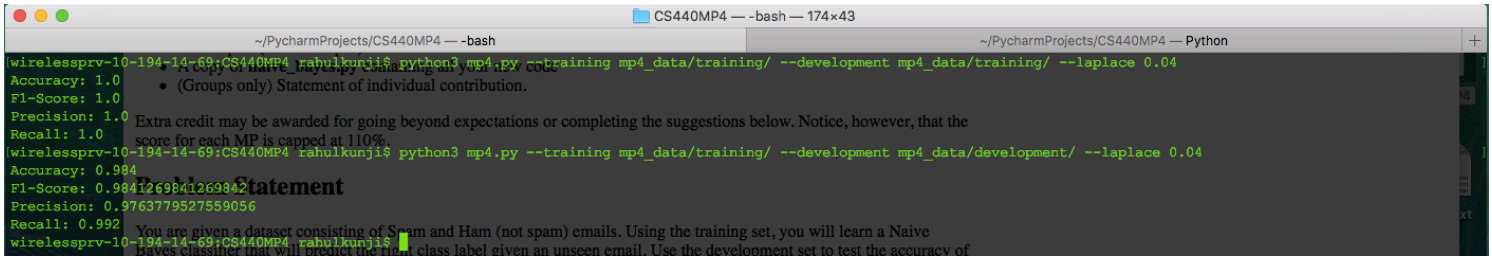
CS440: MP4 - Application of Naive Bayes

Naive Bayes spam email classifier

Date - 22 October 2018



Final Accuracy on Training data: 100%
Final Accuracy on Development data: 98.4%

0) Data Points

Naive MLE (no stemming, default laplace):

```
wirelesssprv-10-194-14-69:CS440MP4 rahulkunji$ python3
mp4.py --training mp4_data/training/ --development
mp4_data/development/
Accuracy: 0.968
F1-Score: 0.9677419354838709
Precision: 0.975609756097561
Recall: 0.96
```

Naive MLE (provided stemming, default laplace):

```
wirelesssprv-10-194-14-69:CS440MP4 rahulkunji$ python3
mp4.py --training mp4_data/training/ --development
mp4_data/development/ --stemming
Accuracy: 0.964
F1-Score: 0.963855421686747
Precision: 0.967741935483871
Recall: 0.96
```

Naive MLE (custom stemming, default laplace):

```
wirelesssprv-10-194-14-69:CS440MP4 rahulkunji$ python3
mp4.py --training mp4_data/training/ --development
mp4_data/development/
Accuracy: 0.976
F1-Score: 0.976
Precision: 0.976
Recall: 0.976
```

TF_IDF (custom stemming, default laplace):

```
wirelesssprv-10-194-14-69:CS440MP4 rahulkunji$ python3
mp4.py --training mp4_data/training/ --development
mp4_data/development/
Accuracy: 0.98
F1-Score: 0.9800796812749003
Precision: 0.9761904761904762
Recall: 0.984
```

TF_IDF (custom stemming, laplace = 0.04): - use train set for IDF

```
wirelesssprv-10-194-14-69:CS440MP4 rahulkunji$ python3
mp4.py --training mp4_data/training/ --development
mp4_data/development/ --laplace 0.04
Accuracy: 0.984
F1-Score: 0.9841269841269842
Precision: 0.9763779527559056
Recall: 0.992
```

TF_IDF (custom stemming, laplace = 0.04): - use dev set for IDF

```
wirelesssprv-10-194-14-69:CS440MP4 rahulkunji$ python3
mp4.py --training mp4_data/training/ --development
mp4_data/development/ --laplace 0.04
Accuracy: 0.984
F1-Score: 0.9841269841269842
Precision: 0.9763779527559056
Recall: 0.992
```

Naive MLE bigram (custom stemming, laplace = 0.04):

```
wirelesssprv-10-194-14-69:CS440MP4 rahulkunji$ python3
mp4.py --training mp4_data/training/ --development
mp4_data/development/ --laplace 0.04
Accuracy: 0.956
F1-Score: 0.9551020408163265
Precision: 0.975
Recall: 0.936
```

Naive MLE bigram (custom stemming, default laplace):

```
wirelesssprv-10-194-14-69:CS440MP4 rahulkunji$ python3
mp4.py --training mp4_data/training/ --development
mp4_data/development/
Accuracy: 0.956
F1-Score: 0.955465587044534
Precision: 0.9672131147540983
Recall: 0.944
```

TF-IDF bigram (custom stemming, default laplace):

```
wirelesssprv-10-194-14-69:CS440MP4 rahulkunji$ python3
mp4.py --training mp4_data/training/ --development
mp4_data/development/
Accuracy: 0.96
F1-Score: 0.9596774193548387
Precision: 0.967479674796748
Recall: 0.952
```

Naive MLE mixture model (custom stemming, default laplace, lambda = 0.04):

```
wirelesssprv-10-194-14-69:CS440MP4 rahulkunji$ python3
mp4.py --training mp4_data/training/ --development
mp4_data/development/
Accuracy: 0.976
F1-Score: 0.976
Precision: 0.976
Recall: 0.976
```

Naive MLE mixture model (custom stemming, laplace = 0.04, lambda = 0.04):

```
wirelesssprv-10-194-14-69:CS440MP4 rahulkunji$ python3
mp4.py --training mp4_data/training/ --development
mp4_data/development/ --laplace 0.04
Accuracy: 0.984
F1-Score: 0.9841269841269842
Precision: 0.9763779527559056
Recall: 0.992
```

TF-IDF mixture model (custom stemming, laplace = 0.04, lambda = 0.25):

```
wirelesssprv-10-194-14-69:CS440MP4 rahulkunji$ python3
mp4.py --training mp4_data/training/ --development
mp4_data/development/ --laplace 0.04
Accuracy: 0.984
F1-Score: 0.984
Precision: 0.984
Recall: 0.984
```

1) Accuracy, recall, and F1 scores on both the training and development sets.

TF-IDF mixture model (custom stemming, laplace = 0.04, lambda = 0.04):

```
wirelesssprv-10-194-14-69:CS440MP4 rahulkunji$ python3
mp4.py --training mp4_data/training/ --development
mp4_data/development/ --laplace 0.04
Accuracy: 0.984
F1-Score: 0.9841269841269842
Precision: 0.9763779527559056
Recall: 0.992
```

TF-IDF mixture model (custom stemming, laplace = 0.04, lambda = 0.04) on **Training**:

```
wirelesssprv-10-194-14-69:CS440MP4 rahulkunji$ python3
mp4.py --training mp4_data/training/ --development
mp4_data/training/ --laplace 0.04
Accuracy: 1.0
F1-Score: 1.0
Precision: 1.0
Recall: 1.0
```

2) The best Laplace smoothing parameter

α

Based on a number of tests, the best laplace smoothing parameter was found to be a low value of alpha. I chose it as 0.04 (note there were other values in its vicinity that also provided same accuracy rates)

The method that I used for this tuning is that I wrote shell loops to try out different values of alpha from 0 to 1 (in steps of 0.01 and captured the performance metrics)

```
wirelesssprv-10-194-14-69:CS440MP4 rahulkunji$ history |
cut -c 8- | grep for | uniq
```

```
for i in $(seq 0 0.01 1); do echo -e "\nLaplace Value
===> $i" 2>&1 | tee -a MP4_laplace_op.txt; python3 mp4.py
--laplace $i 2>&1 | tee -a MP4_laplace_op.txt; done
```

```
for i in $(seq 0 0.01 1); do echo -e "\nLaplace Value
===> $i" 2>&1 | tee -a MP4_laplace_TFIDFop.txt; python3
mp4.py --laplace $i 2>&1 | tee -a
MP4_laplace_TFIDFop.txt; done
```

```
for i in $(seq 0 0.01 1); do echo -e "\nLaplace Value
===> $i" 2>&1 | tee -a MP4_bigram_laplace_op.txt; python3
mp4.py --laplace $i 2>&1 | tee -a
MP4_bigram_laplace_op.txt; done
```

```
for i in $(seq 0 0.01 1); do echo -e "\nLambda Value ===>
$i" 2>&1 | tee -a MP4_mixture_test_op.txt; python3 mp4.py
--laplace $i 2>&1 | tee -a MP4_mixture_test_op.txt; done
```

Link to the performance metric files at each laplace level - https://uofi.box.com/s/xgaytxxuwgpdri92n3arwww9sj0guvl0

3) Stemming

I have reported the numbers above with and without the stemming flag.
Additionally I wrote my own stemmer which did 3 things

1. Used the PortStemmer from nltk to stem words
2. Removed Punctuations
3. Removed \r\n line endings

This provided the better accuracy and other performance metrics than just using the —stemming flag (which invokes PortStemmer)

Discuss the changes in accuracy:

When I just used the —stemming flag, the accuracy actually dropped by a fraction. This could potentially happen when your stemmer stems out certain words which actually had a good distinguishing power.

When I used my custom stemmer (preprocess(email_set) module), the accuracy increased. This is most likely owing to the improved word matching on removing punctuations and line ending differences from similar words.

4) Bigram Model:

Implemented the bigram model by looking at the emails as sets of bigrams. For example “I have a pretty dog” would become (I, have), (have, a), (a, pretty), (pretty, dog) ..

The accuracy rates are reported above.

The bigram model produced a lesser accuracy on the dev set than the unigram model.

I also implemented the mixture model by taking into account different values of lambda. The method is highlighted in part 2 above. I observed that I got the best results for lambda values that put more stress on the unigram results compared to the bigram ones, this is most likely because my unigram model was really well tuned (check section 6 for TF-IDF)

5) Questions:

1. Running naive Bayes on the bigram model relaxes the naive assumption of the model a bit. However, is this always a good thing? Why or why not?

A: The fact about whether relaxing the naive (conditional independence) assumption of the Naive Bayes model depends a lot on the problem context. While it is true that conditional independence assumption takes our model away from how things actually are in real life, more often than not this is not that harmful. The assumption makes our model much simpler and the gains in complexity reduction far outweigh the possible performance gain by relaxing the naive assumption. It can be proven that Naive Bayes is optimal even when features are dependent, but linearly separable. Moreover, sometimes focussing on the dependence between words, like we did with bigrams, reduces the discriminating power of the individual words. For example consider a case where we have "transfer bitcoins" in the training set. Here both "transfer" and "bitcoins" add somewhat to the possibility of being spam. However if you see something like "send bitcoins" or "transfer money" in the dev set, our bigram model will not recognise this from the training set as spam-like, but our naive unigram model would.

2. What would happen if we did an N-gram model where N is a really large number?

A:
Effectively as we increase the N in the N gram model, it should lead to lesser and lesser false positives as spam. However this would come at the cost of reduced total overall detection of spam, as you would end up matching only exact matches of word sequences. Moreover, this would make our classifier overly complex.

3. State the accuracy, recall, and F1 scores on both the training and development sets.

A: This has been done in the section 0

4. State your optimal Laplace smoothing parameter, and how you chose it. Mention other possible parameters you tried and the impact it had on your classification accuracy.

A: 0.04 (Explanation is in section 2)

5. We have said that this algorithm is called "naive" Bayes. What exactly is so naive about it?

A: A Naive Bayes classifier assumes that the presence (or absence) of a particular feature of a class is unrelated to the presence (or absence) of any other feature, given the class variable. Basically, this translated to the fact that we compute the joint probabilities of the various words in the email as a product of the probabilities of individual words. This is something that is rarely met in the real world.

6. Naive Bayes can classify spam quite nicely, but can you imagine classification problems for which naive Bayes performs poorly? Give an example, and explain why naive Bayes may perform poorly.

A: Naive bayes can perform poorly is when the conditional independence assumption does not hold true.
However, Harry Zhang (http://www.aaai.org/Papers/FLAIRS/2004/Flairs04-097.pdf) in his paper 'The Optimality of Naive Bayes' explains how naive bayes performs well not only when features are independent, but also when dependencies of features from each other are similar between features. No matter how strong the dependences among attributes are, naive Bayes can still be optimal if the dependences distribute evenly in classes, or if the dependences cancel each other out.

Given this, an example of where Naive Bayes cannot do classification well is the XOR problem. (Source: https://stackoverflow.com/questions/41682781/why-does-naive-bayes-fail-to-solve-xor)

The XOR problem is the most simple problem that is not linearly separable. Imagine you have two Boolean variables X and Y, and the target value you want to "predict" is the result from XORing the two variables. That is, only when either (but not the other) is 1, you want to predict 1 as outcome, and 0 otherwise. A bit more graphically:

```
Y ^
1 | XOR(x=0,y=1)=1  XOR(x=1,y=1)=0
  |
0 | XOR(x=0,y=0)=0  XOR(x=1,y=0)=1
  +------------------------------>
            0                1     X
```

As you can see, for the four "points" of my "plot" above (X horizontally, Y vertically; imagine the commas are the "points", if you like), there is no way you can draw a straight line that separates the two outcomes (the two 1s in the upper left and lower right, and the two 0s, also in opposing corners). So *linear* classifiers, which model the class separation using straight lines, cannot solve problems of this nature.

Now, as to Naive Bayes, it models *independent* events. Given only X and Y, it can model the distribution of xs and it can model the ys, but it does not model any relation between the two variables. That is, to model the XOR function, the classifier would have to observe both variables *at the same time*. Only making a prediction based on the state of X without taking into account Y's state (and vice versa) cannot lead to a proper solution for this problem.

Further Reading: http://www.ece.utep.edu/research/webfuzzy/docs/kk-thesis/kk-thesis-html/node19.html

6) TF-IDF:

One of the issues that I felt with a naive bayes classifier was that stop words like “the” “if” were also factoring in into our calculations for spam and ham probabilities. Also, words like “Subject” which would be present in mostly all emails also added a slight bias towards one of the spam or ham estimation of a particular email. Based on the number of times these occur, they would tend to overpower actual meaningful words like “bitcoins”, “lucky” , “winner” etc.

So I incorporated the Inverse Document Frequencies in order to reduce the classification powers of these words.

Inverse Document Frequency is calculated as:

IDF(word) = log (M+1 / df(word))

M = total number of documents
df = Document Frequency (no of docs in which the word appears)

Finally probability is calculated as a product of MLE based probability (Term Frequency - no of times a word occurs given context) as well as the IDF.

This gave me a significant performance improvement as shown in the results from section 0.

7) Understanding the code:

The code is divided into 3 sections

MAIN - Driver code, finally using the TF-IDF mixture model. Rest previous approaches are commented.

UTILITY FUNCTIONS - contains the various stemming, and TF, IDF dictionary building methods

CLASSIFIERS - Contains the actual classifiers as described in section 0:

1) classify_dev_set - The most naive unigram MLE (TF) based classifier
2) classify_dev_set_with_idf - A smarter TF-IDF style unigram NB classifer
3) classify_dev_set_bigrams - A Naive Bigram based NB classfier
4) classify_dev_set_with_idf_bigrams - A TF-IDF style bigram classifier
5) classify_dev_set_with_mixture_model - A naive mixture model
6) classify_dev_set_with_mixture_model_idf - The final version - TF -IDF style unigram and bigram calculations