I would like to present a set of scripts that I used to process the enron spam dataset. I will give a general overview of performance criteria and measures here, detailed explanations will follow in the presentation tomorrow.

Initially, I decided to use the features defined for spam_base (https://archive.ics.uci.edu/ml/machine-learning-databases/spambase/) to develop my model. Spambase.names defines 48 variables of the type word_freq, some capital_run_length statistics and char_freq variables. I saved the word_freq and char_freq vaiables in a .txt file to allow for flexibility in changing these later.

1. (Pre)processing ***match_attributes.py***

To process the emails and extract meaningful features I implemented a Python script to index two directories containing the spam and ham (match_attributes.py). This script does the following:

   a. Make a list of all file paths of the files in the spam and ham folders and subfolders, exluding source zip or tar files
   b. Parse every email using the email.parser module to create an object with all the relevant information
   c. Tokenize the words in the body of the email using the tokenizer from spambayes, which has advanced algorithms for recognizing links and other (obfuscated) html attributes.
   d. Make one record per email, in which the relative frequency of the "word" and "char" features and the various capital_run_length variables as defined in spambase are recorded for the body of email, with the exclusion of "stop" words like the, we, I etc. that occur frequently but have a low information content. Word frequencies are assessed for spambayes tokenized words from which capitals and special characters are removed whereas special characters and capital_run_length variables are defined based on the raw data of the email body. Finally, a categorical variable 1 identifies spam emails whereas 0 identifies ham emails. All variables are stored as floats in a .tsv file with a header containing the variable names.

I only processed 'even' emails (%2 == 0) from the spam and ham list, because in a follow-up processing step I bases features on the content of the "even" emails from the spam and ham list. For a fair evaluation of the performance it is desirable to split the data in a training and test set, which I did on several different levels as explained further in sections to come. Also, the maximum number of emails from both spam and ham list is set at 60.000, as memory issues prevented me from processing more emails simultaneously.

After running match_attributes.py we end up with a .tsv file that contains for a large subset of the ham and spam emails the attribute values

2. Running the classifiers: ***email_classifier.py***

The classifier script applies machine learning algorithms to train a model to distinguish spam from ham entries from a training subset taken from the tsv file with parameter feature values. The model performance is subsequently evaluated on a separate "test" dataset for

which we know the categories, but let the model predict these. Comparing the predicted with the actual category (spam or ham) indicates model performance. Details on the parameters, processing etc. can be found in the comments of email_classifier.py.

Model selection

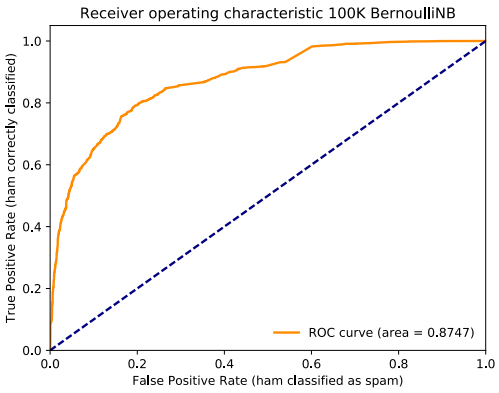
A) Bernoulli Naïve Bayes & spambase features

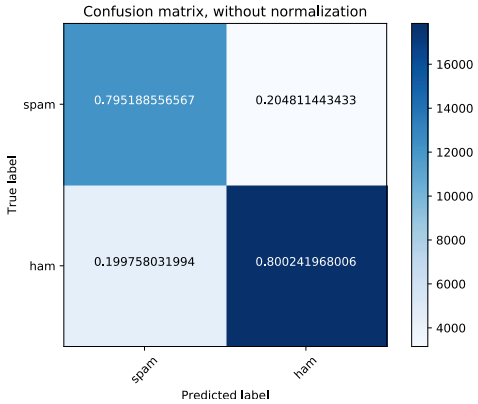
After reading several papers on the performance of different models on spam filtering I decided to try different implementations of naïve Bayes as available in the scikit-learn package, which performed well in previous studies. Among the different naïve Bayes models the Bernoulli naïve Bayes model performed best (results not shown). I used the function train_test_split configures with test_size=0.4 to split the data into 60% for training and 40% for evaluation. As evaluation criteria I selected a few commonly used and informative

measures:

- a Receive Operator Curve (ROC)-curve with the associated area under the curve (AUC)
- A normalized confusion matrix that yields true positive and negative ratios as well as false positive and negative ratios.

Results





For a first attempt the results do not look too shabby, 80% of both spam and ham emails are correctly classified. This is of course not good enough, so I decided to try improving on model performance by selecting different features.

Inspired by the features types of spambase I decided to take the same approach, but now take words that are over-represented in either spam or ham emails and use these features instead of the 48 spambase features. I decided to also increase the number of features to 500, as previous machine learning studies aimed at recognizing spam showed a higher performance of 500 features, but little improvement with more features.

3. Feature extraction (find_features.py)

This script follows the same logic as that described for match_attributes.py, but now we only take “uneven” emails. An equal number of ham and spam emails is processed. Spambayes tokenized word counts are used. For every spambayes-tokenized word the number of occurences in the body of spam and ham emails is recorded. Finally, I selected the 500 words with the highest difference in occurence (e.g. that are overrepresented in either spam or ham emails). Interestingly, this often includes tokenized features associated with (part of) the web address, see the top 11 below:

1. proto:http
2. url:com
3. url:www
4. url:
5. url:gif
6. url:net
7. enron
8. url:images
9. url:info
10. statements
11. url:biz

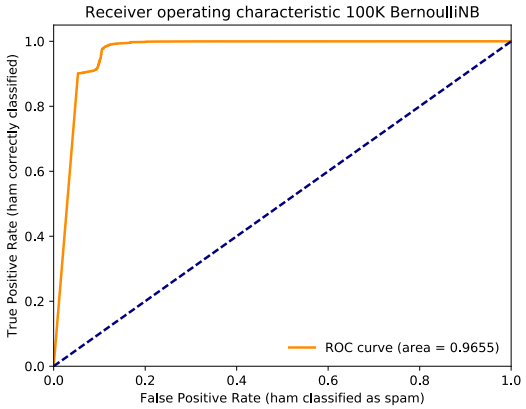
for the total list see “words.txt”

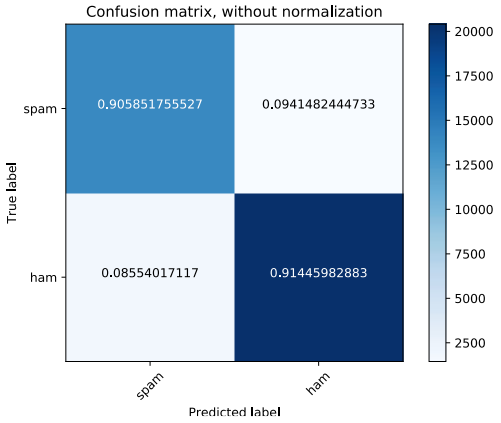
Based on this observation, I decided to also include the spambayes token categories (url, proto, skip) as words and count their relative frequency. This lead to a total of 527 features, of the following type:

500 word_freq (float 0.0 - 1.0), relative frequency of word/token in tokenized email body
18 CAT (float 0.0 - 1.0), relative frequency of category being present (as substring of) tokenized email body:
CAT_url:
CAT_skip:
CAT_at:
CAT_alias:
CAT_email:
CAT_email addr:
CAT_virus:
CAT_re:
CAT_here:
CAT_sent:
CAT_(including:
CAT_from:
CAT_8bit%:
CAT_price:
CAT_cc:
CAT_to:
CAT_subject:
CAT_proto:

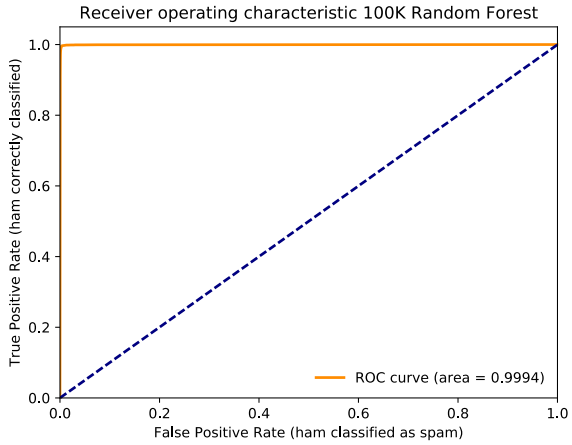
5 char_freq (float 0.0 – 1.0), relative frequency of special character in RAW txt of email body
3 capital_run_lenth type following spambase descriptions

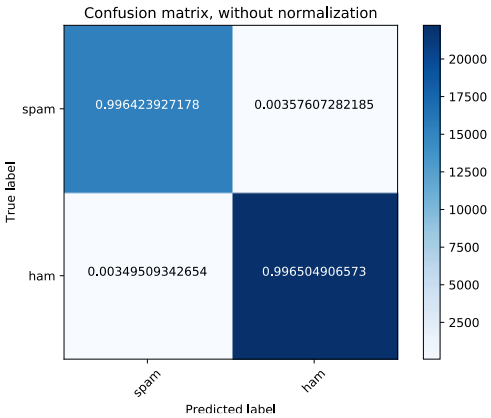
I created a new tsv file using ***email_classifier.py*** (enron_100K_527words.txt.gz which contains 94.243 records) and ran the Bernoulli Naïve bayes classifier again.





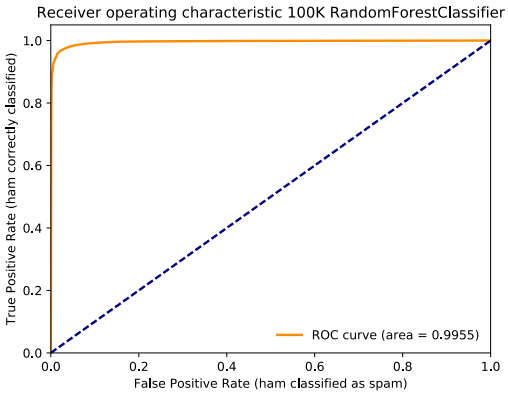
Clearly, with 527 features based on a subset of the ENRON-spam dataset we get a much better performance, Our AUC improves from 0.8747 to 0.9655 and the false positive and false negative rates are more than twice as low, going from ~20% to ~9%. Clearly, this is an improvement, but I still thought I could do better. After some googling on model types that were used frequently in spam filtering I came across a blog post mentioning high performance using Random Forest models, so I decided to give this a try. After some fidgeting with the parameters (I ended up taking RandomForestClassifier with n_estimators=30 and max_features=80), the model performance increased spectacularly:

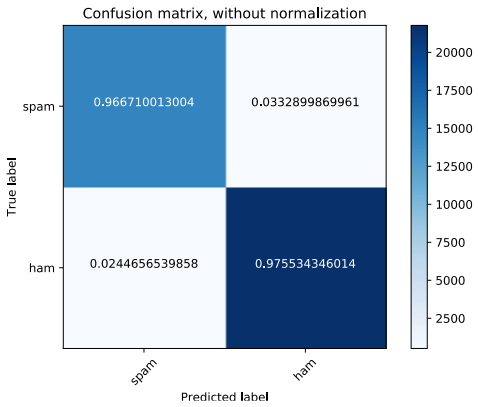




Clearly, model choice is very important. With an AUC of 0.9994 and a false positive and false negative rate of ~0.35% we have a model that closely approaches the state of the art in spam detection, which sits at around 99.9% accuracy. So, we are only 0.25% of! Getting at 99.9% accuracy might require more advance feature extraction, or a better parametrization of the model. Selecting a subset of emails that is send to the same user and comparing it with spam from the same user might similarly also dramatically improve the scoring. For now, I'm quite happy with the result.

So, how would such a classifier be expected to perform on new data? Well, in the worst-case scenario the word and character frequencies of this dataset are not at all representative for a new dataset. But this was also the case for the spambase features, so how does a Random forest classifier perform on the 48 spambase features? Quite well:





Even though the performance is substantially lower compared to a random forest classifier based on our 527 features relevant for this dataset, we still get a 2.4% false negative and 3.3% false negative rate. I would say that this is probably the minimum performance expected on any other spam/ham dataset, but probably it would be a lot better given the advanced parsing and tokenization routing from spambayes.