DATASCI W261: Machine Learning at Scale

- **Sayantan Satpati**
- **sayantan.satpati@ischool.berkeley.edu**

- **W261**
- **Week-1**
- **Assignment-2**
- **Date of Submission: 07-SEP-2015**

This notebook implements a Spam Filter backed by a Multinomial Naive Bayes Classifier

In [47]:
```
# This tells matplotlib not to try opening a new window for each plot.
%matplotlib inline

# Import a bunch of libraries.
import time
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.ticker import MultipleLocator
from sklearn.cross_validation import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.naive_bayes import BernoulliNB
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.grid_search import GridSearchCV
from sklearn.metrics import classification_report

# SK-learn libraries for feature extraction from text.

from sklearn.feature_extraction.text import *

# Set the randomizer seed so results are the same each time.
np.random.seed(0)
```

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 for data sets so large or complex that traditional data processing applications are inadequate, and cannot be processed or analyzed in a single computer. Challenges include analysis, capture, data curation, search, sharing, storage, transfer, visualization, and information privacy. The term often refers simply to the use of predictive analytics or other certain advanced methods to extract value from data, and seldom to a particular size of data set. Big Data is also characterized by the 4 V's: Volume, Velocity, Variety, and Veracity

HW1.0.1

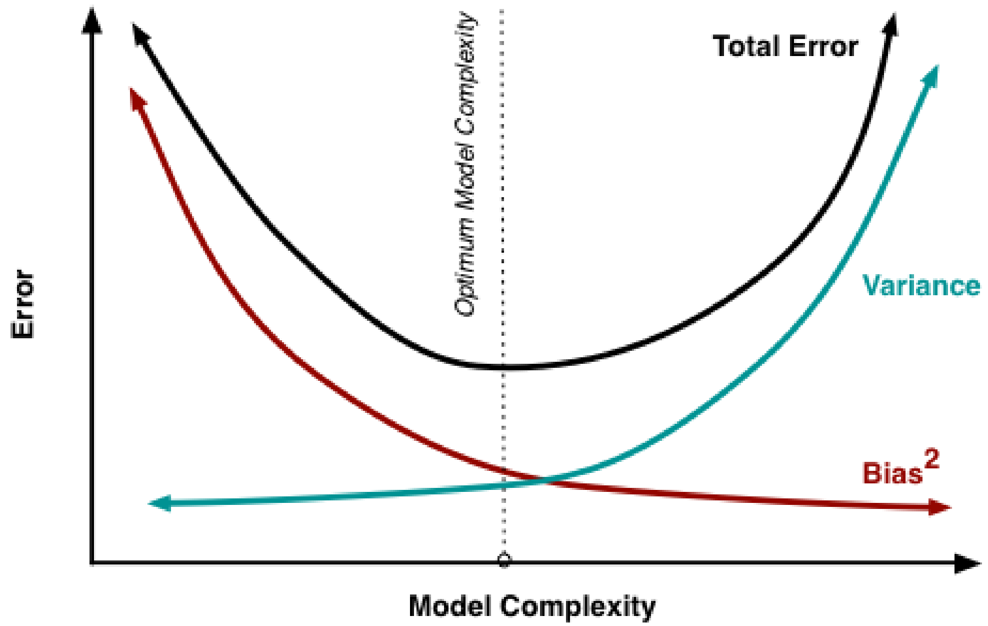
In 500 words (English or pseudo code or a combination) describe how to estimate the bias, the variance, the irreduciable error for a test dataset T when using polynomial regression models of degree 1, 2,3, 4,5 are considered. How would you select a model?

- Error due to Bias: The error due to bias is taken as the difference between the expected (or average) prediction of our model and the correct value which we are trying to predict. Of course you only have one model so talking about expected or average prediction values might seem a little strange. However, imagine you could repeat the whole model building process more than once: each time you gather new data and run a new analysis creating a new model. Due to randomness in the underlying data sets, the resulting models will have a range of predictions. Bias measures how far off in general these models' predictions are from the correct value.
- Error due to Variance: The error due to variance is taken as the variability of a model prediction for a given data point. Again, imagine you can repeat the entire model building process multiple times. The variance is how much the predictions for a given point vary between different realizations of the model.
- If we denote the variable we are trying to predict as Y and our covariates as X, we may assume that there is a relationship relating one to the other such as $Y=f(X)+\epsilon$ where the error term ϵ is normally distributed with a mean of zero.

  Err(x) = Bias^2 + Variance + Irreducible Error

  That third term, irreducible error, is the noise term in the true relationship that cannot fundamentally be reduced by any model. Given the true model and infinite data to calibrate it, we should be able to reduce both the bias and variance terms to 0. However, in a world with imperfect models and finite data, there is a tradeoff between minimizing the bias and minimizing the variance.
- Bias and variance can be visualized with a classic example of a dartboard. We have four different dart throwers, each with different combinations of low/high bias and low/high variance. We represent the locations of each of their dart throws as blue dots:

At its root, dealing with bias and variance is really about dealing with over- and under-fitting. Bias is reduced and variance is increased in relation to model complexity. As more and more parameters are added to a model, the complexity of the model rises and variance becomes our primary concern while bias steadily falls. For example, as more polynomial terms are added to a linear regression, the greater the resulting model's complexity will be. In other words, bias has a negative first-order derivative in response to model complexity while variance has a positive slope.



Map

In [48]:

```
%%writefile mapper_HW12.py
#!/usr/bin/python
import sys
import re

def strip_special_chars(word):
    return re.sub('[^A-Za-z0-9]+', '', word)

count = 0
filename = sys.argv[1]
wordList = sys.argv[2]
wordList = wordList.split()
wordCountDict = {}
with open (filename, "r") as myfile:
    for line in myfile:
        # Split the line by <TAB> delimeter
```

```
        email = re.split(r'\t+', line)

        # Check whether Content is present
        if len(email) < 4:
            continue

        # Get the content as a list of words
        content = email[len(email) - 1].split()

        if len(wordList) == 1 and wordList[0] == '*':
            for w in content:
                w = strip_special_chars(w)
                if w not in wordCountDict:
                    wordCountDict[w] = 1
                else:
                    wordCountDict[w] += 1
        else:
            for w in content:
                w = strip_special_chars(w)
                # Check if word is in word list passed to mapper
                if w in wordList:
                    if w not in wordCountDict:
                        wordCountDict[w] = 1
                    else:
                        wordCountDict[w] += 1

# Print count from each mapper
for k,v in wordCountDict.items():
    print "{0}\t{1}".format(k,v)
```

```
Overwriting mapper_HW12.py
```

In [49]:

```
!chmod a+x mapper_HW12.py
```

Reduce

In [50]:

```
%%writefile reducer_HW12.py

#!/usr/bin/python
import sys
import re
cnt = 0
wordCountDict = {}
for file in sys.argv:
    if cnt == 0:
        cnt += 1
        continue

    with open (file, "r") as myfile:
        for line in myfile:
            wc = re.split(r'\t+', line.strip())
            if wc[0] not in wordCountDict:
                wordCountDict[wc[0]] = int(wc[1])
            else:
                wordCountDict[wc[0]] += int(wc[1])

# Print count from each mapper
for k,v in wordCountDict.items():
    print "{0}\t{1}".format(k,v)
```

```
Overwriting reducer_HW12.py
```

In [51]:

```
!chmod a+x reducer_HW12.py
```

In [52]:

```
# Remove split files from last runs
! rm License.txt.*
```

```
rm: License.txt.*: No such file or directory
```

Write control script 'pNaiveBayes.sh' to a file

In [53]:

```
%%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

## Mapper and Reducer Files are passed to make this script generic
mapper=$3
reducer=$4

## 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 dis
k
    ####
    ####
    ./${mapper} $datachunk "$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} $countfiles > $data.output
####
####

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

## Display the Output
cat $data.output
```

Overwriting pNaiveBayes.sh

In [54]:

```
'''
HW1.1. Read through the provided control script (pNaiveBayes.sh)
'''
print "done"
```

done

Run the file

In [55]:
```
!chmod a+x pNaiveBayes.sh
```

Usage: usage: pGrepCount filename word chuncksize

In [56]:
```
# Test the Program
!./pNaiveBayes.sh 4 'the and of' 'mapper_HW12.py' 'reducer_HW12.py'
```

```
and     631
of      546
the     1217
```

In [57]:
```
'''
HW1.2. Provide a mapper/reducer pair that, when executed by pNaiveBayes.sh
'''
!./pNaiveBayes.sh 4 'assistance' 'mapper_HW12.py' 'reducer_HW12.py'
```

```
assistance      9
```

In [58]:
```
%%writefile mapper_HW15.py
#!/usr/bin/python
import sys
import re

def strip_special_chars(word):
    word = word.strip()

    if not word or word == '':
        return None

    word = re.sub('[^A-Za-z0-9]+', '', word)
    return word.lower()

count = 0
```

```
filename = sys.argv[1]
wordList = sys.argv[2]
wordList = wordList.split()

# (Line#, Spam/Ham, Dict of Word|Count)
mapper_output_list = []
line_num = 0
with open (filename, "r") as myfile:
    for line in myfile:
        # Split the line by <TAB> delimeter
        email = re.split(r'\t+', line)

        # Check whether Content is present
        if len(email) < 4:
            continue

        line_num += 1

        # Get the content as a list of words
        content = email[len(email) - 1].split()

        wordCountDict = {}
        for w in content:
            w = strip_special_chars(w)

            if not w:
                continue

            wordCountDict[w] = wordCountDict.get(w, 0) + 1

        mapper_output_list.append((line_num, email[1], wordCountDict))

# Print output from each mapper
for (line_num, spam, wordCountDict) in mapper_output_list:
    for word,count in wordCountDict.items():
        print "{0}\t{1}\t{2}\t{3}".format(line_num, spam, word, count)
```

```
Overwriting mapper_HW15.py
```

In [59]:
```
!chmod a+x mapper_HW15.py
```

In [60]:
```
%%writefile reducer_HW15.py
#!/usr/bin/python
import sys
import re
import math

# Totals
vocab = 0
vocab_spam = 0
vocab_ham = 0

vocab = {}
word_counts = {
    "1": {},
    "0": {}
}

num_spam = 0
num_ham = 0

cnt = 0
# Calculate the totals in Reducer First Pass
for file in sys.argv[1:]:

    with open (file, "r") as myfile:
        print '[REDUCER] Processing File: {0}'.format(file)
        last_line_num = -1
        last_spam = -1

        for line in myfile:
            tokens = re.split(r'\t+', line.strip())
            line_num = int(tokens[0])
            spam = int(tokens[1])
            word = tokens[2]
            count = float(tokens[3])

            # Init
```

```
            if last_line_num == -1:
                last_line_num = line_num
                last_spam = spam

            # Add Vocab per line
            vocab[word] = vocab.get(word, 0.0) + count
            word_counts[str(spam)][word] = word_counts[str(spam)].get(word, 0.0) + count

            if last_line_num != line_num:
                if last_spam == 1:
                    num_spam += 1
                else:
                    num_ham += 1

            last_line_num = line_num
            last_spam = spam

        # Last Line
        if last_spam == 1:
            num_spam += 1
        else:
            num_ham += 1

# At the end of first pass
print 'Num Spam: {0}, Num Ham: {1}'.format(num_spam, num_ham)
print '''Total Vocab: {0},
       Total Unique Vocab: {1},
       Total Spam Vocab: {2},
       Total Ham Vocab: {3}'''.format(sum(vocab.values()),
                                     len(vocab),
                                     sum(word_counts['1'].values()),
                                     sum(word_counts['0'].values())
                                    )


prior_spam = (num_spam * 1.0) / (num_spam + num_ham)
prior_ham = (num_ham * 1.0) / (num_spam + num_ham)
print '[Priors] Spam: {0}, Ham: {1}'.format(prior_spam, prior_ham)

spam_likelihood_denom = sum(word_counts['1'].values()) + len(vocab)
```

```
ham_likelihood_denom = sum(word_counts['0'].values()) + len(vocab)

# Calculate the Conditionals/Likelihood in Next Pass
reducer_output_list = []
for file in sys.argv[1:]:

    with open (file, "r") as myfile:
        print '[REDUCER] Processing File: {0}'.format(file)
        last_line_num = None
        last_spam = None
        log_prob_spam = 0
        log_prob_ham = 0

        for line in myfile:
            tokens = re.split(r'\t+', line.strip())
            line_num = int(tokens[0])
            spam = int(tokens[1])
            word = tokens[2]
            count = int(tokens[3])

            if last_line_num != line_num:
                # Calculate the Naive Bayes Scores for Document Classification
                spam_score = log_prob_spam + math.log(prior_spam)
                ham_score = log_prob_ham + math.log(prior_ham)
                reducer_output_list.append((last_spam, spam_score, ham_score))
                # Reset log prob
                log_prob_spam = 0
                log_prob_ham = 0
            else:
                # Calcuate the log likelihoods Using Laplace Smoothing
                spam_likelihood = (word_counts['1'].get(word, 0.0) + 1) / spam_likelihood_deno
m
                ham_likelihood = (word_counts['0'].get(word, 0.0) + 1) / ham_likelihood_denom
                log_prob_spam += math.log( spam_likelihood )
                log_prob_ham += math.log( ham_likelihood )

            # For Debug
            #print '[{0}][{1}][{2}][{3}][{4}]'.format(file, last_line_num, line_num, log_pro
b_spam, log_prob_ham)
            last_line_num = line_num
```

```
            last_spam = spam

        # Last Line
        spam_score = log_prob_spam + math.log(prior_spam)
        ham_score = log_prob_ham + math.log(prior_ham)
        reducer_output_list.append((spam, spam_score, ham_score))

total = 0.0
miscat = 0.0
for (spam, spam_score, ham_score) in reducer_output_list:
        total += 1.0
        pred_class = 'HAM'
        if spam_score > ham_score:
            pred_class = 'SPAM'
        if (spam == 1 and pred_class == 'HAM') or (spam == 0 and pred_class == 'SPAM'):
            miscat += 1.0

        print "{0}\t{1}\t{2}\t{3}".format(spam, spam_score, ham_score, pred_class)

error = miscat * 100 / total
print "Accuracy: {0}, Error Rate: {1}, # of Miscats: {2}".format((100 - error), error, miscat)
```

```
Overwriting reducer_HW15.py
```

In [61]:
```
!chmod a+x reducer_HW15.py
```

In [62]:
```
'''
HW1.5. Provide a mapper/reducer pair that, when executed by pNaiveBayes.sh
   will classify the email messages by all words present.
'''
!./pNaiveBayes.sh 4 '*' 'mapper_HW15.py' 'reducer_HW15.py'
```

```
[REDUCER] Processing File: enronemail_1h.txt.chunk.aa.counts
[REDUCER] Processing File: enronemail_1h.txt.chunk.ab.counts
[REDUCER] Processing File: enronemail_1h.txt.chunk.ac.counts
[REDUCER] Processing File: enronemail_1h.txt.chunk.ad.counts
Num Spam: 43, Num Ham: 55
Total Vocab: 30316.0,
       Total Unique Vocab: 5601,
       Total Spam Vocab: 17851.0,
```

```
       Total Ham Vocab: 12465.0
[Priors] Spam: 0.438775510204, Ham: 0.561224489796
[REDUCER] Processing File: enronemail_1h.txt.chunk.aa.counts
[REDUCER] Processing File: enronemail_1h.txt.chunk.ab.counts
[REDUCER] Processing File: enronemail_1h.txt.chunk.ac.counts
[REDUCER] Processing File: enronemail_1h.txt.chunk.ad.counts
None    -0.823767362977 -0.577634293438 HAM
0       -0.823767362977 -0.577634293438 HAM
0       -9.27704050863  -8.58766182192  HAM
0       -2191.11883116  -2021.75440076  HAM
0       -281.802505573  -248.66078934   HAM
0       -916.401633569  -878.181677749  HAM
0       -1653.44319311  -1492.5384975   HAM
0       -395.284181305  -381.866477371  HAM
1       -469.286123972  -514.989803183  SPAM
1       -929.72912698   -1010.91965204  SPAM
1       -580.613850923  -614.383744167  SPAM
0       -308.905756003  -276.061748895  HAM
0       -47.1860789348  -40.2347293477  HAM
0       -950.826286141  -848.81258232   HAM
0       -1072.44352797  -960.329471011  HAM
1       -606.705200099  -633.092993607  SPAM
1       -640.276393173  -676.886377971  SPAM
0       -899.069072412  -813.462660776  HAM
0       -639.436366799  -553.707345035  HAM
0       -486.370815481  -462.444432652  HAM
1       -611.888760379  -658.61975736   SPAM
1       -635.701101835  -666.817949033  SPAM
0       -696.395667351  -629.032419986  HAM
0       -681.79113014   -598.800156466  HAM
0       -369.205477768  -351.507072033  HAM
0       -718.331232072  -652.19066713   HAM
None    -0.823767362977 -0.577634293438 HAM
1       -151.835929783  -160.574088089  SPAM
1       -4309.44846148  -4554.29592822  SPAM
0       -449.539299634  -428.577486601  HAM
0       -4939.06987487  -4573.81524366  HAM
0       -244.865376047  -235.614322214  HAM
1       -311.595394813  -325.941602481  SPAM
1       -831.624836729  -873.780864125  SPAM
```

```
1       -879.235293936  -925.718780195  SPAM
0       -975.250606156  -890.011137371  HAM
0       -504.875426808  -460.29909355   HAM
0       -1898.89679539  -1639.95383108  HAM
0       -1343.30660425  -1220.90014121  HAM
1       -1015.05425275  -1076.07499848  SPAM
1       -770.586716813  -854.775415682  SPAM
0       -2819.20353407  -2554.88871546  HAM
1       -611.888760379  -658.61975736   SPAM
1       -2762.71629433  -3018.46650805  SPAM
1       -720.056436381  -773.525254921  SPAM
1       -3625.33852065  -3812.92515098  SPAM
0       -2212.31442671  -1998.85951051  HAM
0       -345.524100995  -307.123252881  HAM
0       -1926.38342139  -1743.93931076  HAM
0       -3495.9509362   -3146.46329128  HAM
1       -706.085158831  -767.949995342  SPAM
0       -1087.75215399  -962.910070499  HAM
None    -0.823767362977 -0.577634293438 HAM
0       -210.118679734  -187.817228995  HAM
0       -2231.79858251  -1968.3991191   HAM
1       -2137.46598027  -2353.98492953  SPAM
1       -0.823767362977 -0.577634293438 HAM
1       -1395.4197152   -1503.42207516  SPAM
0       -1024.52019829  -952.91587697   HAM
1       -2137.46598027  -2353.98492953  SPAM
1       -6305.60823327  -6829.88011447  SPAM
1       -331.23445788   -356.986489157  SPAM
1       -595.808380241  -635.275255921  SPAM
0       -2096.8884822   -1909.42383997  HAM
0       -583.481548778  -504.150807478  HAM
0       -1877.44961744  -1620.72063087  HAM
0       -683.338107553  -667.292064988  HAM
0       -446.24540735   -414.241507702  HAM
1       -137.14845291   -148.730891811  SPAM
0       -1266.01454826  -1156.42629463  HAM
0       -664.504816859  -577.960068001  HAM
0       -555.948879204  -533.18533743   HAM
1       -9632.30557826  -10393.7781108  SPAM
1       -1152.8268964   -1213.93911154  SPAM
```

```
0       -898.357407207  -816.54883022   HAM
0       -885.61852061   -784.575568784  HAM
0       -1005.27950676  -907.813522375  HAM
1       -1942.1471285   -2080.21140107  SPAM
1       -51.9329350057  -56.2973138263  SPAM
None    -0.823767362977 -0.577634293438 HAM
1       -711.531383123  -746.129028498  SPAM
0       -520.389190282  -477.143503463  HAM
0       -575.927365971  -496.00906614   HAM
0       -105.947550832  -100.847088011  HAM
0       -2678.71129692  -2458.50017481  HAM
1       -693.831992184  -750.107992654  SPAM
1       -827.915052319  -893.677368771  SPAM
0       -590.183866817  -555.989365161  HAM
0       -856.796470852  -760.585553615  HAM
1       -693.831992184  -750.107992654  SPAM
1       -5927.70644169  -6536.54977058  SPAM
1       -687.86927922   -746.725933597  SPAM
1       -470.534905235  -512.398365589  SPAM
0       -333.222151051  -322.840040697  HAM
0       -1887.51232849  -1630.92788297  HAM
0       -344.169934943  -333.542766902  HAM
1       -154.298635668  -164.448798508  SPAM
1       -831.998475902  -873.032087758  SPAM
1       -1760.71166611  -1849.55860198  SPAM
0       -712.460647129  -698.127308355  HAM
1       -1781.02548675  -1910.68924069  SPAM
1       -2039.15695348  -2150.88056932  SPAM
Accuracy: 99.0196078431, Error Rate: 0.980392156863, # of Miscats: 1.0
```

In [63]:

```
# Load Data into Pandas Dataframe
df = pd.read_csv('enronemail_1h.txt', sep='\t', header=None)
df.columns = ['ID', 'SPAM', 'SUBJECT', 'CONTENT']
df.head()
```

Out[63]:

	ID	SPAM	SUBJECT	CONTENT
0	0001.1999-12-10.farmer	0	christmas tree farm pictures	NaN
1	0001.1999-12-10.kaminski	0	re: rankings	thank you.
2	0001.2000-01-17.beck	0	leadership development pilot	sally: what timing, ask and you shall receiv...
3	0001.2000-06-06.lokay	0	key dates and impact of upcoming sap implemen...	NaN
4	0001.2001-02-07.kitchen	0	key hr issues going forward	a) year end reviews-report needs generating l...

In [64]:

```
# Remove missing values
print df.count()
df = df.dropna()
print df.count()
```

```
ID          100
SPAM        100
SUBJECT      98
CONTENT      96
dtype: int64
ID          94
SPAM        94
SUBJECT     94
CONTENT     94
dtype: int64
```

In [65]:
```
data = df['CONTENT'].values

labels = df['SPAM'].values
print data[:1], labels[:1]
# Split into Train and Test
train_data, test_data, train_labels, test_labels = train_test_split(data, labels, train_size =
0.8)
print train_data.shape, train_labels.shape
print test_data.shape, test_labels.shape
```

```
[' thank you.'] [0]
(75,) (75,)
(19,) (19,)
```

In [66]:
```
# Extract features from Dataset
cv = CountVectorizer(analyzer='word')
train_counts = cv.fit_transform(data)
print "Shape of training/feature vector", train_counts.shape
print "Size of the Vocabulary", len(cv.vocabulary_)

# Run Multinomial NB (sklearn)
mNB = MultinomialNB()
mNB.fit(train_counts, labels)
print "Multinomial NB Training Accuracy: {0}".format(mNB.score(train_counts, labels))

#Run Bernoulli MB (sklearn)
bNB = BernoulliNB()
bNB.fit(train_counts, labels)
print "Bernoulli NB Training Accuracy: {0}".format(bNB.score(train_counts, labels))
```

```
Shape of training/feature vector (94, 5224)
Size of the Vocabulary 5224
Multinomial NB Training Accuracy: 0.989361702128
Bernoulli NB Training Accuracy: 0.765957446809
```

HW1.6

- Training Accuracy from Multinomial NB: 0.989361702128
- Training Accuray from Bernoulli NB: 0.765957446809
- Training Accuracy from Multinomial NB from HW1.5: 66.3265306122

#	Naive Bayes Algorithm	Accuracy
1	Multinomial NB	0.989
1	Bernoulli NB	0.766
1	Multinomial NB HW1.5	0.990

Explain/justify any differences in terms of training error rates over the dataset in HW1.5 between your Multinomial Naive Bayes implementation (in Map Reduce) versus the Multinomial Naive Bayes implementation in SciKit-Learn

- There is very less difference between the Multinomial Approaches taken in HW1.5 & sklearn. How we pre-processed the data might have caused the small difference. In HW1.5, I have tokenized the data before removing special chars in words. For Ex: 'assistance,', 'assistance:' have become 'assistance'. Also, I have converted everything into lower case, which sklearn seems to be doing as well.

Discuss the performance differences in terms of training error rates over the dataset in HW1.5 between the Multinomial Naive Bayes implementation in SciKit-Learn with the Bernoulli Naive Bayes implementation in SciKit-Learn

- Empirical comparisons provide evidence that the multinomial model tends to outperform the multi-variate Bernoulli model if the vocabulary size is relatively large. And we observe the same behavior from our results as well.

```
In [ ]:
```