Name: Patrick Ng
Email: patng@ischool.berkeley.edu
Class: W261-2
Week: 01
Date of submission: Jan 18, 2016

HW1.0.0

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

Big data is broad term for data sets so large or complex that traditional data-processing applications are inadequate. For instance, today a high end laptop may have 1 TB of harddisk and 8GB of memory. However, for many big data problem, the amount of disk and memory needed is way over what can be fit in such a high end laptop.

IBM has also characterized big data by its 4 V's: Volume (scale of data), Velocity (analysis of streaming data), Variety (different forms of data) and Veractiy (uncertainty of data).

For example, for a popular website which attracts ten millions visits each day, the amount of web log data generated each day is about 50GB. The website also uses a recommendation engine to generate recommendations to the visitors. In order to train the engine each night, we need to cleanse (e.g. remove logs generated by suspected robots) and transform the web log data into a format usable by the training process, and the training itself has to process all the new web log data, together with all data generated from the past. The whole process has to complete within 6 hours due to business needs. However, using traditional data-processing techniques the whole proceess could take more than 8 hours. As a result, we need to make use of big data techniques such as HDFS and parallel computating in order to complete the processing within the required period.

HW1.0.1

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

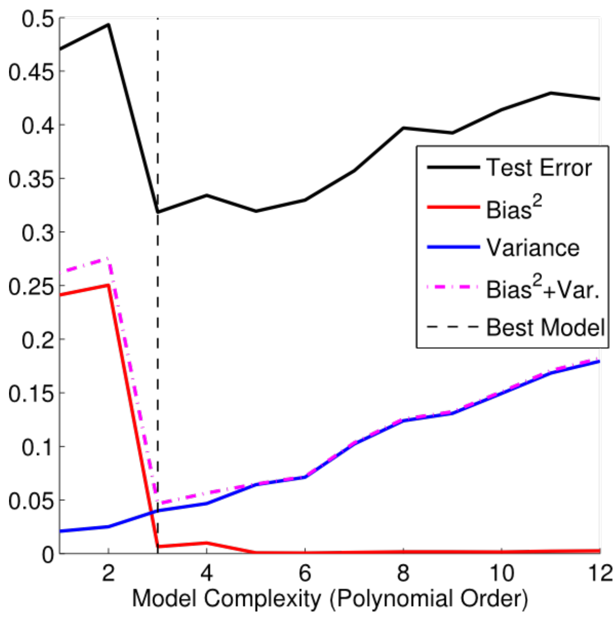
To estimate the bias, the variance, the irreducible error for a test dataset T when using polynomial regression models of degree 1, 2, 3, 4 and 5, the strategy is:

- First, please note that each regression model will produce an estimator *g(x)* of the true function *f(x)*.
- For each model:
  - Using bootstrapping, generate datasets S_1, S_2, ..., S_B from the dataset T.
  - For each S_b:
    - Use S_b to train the model to produce an estimator $g_b(x)$.
    - Let the dataset $T_b = T \setminus S_b$ be the data points that do not appear in S_b
    - Calculate the predicted value $g_b(x)$ for each x in T_b.
  - For each data point x in T :
    - We have calculated several predictions $y_1, y_2, \ldots, y_K$. From them we can calculate the average prediction $\underline{h} = \sum_K y_k / K$
      - K is the number of predictions made for data point x
      - y_k is the k-th prediction made for data point x
    - And from that we can calculate the bias, the variance and the irreducible error of the model for the data point x:
      - $Bias = \underline{h} - y$
      - $Variance = \sum_K \left(y_k - \underline{h}\right)^2 / (K - 1)$
      - $IrreducibleError/Noise$:
        - Since we do not have the true function f(x), we have to estimate the noise at each data point x:
          - In T, if several data points exists at x, then:
            - noise = variance of the multiple y's from those data points
          - Otherwise, assume noise = 0 for x

To select a model, for each model we calculate the *expected prediction error*:

- Expected prediction error = $\mathrm{E}_X \left[Variance + \mathrm{Bias}^2 + Noise^2\right]$
  - E_X means we average it over all data points.

We then plot the numbers, similar to the graph below. Please note that we do not have the figures for the *Test Error* line.



From the graph, we identify the point where $\mathrm{E}_X \left[Variance + \mathrm{Bias}^2\right]$ is the smallest. The

Prepare the control script

In [40]:
```
%%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., "th
##
## Instructions: Read this script and its comments closely.
##               Do your best to understand the purpose of each command,
##               and focus on how arguments are supplied to mapper.py/re
##               as this will determine how the python scripts take inpu
##               When you are comfortable with the unix code below,
##               answer the questions on the LMS for HW1 about the start

## 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
    ####
    ####
    ./mapper.py $datachunk "$wordlist" > $datachunk.counts &
    ####
    ####
done
## wait for the mappers to finish their work
wait

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

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

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

```
Overwriting pNaiveBayes.sh
```

HW1.1

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

In [41]:
```
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.

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.

In [1]:
```
%%writefile mapper.py
#!/usr/bin/python
## mapper.py
## Author: Patrick Ng
## Description: mapper code for HW1.2

import sys
import re

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

findword = findwords[0] # for HW1.2 we handle only a single word
regex = re.compile(r'\b' + re.escape(findword) + r'\b')
count = 0

# Loop thru all the lines in the file and count the total occurrence cou
with open(filename, 'r') as f:
    for line in f:
        parts = re.split("\t", line)

        subject = "" if parts[2].strip() == "NA" else parts[2]
        body = "" if parts[3].strip() == "NA" else parts[3]
        text = subject + " " + body

        count += len(regex.findall(text))

print '%s\t%d' % (findword, count)
```

```
Overwriting mapper.py
```

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

        import sys
        import re

        ## collect user input
        filenames = sys.argv[1:]

        totalCount = 0
        findword = None

        # Go thru all the files, and sum up the number of occurrence counters.
        for filename in filenames:
            with open(filename, 'r') as f:
                line = f.readline()
                parts = line.split('\t')
                findword = parts[0]
                count = int(parts[1])
                totalCount += count

        print '%s\t%d' % (findword, totalCount)
```

```
Overwriting reducer.py
```

```
In [3]: !chmod +x mapper.py; chmod +x reducer.py; chmod +x pNaiveBayes.sh;
```

Results:

```
In [4]: !./pNaiveBayes.sh 4 assistance
        !cat enronemail_1h.txt.output
```

```
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

that performs a single word Naive Bayes classification. For multinomial Naive Bayes, the Pr(X=“assistance”|Y=SPAM) is calculated as follows:

```
the number of times “assistance” occurs in SPAM labeled documents
 / the number of words in documents labeled SPAM
```

NOTE if “assistance” occurs 5 times in all of the documents Labeled SPAM, and the length in terms of the number of words in all documents labeld as SPAM (when concatenated) is 1,000. Then Pr(X=“assistance”|Y=SPAM) = 5/1000. Note this is a multinomial estimated of the class conditional for a Naive Bayes Classifier. No smoothing is needed in this HW.

In [5]:
```
%%writefile mapper.py
#!/usr/bin/python
## mapper.py
## Author: Patrick Ng
## Description: mapper code for HW1.3

import sys
import re

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

findword = findwords[0] # for HW1.3 we handle only a single word

# Regex for counting the number of findword in msg
regex = re.compile(r'\b' + re.escape(findword) + r'\b')

# Regex for counting the total number of words in a msg
regexAll = re.compile(r'\b\w+\b')

# For each of the count list below, index=0 means non-spam, index=1 mean
findwordCounts = [0, 0] # The findword counts among the two classes
totalwordCounts = [0, 0] # The total counts among the two classes
msgCounts = [0, 0]  # Total count of msg among the two classes

# For each msg: MsgId, its True class, and the occurrence count of findw
findwordInMessages = {}

# Go through each line of the file, and update the various counters
with open(filename, 'r') as f:
    for line in f:
        parts = re.split("\t", line)

        msgId = parts[0]
        spamIndicator = int(parts[1])

        subject = "" if parts[2].strip() == "NA" else parts[2]
        body = "" if parts[3].strip() == "NA" else parts[3]
        text = subject + " " + body

        # Update the various counters

        msgCounts[spamIndicator] += 1

        findwordCount = len(regex.findall(text))
        findwordCounts[spamIndicator] += findwordCount
        findwordInMessages[msgId] = [spamIndicator, findwordCount]

        totalwordCounts[spamIndicator] += len(regexAll.findall(text))


# Msg count in each class
print "%d\t%d" % (msgCounts[0], msgCounts[1])

# Occurrence count of the findword in each class
print "%d\t%d" % (findwordCounts[0], findwordCounts[1])

# Total word count in each class
print "%d\t%d" % (totalwordCounts[0], totalwordCounts[1])

# For each msg: MsgId, occurrence count of findword, its True class
for k, v in findwordInMessages.iteritems():
    print "%s\t%d\t%d" % (k, v[0], v[1])
```

In [6]:
```
%%writefile reducer.py
#!/usr/bin/python
## reducer.py
## Author: Patrick Ng
## Description: reducer code for HW1.3

import sys
import re
import math
import collections

# Helper function to read the findwordInMessages data from the file
def readFindwordInMessages(f, countDict):
    for line in f:
        parts = re.split("\t", line)
        msgId = parts[0]
        trueClass = int(parts[1])
        count = int(parts[2])
        countDict[msgId] = [trueClass, count]

# Helper function to read two tab separated counters from a line in the
def readCounts(f, counts):
    line = f.readline()
    parts = re.split("\t", line)
    counts[0] += int(parts[0])
    counts[1] += int(parts[1])

# Train the Multinomial Naive Bayes model and return the Priors and
# the conditional Probabilities
def trainMultinomialNB(msgCounts, findwordCounts, totalwordCounts):
    """
    Train the Multinomial Naive Bayes model.

    Parameters
    ----------
        msgCounts : list of int
            Count of messages in each class (0=non-spam, 1=spam)

        findwordCounts : list of int
            Count of findword occurrence in each class (0=non-spam, 1=sp

        totalwordCounts : list of int
            Total number of words in each class (0=non-spam, 1=spam)

    Returns
    -------
        priors : dict of float
            The priors (Pr(c)) of each class.

        condProbs : dict of float
            The conditional probability (Pr(X=t|c)) of each class.
    """
    priors = {} # Pr(c)
    condProbs = {} # Pr(X=t|c)
    msgTotal = sum(msgCounts)

    # 0: non-spam, 1: spam
    for c in [0, 1]:
        priors[c] = float(msgCounts[c]) / msgTotal
        condProbs[c] = float(findwordCounts[c]) / totalwordCounts[c]

    return priors, condProbs
```

In [7]:
```
!./pNaiveBayes.sh 4 assistance
!cat enronemail_1h.txt.output
```

```
0001.1999-12-10.farmer  0       0
0001.1999-12-10.kaminski        0       0
0001.2000-01-17.beck    0       0
0001.2000-06-06.lokay   0       0
0001.2001-02-07.kitchen 0       0
0001.2001-04-02.williams        0       0
0002.1999-12-13.farmer  0       0
0002.2001-02-07.kitchen 0       0
0002.2001-05-25.SA_and_HP       1       0
0002.2003-12-18.GP      1       0
0002.2004-08-01.BG      1       1
0003.1999-12-10.kaminski        0       0
0003.1999-12-14.farmer  0       0
0003.2000-01-17.beck    0       0
0003.2001-02-08.kitchen 0       0
0003.2003-12-18.GP      1       0
0003.2004-08-01.BG      1       0
0004.1999-12-10.kaminski        0       1
0004.1999-12-14.farmer  0       0
0004.2001-04-02.williams        0       0
0004.2001-06-12.SA_and_HP       1       0
0004.2004-08-01.BG      1       0
0005.1999-12-12.kaminski        0       1
0005.1999-12-14.farmer  0       0
0005.2000-06-06.lokay   0       0
0005.2001-02-08.kitchen 0       0
0005.2001-06-23.SA_and_HP       1       0
0005.2003-12-18.GP      1       0
0006.1999-12-13.kaminski        0       0
0006.2001-02-08.kitchen 0       0
0006.2001-04-03.williams        0       0
0006.2001-06-25.SA_and_HP       1       0
0006.2003-12-18.GP      1       0
0006.2004-08-01.BG      1       0
0007.1999-12-13.kaminski        0       0
0007.1999-12-14.farmer  0       0
0007.2000-01-17.beck    0       0
0007.2001-02-09.kitchen 0       0
0007.2003-12-18.GP      1       0
0007.2004-08-01.BG      1       0
0008.2001-02-09.kitchen 0       0
0008.2001-06-12.SA_and_HP       1       0
0008.2001-06-25.SA_and_HP       1       0
0008.2003-12-18.GP      1       0
0008.2004-08-01.BG      1       0
0009.1999-12-13.kaminski        0       0
0009.1999-12-14.farmer  0       0
0009.2000-06-07.lokay   0       0
0009.2001-02-09.kitchen 0       0
0009.2001-06-26.SA_and_HP       1       0
0009.2003-12-18.GP      1       0
0010.1999-12-14.farmer  0       0
0010.1999-12-14.kaminski        0       0
0010.2001-02-09.kitchen 0       0
0010.2001-06-28.SA_and_HP       1       1
0010.2003-12-18.GP      1       0
0010.2004-08-01.BG      1       0
0011.1999-12-14.farmer  0       0
0011.2001-06-28.SA_and_HP       1       1
0011.2001-06-29.SA_and_HP       1       0
0011.2003-12-18.GP      1       0
```

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) 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. No smoothing is needed in this HW.

In [8]:
```
%%writefile mapper.py
#!/usr/bin/python
## mapper.py
## Author: Patrick Ng
## Description: mapper code for HW1.4

import sys
import re
from collections import defaultdict
import cPickle

## collect user input
filename = sys.argv[1]
findwords = re.split(" ",sys.argv[2].lower())
regexes = {}

# Create the list of compiled regex, one for each findword
for word in findwords:
    regexes[word] = re.compile(r'\b' + re.escape(word) + r'\b')

# Regex for counting the total number of words in a msg
regexAll = re.compile(r'\b\w+\b')

# For each of the count lists below, index=0 means non-spam, index=1 mea

# For each class, the occurrence count of each word.
findwordCounts = [defaultdict(int), defaultdict(int)]
totalwordCounts = [0, 0] # The total counts among the two classes.
msgCounts = [0, 0]  # Total count of msg among the two classes.

# A dictionary holding the findword info for each message.
# Key = MsgId
# Value = a list: [message's true class, the occurrence count of each wo
# The "occurrence count of each word" is stored as a dictionary where th
# and the value is the count.
findwordInMessages = {}

with open(filename, 'r') as f:
    for line in f:
        parts = re.split("\t", line)

        msgId = parts[0]
        spamIndicator = int(parts[1])

        subject = "" if parts[2].strip() == "NA" else parts[2]
        body = "" if parts[3].strip() == "NA" else parts[3]
        text = subject + " " + body

        # Update the various counters

        msgCounts[spamIndicator] += 1

        wordCountsOfOneMsg = {} # Store the occurrence count of each wor
        for word in findwords:
            count = len(regexes[word].findall(text))
            if count > 0:
                findwordCounts[spamIndicator][word] += count
                wordCountsOfOneMsg[word] = count

        findwordInMessages[msgId] = [spamIndicator, wordCountsOfOneMsg]

        totalwordCounts[spamIndicator] += len(regexAll.findall(text))

# Use cPickle to dump all the data
```

In [11]:
```
%%writefile reducer.py
#!/usr/bin/python
## reducer.py
## Author: Patrick Ng
## Description: reducer code for HW1.4

import sys
import re
import math
from collections import defaultdict
from collections import Counter
import collections
import cPickle

def trainMultinomialNB(msgCounts, findwordCounts, totalwordCounts, vocab
    """
    Train the Multinomial Naive Bayes model.

    Parameters
    ----------
        msgCounts : list of int
            Count of messages in each class (0=non-spam, 1=spam)

        findwordCounts : list of dict of int
            The list contains the occurrence count info in each class (0
            In each dict, each item's key is the word, and each item's v
            count of that word.

        totalwordCounts : list of int
            Total number of words in each class (0=non-spam, 1=spam)

        vocab : set of str
            The vocabulary set

    Returns
    -------
        priors : dict of float
            The priors (Pr(c)) of each class.

        condProbs : dict of dict of float
            The outer dictionary contains the conditional probabilities
            For each of the inner dict, each item's key is the word, and
            is the conditional probability (Pr(X=t|c)) of that word.
    """
    priors = {} # Pr(c)
    condProbs = {0:{}, 1:{}} # Pr(X=t|c) of each word in each class
    msgTotal = sum(msgCounts)

    # 0: non-spam, 1: spam
    for c in [0, 1]:
        priors[c] = float(msgCounts[c]) / msgTotal
        for term in vocab:
            # By default we'll set word count to be zero if the term isn
            wordCount = findwordCounts[c].get(term, 0)

            # Note: we will use Laplace smoothing because the word "enla
            #       isn't found in the whole corpus.
            condProbs[c][term] = float(wordCount + 1) / (totalwordCounts

    return priors, condProbs

## collect user input
filenames = sys.argv[1:]
```

In [12]:
```
!./pNaiveBayes.sh 4 "assistance valium enlargementWithATypo"
!cat enronemail_1h.txt.output
```

```
0001.1999-12-10.farmer  0       0
0001.1999-12-10.kaminski        0       0
0001.2000-01-17.beck    0       0
0001.2000-06-06.lokay   0       0
0001.2001-02-07.kitchen 0       0
0001.2001-04-02.williams        0       0
0002.1999-12-13.farmer  0       0
0002.2001-02-07.kitchen 0       0
0002.2001-05-25.SA_and_HP       1       0
0002.2003-12-18.GP      1       0
0002.2004-08-01.BG      1       1
0003.1999-12-10.kaminski        0       0
0003.1999-12-14.farmer  0       0
0003.2000-01-17.beck    0       0
0003.2001-02-08.kitchen 0       0
0003.2003-12-18.GP      1       0
0003.2004-08-01.BG      1       0
0004.1999-12-10.kaminski        0       1
0004.1999-12-14.farmer  0       0
0004.2001-04-02.williams        0       0
0004.2001-06-12.SA_and_HP       1       0
0004.2004-08-01.BG      1       0
0005.1999-12-12.kaminski        0       1
0005.1999-12-14.farmer  0       0
0005.2000-06-06.lokay   0       0
0005.2001-02-08.kitchen 0       0
0005.2001-06-23.SA_and_HP       1       0
0005.2003-12-18.GP      1       0
0006.1999-12-13.kaminski        0       0
0006.2001-02-08.kitchen 0       0
0006.2001-04-03.williams        0       0
0006.2001-06-25.SA_and_HP       1       0
0006.2003-12-18.GP      1       0
0006.2004-08-01.BG      1       0
0007.1999-12-13.kaminski        0       0
0007.1999-12-14.farmer  0       0
0007.2000-01-17.beck    0       0
0007.2001-02-09.kitchen 0       0
0007.2003-12-18.GP      1       0
0007.2004-08-01.BG      1       0
0008.2001-02-09.kitchen 0       0
0008.2001-06-12.SA_and_HP       1       0
0008.2001-06-25.SA_and_HP       1       0
0008.2003-12-18.GP      1       0
0008.2004-08-01.BG      1       0
0009.1999-12-13.kaminski        0       0
0009.1999-12-14.farmer  0       0
0009.2000-06-07.lokay   0       0
0009.2001-02-09.kitchen 0       0
0009.2001-06-26.SA_and_HP       1       0
0009.2003-12-18.GP      1       1
0010.1999-12-14.farmer  0       0
0010.1999-12-14.kaminski        0       0
0010.2001-02-09.kitchen 0       0
0010.2001-06-28.SA_and_HP       1       1
0010.2003-12-18.GP      1       0
0010.2004-08-01.BG      1       0
0011.1999-12-14.farmer  0       0
0011.2001-06-28.SA_and_HP       1       1
0011.2001-06-29.SA_and_HP       1       0
0011.2003-12-18.GP      1       0
```

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.

In all cases, mapper.py will read in a portion of the email data, count some words and print out counts to a file.

In [17]:
```
%%writefile mapper.py
#!/usr/bin/python
## mapper.py
## Author: Patrick Ng
## Description: mapper code for HW1.5

import sys
import re
from collections import defaultdict
import cPickle

## collect user input
filename = sys.argv[1]
findwords = re.split(" ",sys.argv[2].lower())
regexes = {}

useAllWords = False

# See if we are asked to use all words
if findwords[0].strip() == '*':
    useAllWords = True
else:
    # Create the list of compiled regex, one for each findword
    for word in findwords:
        regexes[word] = re.compile(r'\b' + re.escape(word) + r'\b')

# Regex for counting the total number of words in a msg
regexAll = re.compile(r'\b\w+\b')

# For each of the count lists below, index=0 means non-spam, index=1 mea

# For each class (each item in the list), the occurrence count of each w
# each item's key is the word, and the item's value is the count)
findwordCounts = [defaultdict(int), defaultdict(int)]
totalwordCounts = [0, 0] # The total counts among the two classes.
msgCounts = [0, 0]  # Total count of msg among the two classes.

# A dictionary holding the findword info for each message.
# Key = MsgId
# Value = a list: [message's true class, the occurrence count of each wo
# The "occurrence count of each word" is stored as a dictionary where th
# and the value is the count.
findwordInMessages = {}

with open(filename, 'r') as f:
    for line in f:
        parts = re.split("\t", line)

        msgId = parts[0]
        spamIndicator = int(parts[1])

        subject = "" if parts[2].strip() == "NA" else parts[2]
        body = "" if parts[3].strip() == "NA" else parts[3]
        text = subject + " " + body

        # Update the various counters

        msgCounts[spamIndicator] += 1

        wordCountsOfOneMsg = {} # Occurrence count of each word in this

        if useAllWords:
            wordsInMsg = regexAll.findall(text) # The vocab in this mess
        else:
```

In [18]:
```
%%writefile reducer.py
#!/usr/bin/python
## reducer.py
## Author: Patrick Ng
## Description: reducer code for HW1.5

import sys
import re
import math
from collections import defaultdict
from collections import Counter
import collections
import cPickle

def trainMultinomialNB(msgCounts, findwordCounts, totalwordCounts, vocab
    """
    Train the Multinomial Naive Bayes model.

    Parameters
    ----------
        msgCounts : list of int
            Count of messages in each class (0=non-spam, 1=spam)

        findwordCounts : list of dict of int
            The list contains the occurrence count info in each class (0
            In each dict, each item's key is the word, and each item's v
            count of that word.

        totalwordCounts : list of int
            Total number of words in each class (0=non-spam, 1=spam)

        vocab : set of str
            The vocabulary set

    Returns
    -------
        priors : dict of float
            The priors (Pr(c)) of each class.

        condProbs : dict of dict of float
            The outer dictionary contains the conditional probabilities
            For each of the inner dict, each item's key is the word, and
            is the conditional probability (Pr(X=t|c)) of that word.
    """
    priors = {} # Pr(c)
    condProbs = {0:{}, 1:{}} # Pr(X=t|c) of each word in each class
    msgTotal = sum(msgCounts)

    # 0: non-spam, 1: spam
    for c in [0, 1]:
        priors[c] = float(msgCounts[c]) / msgTotal
        for term in vocab:
            # By default we'll set word count to be zero if the term isn
            wordCount = findwordCounts[c].get(term, 0)

            # Note: we will use Laplace smoothing because the word "enla
            #       isn't found in the whole corpus.
            condProbs[c][term] = float(wordCount + 1) / (totalwordCounts

    return priors, condProbs


## collect user input
filenames = sys.argv[1:]
```

In [19]:
```
!./pNaiveBayes.sh 4 "*"
!cat enronemail_1h.txt.output
```

```
0001.1999-12-10.farmer  0       0
0001.1999-12-10.kaminski        0       1
0001.2000-01-17.beck    0       0
0001.2000-06-06.lokay   0       0
0001.2001-02-07.kitchen 0       0
0001.2001-04-02.williams        0       1
0002.1999-12-13.farmer  0       0
0002.2001-02-07.kitchen 0       1
0002.2001-05-25.SA_and_HP       1       1
0002.2003-12-18.GP      1       1
0002.2004-08-01.BG      1       1
0003.1999-12-10.kaminski        0       0
0003.1999-12-14.farmer  0       0
0003.2000-01-17.beck    0       0
0003.2001-02-08.kitchen 0       0
0003.2003-12-18.GP      1       1
0003.2004-08-01.BG      1       1
0004.1999-12-10.kaminski        0       0
0004.1999-12-14.farmer  0       0
0004.2001-04-02.williams        0       1
0004.2001-06-12.SA_and_HP       1       1
0004.2004-08-01.BG      1       1
0005.1999-12-12.kaminski        0       0
0005.1999-12-14.farmer  0       0
0005.2000-06-06.lokay   0       1
0005.2001-02-08.kitchen 0       0
0005.2001-06-23.SA_and_HP       1       1
0005.2003-12-18.GP      1       1
0006.1999-12-13.kaminski        0       1
0006.2001-02-08.kitchen 0       0
0006.2001-04-03.williams        0       1
0006.2001-06-25.SA_and_HP       1       1
0006.2003-12-18.GP      1       1
0006.2004-08-01.BG      1       1
0007.1999-12-13.kaminski        0       0
0007.1999-12-14.farmer  0       0
0007.2000-01-17.beck    0       0
0007.2001-02-09.kitchen 0       0
0007.2003-12-18.GP      1       1
0007.2004-08-01.BG      1       1
0008.2001-02-09.kitchen 0       0
0008.2001-06-12.SA_and_HP       1       1
0008.2001-06-25.SA_and_HP       1       1
0008.2003-12-18.GP      1       1
0008.2004-08-01.BG      1       1
0009.1999-12-13.kaminski        0       0
0009.1999-12-14.farmer  0       0
0009.2000-06-07.lokay   0       0
0009.2001-02-09.kitchen 0       0
0009.2001-06-26.SA_and_HP       1       1
0009.2003-12-18.GP      1       1
0010.1999-12-14.farmer  0       0
0010.1999-12-14.kaminski        0       0
0010.2001-02-09.kitchen 0       0
0010.2001-06-28.SA_and_HP       1       1
0010.2003-12-18.GP      1       1
0010.2004-08-01.BG      1       1
0011.1999-12-14.farmer  0       0
0011.2001-06-28.SA_and_HP       1       1
0011.2001-06-29.SA_and_HP       1       1
0011.2003-12-18.GP      1       1
```

```
In [ ]:
```