James Wang
PennID: 46576241

Improved Spam Filter Report

11/10/16

General System Design:
The system is rather simple: 2 global functions *load_tokens* and *log_probs* and 1 class *SpamFilter*. *load_tokens* handles creating the features and *log_probs* handles calculating the log-probabilities of those features. *SpamFilter* initializes an object with a 2 dictionaries for tokens in spam and ham and their respective log-probabilities, as well as a hard-coded smoothing parameter, and finally the class probabilities of spam and ham. The *is_spam* method checks to see if a given email is either spam or ham, using an implementation of the Naïve Bayes classifier.

Pre-Processing and Tokenization:
Pre-processing the data was handled in the *load_tokens* global function. Not too much pre-processing was actually performed on the data though – after getting the line-level data from the email using the *email.iterators.body_line_iterator* method, each element was stripped of white-space, then all strings were joined together by a single space each, and finally split again on white space. Afterwards, all blank values in the list were filtered out. This tokenization process creates one-gram features.

Experimentation:
Additional features were created within the *load_tokens* function. I also tried tuning the smoothing parameter at different values. Here were some of the features and combinations of features that I experimented with.

	(1) Base Case	(2) Lower smoothing	(3) One-gram & two-gram features
Smoothing Parameter Value	1e-05	1e-10	1e-10
Performance (Accuracy)	('dev/spam', 0.945) ('dev/ham', 0.995)	('dev/spam', 0.95) ('dev/ham', 0.995)	('dev/spam', 0.955) ('dev/ham', 0.995)

	(4) One-gram & "Numbers"	(5) One-gram & "Length"	(6) One-gram, two-gram, & "Length"
Smoothing Parameter Value	1e-10	1e-10	1e-10
Performance (Accuracy)	('dev/spam', 0.98) ('dev/ham', 0.995)	('dev/spam', 0.985) ('dev/ham', 0.995)	('dev/spam', 0.99) ('dev/ham', 0.995)

Smoothing parameter values I tried were 1e-05, 1e-10, and 1e-20 for all models. It seemed that any smoothing value lower than 1e-10 would result in the overall accuracy as 1e-10. A smoothing value of 1e-10 always outperformed a smoothing value of 1e-05. This is interesting, because it seems to suggest that words in testing, that did not appear in training, are not that useful in helping us classify spam or ham, because the model performs better when we assign these words a value that's closer to 0.

The first additional features I created were two-gram features off of the one-gram features. Coupled with a smoothing value of 1e-10, this seemed to give a boost in accuracy on dev/spam from 94.5% in base case to 95.5%.

The next feature I created was "Numbers", which added a "new_feature_numbers" token to the document every time an existing token in the document contained a character that was a number. The hypothesis here is that spam emails typically try to solicit money, therefore they'll use more digits in their emails as part of dollar figures. Adding this feature saw an increase in dev/spam accuracy up to 98%. However, combining "two-grams" and "Numbers" saw no increase in accuracy, so the hunt was still on for more features.

The next feature I tried was "Length". "Length" added a "new_feature_length" token to the document every time an existing token's length exceeded 20 characters. The hypothesis here is that spam emails typically have more HTML soup because they try to decorate their emails with flashier text and design. Different lengths were tested: 10, 20, and 30. 20 characters seemed to be the most optimal length with regards to performance accuracy, which saw a boost up to 98.5%.

Finally, combining "two-grams" and "Length" features yielded a model with the best performance: 99% accuracy on dev/spam. This means that the model predicted with 99% accuracy on spam emails it had never seen before, and 99.5% accuracy on ham emails it had never seen before. Adding "Numbers" resulted in no difference in model performance, so I opted to leave it out to reduce model complexity, thereby controlling for potential overfitting.

Analysis of Errors:
I was able to trace the emails that the model misclassified by specifying that it to print out the file path of the email every time a misclassification occurred. The emails that were spam but the model misclassified as ham were: dev222 and dev283. The emails that were ham but my model misclassified as spam were: dev118. Taking a look at the dev222 and dev283 emails, a commonality I empirically observed was that they did not contain a lot of HTML tags and the tokens were not too long in length. On the contrary, dev118 had a lot of HTML tags, which may have triggered its classification as spam. The best way to improve model performance is to either get more quality data or to create better features. However, one must be careful when creating features based off the misclassified emails attributes – this could potentially lead to overfitting.

	Log-Probability (Spam)	Log-Probability (Ham)	*Comments*
dev222	-3614.4188	-3585.1230	*The model was really close in getting this one right!*
dev283	-524003.1069	-511235.4080	*The model was rather far off on this one.*
dev118	-7073.2379	-7756.6768	*The model was also rather close on this one too.*