DATASCI W261, Machine Learning at Scale

Assignement: week#1

Lei Yang (leiyang@berkeley.edu)

Due: 2016-01-19

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

HW1.0.0 Answer:

Big-data problem usually has the characteristics of **4V's (http://www.ibmbigdatahub.com/infographic/four-vs-big-data)**, namely volume, velocity, variety, and veracity

- *volume*: in the range of petabyte at least, where regular single server cannot handle, and must resort to distributed computing
- *velocity*: constant streaming data instead of batch data, information is updated in real-time
- *variety*: unstructured data from various sources, fusion together for decision making
- *veracity*: the uncertainty of data, manifested by poor data quality, incomplete record, invalid value, etc.

Example from my domain: I work in manufacturing industry, where numerous sensors are mounted on the equipment to collect process data, such as temperature, flow rate, pressure etc. The real-time data is used for process and equipment control, where algorithms and statistics are applied. As the production capacity increases, challenges for data analysis rise in various areas:

- *volume*: with more sensors installed on the equipment, higher sampling rate required by customer, longer processing time, bigger production capacity, data volume keeps increasing and is approaching the big-data scale
- *velocity*: in the highly automated manufacturing environment, wafer processing is fast, which requires fast data processing for decision making. For fault detection problem, any abnormal phenomenon must be detected promptly before the next wafer starts. For classification problem, accurate prediction is desired to facilitate fault diagnosis.

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 polynomial regression models of degree 1, 2, 3, 4, 5 are considered. How would you select a model?

HW1.0.1 Answer:

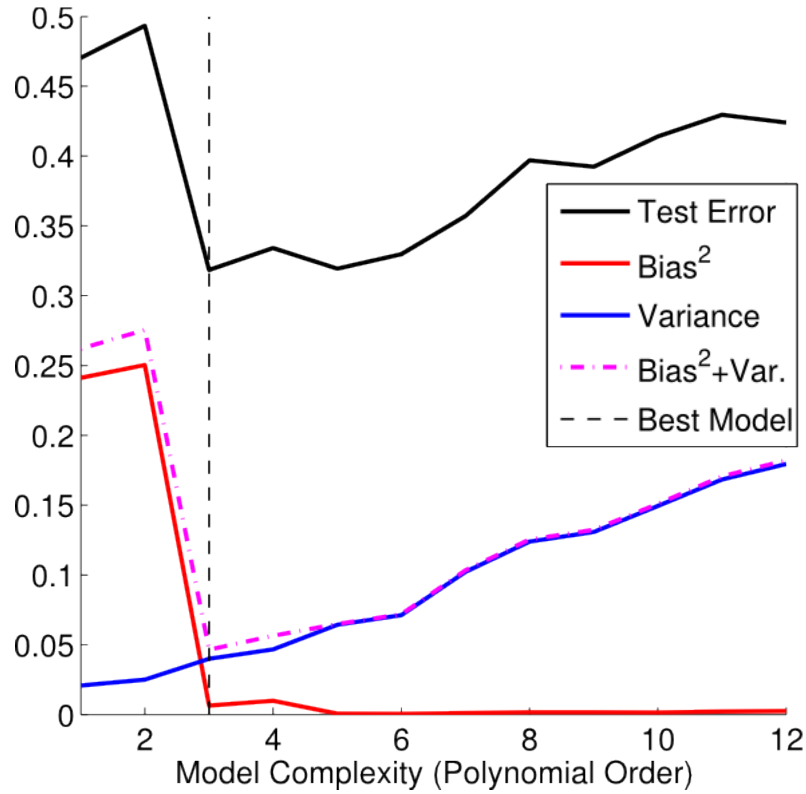
The expected prediction error $\mathbb{E}[(g(x)-y)^2]$ of an estimator, $g(x)$, can be decomposed into 3 parts:

- *variance of the estimator*: $\mathbb{E}[(g(x)-\mathbb{E}[g(x)])^2]$
- *bias of the estimator*: $(\mathbb{E}[g(x)]-f(x))^2$
- *irreducible error*: $\mathbb{E}[(y-f(x))^2]$

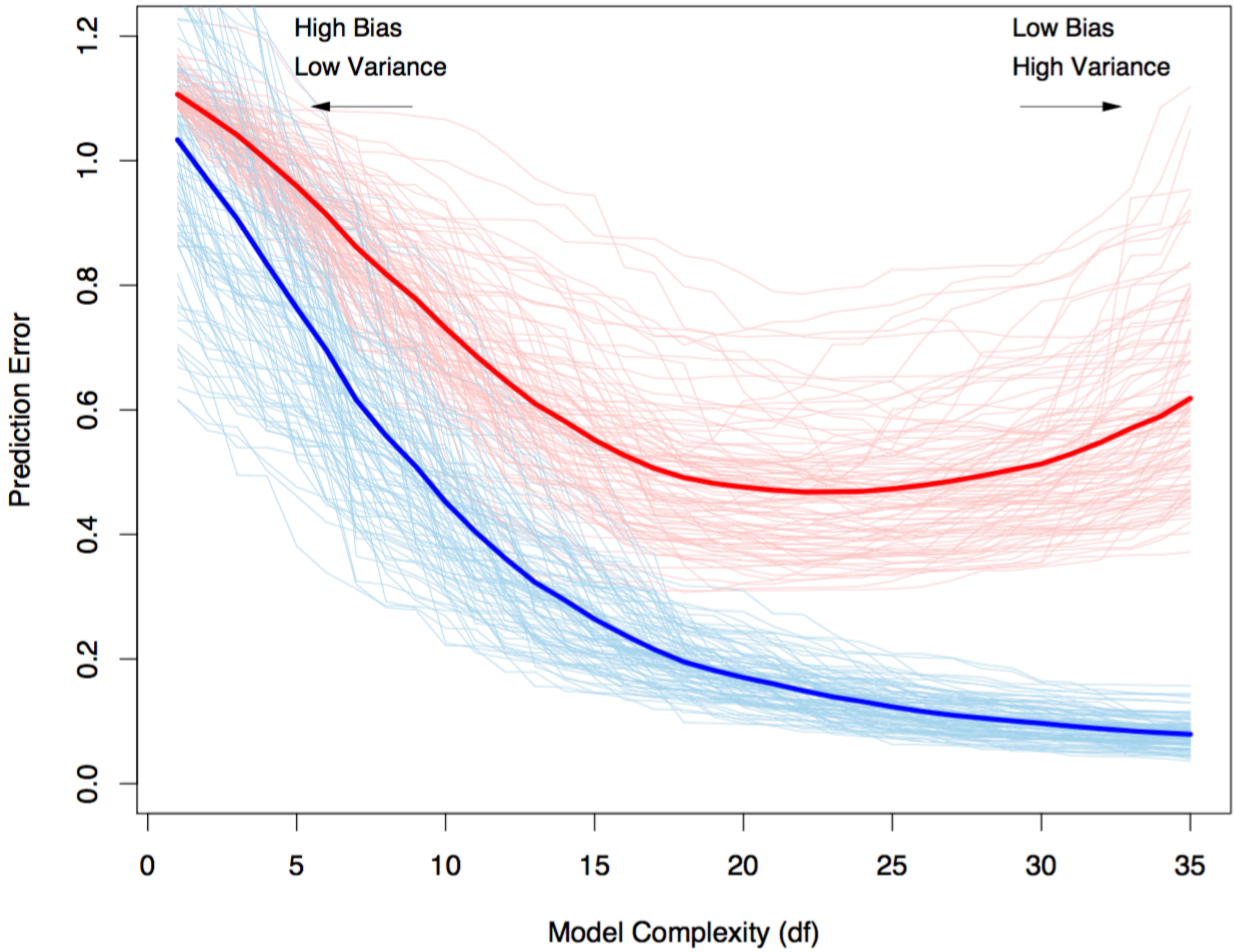
Because the underlying truth $f(x)$ is generally unknown, it is challenging to estimate the variance and bias. One approach to numerically approximate is **bootstrap**, in which a group of sample data (re-draw with replacement) is generated, the steps are:

- with dataset T, bootstrap to generate K datasets with N samples
- for each dataset K_i, estimate a polynomial model g_k with order n
- for each (x, y) in K_i:
  - calculate average prediction $\bar{g} = \frac{1}{K}\sum_{i=1}^{K} g_k(x)$
  - bias is then: $b = \bar{g} - y$
  - variance: $v = \frac{\sum_k (y_k - \bar{g})^2}{K-1}$
  - prediction error: $e = (g(x)-y)^2$
  - irreducible error: $e - v - b$

Repeat the above steps for all order 1 through 5, we can plot the above components for each model. And we would get a plot similar like:



In general, there is a trade-off between bias and variance of any estimator, as illustrated in graph below:



as the model complexity increases, it becomes more flexible fitting the training data and hence the bias will decrease. On the other hand, overly complicate model will fit noise in the data and increase the uncertainty of predictions, thus the variance is increasing. To minimize the prediction error, one should choose the model that comes with the minimum sum of variance and bias. In the example above, model with order 3 has the minimum sum of bias and variance, and should be taken as optimal selection. Taking it holistically, there are a few critiria for model selection, including AIC, BIC, MDL, cross-validation etc.

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

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.

Let's define *pNaiveBayes.sh* script first, we only need to do this once since it is the same throughout HW1

In [2]:
```
%%writefile pNaiveBayes.sh
## pNaiveBayes.sh
## Author: Jake Ryland Williams
## Usage: pNaiveBayes.sh m wordlist
## Input:
##       m = number of processes (maps), e.g., 4
##       wordlist = a space-separated list of words in quotes, e.g., "the and of"
##
## Instructions: Read this script and its comments closely.
##               Do your best to understand the purpose of each command,
##               and focus on how arguments are supplied to mapper.py/reducer.py,
##               as this will determine how the python scripts take input.
##               When you are comfortable with the unix code below,
##               answer the questions on the LMS for HW1 about the starter code.

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

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

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

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

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

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

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

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

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

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

```
Overwriting pNaiveBayes.sh
```

Define mapper.py & reducer.py, and make all scripts executable

- *mapper.py* counts the single specified word for the chunk, and output an integer
- *reducer.py* collates counts from all chunks, and output the total count of the single specified word

In [3]:
```
%%writefile mapper.py
#!/usr/bin/python
import sys
import re
count = 0
WORD_RE = re.compile(r"[\w']+")
filename = sys.argv[1]
countword = sys.argv[2].lower()
with open (filename, "r") as myfile:
    for line in myfile.readlines():
        for word in line.lower().split()[2:]:
            if countword in word:
                count += 1
print countword + ' ' + str(count)
```

```
Overwriting mapper.py
```

In [4]:
```
%%writefile reducer.py
#!/usr/bin/python
import sys
sum = 0
for filename in sys.argv[1:]:
    with open (filename, "r") as myfile:
        for line in myfile.readlines():
            temp = line.split()
            word = temp[0]
            sum += int(temp[1])
print word + ': ' + str(sum)
```

```
Overwriting reducer.py
```

In [5]:
```
!chmod a+x pNaiveBayes.sh
!chmod a+x mapper.py
!chmod a+x reducer.py
```

HW1.2 Results: by checking the ouput file, we know there are 10 counts of word 'assistance'.

In [6]:
```
!./pNaiveBayes.sh 5 "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.

Define mapper.py:

- obtains count for each word from the chunk, for spam and non-spam email separately,
- records all counts in a dictionary,
- outputs the dictionaries (non-spam count, and spam count), (non)spam counts, and keyword.

In [7]:
```
%%writefile mapper.py
#!/usr/bin/python
import sys
import re
# let's use two dictionaries to hold the word counts for spam and non-spam
n_count, s_count = {}, {}
nSpam, nNormal = 0, 0
WORD_RE = re.compile(r"[\w']+")
filename = sys.argv[1]
keyword = sys.argv[2].lower()
with open (filename, "r") as myfile:
    for email in myfile.readlines():
        isSpam = email.split('\t')[1] == '1'
        if isSpam:
            nSpam += 1
            for word in email.lower().split()[2:]: # only use subject & content for modeling
                if word not in s_count:
                    s_count[word] = 1
                else:
                    s_count[word] += 1
        else:
            nNormal += 1
            for word in email.lower().split()[2:]: # only use subject & content for modeling
                if word not in n_count:
                    n_count[word] = 1
                else:
                    n_count[word] += 1
print n_count
print s_count
print nNormal
print nSpam
print "'" + keyword + "'"
```

```
Overwriting mapper.py
```

Define reducer.py:

- collapse wrod counts from all chunks
- estimate NB model parameters: prior and conditional probabilities
- classify messages that contains the keyword
- **Note:** for messages that don't contain the keyword, the decision is solely based on prior probability, which will always give non-spam prediction, thus we skip those messages and only focus on those with the specified keyword
- output results

Parameter estimation background:

Assuming *positional independence*, and with *add-one Laplace smoothing*, the multinomial NB conditional probability $P(t|c)$ can be estimated as:

$$\hat{P}(t \mid c) = \frac{T_{ct} + 1}{(\sum_{t' \in V} T_{ct'}) + B},$$

where $B = |V|$ is the number of terms in the vocabulary V (including all text classes), and T_{ct} is the count of word *t* in class *c*.

To classify a message, the posterior probability of class c can be calculated as:

$$c_{map} = \arg\max_{c \in \mathbb{C}} [\log \hat{P}(c) + \sum_{1 \leqslant k \leqslant n_d} \log \hat{P}(t_k \mid c)],$$

where $\hat{P}(t_k \mid c)$ is estimated above with *positional independence* assumption as $\hat{P}(t \mid c)$.

In [9]:
```
%%writefile reducer.py
#!/usr/bin/python
import sys
import math
from sets import Set

n_count, s_count = {}, {}
nSpam, nNormal = 0, 0
counts = []

# scan through each output file from the chunks
for filename in sys.argv[1:]:
    # we first read out the 2 count dictionaries
    with open (filename, "r") as myfile:
        for line in myfile.readlines():
            cmd = 'counts.append(' + line + ')'
            exec cmd

    # we then combine word counts, for non-spam and spam messages, respectively
    for word in counts[0]:
        if word not in n_count:
            n_count[word] = counts[0][word]
        else:
            n_count[word] += counts[0][word]

    for word in counts[1]:
        if word not in s_count:
            s_count[word] = counts[1][word]
        else:
            s_count[word] += counts[1][word]

    # combine spam and non-spam count
    nNormal += int(counts[2])
    nSpam += int(counts[3])

    # pass along the keyword for classification
    keyword = counts[4]

    # clear counts for next chunk
    counts = []

# we now estimate NB parameters for the specified word, according to the formular above
testfile = 'enronemail_1h.txt'
print 'Classify messages with key word: ' + keyword
B = len(Set(s_count.keys() + n_count.keys()))
tot_n = sum(n_count.values())
tot_s = sum(s_count.values())
p_word_s = 1.0*((s_count[keyword] if keyword in s_count else 0) + 0) / (tot_s + B) # no smoothing
p_word_n = 1.0*((n_count[keyword] if keyword in n_count else 0) + 0) / (tot_n + B)

# finally we classify the messages which contains the specified word
#### prior probability: same for every message, since it's determined by training data ####
p_s = 1.0*nSpam/(nSpam+nNormal)
p_n = 1.0*nNormal/(nSpam+nNormal)

# print model parameters
print '\n============= Model Parameters ============='
print 'P(spam) = %f' %(p_s)
print 'P(non-spam) = %f' %(p_n)
print 'P(%s|spam) = %f' %(keyword, p_word_s)
print 'P(%s|non-spam) = %f' %(keyword, p_word_n)

#### likelihood: dependend on the frequency of specified word ####
print '\n============= Classification Results ============='
print 'TRUTH \t CLASS \t ID'
with open (testfile, "r") as myfile:
    for line in myfile.readlines():
        msg = line.lower().split()
        words = msg[2:] # only include words in subject and content
        n_word = sum([1 if keyword in word else 0 for word in words])
        # if the message doesn't contain our keyword, skip it;
        if n_word == 0:
            continue
        #### posterior probability ####
        p_s_word = math.log(p_s) + n_word * math.log(p_word_s)
        p_n_word = math.log(p_n) + n_word * math.log(p_word_n)
        isSpam = True if p_s_word > p_n_word else False
        # print results
        print ('spam' if int(msg[1]) else 'ham') + '\t' + ('spam' if isSpam else 'ham') + '\t' + msg[0]
```

```
Overwriting reducer.py
```

HW1.3 Results: run the NB classifier with keyword 'assistance', the output file are displayed below:

- **Model parameters**:
  - prior
  - likelihood
- **Classification results**:
  - TRUTH: original label
  - CLASS: filter result
  - ID: message ID

In [10]:
```
!./pNaiveBayes.sh 2 "assistance"
!cat enronemail_1h.txt.output
```

```
Classify messages with key word: assistance

============= Model Parameters =============
P(spam) = 0.440000
P(non-spam) = 0.560000
P(assistance|spam) = 0.000189
P(assistance|non-spam) = 0.000047

============= Classification Results =============
TRUTH    CLASS   ID
spam    spam    0002.2004-08-01.bg
ham     spam    0004.1999-12-10.kaminski
ham     spam    0005.1999-12-12.kaminski
spam    spam    0010.2001-06-28.sa_and_hp
spam    spam    0011.2001-06-28.sa_and_hp
spam    spam    0013.2004-08-01.bg
spam    spam    0018.2001-07-13.sa_and_hp
spam    spam    0018.2003-12-18.gp
```

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.

Definition of mapper.py remains the same as it still just counts words for both classes

In [11]:
```
%%writefile mapper.py
#!/usr/bin/python
import sys
import re
# let's use two dictionaries to hold the word counts for spam and non-spam
n_count, s_count = {}, {}
nSpam, nNormal = 0, 0
WORD_RE = re.compile(r"[\w']+")
filename = sys.argv[1]
keywords = sys.argv[2].lower()
with open (filename, "r") as myfile:
    for email in myfile.readlines():
        isSpam = email.split('\t')[1] == '1'
        if isSpam:
            nSpam += 1
            for word in email.lower().split()[2:]: # only use subject & content for modeling
                if word not in s_count:
                    s_count[word] = 1
                else:
                    s_count[word] += 1
        else:
            nNormal += 1
            for word in email.lower().split()[2:]: # only use subject & content for modeling
                if word not in n_count:
                    n_count[word] = 1
                else:
                    n_count[word] += 1
print n_count
print s_count
print nNormal
print nSpam
print "'" + keywords + "'"
```

```
Overwriting mapper.py
```

Definition of reducer.py is modified to consider multiple keywords, which we use dictionaries to represent

In [14]:
```
%%writefile reducer.py
```

```
#!/usr/bin/python
import sys
import math
from sets import Set

n_count, s_count = {}, {}
nSpam, nNormal = 0, 0
counts = []

# scan through each output file from the chunks
for filename in sys.argv[1:]:
    # we first read out the 2 count dictionaries
    with open (filename, "r") as myfile:
        for line in myfile.readlines():
            cmd = 'counts.append(' + line + ')'
            exec cmd

    # we then combine word counts, for non-spam and spam messages, respectively
    for word in counts[0]:
        if word not in n_count:
            n_count[word] = counts[0][word]
        else:
            n_count[word] += counts[0][word]

    for word in counts[1]:
        if word not in s_count:
            s_count[word] = counts[1][word]
        else:
            s_count[word] += counts[1][word]

    # combine spam and non-spam count
    nNormal += int(counts[2])
    nSpam += int(counts[3])

    # pass along the keyword for classification
    keywords = counts[4].split()

    # clear counts for next chunk
    counts = []

testfile = 'enronemail_1h.txt'
print 'Classify messages with keywords: ' + str(keywords)

# we now estimate NB parameters for the specified word, according to the formular above
B = len(Set(s_count.keys() + n_count.keys()))
tot_n = sum(n_count.values())
tot_s = sum(s_count.values())

#### prior probability ####
p_s = 1.0*nSpam/(nSpam+nNormal)
p_n = 1.0*nNormal/(nSpam+nNormal)

#### conditional probabilities for words ####
p_word_s, p_word_n = {}, {}
for word in keywords:
    p_word_s[word] = 1.0*((s_count[word] if word in s_count else 0) + 1) / (tot_s + B)
    p_word_n[word] = 1.0*((n_count[word] if word in n_count else 0) + 1) / (tot_n + B)

# finally we classify the messages which contains the specified word
#### print model parameters ####
print '\n============= Model Parameters ============='
print 'P(spam) = %f' %(p_s)
print 'P(non-spam) = %f' %(p_n)
for word in keywords:
    print 'P(%s|spam) = %f' %(word, p_word_s[word])
    print 'P(%s|non-spam) = %f' %(word, p_word_n[word])

#### likelihood: dependend on the frequency of specified word ####
print '\n============= Classification Results ============='
print 'TRUTH \t CLASS \t ID'
with open (testfile, "r") as myfile:
    for line in myfile.readlines():
        msg = line.lower().split()
        words = msg[2:] # only include words in subject and content
        #### initialize posterior probability ####
        p_s_word = math.log(p_s)
        p_n_word = math.log(p_n)

        #### add likelihood for each keyword ####
        n_word = 0
        for key in keywords:
            n_key = sum([1 if key in word else 0 for word in words])
            n_word += n_key
            p_s_word += n_key * math.log(p_word_s[key])
            p_n_word += n_key * math.log(p_word_n[key])
```

```
        # if the message doesn't contain any keyword, skip it;
        if n_word == 0:
            continue
        isSpam = True if p_s_word > p_n_word else False
        # print results
        print ('spam' if int(msg[1]) else 'ham') + '\t' + ('spam' if isSpam else 'ham') + '\t' + msg[0]
```

```
Overwriting reducer.py
```

HW1.4 Results: run the NB classifier with keywords 'assistance', 'valium' and 'enlargementWithATypo', the output file are displayed below:

In [15]:
```
!./pNaiveBayes.sh 2 "assistance valium enlargementWithATypo"
!cat enronemail_1h.txt.output
```

```
Classify messages with keywords: ['assistance', 'valium', 'enlargementwithatypo']

============= Model Parameters =============
P(spam) = 0.440000
P(non-spam) = 0.560000
P(assistance|spam) = 0.000227
P(assistance|non-spam) = 0.000093
P(valium|spam) = 0.000038
P(valium|non-spam) = 0.000047
P(enlargementwithatypo|spam) = 0.000038
P(enlargementwithatypo|non-spam) = 0.000047

============= Classification Results =============
TRUTH    CLASS   ID
spam    spam    0002.2004-08-01.bg
ham     spam    0004.1999-12-10.kaminski
ham     spam    0005.1999-12-12.kaminski
spam    ham     0009.2003-12-18.gp
spam    spam    0010.2001-06-28.sa_and_hp
spam    spam    0011.2001-06-28.sa_and_hp
spam    spam    0013.2004-08-01.bg
spam    ham     0016.2003-12-19.gp
spam    ham     0017.2004-08-01.bg
spam    spam    0018.2001-07-13.sa_and_hp
spam    spam    0018.2003-12-18.gp
```

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

Definition of mapper.py remains the same as it still just counts words for both classes

In [16]:
```
%%writefile mapper.py
#!/usr/bin/python
import sys
import re
# let's use two dictionaries to hold the word counts for spam and non-spam
n_count, s_count = {}, {}
nSpam, nNormal = 0, 0
WORD_RE = re.compile(r"[\w']+")
filename = sys.argv[1]
#keywords = sys.argv[2].lower()
with open (filename, "r") as myfile:
    for email in myfile.readlines():
        isSpam = email.split('\t')[1] == '1'
        if isSpam:
            nSpam += 1
            for word in email.lower().split()[2:]: # only use subject & content for modeling
                if word not in s_count:
                    s_count[word] = 1
                else:
                    s_count[word] += 1
        else:
            nNormal += 1
            for word in email.lower().split()[2:]: # only use subject & content for modeling
                if word not in n_count:
                    n_count[word] = 1
                else:
                    n_count[word] += 1
print n_count
print s_count
print nNormal
print nSpam
```

```
Overwriting mapper.py
```

Definition of reducer.py is modified to consider all present words:

In [17]:

```
%%writefile reducer.py
#!/usr/bin/python
import sys
import math
from sets import Set

n_count, s_count = {}, {}
nSpam, nNormal = 0, 0
counts = []

# scan through each output file from the chunks
for filename in sys.argv[1:]:
    # we first read out the 2 count dictionaries
    with open (filename, "r") as myfile:
        for line in myfile.readlines():
            cmd = 'counts.append(' + line + ')'
            exec cmd

    # we then combine word counts, for non-spam and spam messages, respectively
    for word in counts[0]:
        if word not in n_count:
            n_count[word] = counts[0][word]
        else:
            n_count[word] += counts[0][word]

    for word in counts[1]:
        if word not in s_count:
            s_count[word] = counts[1][word]
        else:
            s_count[word] += counts[1][word]

    # combine spam and non-spam count
    nNormal += int(counts[2])
    nSpam += int(counts[3])

    # clear counts for next chunk
    counts = []

testfile = 'enronemail_1h.txt'
print 'Classify messages with all words'

# we now estimate NB parameters for all present words
allwords = Set(s_count.keys() + n_count.keys())
B = len(allwords)
tot_n = sum(n_count.values())
tot_s = sum(s_count.values())

#### prior probability ####
p_s = 1.0*nSpam/(nSpam+nNormal)
p_n = 1.0*nNormal/(nSpam+nNormal)

#### conditional probabilities for words ####
p_word_s, p_word_n = {}, {}
for word in allwords:
    p_word_s[word] = 1.0*((s_count[word] if word in s_count else 0) + .1) / (tot_s + B) #Laplace add 1 smoothing
    p_word_n[word] = 1.0*((n_count[word] if word in n_count else 0) + .1) / (tot_n + B)

# finally we classify the messages which contains the specified word
#### we won't print model parameters, to save some space ####
#### likelihood: dependend on the frequency of current word ####
print '\n============= Classification Results ============='
print 'TRUTH \t CLASS \t ID'
n_correct = 0
with open (testfile, "r") as myfile:
    for line in myfile.readlines():
        msg = line.lower().split()
        words = msg[2:] # only include words in subject and content
        #### initialize posterior probability ####
        p_s_word = math.log(p_s)
        p_n_word = math.log(p_n)

        #### add likelihood for each keyword ####
        for key in Set(words):
            n_key = sum([1 if key in word else 0 for word in words])
            p_s_word += n_key * math.log(p_word_s[key])
            p_n_word += n_key * math.log(p_word_n[key])

        isSpam = True if p_s_word > p_n_word else False
        n_correct += isSpam == int(msg[1])
        # print results
        print ('spam' if int(msg[1]) else 'ham') + '\t' + ('spam' if isSpam else 'ham') + '\t' + msg[0]

print '\nOur multinomial NB training error: %f' %(1-1.0*n_correct/(nSpam+nNormal))
```

Overwriting reducer.py

HW1.5 Results: run the NB classifier all present words, the output file are displayed below:

In [18]:
```
!./pNaiveBayes.sh 4 "dummy"
!cat enronemail_1h.txt.output
```

```
Classify messages with all words

============= Classification Results =============
TRUTH    CLASS   ID
ham     ham     0001.1999-12-10.farmer
ham     ham     0001.1999-12-10.kaminski
ham     ham     0001.2000-01-17.beck
ham     ham     0001.2000-06-06.lokay
ham     ham     0001.2001-02-07.kitchen
ham     ham     0001.2001-04-02.williams
ham     ham     0002.1999-12-13.farmer
ham     ham     0002.2001-02-07.kitchen
spam    spam    0002.2001-05-25.sa_and_hp
spam    spam    0002.2003-12-18.gp
spam    spam    0002.2004-08-01.bg
ham     ham     0003.1999-12-10.kaminski
ham     ham     0003.1999-12-14.farmer
ham     ham     0003.2000-01-17.beck
ham     ham     0003.2001-02-08.kitchen
spam    spam    0003.2003-12-18.gp
spam    spam    0003.2004-08-01.bg
ham     ham     0004.1999-12-10.kaminski
ham     ham     0004.1999-12-14.farmer
ham     ham     0004.2001-04-02.williams
spam    spam    0004.2001-06-12.sa_and_hp
spam    spam    0004.2004-08-01.bg
ham     ham     0005.1999-12-12.kaminski
ham     ham     0005.1999-12-14.farmer
ham     ham     0005.2000-06-06.lokay
ham     ham     0005.2001-02-08.kitchen
spam    spam    0005.2001-06-23.sa_and_hp
spam    spam    0005.2003-12-18.gp
ham     ham     0006.1999-12-13.kaminski
ham     ham     0006.2001-02-08.kitchen
ham     ham     0006.2001-04-03.williams
spam    spam    0006.2001-06-25.sa_and_hp
spam    spam    0006.2003-12-18.gp
spam    spam    0006.2004-08-01.bg
ham     ham     0007.1999-12-13.kaminski
ham     ham     0007.1999-12-14.farmer
ham     ham     0007.2000-01-17.beck
ham     ham     0007.2001-02-09.kitchen
spam    spam    0007.2003-12-18.gp
spam    spam    0007.2004-08-01.bg
ham     ham     0008.2001-02-09.kitchen
spam    spam    0008.2001-06-12.sa_and_hp
spam    spam    0008.2001-06-25.sa_and_hp
spam    spam    0008.2003-12-18.gp
spam    spam    0008.2004-08-01.bg
ham     ham     0009.1999-12-13.kaminski
ham     ham     0009.1999-12-14.farmer
ham     ham     0009.2000-06-07.lokay
ham     ham     0009.2001-02-09.kitchen
spam    spam    0009.2001-06-26.sa_and_hp
spam    spam    0009.2003-12-18.gp
ham     ham     0010.1999-12-14.farmer
ham     ham     0010.1999-12-14.kaminski
ham     ham     0010.2001-02-09.kitchen
spam    spam    0010.2001-06-28.sa_and_hp
spam    spam    0010.2003-12-18.gp
spam    spam    0010.2004-08-01.bg
ham     ham     0011.1999-12-14.farmer
spam    spam    0011.2001-06-28.sa_and_hp
spam    spam    0011.2001-06-29.sa_and_hp
spam    spam    0011.2003-12-18.gp
spam    spam    0011.2004-08-01.bg
ham     ham     0012.1999-12-14.farmer
ham     ham     0012.1999-12-14.kaminski
ham     ham     0012.2000-01-17.beck
ham     ham     0012.2000-06-08.lokay
ham     ham     0012.2001-02-09.kitchen
spam    spam    0012.2003-12-19.gp
ham     ham     0013.1999-12-14.farmer
ham     ham     0013.1999-12-14.kaminski
ham     ham     0013.2001-04-03.williams
spam    spam    0013.2001-06-30.sa_and_hp
spam    spam    0013.2004-08-01.bg
ham     ham     0014.1999-12-14.kaminski
```

```
ham     ham     0014.1999-12-15.farmer
ham     ham     0014.2001-02-12.kitchen
spam    spam    0014.2001-07-04.sa_and_hp
spam    spam    0014.2003-12-19.gp
spam    spam    0014.2004-08-01.bg
ham     ham     0015.1999-12-14.kaminski
ham     ham     0015.1999-12-15.farmer
ham     ham     0015.2000-06-09.lokay
ham     ham     0015.2001-02-12.kitchen
spam    spam    0015.2001-07-05.sa_and_hp
spam    spam    0015.2003-12-19.gp
ham     ham     0016.1999-12-15.farmer
ham     ham     0016.2001-02-12.kitchen
spam    spam    0016.2001-07-05.sa_and_hp
spam    spam    0016.2001-07-06.sa_and_hp
spam    spam    0016.2003-12-19.gp
spam    spam    0016.2004-08-01.bg
ham     ham     0017.1999-12-14.kaminski
ham     ham     0017.2000-01-17.beck
ham     ham     0017.2001-04-03.williams
spam    spam    0017.2003-12-18.gp
spam    spam    0017.2004-08-01.bg
spam    spam    0017.2004-08-02.bg
ham     ham     0018.1999-12-14.kaminski
spam    spam    0018.2001-07-13.sa_and_hp
spam    spam    0018.2003-12-18.gp

Our multinomial NB training error: 0.000000
```

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

- Feature vectorization for the emails
- Run the Multinomial Naive Bayes algorithm (using default settings) from SciKit-Learn
- Run the Bernoulli Naive Bayes algorithm from SciKit-Learn (using default settings)
- Run the Multinomial Naive Bayes algorithm from **HW1.5**
- Report Training error

In [19]:

```
from sklearn.naive_bayes import BernoulliNB
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import *

import csv
import numpy as np

# read email message, and organize training data
with open('enronemail_1h.txt', 'r') as f:
    reader = csv.reader(f, delimiter="\t")
    emails = list(reader)
train_label = [msg[1] for msg in emails]
train_data = [msg[2]+msg[3] if len(msg)==4 else msg[2] for msg in emails]
msg_id = [msg[0].lower() for msg in emails]

# feature vectorization
uniVectorizer = CountVectorizer()
dtmTrain = uniVectorizer.fit_transform(train_data)

# multinomial Naive Bayes Classifier from sklearn
mnb = MultinomialNB()
mnb.fit(dtmTrain, train_label)
pred_mnb = mnb.predict(dtmTrain)
training_error_mnb = 1.0*sum(pred_mnb != train_label) / len(train_label)

# Bernoulli Naive Bayes Classifier from sklearn
bnb = BernoulliNB()
bnb.fit(dtmTrain, train_label)
pred_bnb = bnb.predict(dtmTrain)
training_error_bnb = 1.0*sum(pred_bnb != train_label) / len(train_label)

# multinomial Naive Bayes Classifier from HW1.5
!./pNaiveBayes.sh 4 "dummy"

# load results from HW1.5 and generate comparison matrix
print 'TRUTH \t MNB_HW1.5 \t MNB_SK \t BNB_SK \t ID'
with open ('enronemail_1h.txt.output', "r") as myfile:
    for line in myfile.readlines():
        if line.startswith('ham') or line.startswith('spam'):
            result = line.split()
            idx = msg_id.index(result[2])
            result.insert(2, 'spam' if pred_mnb[idx]=='1' else 'ham')
            result.insert(3, 'spam' if pred_bnb[idx]=='1' else 'ham')
            print str.join('\t', result)

        if line.startswith('Our multinomial NB'):
            print '\n' + line.strip('\n')

print 'SK- multinomial NB training error: %f' %training_error_mnb
print 'SK- Bernoulli   NB training error: %f' %training_error_bnb
```

```
TRUTH    MNB_HW1.5       MNB_SK          BNB_SK          ID
ham     ham     ham     ham     0001.1999-12-10.farmer
ham     ham     ham     ham     0001.1999-12-10.kaminski
ham     ham     ham     ham     0001.2000-01-17.beck
ham     ham     ham     ham     0001.2000-06-06.lokay
ham     ham     ham     ham     0001.2001-02-07.kitchen
ham     ham     ham     ham     0001.2001-04-02.williams
ham     ham     ham     ham     0002.1999-12-13.farmer
ham     ham     ham     ham     0002.2001-02-07.kitchen
spam    spam    spam    ham     0002.2001-05-25.sa_and_hp
spam    spam    spam    spam    0002.2003-12-18.gp
spam    spam    spam    ham     0002.2004-08-01.bg
ham     ham     ham     ham     0003.1999-12-10.kaminski
ham     ham     ham     ham     0003.1999-12-14.farmer
ham     ham     ham     ham     0003.2000-01-17.beck
ham     ham     ham     ham     0003.2001-02-08.kitchen
spam    spam    spam    ham     0003.2003-12-18.gp
spam    spam    spam    ham     0003.2004-08-01.bg
ham     ham     ham     ham     0004.1999-12-10.kaminski
ham     ham     ham     ham     0004.1999-12-14.farmer
ham     ham     ham     ham     0004.2001-04-02.williams
spam    spam    spam    spam    0004.2001-06-12.sa_and_hp
spam    spam    spam    ham     0004.2004-08-01.bg
ham     ham     ham     ham     0005.1999-12-12.kaminski
ham     ham     ham     ham     0005.1999-12-14.farmer
ham     ham     ham     ham     0005.2000-06-06.lokay
ham     ham     ham     ham     0005.2001-02-08.kitchen
spam    spam    spam    ham     0005.2001-06-23.sa_and_hp
spam    spam    spam    spam    0005.2003-12-18.gp
ham     ham     ham     ham     0006.1999-12-13.kaminski
ham     ham     ham     ham     0006.2001-02-08.kitchen
ham     ham     ham     ham     0006.2001-04-03.williams
spam    spam    spam    ham     0006.2001-06-25.sa_and_hp
spam    spam    spam    spam    0006.2003-12-18.gp
```

```
spam    spam    spam    spam    0006.2004-08-01.bg
ham     ham     ham     ham     0007.1999-12-13.kaminski
ham     ham     ham     ham     0007.1999-12-14.farmer
ham     ham     ham     ham     0007.2000-01-17.beck
ham     ham     ham     ham     0007.2001-02-09.kitchen
spam    spam    spam    spam    0007.2003-12-18.gp
spam    spam    spam    spam    0007.2004-08-01.bg
ham     ham     ham     ham     0008.2001-02-09.kitchen
spam    spam    spam    spam    0008.2001-06-12.sa_and_hp
spam    spam    spam    spam    0008.2001-06-25.sa_and_hp
spam    spam    spam    ham     0008.2003-12-18.gp
spam    spam    spam    spam    0008.2004-08-01.bg
ham     ham     ham     ham     0009.1999-12-13.kaminski
ham     ham     ham     ham     0009.1999-12-14.farmer
ham     ham     ham     ham     0009.2000-06-07.lokay
ham     ham     ham     ham     0009.2001-02-09.kitchen
spam    spam    spam    spam    0009.2001-06-26.sa_and_hp
spam    spam    spam    spam    0009.2003-12-18.gp
ham     ham     ham     ham     0010.1999-12-14.farmer
ham     ham     ham     ham     0010.1999-12-14.kaminski
ham     ham     ham     ham     0010.2001-02-09.kitchen
spam    spam    spam    spam    0010.2001-06-28.sa_and_hp
spam    spam    spam    ham     0010.2003-12-18.gp
spam    spam    spam    spam    0010.2004-08-01.bg
ham     ham     ham     ham     0011.1999-12-14.farmer
spam    spam    spam    spam    0011.2001-06-28.sa_and_hp
spam    spam    spam    spam    0011.2001-06-29.sa_and_hp
spam    spam    spam    ham     0011.2003-12-18.gp
spam    spam    spam    ham     0011.2004-08-01.bg
ham     ham     ham     ham     0012.1999-12-14.farmer
ham     ham     ham     ham     0012.1999-12-14.kaminski
ham     ham     ham     ham     0012.2000-01-17.beck
ham     ham     ham     ham     0012.2000-06-08.lokay
ham     ham     ham     ham     0012.2001-02-09.kitchen
spam    spam    spam    ham     0012.2003-12-19.gp
ham     ham     ham     ham     0013.1999-12-14.farmer
ham     ham     ham     ham     0013.1999-12-14.kaminski
ham     ham     ham     ham     0013.2001-04-03.williams
spam    spam    spam    spam    0013.2001-06-30.sa_and_hp
spam    spam    spam    spam    0013.2004-08-01.bg
ham     ham     ham     ham     0014.1999-12-14.kaminski
ham     ham     ham     ham     0014.1999-12-15.farmer
ham     ham     ham     ham     0014.2001-02-12.kitchen
spam    spam    spam    spam    0014.2001-07-04.sa_and_hp
spam    spam    spam    ham     0014.2003-12-19.gp
spam    spam    spam    ham     0014.2004-08-01.bg
ham     ham     ham     ham     0015.1999-12-14.kaminski
ham     ham     ham     ham     0015.1999-12-15.farmer
ham     ham     ham     ham     0015.2000-06-09.lokay
ham     ham     ham     ham     0015.2001-02-12.kitchen
spam    spam    spam    spam    0015.2001-07-05.sa_and_hp
spam    spam    spam    spam    0015.2003-12-19.gp
ham     ham     ham     ham     0016.1999-12-15.farmer
ham     ham     ham     ham     0016.2001-02-12.kitchen
spam    spam    spam    spam    0016.2001-07-05.sa_and_hp
spam    spam    spam    spam    0016.2001-07-06.sa_and_hp
spam    spam    spam    spam    0016.2003-12-19.gp
spam    spam    spam    ham     0016.2004-08-01.bg
ham     ham     ham     ham     0017.1999-12-14.kaminski
ham     ham     ham     ham     0017.2000-01-17.beck
ham     ham     ham     ham     0017.2001-04-03.williams
spam    spam    spam    ham     0017.2003-12-18.gp
spam    spam    spam    spam    0017.2004-08-01.bg
spam    spam    spam    spam    0017.2004-08-02.bg
ham     ham     ham     ham     0018.1999-12-14.kaminski
spam    spam    spam    spam    0018.2001-07-13.sa_and_hp
spam    spam    spam    spam    0018.2003-12-18.gp

Our multinomial NB training error: 0.000000
SK- multinomial NB training error: 0.000000
SK- Bernoulli   NB training error: 0.160000
```

In [ ]: