- Name : Prabhakar Gundugola
- Email address: prabhakar@berkeley.edu
- W261-3 : Spring 2016
- Week 1 : Homework 1
- Date : January 19, 2016

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 used to mainly describe the data that have the following characteristics:

- 1) Volume - massive volume of generated and stored data that has the potential to be mined for information.
- 2) Variety - type and nature of data, which could be structured, semi-structured, and unstructured.
- 3) Velocity - speed at which the data is generated and processed.
- 4) Veracity - quality of data
- 5) Complexity - so large and complex that traditional database and software techniques are not adequate.

I work in a Data center company that owns over 145 data centers all over the world. Each and every data center has hundreds of instruments that are IoT enabled. These instruments emit events every minute that we capture and store in a big data lake for data mining and analytics. In addition to that, we also store log files generated by Infrastructure systems. As you see, this problem has all the above big data characteristics and we have to use big data technologies to acquire, store, and process the data as traditional data procecssing techniques cannot scale to meet the needs.

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?

Answer:

Definitions

- Bias and Variance are 2 different sources of reducible errors that affect model accuracy.
- Bias is the difference between the expected predicted value and the actual value for any given observation.
- Variance is the variability of model prediction for any given observation.
- Irreducible error is the noise that cannot fundamentally be reduced by any model.

Relationship between bias, variance, and model complexity

- Dealing with bias and variance is all about dealing with over-fitting and under-fitting.
- When the degree of polynomial (model complexity) is increased, it results in over-fitting. This leads to decrease in bias and increase in variance.
- Conversely, under-fitting results in increase in bias and decrease in variance

Estimation

Step 1: Create training, validation, and test data sets

- If the size of test dataset T is not large, then apply bootstrapping (process of resampling the dataset with replacement) from T and generate multiple data sets, say 100 datasets.
- For each of these 100 datasets, split the dataset into training, validation, and test datasets in the proportion of 50%, 25%, 25%.
- Fit each of the 100 training sets with polynomials of degree 1, 2, 3, 4, 5. It will result in 100 models for each degree.

Step 2: Estimate Bias

- Determine the bias for each observation x, which is the difference between the expected predicted value and the actual value.

$$Bias = E[y] - f(x)$$

Step 3: Estimate Variance

- Determinie the variance, which is the squared sum of differences between the predicted value and the expected predicted value .

$$Variance = E[(y - E[y])^2]$$

Step 4: Estimate Noise

- Determine the noise, which is the squared sum of differences between the predicted value and the actual value.

$$Noise = E[(y - f(x)^2] = \sigma^2$$

In [9]:

```python
from IPython.display import Image
Image(filename='BiasVarianceEquation.png', width=500, height=500)
```

Out[9]:

Bias-variance decomposition (2)

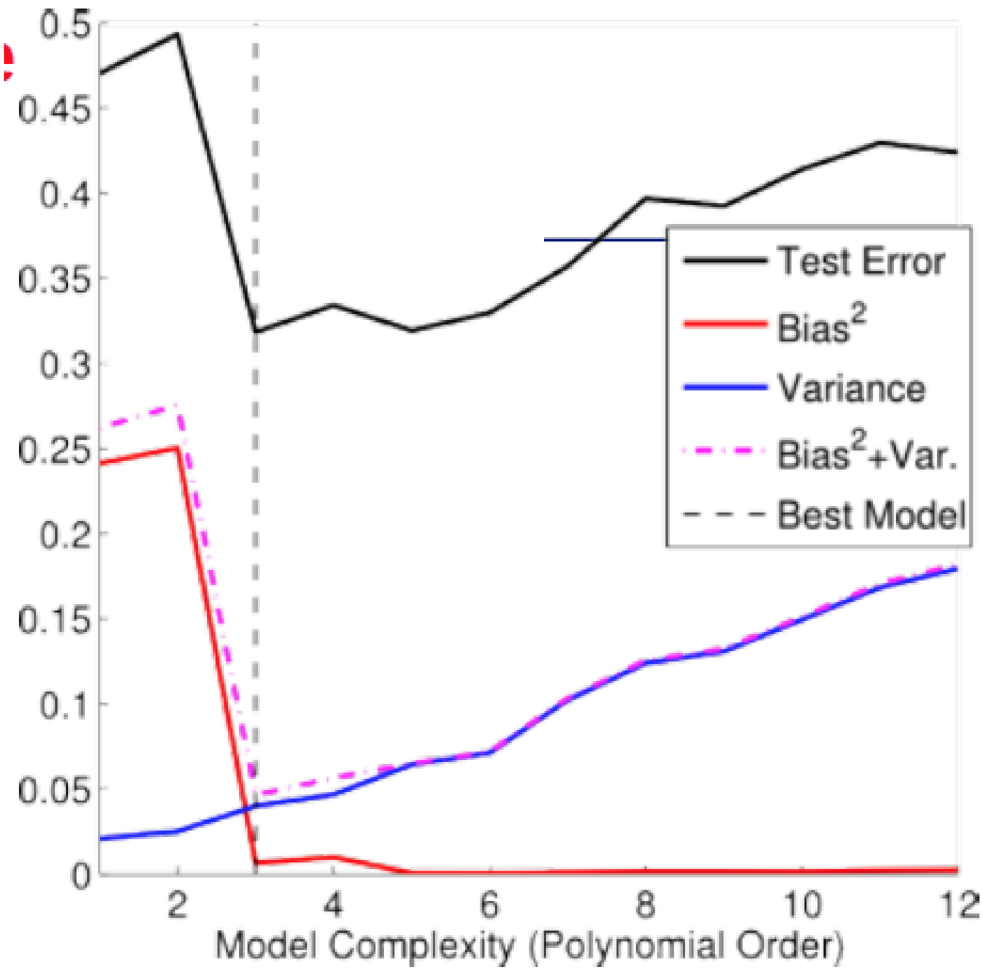
- Putting everything together, we have:

$$\begin{aligned}
E_P\left[(y - h(\mathbf{x}))^2\right] &= E_P\left[(h(\mathbf{x}) - \bar{h}(\mathbf{x}))^2\right] + \\
&\quad \bar{h}(\mathbf{x})^2 - 2f(\mathbf{x})\bar{h}(\mathbf{x}) + f(\mathbf{x})^2 + \\
&\quad E_P\left[(y - f(\mathbf{x}))^2\right] \\
&= E_P\left[(h(\mathbf{x}) - \bar{h}(\mathbf{x}))^2\right] + \qquad \text{(variance)} \\
&\quad (h(\mathbf{x}) - f(\mathbf{x}))^2 + \qquad \text{(bias)}^2 \\
&\quad E_P\left[(y - f(\mathbf{x}))^2\right] \qquad \text{(noise)} \\
&= \text{Var}[h(\mathbf{x})] + \text{Bias}[h(\mathbf{x})]^2 + E_P[\varepsilon^2] \\
&= \text{Var}[h(\mathbf{x})] + \text{Bias}[h(\mathbf{x})]^2 + \sigma^2
\end{aligned}$$

- Expected prediction error = Variance + Bias^2 + Noise^2

```
In [2]: Image(filename='BiasVarianceGraph.png')
```

Out[2]:



Model selection

The optimum model is the level of degree at which the increase in bias is equivalent to the reduction in variance. In practice, there is no way to find this equivalence.

For model selection, we need to determine the accurate measure of expected prediction error for different degrees and then choose the degree that minimizes the overall error.

The below graph depicts the Bias-Variance trade-off and the optimum degree as per the below graph is 3.

Run control script

In [9]:
```
%%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 a
nd 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/reduc
er.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
mapper=$3 ## mapper program
reducer=$4 ## reducer program

## 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 $data.chunk.*; do
    ## feed word list to the python mapper here and redirect STDOUT to a te
mporary file on disk
    ####
    ####
    ./$mapper.py $datachunk "$wordlist" > $datachunk.counts &
    ####
    ####
done
## wait for the mappers to finish their work
wait

## 'ls' makes a list of the temporary count files
```

```
## 'perl -pe' regex replaces line breaks with spaces
countfiles=`\ls $data.chunk.*.counts | perl -pe 's/\n/ /'`

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

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

```
Overwriting pNaiveBayes.sh
```

In [2]:

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

Data validation and cleansing

By exploring the data in the input dataset, 2 problems with 3 records are identified:

- There are 2 records with only 3 fields instead of 4 fields.
- There is 1 record with extra new line character in the body field

Data cleansing algorithm

- Open enronemail_1new.txt with "w" permissions
- Initialize prev_line = ""
- For each line as line in enronemail_1h.txt file
    - tokenize with delimiter "\t"
    - If number of tokens >= 3 then
        - If number of tokens == 3 then
            - Add "\t" as another token between 2nd and 3rd tokens. Now total number of tokens = 4.
            - Update line by concatenating all the 4 okens
        - If prev_line != "" then write prev_line in enronemail_1new.txt
        - prev_line = line
    - If number of tokens == 1 then
        - prev_line = prev_line + line

In [3]:
```
# Data cleansing algorithm

import os
import re

# Open enronemail_1new.txt with "w" permissions.
with open("enronemail_1new.txt", "w") as new:
    with open("enronemail_1h.txt", "rU") as old:
        # curr_line is the line to be written to new file. Initially it is
set to "".
        prev_line = ""

        # For every line in enronemail_1h.txt file
        for line in old:

            line = line.strip()
            # Split the line into tokens
            tokens = line.split('\t')

            if len(tokens) >= 3:

                # If subject field is missed out, add blank token and recon
struct the line
                if len(tokens) == 3:
                    line = tokens[0] + '\t' + tokens[1] + '\t' + '' + '\t'
+ tokens[2]

                # If len(tokens) == 4 then this line is valid. Keep it in b
uffer.
                # Now copy the previous line (if not blank).
                if prev_line != "":
                    prev_line += '\n'
                    new.write(prev_line)
                prev_line = line

            # If there is only one field, it must be because of an
            # extra new line character in the previous line body field.
            if len(tokens) == 1:
                # Add this line too to the previous line
                prev_line += line

        # Add the last line to the new file
        new.write(prev_line)

# Now rename enronemail_1new.txt to enronemail_1h.txt
os.rename('enronemail_1new.txt', 'enronemail_1h.txt')

print "Cleanup completed"
```

```
Cleanup completed
```

HW1.1

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

In [4]:
```
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.

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

Input

- 2 Input arguments
  - dataset file name
  - list of words separated by space in double quoted string

Output

- Outputs a tab delimited file with 2 fields:
  - word
  - count

In [93]:
```
%%writefile mapper12.py
#!/usr/bin/python
## mapper12.py
## Author: Prabhakar Gundugola
## Description: mapper code for HW1.2

import sys
import re
import string

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

with open (filename, "rU") as myfile:
    for line in myfile:
        tokens = line.lower().split('\t')

        # Concatenate subject and body fields and store it in word_string
        word_string = tokens[2] + ' ' + tokens[3].strip()

        # Remove punctuation
        word_string = word_string.translate(string.maketrans("",""),
                                            string.punctuation)

        for word in findwords:
            if word in word_string:
                print word + '\t' + str(word_string.count(word))
```

```
Overwriting mapper12.py
```

Reducer

Input

- Mapper output files

Output

- Prints the following output fields separated by '\t'
    - word
    - count

In [11]:
```
%%writefile reducer12.py
#!/usr/bin/python
## reducer12.py
## Author: Prabhakar Gundugola
## Description: reducer code for HW1.2-1.4

import sys

filenames = sys.argv[1:]

word_count = {}

for filename in filenames:
    with open(filename, "r") as myfile:
        for line in myfile:
            word, value = line.split('\t', 1)
            if word not in word_count:
                word_count[word] = int(value)
            else:
                word_count[word] += int(value)

for word in word_count:
    print word + '\t' + str(word_count[word])
```

```
Writing reducer12.py
```

In [12]:
```
!chmod a+x mapper12.py
!chmod a+x reducer12.py
```

In [13]:
```
!./pNaiveBayes.sh 4 "assistance" "mapper12" "reducer12"
!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

Mapper

In [24]:

```
%%writefile mapper13.py
#!/usr/bin/python
## mapper13.py
## Author: Prabhakar Gundugola
## Description: mapper code for HW1.3

import sys
import re
import string

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

with open (filename, "r") as myfile:
    for line in myfile:
        tokens = line.lower().split('\t')

        # Concatenate subject and body fields and store it in word_string
        word_string = tokens[2] + ' ' + tokens[3].strip()

        # Remove punctuation
        word_string = word_string.translate(string.maketrans("",""),
                                            string.punctuation)

        for word in findwords:
            key = tokens[0] + '\t' + tokens[1]
            key += '\t' + word + '\t' + str(len(word_string.split()))
            print key + '\t' + str(word_string.count(word))
```

```
Overwriting mapper13.py
```

Reducer

In [94]:
```
%%writefile reducer13.py
#!/usr/bin/python
## reducer13.py
## Author: Prabhakar Gundugola
## Description: reducer code for HW1.3

import sys
import math

def isspam(true_class):
    if true_class == 1:
        return 'SPAM'
    else:
        return 'HAM'

filenames = sys.argv[1:]

spam_email_count = 0
ham_email_count = 0

spam_word_count = 0
ham_word_count = 0

spam_findword_count = 0
ham_findword_count = 0

total_cases = 0
correct_cases = 0

for filename in filenames:
    with open(filename, "r") as myfile:
        for line in myfile:
            tokens = line.split('\t')
            doc_id = tokens[0]
            true_class = int(tokens[1])
            findword = tokens[2]
            findword_count = int(tokens[4])
            word_count = int(tokens[3])

            # Determine number of SPAM/HAM emails that contain the input wo
rd. HAM is not SPAM
            if true_class == 1:
                spam_email_count += 1
                spam_word_count += word_count
                spam_findword_count += findword_count
            else:
                ham_email_count += 1
                ham_word_count += word_count
                ham_findword_count += findword_count

# Calculate the prior probabilities of both SPAM and HAM
spam_prior = math.log((1.0*spam_email_count)/(spam_email_count + ham_emai
l_count))
ham_prior = math.log((1.0*ham_email_count)/(ham_email_count + spam_email_co
unt))
```

```
spam_findword_prob = math.log((1.0*spam_findword_count/spam_word_count))
ham_findword_prob = math.log((1.0*ham_findword_count/ham_word_count))

# Naive Bayes classification
for filename in filenames:
    with open(filename, "r") as myfile:
        for line in myfile:
            total_cases += 1
            tokens = line.split('\t')
            doc_id = tokens[0]
            true_class = int(tokens[1])
            findword_count = int(tokens[4])

            spam_doc_prob = spam_prior + spam_findword_prob*findword_count
            ham_doc_prob = ham_prior + ham_findword_prob*findword_count

            result = doc_id.ljust(30) + '\t\t' + isspam(true_class) +
'\t\t'
            if spam_doc_prob > ham_doc_prob:
                predicted = 1
            else:
                predicted = 0
            result += isspam(predicted)
            print result

            if true_class == predicted:
                correct_cases += 1

accuracy = 1.0*correct_cases/total_cases
print "------------------------"
print "Accuracy: " + str(accuracy*100) + '%'

print "Total number of documents considered for classification: ", total_ca
ses
print "Number of documents correctly classified: ", correct_cases
```

```
Overwriting reducer13.py
```

In [95]:
```
!chmod a+x mapper13.py
!chmod a+x reducer13.py

!./pNaiveBayes.sh 4 "assistance" "mapper13" "reducer13"
!cat "enronemail_1h.txt.output"
```

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	HAM	HAM
0002.2001-05-25.sa_and_hp	SPAM	HAM
0002.2003-12-18.gp	SPAM	HAM
0002.2004-08-01.bg	SPAM	SPAM
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	HAM	HAM
0003.2003-12-18.gp	SPAM	HAM
0003.2004-08-01.bg	SPAM	HAM
0004.1999-12-10.kaminski	HAM	SPAM
0004.1999-12-14.farmer	HAM	HAM
0004.2001-04-02.williams	HAM	HAM
0004.2001-06-12.sa_and_hp	SPAM	HAM
0004.2004-08-01.bg	SPAM	HAM
0005.1999-12-12.kaminski	HAM	SPAM
0005.1999-12-14.farmer	HAM	HAM
0005.2000-06-06.lokay	HAM	HAM
0005.2001-02-08.kitchen	HAM	HAM
0005.2001-06-23.sa_and_hp	SPAM	HAM
0005.2003-12-18.gp	SPAM	HAM
0006.1999-12-13.kaminski	HAM	HAM
0006.2001-02-08.kitchen	HAM	HAM
0006.2001-04-03.williams	HAM	HAM
0006.2001-06-25.sa_and_hp	SPAM	HAM
0006.2003-12-18.gp	SPAM	HAM
0006.2004-08-01.bg	SPAM	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	HAM	HAM
0007.2003-12-18.gp	SPAM	HAM
0007.2004-08-01.bg	SPAM	HAM
0008.2001-02-09.kitchen	HAM	HAM
0008.2001-06-12.sa_and_hp	SPAM	HAM
0008.2001-06-25.sa_and_hp	SPAM	HAM
0008.2003-12-18.gp	SPAM	HAM
0008.2004-08-01.bg	SPAM	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	HAM	HAM
0009.2001-06-26.sa_and_hp	SPAM	HAM
0009.2003-12-18.gp	SPAM	HAM
0010.1999-12-14.farmer	HAM	HAM
0010.1999-12-14.kaminski	HAM	HAM
0010.2001-02-09.kitchen	HAM	HAM
0010.2001-06-28.sa_and_hp	SPAM	SPAM

```
0010.2003-12-18.gp                       SPAM            HAM
0010.2004-08-01.bg                       SPAM            HAM
0011.1999-12-14.farmer                   HAM             HAM
0011.2001-06-28.sa_and_hp                SPAM            SPAM
0011.2001-06-29.sa_and_hp                SPAM            HAM
0011.2003-12-18.gp                       SPAM            HAM
0011.2004-08-01.bg                       SPAM            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                  HAM             HAM
0012.2003-12-19.gp                       SPAM            HAM
0013.1999-12-14.farmer                   HAM             HAM
0013.1999-12-14.kaminski                 HAM             HAM
0013.2001-04-03.williams                 HAM             HAM
0013.2001-06-30.sa_and_hp                SPAM            HAM
0013.2004-08-01.bg                       SPAM            SPAM
0014.1999-12-14.kaminski                 HAM             HAM
0014.1999-12-15.farmer                   HAM             HAM
0014.2001-02-12.kitchen                  HAM             HAM
0014.2001-07-04.sa_and_hp                SPAM            HAM
0014.2003-12-19.gp                       SPAM            HAM
0014.2004-08-01.bg                       SPAM            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                  HAM             HAM
0015.2001-07-05.sa_and_hp                SPAM            HAM
0015.2003-12-19.gp                       SPAM            HAM
0016.1999-12-15.farmer                   HAM             HAM
0016.2001-02-12.kitchen                  HAM             HAM
0016.2001-07-05.sa_and_hp                SPAM            HAM
0016.2001-07-06.sa_and_hp                SPAM            HAM
0016.2003-12-19.gp                       SPAM            HAM
0016.2004-08-01.bg                       SPAM            HAM
0017.1999-12-14.kaminski                 HAM             HAM
0017.2000-01-17.beck                     HAM             HAM
0017.2001-04-03.williams                 HAM             HAM
0017.2003-12-18.gp                       SPAM            HAM
0017.2004-08-01.bg                       SPAM            HAM
0017.2004-08-02.bg                       SPAM            HAM
0018.1999-12-14.kaminski                 HAM             HAM
0018.2001-07-13.sa_and_hp                SPAM            SPAM
0018.2003-12-18.gp                       SPAM            SPAM
-----------------------
Accuracy: 60.0%
Total number of documents considered for classification:  100
Number of documents correctly classified:  60
```

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.

No smoothing is needed in this HW.

Mapper

In [78]:

```
%%writefile mapper14.py
#!/usr/bin/python
## mapper14.py
## Author: Prabhakar Gundugola
## Description: mapper code for HW1.4

import sys
import re
import string

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

with open (filename, "r") as myfile:
    for line in myfile:
        tokens = line.lower().split('\t')
        word_string = tokens[2] + ' ' + tokens[3].strip()
        word_string = word_string.translate(string.maketrans("",""), strin
g.punctuation)

        key = tokens[0] + '\t' + tokens[1] + '\t' + str(len(word_string.spl
it()))
        for word in findwords:
            key += '\t' + word + '\t' + str(word_string.count(word))
        print key
```

```
Overwriting mapper14.py
```

Reducer

In [96]:
```
%%writefile reducer14.py
#!/usr/bin/python
## reducer13.py
## Author: Prabhakar Gundugola
## Description: reducer code for HW1.4

import sys
import math

def isspam(true_class):
    if true_class == 1:
        return 'SPAM'
    else:
        return 'HAM'

filenames = sys.argv[1:]

spam_email_count = 0
ham_email_count = 0

spam_word_count = 0
ham_word_count = 0

spam_findword = {}
ham_findword = {}

total_cases = 0
correct_cases = 0

for filename in filenames:
    with open(filename, "r") as myfile:
        for line in myfile:
            tokens = line.split('\t')
            doc_id = tokens[0]
            true_class = int(tokens[1])
            #findword = tokens[3]
            #findword_count = int(tokens[4])
            word_count = int(tokens[2])

            if true_class == 1:
                spam_email_count += 1
                spam_word_count += word_count
            else:
                ham_email_count += 1
                ham_word_count += word_count

            if len(tokens) > 3:
                for i in range(3, len(tokens), 2):
                    findword = tokens[i]
                    findword_count = int(tokens[i+1])

                    if true_class == 1:
                        if findword not in spam_findword:
                            spam_findword[findword] = findword_count
                        else:
```

```
                    spam_findword[findword] += findword_count
                else:
                    if findword not in ham_findword:
                        ham_findword[findword] = findword_count
                    else:
                        ham_findword[findword] += findword_count

# Calculate the prior probabilities of both SPAM and HAM
spam_prior = math.log((1.0*spam_email_count)/(spam_email_count + ham_emai
l_count))
ham_prior = math.log((1.0*ham_email_count)/(ham_email_count + spam_email_co
unt))
spam_findword_prob = {}
ham_findword_prob = {}

for word in spam_findword:
    if spam_findword[word] > 0:
        spam_findword_prob[word] = math.log((1.0*spam_findword[word]/spam_w
ord_count))
    else:
        spam_findword_prob[word] = float('-inf')
for word in ham_findword:
    if ham_findword[word] > 0:
        ham_findword_prob[word] = math.log((1.0*ham_findword[word]/ham_wor
d_count))
    else:
        ham_findword_prob[word] = float('-inf')

# Naive Bayes classification
for filename in filenames:
    with open(filename, "r") as myfile:
        for line in myfile:
            total_cases += 1
            tokens = line.split('\t')
            doc_id = tokens[0]
            true_class = int(tokens[1])
            vocab = {}
            if len(tokens) > 3:
                for i in range(3, len(tokens), 2):
                    findword = tokens[i]
                    findword_count = int(tokens[i+1])
                    vocab[findword] = findword_count

            spam_doc_prob, ham_doc_prob = 0.0, 0.0
            for key, value in vocab.iteritems():
                if spam_findword_prob[key] == float('-inf'):
                    if value == 0:
                        spam_doc_prob += 0
                    else:
                        spam_doc_prob += float('-inf')
                else:
                    spam_doc_prob += spam_findword_prob[key]*value

            for key, value in vocab.iteritems():
                if ham_findword_prob[key] == float('-inf'):
                    if value == 0:
```

```
                        ham_doc_prob += 0
                    else:
                        ham_doc_prob += float('-inf')
                else:
                    ham_doc_prob += ham_findword_prob[key]*value

            spam_doc_prob += spam_prior
            ham_doc_prob += ham_prior

            result = doc_id.ljust(30) + '\t\t' + isspam(true_class) +
'\t\t'
            if spam_doc_prob > ham_doc_prob:
                predicted = 1
            else:
                predicted = 0
            result += isspam(predicted)
            print result

            if true_class == predicted:
                correct_cases += 1

accuracy = 100.0*correct_cases/total_cases
print "-----------------------"
print "Accuracy: " + str(accuracy) + '%'
print "Total number of documents considered for classification: ", total_ca
ses
print "Number of documents correctly classified: ", correct_cases
```

```
Overwriting reducer14.py
```

In [97]:

```
!chmod a+x mapper14.py
!chmod a+x reducer14.py

!./pNaiveBayes.sh 1 "assistance valium enlargementWithATypo" "mapper14" "re
ducer14"
!cat "enronemail_1h.txt.output"
```

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	HAM	HAM
0002.2001-05-25.sa_and_hp	SPAM	HAM
0002.2003-12-18.gp	SPAM	HAM
0002.2004-08-01.bg	SPAM	SPAM
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	HAM	HAM
0003.2003-12-18.gp	SPAM	HAM
0003.2004-08-01.bg	SPAM	HAM
0004.1999-12-10.kaminski	HAM	SPAM
0004.1999-12-14.farmer	HAM	HAM
0004.2001-04-02.williams	HAM	HAM
0004.2001-06-12.sa_and_hp	SPAM	HAM
0004.2004-08-01.bg	SPAM	HAM
0005.1999-12-12.kaminski	HAM	SPAM
0005.1999-12-14.farmer	HAM	HAM
0005.2000-06-06.lokay	HAM	HAM
0005.2001-02-08.kitchen	HAM	HAM
0005.2001-06-23.sa_and_hp	SPAM	HAM
0005.2003-12-18.gp	SPAM	HAM
0006.1999-12-13.kaminski	HAM	HAM
0006.2001-02-08.kitchen	HAM	HAM
0006.2001-04-03.williams	HAM	HAM
0006.2001-06-25.sa_and_hp	SPAM	HAM
0006.2003-12-18.gp	SPAM	HAM
0006.2004-08-01.bg	SPAM	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	HAM	HAM
0007.2003-12-18.gp	SPAM	HAM
0007.2004-08-01.bg	SPAM	HAM
0008.2001-02-09.kitchen	HAM	HAM
0008.2001-06-12.sa_and_hp	SPAM	HAM
0008.2001-06-25.sa_and_hp	SPAM	HAM
0008.2003-12-18.gp	SPAM	HAM
0008.2004-08-01.bg	SPAM	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	HAM	HAM
0009.2001-06-26.sa_and_hp	SPAM	HAM
0009.2003-12-18.gp	SPAM	SPAM
0010.1999-12-14.farmer	HAM	HAM
0010.1999-12-14.kaminski	HAM	HAM
0010.2001-02-09.kitchen	HAM	HAM
0010.2001-06-28.sa_and_hp	SPAM	SPAM

0010.2003-12-18.gp	SPAM	HAM
0010.2004-08-01.bg	SPAM	HAM
0011.1999-12-14.farmer	HAM	HAM
0011.2001-06-28.sa_and_hp	SPAM	SPAM
0011.2001-06-29.sa_and_hp	SPAM	HAM
0011.2003-12-18.gp	SPAM	HAM
0011.2004-08-01.bg	SPAM	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	HAM	HAM
0012.2003-12-19.gp	SPAM	HAM
0013.1999-12-14.farmer	HAM	HAM
0013.1999-12-14.kaminski	HAM	HAM
0013.2001-04-03.williams	HAM	HAM
0013.2001-06-30.sa_and_hp	SPAM	HAM
0013.2004-08-01.bg	SPAM	SPAM
0014.1999-12-14.kaminski	HAM	HAM
0014.1999-12-15.farmer	HAM	HAM
0014.2001-02-12.kitchen	HAM	HAM
0014.2001-07-04.sa_and_hp	SPAM	HAM
0014.2003-12-19.gp	SPAM	HAM
0014.2004-08-01.bg	SPAM	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	HAM	HAM
0015.2001-07-05.sa_and_hp	SPAM	HAM
0015.2003-12-19.gp	SPAM	HAM
0016.1999-12-15.farmer	HAM	HAM
0016.2001-02-12.kitchen	HAM	HAM
0016.2001-07-05.sa_and_hp	SPAM	HAM
0016.2001-07-06.sa_and_hp	SPAM	HAM
0016.2003-12-19.gp	SPAM	SPAM
0016.2004-08-01.bg	SPAM	HAM
0017.1999-12-14.kaminski	HAM	HAM
0017.2000-01-17.beck	HAM	HAM
0017.2001-04-03.williams	HAM	HAM
0017.2003-12-18.gp	SPAM	HAM
0017.2004-08-01.bg	SPAM	SPAM
0017.2004-08-02.bg	SPAM	HAM
0018.1999-12-14.kaminski	HAM	HAM
0018.2001-07-13.sa_and_hp	SPAM	SPAM
0018.2003-12-18.gp	SPAM	SPAM

Accuracy: 63.0%

Total number of documents considered for classification: 100

Number of documents correctly classified: 63