Zhiping Xiu

1):

We have the equation for this task, but we still need to make decision for following things:

1: How should we parse the email, this can hugely influence the dictionary.

2: How should we deal with 0 probability,

e.g:P[word='laurance' I spam='True'] = 0

Those zeros can lead the sum of logs to -inf (or product probabilities to 0)

The task for this question is done by function makeDict(spam, ham, dictionary_filename) and parse(text_file) defined in spamfilter.py

Parse the email simply by split the raw input with \s is not a good choice, this will introduce too many un-words into our dictionary, E.g: <\br><\br><\br><\br><\br> as a word.
I used python's built in email parser to extract the payload of the email, and then removed the numbers and none letters, and change all the uppercase to lowercase. As my way to parse the email.

To deal with 0 probability, after I counted the appearance of each words, I removed the words that only showed in one label, E.g: 'laurance' only showed up in spam email but there are no ham include 'laurance', so the program will remove this word.

Update:
Professor Pardo introduced pseudo count in the class, and it turns out to be a better way (lower error rate during cross validation). So I applied pseudo count eventually in my code. And it works better. Then I tried to change the parse code, and it caused worse result, so I choose to not changing the parse code (commented out)

Please see more details in code.

2):

The function for this task is spamSort() and isSpam() defined in spamfilter.py

They are pretty much coded totally according to the equations. With two little change:

1: In my program, a yielder will feed training and testing set file directly to the filter function (instead of the dir of them, as defined in start up code).

2: spamSort() will return the number of spam it classified (positive result), this return value will be used to evaluate the performance. (see more in question 4 where I stated my method of doing cross validation).

You can test the code using:

```
~ python spamfilter.py 'trainingDir/spam' 'trainingDir/ham' 'testingDir'
```

This function will split the results in testingDir to sorted_ham and sorted_spam
Please see more details in code.

3):

If there were no underflow problem, equation 6 and 7 will always return the same result.

Prove:

Log is a monotonic function, since we are maximizing the products in equation 6, put this result with log will not influence the parameter (in this case is v_i) that lead to maximize this value.

Since log the product of a serial of values is equal to sum the log all a serial of values, we can then change the product equation to sum equation.

If we have underflow problem, since there were too much very small percentage value in our dictionary, trying to get the product of them will easily lead to underflow problem, if this happen, we probably get two 0 (or -inf) for input v_i = spam or v_i = ham, returning same result for two different assert. So in good chance we will not get the same result.

4):

A):

1: Datasets: 20030228_spam.tar.bz2 & 20021010_easy_ham.tar.bz2 as spam and ham.

2: Test & Training Sets: I used cross validation, see more explanation in 4_B.

3: Since I used cross validation, my testing set and training set are not same.

4: I choose to use cross validation because I want to remove the RNG caused by manually splitting test and training set.

5: The data set contains 3,051 emails, including 2551 ham and 500 spam.

6: I don't think this rate of spam can correctly represent the spam rate in real world, this is just a guess, but usually I don't see so many spam in my email box (or spam box). So I think the real world spam rate should be lower. I'm not sure about this those, because I'm just assuming this according to my own experience. There are might be more spam out there which I didn't suffer.

B):

Run following command will initiate a 10 fold cross validation on spam filter.

```
~ python cv.py 'sourceDataSet' 'resultDir' 10

# ~ python cv.py <sourceDir> <resultDir> <n-fold cross validation>
```

Please Notice:

Don't change the <sourceDir>, unless you want to use your own dataset.

Every time when you call the function, the program will generate a dir with the <resultDir> you given, and a 16 length random string (to prevent reuse dir).
You may want to clear the generated dir manually.

The program will first split the input datasets to n fold, and store them on <resultDir> under sub-dir: cvInput/. This is done by splitKFold(sourceDir, targetDir, k) method defined in cvGen.py.

Then, a reader will read from cvInput/, and yield the training set and testing set for each cross validation. This is done by cvRead(cvDir, k) method defined in cvGen.py

For each cross validation, now we have get the training set and testing set, I built the dictionary with training set and test them on the testing set.

The measure the error using error_rate. As described in question 1-B, spamSort() will return number of spam it classified (positive result). I split the testing according to their true classification, and feed them to spamSort() respectively. So, it's easy to reason when we feed spamSort() with all spam testing set, we will get true positive count, when we feed spamSort() with all ham testing set, we will get false positive count. Then I can count error rate according the true and false positive count and the total count of testing set easily.

Cross Validation is performed by cv() defined in spamfilter.py.

C):

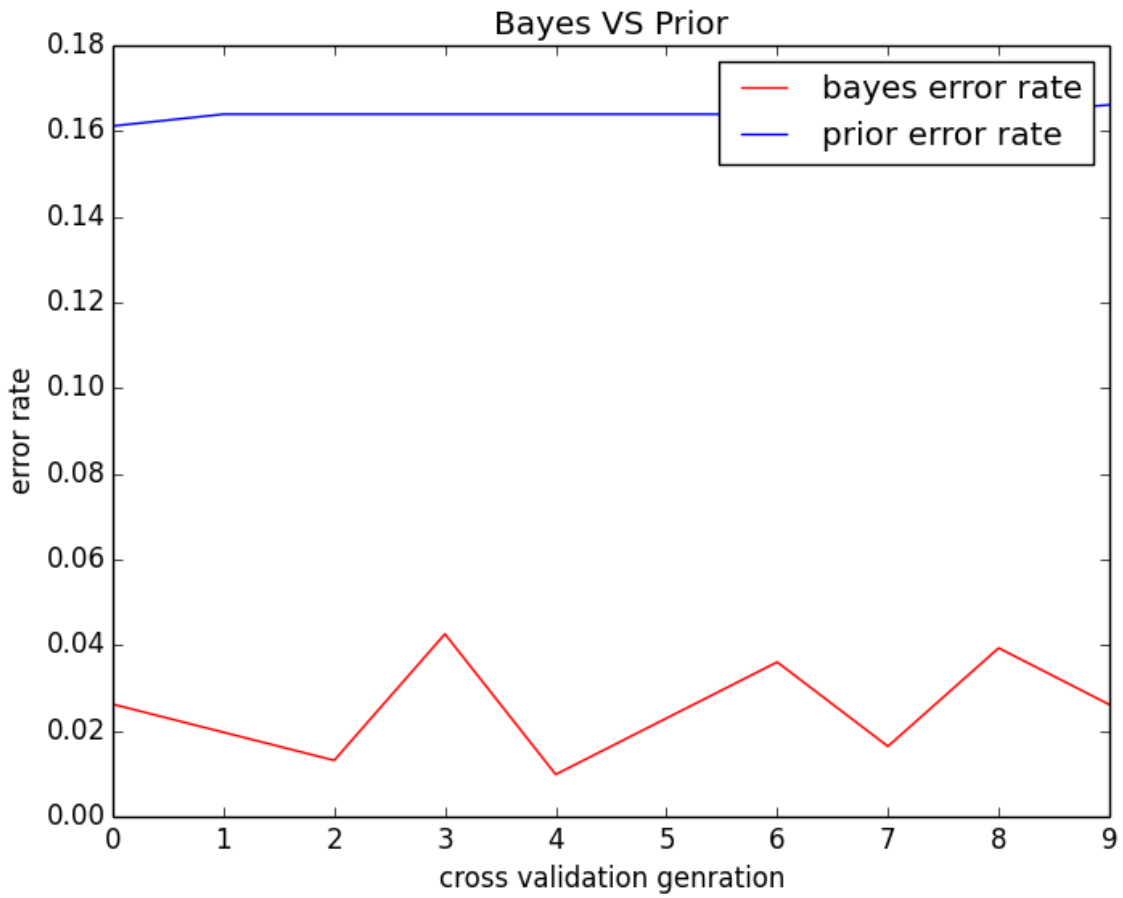
The performance of Naive Bayes classifier surpasses the performance of Prior classifier.

You can use the batch compare script to see the result by yourself:

```
~ python compare.py
```

Every time when you call the function, the program will generate a two dir 'resultDirBayes' and 'resultDirPrior', and a 16 length random string appended to each dir (to prevent reuse dir). You may need to remove resultDirBayes and resultDirPrior by hand

Script compare.py will run 10-fold cross validation for both methods, and compare their result. Plotting following chart and result output with statical test. (see in last page)



Zhiping Xiu

Output by comparing Bayes and Prior classifier. After you test it, remember to remove resultDirBayes and resultDirPrior before your next call.

```
➜  Zhiping_Xiu_hw5 python compare.py
Bayes's turn
cv_0: trained on 2746 emails, tested on 305 emails, misclassified: 8 emails, errorRate: 0.026230
cv_1: trained on 2746 emails, tested on 305 emails, misclassified: 6 emails, errorRate: 0.019672
cv_2: trained on 2746 emails, tested on 305 emails, misclassified: 4 emails, errorRate: 0.013115
cv_3: trained on 2746 emails, tested on 305 emails, misclassified: 13 emails, errorRate: 0.042623
cv_4: trained on 2746 emails, tested on 305 emails, misclassified: 3 emails, errorRate: 0.009836
cv_5: trained on 2746 emails, tested on 305 emails, misclassified: 7 emails, errorRate: 0.022951
cv_6: trained on 2746 emails, tested on 305 emails, misclassified: 11 emails, errorRate: 0.036066
cv_7: trained on 2746 emails, tested on 305 emails, misclassified: 5 emails, errorRate: 0.016393
cv_8: trained on 2746 emails, tested on 305 emails, misclassified: 12 emails, errorRate: 0.039344
cv_9: trained on 2745 emails, tested on 306 emails, misclassified: 8 emails, errorRate: 0.026144
Mean error rate: 0.025237
Prior's turn
cv_0: trained on 2747 emails, tested on 304 emails, misclassified: 49 emails, errorRate: 0.161184
cv_1: trained on 2746 emails, tested on 305 emails, misclassified: 50 emails, errorRate: 0.163934
cv_2: trained on 2746 emails, tested on 305 emails, misclassified: 50 emails, errorRate: 0.163934
cv_3: trained on 2746 emails, tested on 305 emails, misclassified: 50 emails, errorRate: 0.163934
cv_4: trained on 2746 emails, tested on 305 emails, misclassified: 50 emails, errorRate: 0.163934
cv_5: trained on 2746 emails, tested on 305 emails, misclassified: 50 emails, errorRate: 0.163934
cv_6: trained on 2746 emails, tested on 305 emails, misclassified: 50 emails, errorRate: 0.163934
cv_7: trained on 2746 emails, tested on 305 emails, misclassified: 50 emails, errorRate: 0.163934
cv_8: trained on 2746 emails, tested on 305 emails, misclassified: 50 emails, errorRate: 0.163934
cv_9: trained on 2744 emails, tested on 307 emails, misclassified: 51 emails, errorRate: 0.166124
Mean error rate: 0.163878
Mean error rate for Bayes Classifier: 0.025237
Mean error rate for Prior Classifier: 0.163878
p-value: 7.27707814834e-19
```