Data Science W261: Machine Learning at Scale

Safyre Anderson

safyre@berkeley.edu

January 21, 2016, 8am

W261-3

Week 1 HW

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

Big data refers to the problem of storage and analysis of datasets that overwhelm a traditional single computer system, and typically characterized by the "3 V's": volume, velocity, and veracity. The extent of either one of these 3 V's (the size of the dataset, the speed of ingestion, and the variety of features) prevents Big Data from being analyzed on the limited hardware of a personal computer or laptop. Currently, I work in FinTech and one of the most important Big Data problems that exists is the near real-time detection of fraudulent transactions. There are over $1 billion people online that transact between 400 billion to 1 billion times per day (depending on what article you read), with variability in customers, transfers, purchases, payment methods, and many more kinds of recorded data--easily fulfilling the 3 V's.

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

One of the recommended readings provided an excellent framework for this problem: https://theclevermachine.wordpress.com/2013/04/21/model-selection-underfitting-overfitting-and-the-bias-variance-tradeoff/ (https://theclevermachine.wordpress.com/2013/04/21/model-selection-underfitting-overfitting-and-the-bias-variance-tradeoff/).

Essentially, total error of a model is the sum of the squared error of bias, variance, and residual squared (noise). Thus on a high level, we will want to to determine the bias, variance, and total error of each model we test. From these three parameters, we can estimate the noise of each estimated true model by subtracting the mean squared error from the total test error:

Expected Prediction Error $= E[(g(x^*) - E[g(x^*)]^2] + (E[g(x^*)] - f(x^*))^2 + E[(y^* - f(x))^2] = E[(g(x^*) - y^*)^2]$

In our case, $f(x)$ is unknown. However, we are given y^* and we can bootstrap training and test data to get $g(x^*)$ and E[g(x^*)]$ over all models (of the same degree).

In general:

Given a dataset:

```
for each polynomial of degree 1 through 5:

   for each bootstrapped sample of the dataset:
       subset $x%$ to training
       subset $1-x%$ to testing, where x is a fraction of the whole dataset

       fit a model on the training data
       predict $g(x^*)$ on the testing data

       calculate estimated predicted error (mean squared difference of observed $y^*_{test}$ from predicted y ($g(x^*)
   $).
       calculate variance (mean of the variance of $g(x^*)$ across all datasets)
```

Here, we should already by able to select the best model by choosing the parameter(s) that achieved the lowest estimated predicted error. What about bias and noise? We still haven't figured those out yet.

Since noise is independent of the model, it is a constant term in the above equation. Thus, we can rethink this problem as such: $Expected\ Error - Variance = Bias + Constant$. To estimate the noise, we can take advantage of the bias variance tradeoff-- with the highest complexity model, we can assume bias approaches 0. Thus the equation becomes $Expected\ Error - Variance = Constant$. At polynomial N = 5, we can estimate the noise and finally solve for the bias: $Bias = Expected\ Error - Variance - Constant$.

Part 2

In the remainder of this assignment you will produce a spam filter that is backed by a multinomial naive Bayes classifier b (see http://nlp.stanford.edu/IR-book/html/htmledition/properties-of-naive-bayes-1.html (http://nlp.stanford.edu/IR-book/html/htmledition/properties-of-naive-bayes-1.html)), which counts words in parallel via a unix, poor-man's map-reduce framework.

For the sake of this assignment we will focus on the basic construction of the parallelized classifier, and not consider its validation or calibration, and so you will have the classifier operate on its own training data (unlike a field application where one would use non-overlapping subsets for training, validation and testing).

The data you will use is a curated subset of the Enron email corpus (whose details you may find in the file enronemail_README.txt in the directory surrounding these instructions).

=====Instructions/Goals=====

In this directory you will also find starter code (pNaiveBayes.sh), (similar to the pGrepCount.sh code that was presented in this weeks lectures), which will be used as control script to a python mapper and reducer that you will supply at several stages. Doing some exploratory data analysis you will see (with this very small dataset) the following\:

```
> wc -l enronemail_1h.txt #100 email records

     100 enronemail_1h.txt


> cut -f2 -d$'\t' enronemail_1h.txt|wc #extract second field which is SPAM flag

     101     394    3999


JAMES-SHANAHANs-Desktop-Pro-2:HW1-Questions jshanahan\$ cut -f2 -d$'\t' enronemail_1h.txt|head
```

0

0

0

0

0

0

0

0

1

1

```
> head -n 100 enronemail_1h.txt|tail -1|less #an example SPAM email record

018.2001-07-13.SA_and_HP 1

[ilug] we need your assistance to invest in your country dear sir/madam, i am well confident of your capability to assist
me in a transaction for mutual benefit of both parties, ie (me and you) i am also believing that you will not expose or
betray the trust and confidence i am about to establish with you. i have decided to contact you with greatest delight and
personal respect. well, i am victor sankoh, son to mr. foday sankoh who was arrested by the ecomog peace keeping force
months ago in my country sierra leone.
```

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 [1]:
```
# just to run it to show i've read it. I know there aren't mappers and reducers yet!
!chmod +x pNaiveBayes.sh
!./pNaiveBayes.sh
```

```
(standard_in) 1: parse error
split: enronemail_1h.txt: illegal line count
Traceback (most recent call last):
  File "./reducer.py", line 27, in <module>
    spam_total += int(spam_count)
NameError: name 'spam_count' is not defined
```

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.

CROSSCHECK: >grep assistance enronemail_1h.txt|cut -d$'\t' -f4| grep assistance|wc -l 8

In [27]:
```
# download into current directory
!wget https://www.dropbox.com/sh/jylzkmauxkostck/AAC_6JZH7yqMcxfEGPc4-_xJa/enronemail_1h.txt?dl=0 -O enronemail_1h.txt

# running wc -l enronemail_1h.txt outputs linecount = 0
# due to \r line break, need to change to \n line break (mac, unix, respectively)
!perl -pi -e 's/\r/\n/g' enronemail_1h.txt

!wc -l enronemail_1h.txt
```

```
--2016-01-22 07:36:56--  https://www.dropbox.com/sh/jylzkmauxkostck/AAC_6JZH7yqMcxfEGPc4-_xJa/enronemail_1h.txt?dl
=0
Resolving www.dropbox.com... 108.160.172.238, 108.160.172.206
Connecting to www.dropbox.com|108.160.172.238|:443... connected.
HTTP request sent, awaiting response... 302 FOUND
Location: https://dl.dropboxusercontent.com/content_link/UPXdQChI66QKyyBxhl07rNOngofwByuL0wSayFJjt1CrDsxj34imSfJI6
bJFxMTZ/file [following]
--2016-01-22 07:36:59--  https://dl.dropboxusercontent.com/content_link/UPXdQChI66QKyyBxhl07rNOngofwByuL0wSayFJjt1
CrDsxj34imSfJI6bJFxMTZ/file
Resolving dl.dropboxusercontent.com... 199.47.217.5
Connecting to dl.dropboxusercontent.com|199.47.217.5|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 204579 (200K) [text/plain]
Saving to: 'enronemail_1h.txt'

enronemail_1h.txt   100%[=====================>] 199.78K   399KB/s   in 0.5s

2016-01-22 07:37:01 (399 KB/s) - 'enronemail_1h.txt' saved [204579/204579]

      99 enronemail_1h.txt
```

In [40]:
```
%%writefile mapper.py
#!/usr/bin/python
import re
import sys
import os
import numpy as np

# store a regex expression into a pattern object
# that seeks words including underscores and single quotes
WORD_RE = re.compile(r"[\w']+")

# file input
filename = sys.argv[1]

# for this part, just assume word_list is length 1
word_list = sys.argv[2]
count = 0

with open(filename, 'rU') as f:
    for line in f.readlines():
        for word in word_list.split():
            counts = [int(1) if x == word else int(0) for x in WORD_RE.findall(line)]
            counts = np.array(counts)

            if counts.sum() > 0:
                print word + " " + str(counts.sum())
```

```
Overwriting mapper.py
```

Making the mapper.py an executable:

In [41]:
```
!chmod a+x mapper.py
```

In [48]:
```
%%writefile reducer.py
#!/usr/bin/python

import sys

files_list = sys.argv[1].split()

cur_word = ""

word_counts = {}

for f in files_list:
    with open(f, 'rU') as countfile:
        for line in countfile.readlines():
            new_word, count = line.split()
            if new_word != cur_word:
                # print the final count of the current word and start over
                cur_word = new_word
                word_counts[cur_word] = word_counts.get(cur_word, 0) + int(count)

            else:
                word_counts[new_word] += int(count)

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

```
Overwriting reducer.py
```

In [49]:
```
!chmod a+x reducer.py
```

In [62]:
```
%%writefile pNaiveBayes.sh
#!/bin/bash
## 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/ /'`

## 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 [63]:
```
!chmod a+x pNaiveBayes.sh
```

Run for 1.2, first argument is number of mappers, second argument is a list of words. File list was printed only for debugging.

In [52]:
```
!./pNaiveBayes.sh 4 assistance
!head enronemail_1h.txt.output
```

```
assistance      10
```

Confirm answers with bash commands (note there are actually 10 instances of assistance, but only 8 lines):

In [14]:
```
!grep assistance enronemail_1h.txt|cut -d$'\t' -f4| grep assistance|wc -l
```

```
       8
```

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 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.

In [64]:
```
%%writefile mapper.py
#!/usr/bin/python
import re
import sys
import os
import numpy as np

# store a regex expression into a pattern object
# that seeks words including underscores and single quotes
WORD_RE = re.compile(r"[\w']+")
TRUTH_RE = re.compile(r"\t(\d)\t")

# file input
filename = sys.argv[1]

# for this part, just assume word_list is length 1
word_list = sys.argv[2]

# Avoid KeyError if no data in chunk
#counts_dict = dict.fromkeys(['0', '1'], 0)
counts_dict = {}
doc_len    = 0
spam_count = 0
ham_count  = 0

with open(filename, 'rU') as f:
    for line in f.readlines():
        # Remove punctuation
        line = re.sub(r'[^\w\s]','',line)

        # truth is the actual label provided in the data
        # 1 = spam, 0 = ham
        truth = TRUTH_RE.findall(line)[0]

        '''
        if truth == '1':
            spam_count += 1
        else:
            ham_count += 1
        '''
        for category in ['0','1']:
            counts_dict[category] = {}

        for word in word_list.split():
            doc_len = len(word_list.split())
            counts = [1 if x == word else 0 for x in WORD_RE.findall(line)]
            counts = np.array(counts)

            if counts.sum() > 0:
                count = counts.sum()
                counts_dict[truth][word] = counts_dict[truth].get(word, 0) + int(count)
            else:
                counts_dict[truth][word] = 0


for category, word_dictionary in counts_dict.iteritems():
    for words, count in counts_dict[category].iteritems():
        print category + "\t" + words + "\t" + str(count) + "\t" + str(doc_len)
```

```
Overwriting mapper.py
```

In [65]:
```
!chmod a+x mapper.py
```

In [69]:
```
%%writefile reducer.py
#!/usr/bin/python

import sys
import numpy as np
import re
from math import log

WORD_RE = re.compile(r"[\w']+")
TRUTH_RE = re.compile(r"\t(\d)\t")
files_list = sys.argv[1].split()

## training, gather all the counts and calculate corpus-wide priors, etc
## data come in as strings,
## TRUTH WORD COUNT SPAM_COUNT HAM_COUNT
counts_dict = {}
for category in ['0','1']:
    counts_dict[category] = {}

spam_total = 0
ham_total  = 0

for f in files_list:
    with open(f, 'rU') as countfile:
        for line in countfile.readlines():
            truth, word, count, doc_len = line.split()
            counts_dict[truth][word] = counts_dict[truth].get(word, 0) + int(count)
            if truth == '1':
                spam_total += int(doc_len)
            else:
                ham_total  += int(doc_len)

priors = {'0': float(ham_total)/(spam_total+ham_total),
          '1': float(spam_total)/(spam_total+ham_total)}

prior_counts = {'0': float(ham_total),
          '1': float(spam_total)}

print "Priors are: "
for category in priors:
    print category + " " + str(priors[category]) + "\n"

spam_vocab = counts_dict['1'].keys()
ham_vocab  = counts_dict['0'].keys()

spam_vocab_n = len(counts_dict['1'].keys())
ham_vocab_n  = len(counts_dict['0'].keys())

## although we are not implementing Laplace Transform
# probably don't need these next two lines
#vocab = union(spam_vocab, ham_vocab)
#vocab_n = len(vocab)

## Calculate conditional probabilities
## P(word | class)
posteriors = {}
for category in ['0', '1']:
    posteriors[category] = {}
    for word in counts_dict[category].keys():
        posteriors[category][word] = float(counts_dict[category][word])/prior_counts[category]

print "\nPosteriors are: "
for category in posteriors:
    for word in posteriors[category]:
        print word + " in class " + category + " " + str(posteriors[category][word]) + "\n"

## Testing the classifer
## Without laplacian transform
print "DOC_ID | TRUTH | CLASS "
print "======================\n"

doc_id = 0
correct = 0
with open("enronemail_1h.txt", 'rU') as testdata:
    for line in testdata.readlines():
        score = [0,0]
        line = re.sub(r'[^\w\s]','',line)
        truth = TRUTH_RE.findall(line)[0]

        for category in ['0', '1']:
            idx = int(category)
            score[idx] =  priors[category]
            for word in posteriors[category]:
                if word in WORD_RE.findall(line):
                    score[idx] *= float(posteriors[category][word])
                #print "\n", idx, score[idx], category, word
        score = np.array(score)
        prediction = score.argmax()
        doc_id +=1

        if int(prediction) == int(truth):
            correct +=1
        print str(doc_id) + "\t" + truth + "\t" +str(prediction) +  " " +str(score[0]) + " " + str(score[1])

accuracy = float(correct)/doc_id*100.0
print "Accuracy: ", accuracy
```

```
Overwriting reducer.py
```

In [70]:
```
!chmod a+x reducer.py
```

In [71]:
```
!./pNaiveBayes.sh 4 assistance
!cat enronemail_1h.txt.output
```

```
Priors are:
1 0.5

0 0.5


Posteriors are:
assistance in class 1 0.5

assistance in class 0 0.0

DOC_ID | TRUTH | CLASS
=======================

1       0       0 0.5 0.5
2       0       0 0.5 0.5
3       0       0 0.5 0.5
4       0       0 0.5 0.5
5       0       0 0.5 0.5
6       0       0 0.5 0.5
7       0       0 0.5 0.5
8       0       0 0.5 0.5
9       1       0 0.5 0.5
10      1       0 0.5 0.5
11      1       1 0.0 0.25
12      0       0 0.5 0.5
13      0       0 0.5 0.5
14      0       0 0.5 0.5
15      0       0 0.5 0.5
16      1       0 0.5 0.5
17      1       0 0.5 0.5
18      0       1 0.0 0.25
19      0       0 0.5 0.5
20      0       0 0.5 0.5
21      1       0 0.5 0.5
22      1       0 0.5 0.5
23      0       1 0.0 0.25
24      0       0 0.5 0.5
25      0       0 0.5 0.5
26      0       0 0.5 0.5
27      1       0 0.5 0.5
28      1       0 0.5 0.5
29      0       0 0.5 0.5
30      0       0 0.5 0.5
31      0       0 0.5 0.5
32      1       0 0.5 0.5
33      1       0 0.5 0.5
34      1       0 0.5 0.5
35      0       0 0.5 0.5
36      0       0 0.5 0.5
37      0       0 0.5 0.5
38      0       0 0.5 0.5
39      1       0 0.5 0.5
```

```
40      1       0 0.5 0.5
41      0       0 0.5 0.5
42      1       0 0.5 0.5
43      1       0 0.5 0.5
44      1       0 0.5 0.5
45      1       0 0.5 0.5
46      0       0 0.5 0.5
47      0       0 0.5 0.5
48      0       0 0.5 0.5
49      0       0 0.5 0.5
50      1       0 0.5 0.5
51      1       0 0.5 0.5
52      0       0 0.5 0.5
53      0       0 0.5 0.5
54      0       0 0.5 0.5
55      1       1 0.0 0.25
56      1       0 0.5 0.5
57      1       0 0.5 0.5
58      0       0 0.5 0.5
59      1       1 0.0 0.25
60      1       0 0.5 0.5
61      1       0 0.5 0.5
62      1       0 0.5 0.5
63      0       0 0.5 0.5
64      0       0 0.5 0.5
65      0       0 0.5 0.5
66      0       0 0.5 0.5
67      0       0 0.5 0.5
68      1       0 0.5 0.5
69      0       0 0.5 0.5
70      0       0 0.5 0.5
71      0       0 0.5 0.5
72      1       0 0.5 0.5
73      1       1 0.0 0.25
74      0       0 0.5 0.5
75      0       0 0.5 0.5
76      0       0 0.5 0.5
77      1       0 0.5 0.5
78      1       0 0.5 0.5
79      1       0 0.5 0.5
80      0       0 0.5 0.5
81      0       0 0.5 0.5
82      0       0 0.5 0.5
83      0       0 0.5 0.5
84      1       0 0.5 0.5
85      1       0 0.5 0.5
86      0       0 0.5 0.5
87      0       0 0.5 0.5
88      1       0 0.5 0.5
89      1       0 0.5 0.5
90      1       0 0.5 0.5
91      1       0 0.5 0.5
92      0       0 0.5 0.5
93      0       0 0.5 0.5
94      0       0 0.5 0.5
95      1       0 0.5 0.5
96      1       0 0.5 0.5
97      1       0 0.5 0.5
98      0       0 0.5 0.5
99      1       1 0.0 0.25
100     1       1 0.0 0.25
Accuracy:  60.0
```

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 To do so, make sure that

- mapper.py counts all occurrences of a list of words, and
- reducer.py

  performs the multiple-word Naive Bayes classification via the chosen list.

In [72]:
```
# 1.4

!./pNaiveBayes.sh 3 "assistance valium enlargementWithATypo"
!cat enronemail_1h.txt.output
```

```
Priors are:
1 1.0

0 0.0


Posteriors are:
assistance in class 1 0.037037037037

enlargementWithATypo in class 1 0.0

valium in class 1 0.0

DOC_ID | TRUTH | CLASS
=======================

1       0       1 0.0 1.0
2       0       1 0.0 1.0
3       0       1 0.0 1.0
4       0       1 0.0 1.0
5       0       1 0.0 1.0
6       0       1 0.0 1.0
7       0       1 0.0 1.0
8       0       1 0.0 1.0
9       1       1 0.0 1.0
10      1       1 0.0 1.0
11      1       1 0.0 0.037037037037
12      0       1 0.0 1.0
13      0       1 0.0 1.0
14      0       1 0.0 1.0
15      0       1 0.0 1.0
16      1       1 0.0 1.0
17      1       1 0.0 1.0
18      0       1 0.0 0.037037037037
19      0       1 0.0 1.0
20      0       1 0.0 1.0
21      1       1 0.0 1.0
22      1       1 0.0 1.0
23      0       1 0.0 0.037037037037
24      0       1 0.0 1.0
25      0       1 0.0 1.0
26      0       1 0.0 1.0
27      1       1 0.0 1.0
28      1       1 0.0 1.0
29      0       1 0.0 1.0
30      0       1 0.0 1.0
31      0       1 0.0 1.0
32      1       1 0.0 1.0
33      1       1 0.0 1.0
34      1       1 0.0 1.0
35      0       1 0.0 1.0
36      0       1 0.0 1.0
37      0       1 0.0 1.0
38      0       1 0.0 1.0
39      1       1 0.0 1.0
40      1       1 0.0 1.0
41      0       1 0.0 1.0
42      1       1 0.0 1.0
43      1       1 0.0 1.0
44      1       1 0.0 1.0
45      1       1 0.0 1.0
46      0       1 0.0 1.0
47      0       1 0.0 1.0
48      0       1 0.0 1.0
49      0       1 0.0 1.0
50      1       1 0.0 1.0
51      1       0 0.0 0.0
52      0       1 0.0 1.0
53      0       1 0.0 1.0
54      0       1 0.0 1.0
55      1       1 0.0 0.037037037037
56      1       1 0.0 1.0
57      1       1 0.0 1.0
58      0       1 0.0 1.0
59      1       1 0.0 0.037037037037
60      1       1 0.0 1.0
61      1       1 0.0 1.0
62      1       1 0.0 1.0
63      0       1 0.0 1.0
64      0       1 0.0 1.0
```

```
65      0       1 0.0 1.0
66      0       1 0.0 1.0
67      0       1 0.0 1.0
68      1       1 0.0 1.0
69      0       1 0.0 1.0
70      0       1 0.0 1.0
71      0       1 0.0 1.0
72      1       1 0.0 1.0
73      1       1 0.0 0.037037037037
74      0       1 0.0 1.0
75      0       1 0.0 1.0
76      0       1 0.0 1.0
77      1       1 0.0 1.0
78      1       1 0.0 1.0
79      1       1 0.0 1.0
80      0       1 0.0 1.0
81      0       1 0.0 1.0
82      0       1 0.0 1.0
83      0       1 0.0 1.0
84      1       1 0.0 1.0
85      1       1 0.0 1.0
86      0       1 0.0 1.0
87      0       1 0.0 1.0
88      1       1 0.0 1.0
89      1       1 0.0 1.0
90      1       0 0.0 0.0
91      1       1 0.0 1.0
92      0       1 0.0 1.0
93      0       1 0.0 1.0
94      0       1 0.0 1.0
95      1       1 0.0 1.0
96      1       0 0.0 0.0
97      1       1 0.0 1.0
98      0       1 0.0 1.0
99      1       1 0.0 0.037037037037
100     1       1 0.0 0.037037037037
Accuracy:  41.0
```

After looking through the predictions for 1.3 and 1.4 more closely, it appears I wasn't able to make any positive predictions of spam. It is very likely that laplacian smoothing would greatly improve the accuracy as our vocabulary is extremely sparse compared to the entire vocabulary of the corpus. Furthermore, it appears that for reasons I wasn't able to figure out, I was unable to accurately count the lines that contained "valium". This is probably why the prediction for 1.4 is practically identical to 1.3.

```
In [ ]:
```