-09.kitchen                0    0
0011.1999-12-14.farmer                 0    0
0013.1999-12-14.kaminski               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
0005.1999-12-14.farmer                 0    0
0003.2001-02-08.kitchen                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    0
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    0
0018.2001-07-13.SA_and_HP         1    0
0002.2001-02-07.kitchen           0    0
0007.2004-08-01.BG                1    0
0012.1999-12-14.kaminski          0    0
0005.2001-06-23.SA_and_HP         1    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    0
0002.2001-05-25.SA_and_HP         1    0
0011.2003-12-18.GP                1    0

Accuracy: 59.00%
```

I had already finished most of 1.5 and 1.6 when we were told to skip these questions -- they are included in case any feedback is provided

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

To do so, make sure that mapper.py counts all occurrences of all words, and reducer.py performs a word-distribution-wide Naive Bayes classification.

Mapper

```
In [19]:  %%writefile mapper.py
          #!/usr/bin/python
          ## mapper.py
          ## Author: Miki Seltzer
          ## Description: mapper code for HW1.5

          import sys
          import re
          import string

          # Collect input
          filename = sys.argv[1]
          findwords = re.split(" ",sys.argv[2].lower())

          # Initialize dictionary to empty
          word_count = {}

          with open(filename, "rU") as myfile:
              for line in myfile:
                  # Format each line, fields separated by \t according to enrone
          mail_README.txt
                  fields = re.split("\t", line)
                  fields[3] = fields[3].replace("\n", "")
                  subj = fields[2].translate(string.maketrans("",""), string.pun
          ctuation)
                  body = fields[3].translate(string.maketrans("",""), string.pun
          ctuation)

                  # For each word, count occurrences in subj and body
                  # Key is (document id, spam, word)
                  if findwords[0] == "*":
                      full_text = subj + " " + body
                      for word in full_text.split():
                          my_key = (fields[0], fields[1], word)
                          if my_key not in word_count:
                              word_count[my_key] = 0.0
                          word_count[my_key] += 1
                  else:
                      for word in findwords:
                          my_key = (fields[0], fields[1], word)
                          if my_key not in word_count:
                              word_count[my_key] = 0.0
                          word_count[my_key] += subj.count(word) + body.count(wo
          rd)

          for key in word_count:
              print [key, word_count[key]]
```

```
Overwriting mapper.py
```

Reducer

In [29]:

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

import sys
import math

filenames = sys.argv[1:]

doc_ids = {}
document_words = {}
class_counts = {'1':0.0, '0':0.0}
vocab = {}
word_counts = {
    '0': {},
    '1': {}
}

for file in filenames:
    # Train classifier with data from mapper.py
    with open(file, "r") as myfile:
        for line in myfile:
            pair = eval(line)
            doc_id = pair[0][0]
            spam = pair[0][1]
            word = pair[0][2]
            count = int(pair[1])

            # We need to aggregate counts on specific levels. We need:
            #    - number of total documents
            #    - number of documents per class
            #    - our entire vocabulary
            #    - word counts per class
            #    - word counts per document (for testing purposes)
            if doc_id not in doc_ids:
                doc_ids[doc_id] = spam
                class_counts[spam] += 1
                document_words[doc_id] = {}
            if word not in vocab: vocab[word] = 0.0
            vocab[word] += count
            if word not in word_counts[spam]: word_counts[spam][word]
= 0.0
            word_counts[spam][word] += count
            if word not in document_words[doc_id]: document_words[doc_
id][word] = 0.0
            document_words[doc_id][word] += count


prior_spam = class_counts['1'] / len(doc_ids)
prior_ham = class_counts['0'] / len(doc_ids)
```

```
for doc in document_words:
    pred = 'none'

    log_prob_spam_word = math.log(prior_spam)
    log_prob_ham_word = math.log(prior_ham)

    # Test classifier using training data
    for word in document_words[doc]:


        # Make sure that the word was in the training data
        if word not in vocab: continue

        # Calculate P(word)
        p_word = vocab[word] / sum(vocab.values())

        # Calculate P(word|spam) and P(word|ham)
        # Use add-1 smoothing
        if word not in word_counts['1']: num_word_spam = 0
        else: num_word_spam = word_counts['1'][word]
        p_word_spam = (num_word_spam + 1.0) / (sum(word_counts['1'].va
lues()) + len(vocab))

        # If we were to not use smoothing:
        # p_word_spam = (num_word_spam) / sum(word_counts['1'].values(
))

        if word not in word_counts['0']: num_word_ham = 0
        else: num_word_ham = word_counts['0'][word]
        p_word_ham = (num_word_ham + 1.0) / (sum(word_counts['0'].valu
es()) + len(vocab))

        # If we were to not use smoothing:
        #p_word_ham = (num_word_ham) / sum(word_counts['0'].values())

        # Update probabilities
        log_prob_spam_word += math.log(p_word_spam) * document_words[d
oc][word]
        log_prob_ham_word += math.log(p_word_ham) * document_words[doc
][word]

    if log_prob_spam_word > log_prob_ham_word: pred = '1'
    else: pred = '0'

    print doc + "\t" + doc_ids[doc] + "\t" + pred
```

```
Overwriting reducer.py
```

Run shell script

In [30]:

```
# Change file permissions
!chmod a+x mapper.py
!chmod a+x reducer.py

# Run shell script
!./pNaiveBayes.sh 4 "*"
```

Formatted Output of HW1.5

In [31]:
```
print_predictions()
```

```
document id                           true pred
---------------------------------------------
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
0018.2001-07-13.SA_and_HP             1    1
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
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
0011.1999-12-14.farmer                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
0005.1999-12-14.farmer                0    0
0003.2001-02-08.kitchen               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
0002.2004-08-01.BG                 1    1
0010.1999-12-14.kaminski           0    0
0007.2000-01-17.beck               0    0
0003.1999-12-14.farmer             0    0
0003.2004-08-01.BG                 1    1
0017.2004-08-01.BG                 1    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
0017.1999-12-14.kaminski           0    0
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
0005.2000-06-06.lokay              0    0
0013.1999-12-14.kaminski           0    0
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
0002.2001-05-25.SA_and_HP          1    1
0011.2003-12-18.GP                 1    1

Accuracy: 100.00%
```

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

Set up SK-learn libraries and data ingestion

In [23]:
```
# General libraries
import numpy as np
from __future__ import division

# SK-learn libraries for learning
from sklearn.pipeline import Pipeline
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import BernoulliNB

# SK-learn libraries for feature extraction from text.
from sklearn.feature_extraction.text import *
```

In [24]:
```
# Read data in and create data and label arrays
ids, X, Y = [], [], []

with open('enronemail_1h.txt', 'rU') as myfile:
    for line in myfile:
        fields = line.split("\t")
        text = fields[2] + " " + fields[3]
        text = text.replace("\n", "")
        X.append(text)
        Y.append(fields[1])
        ids.append(fields[0])

# Convert these to numpy arrays
X = np.array(X)
Y = np.array(Y)

# Check that the shapes look correct
print X.shape, Y.shape
```

```
(100,) (100,)
```

- 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
- 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
- Run the Multinomial Naive Bayes algorithm you developed for HW1.5 over the same data used HW1.5 and report the training error
- Please prepare a table to present your results

In [32]:

```
def hw1_6():
    train_errors = {}

    ##### MULTINOMIAL NB
    # Create Pipeline to get feature vectors and train
    # Use CountVectorizer to get feature arrays
    # Classify using Multinomial NB
    mnb_pipe = Pipeline([('vect', CountVectorizer()),
                         ('clf', MultinomialNB()),
                        ])

    # Fit training data and labels
    mnb_pipe.fit(X, Y)

    # Print training error
    mnb_predictions = mnb_pipe.predict(X)
    train_errors["Multinomial"] = sum(mnb_predictions != Y) / Y.size

    ##### BERNOULLI NB
    # Create Pipeline to get feature vectors and train
    # Use CountVectorizer to get feature arrays
    # Classify using Bernoulli NB
    bnb_pipe = Pipeline([('vect', CountVectorizer()),
                         ('clf', BernoulliNB()),
                        ])

    # Fit training data and labels
    bnb_pipe.fit(X, Y)

    # Print training error
    bnb_predictions = bnb_pipe.predict(X)
    train_errors["Bernoulli"] = sum(bnb_predictions != Y) / Y.size

    ##### CLASSIFIER in HW1.5
    # Read output from enronemail_1h.txt.output

    incorrect = 0.0
    total = 0.0

    with open('enronemail_1h.txt.output', 'rU') as myfile:
        for line in myfile:
            fields = line.split("\t")
            fields[-1] = fields[-1].replace("\n", "")
            total += 1
            if fields[1] != fields[2]: incorrect += 1

    train_errors["HW1.5"] = incorrect/total

    ##### TABLE OF TRAINING ERRORS
    print "{:<14s}{:6s}".format("Method", "Error")
    print "--------------------"
    for method in train_errors:
        print "{:<14s}{:>4.2%}".format(method, train_errors[method])
```

```
In [33]: hw1_6()
```

```
Method         Error
--------------------
Multinomial   0.00%
Bernoulli     16.00%
HW1.5         0.00%
```