COS424 Assignment 1: Email Classification

Eddie D Zhou
ORFE '16
Princeton University
edzhou@princeton.edu

Daway Chou-Ren
COS '16
Princeton University
dchouren@princeton.edu

Abstract

Spam has plagued the inboxes of email users for decades now, and new companies and technologies have risen to fight the influx in past years. In this assignment, we address the problem of classifying emails into spam or ham using two different feature sets, and a variety of different classifiers, with the TREC 2007 spam dataset. We find that logistic regression in conjunction with bag-of-words features achieves the highest accuracy, but when examining different classifiers in a more refined and smaller feature space, a linear support vector machine achieves the best performance.

1 Introduction

We first use the given "vanilla" feature extraction script to extract bag-of-word features from our dataset. Using cross-validation, we obtain a validation score for each of the following three models – Naive Bayes, Logistic Regression, and AdaBoost. Picking the highest validation score, we re-train the model on the full training set, and obtain the final test accuracy upon the testing set originally given.

Next, as more of an experiment, we use custom features, extracted from the same dataset. Splitting the training set into a simple training / validation partition, we train the following models on the sub-training set – Random Forests, SVM (Gaussian), SVM (Sigmoid), SVM (Linear), Deep Neural Network, and Logistic Regression. We take the model with the highest accuracy on the untouched validation set, and then re-train said model on the full training set. Finally, we evaluate this model on the testing set to obtain an unbiased test accuracy.

1.1 Data processing

The TREC 2007 dataset was downloaded from the COS424 Piazza website on February 13th, 2015, with the folder already partitioned into a 90/10 training/testing split. The test set had 5000 emails classified 50/50 as Spam/Not Spam. Similarly, the training set was split into 22,500 Spam and 22,500 Not Spam, for a total of 50,000 emails.

Email Format

Each individual email was stored as a text file, along with email metadata. The format was similar for each email as follows: The first line included sender information, including email address and date/time sent. Following this there was return path information (the sender's email address), receiver information, and message IDs. After this there was typical email header information, again including a "From", "Reply To" (if applicable), "To", "Subject", "Date", mail software, MIME version, Content-type, various priority, Status, virus scan, content length, and line fields.

Many of the fields contained redundant information, such as the multiple identifications of the email sender. Each email contained slight variations on which fields were present, such as whether or not

the email had been scanned for viruses. This was obviously dependent on email software or services used. Following the metadata was the actual email body, encoded in plain text with HTML tags and email formatting present. Email footers were also retained.

Data Extraction

Using the `email_process.py` script that was written for us (with a small modification of dictionary threshold 200 lowered to 100), we extracted enough word counts for each training and testing email to form a dataframe with the 45000 examples, and 15228 features per email.

For the custom features, we modified `email_process.py` as `custom_email_process.py`. Originally this script was used to extract 179 custom features, but we discarded all except 68 because the variance for these features was not high enough to distinguish between classes.

The features kept were as follows and stored in this order, where W is the total word count and C is the total character count of an email:

1: hapax legomena / W
2-9 Ratio of punctuation characters / C: , . ! ? ; : ' "
11-32 Ratio of special characters / C: ~@ # $ % ^& * ( ) { } [ ] <>/ \- _ + =
33-37 Types of character ratios: Uppercase characters / C, Lowercase characters / C, Digit characters / C, Special characters / C, Punctuation characters / C
38-39: Total characters, total words
40-43 Time sent: Hour, minute, second, and day of week (0 = Sunday)
44-63 Word length counts: Number of words length 1 / W, length 2, ... length 19, length 20+

Data extraction for these custom features did not tokenize or stem. We did not stem because we cared about precise character counts and word lengths, and we didn't tokenize because we wanted to preserve punctuation.

1.2 Classification methods

We use three different classification methods from the SciKitLearn Python libraries for the vanilla feature set (all parameterizations are the default unless specified)

1. *Naive Bayes* (NB): Using the default parameters of MultinomialNB()
2. *Logistic regression with ℓ_2 penalty* (LOG): built on liblinear library
3. *AdaBoost* (AdB): using 50 decision trees as weak learners

For the custom feature set, we use six different classification methods in R:

1. *Random Forest* (RF): default params of randomForest
2. *Support Vector Machine (Gaussian)* (SVMG): default params
3. *Support Vector Machine (sigmoid)* (SVMS): default params
4. *Support Vector Machine (linear)* (SVML): default params
5. *Deep Neural Network* (DNN): 100 epochs, tanh activation, 3 layers of 50 nodes, 50% dropout for each layer, 20% of inputs dropped

1.3 Evaluation

In the vanilla feature space, we used 5-fold cross validation on the full training set to obtain a full generalization error for each of the three models. This is a simple accuracy metric – we sum the total number of errors each model incurred on each validation fold, and subtract it from 1. The k-fold cross validation gives us a prediction for each training sample when it is unseen, allowing us to use that total error as a metric to choose between models. In other words, we have the metric A_q for model q:

$$A_q = 1 - \frac{\sum_{i=1}^{5}(FP_{fi} + FN_{fi})}{N}$$

where FP_{fi} is the number of false positives obtained from training on all folds but fold i, and predicting on fold i (likewise for the false negatives FN_{fi}.

For our experimental custom feature set, we simply set aside 20% of the training set as a validation set, to avoid the cost of training the same model k times in the cross validation. Here, we also obtain some more in-depth metrics rather than just the validation accuracy: we also use the precision, recall, F_1-score, and log loss metrics, defined as

$$\text{precision} = \frac{TP}{TP + FP}, \qquad \text{recall} = \frac{TP}{TP + FN}$$

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}, \qquad \text{log-loss} = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{M} y_{i,j} \log(p_{i,j})$$

where for the log-loss, there are N samples, M classes (2 in our case), $y_{i,j}$ is 1 if sample i is in class j, and 0 otherwise, and $p_{i,j}$ is the model's probabilistic estimate of sample i belonging in class j.

2 Results

2.1 Evaluation results

2.1.1 Vanilla Features

For our vanilla features and three models, we include the number of misclassified samples in each validation fold (indexing from f0 to f4 instead of f1 to f5), the average number of misclassifications per fold, as well as the total number of misclassifications, which is easily manipulated into the generalization error (and accuracy). While it's clear that the total validation accuracy is extremely

	f0	f1	f2	f3	f4	average	total	gen_acc
NB	29	18	21	38	25	26.2	131	0.997089
LOG	12	9	9	15	10	11.0	55	0.998778
AdB	13	19	15	26	21	18.8	94	0.997911

high for all three methods, it is clear that the Logistic Regression model obtains the highest accuracy with respect to the entire validation process.

Taking the logistic regression as our best model, we re-train it on the full training set and then evaluate it in an unbiased fashion on the test set. Doing so gives us an accuracy that is comparable to the generalization accuracy obtained during the validation phase – **99.762%**. Given this extraordinarily high test accuracy, we should expect to see a relatively strange ROC curve, one that spikes up immediately as the threshold moves down from 99.99...% to 0%. This tells us that we are near our perfectly desired $(0, 1)$ point. This makes sense, because when we have a very high threshold, everything is classified as negative, and, as with all ROC curves, the first point is at $(0, 0)$. But because the classifier is so strong, its predicted probabilities are very close to 0 and 1, so the threshold only needs to be relaxed a small amount before the true positive rate shoots up. The ROC curves of such successful classifiers are almost misleading – with the points overlaid on the curve, we see that the fpr value simply jumps from 0.09 to 1 as the threshold goes to 0.

2.1.2 Custom Features

Because the more refined feature space is defined by only 63 features, the classifiers are working with less data and should be expected to have lower validation and testing scores. The idea here is that with much lower space, time, and computational costs, we might be able to achieve decent performance.

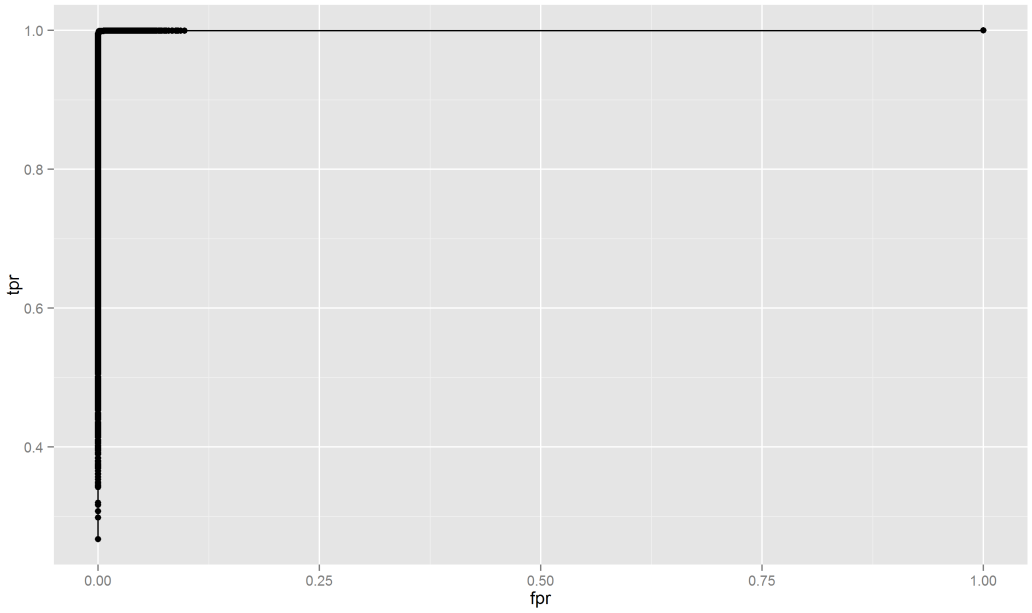


Figure 1: ROC curve for Logistic Regression with vanilla feature set

	validation accuracy	log loss	precision	recall	F_1
RF	0.5796667	0.3951222	0.8917411	0.1781494	0.2969708
SVM_gauss	0.4983333	0.1532875	0.4983333	1.0000000	0.6651835
SVM_sigmoid	0.3371111	3.6592386	0.3623350	0.4345596	0.3951744
SVM_linear	0.8593333	0.6662509	0.8217070	0.9166109	0.8665683
DNN	0.5195556	0.8806850	0.5176342	0.5268673	0.4307182
Logistic	0.4610000	10.2988752	0.4563246	0.4263099	0.4493372

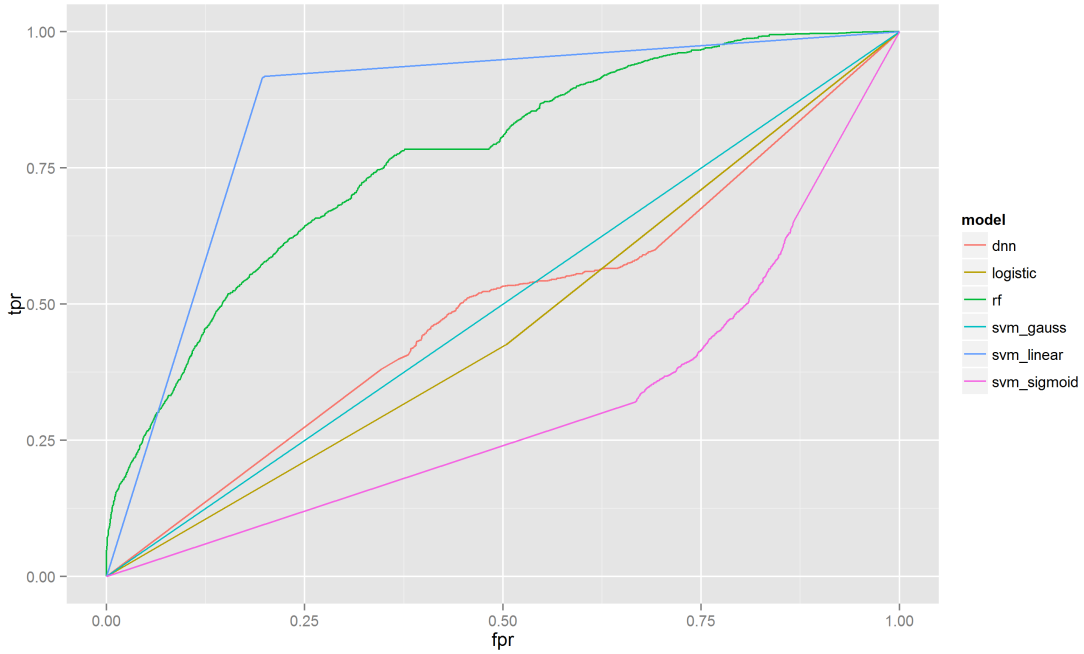
It is clear that the linear support vector machine blew all of the other models out of the water, with a much higher validation accuracy, precision, recall, and F_1 score. Since we have more models and computational capacity here, we also provide the ROC curves of every model on the validation set: The ROC curves confirm the numbers, telling us that the linear support vector machine is the model to move forward with. Re-training on the full training set, and then applying to the testing set gives us **88.98% accuracy**, which is very comparable to the vanilla feature set, especially considering the space, time, and computational improvements.

2.2 Computational speed

As expected, all of the classifiers' had much shorter training times on the custom features set than the times on the higher-dimensional vanilla feature set. Among the vanilla feature models, KNN was originally considered, but the prediction phase took too long given the massive distance calculations involved. Of the three, Naive Bayes was quicker than the Logistic Regression, which was much quicker than AdaBoost (which requires training all the weak learner decision trees). For the custom feature set, the support vector machines all took longer than the random forest, neural network, and logistic regression (with linear taking the longest).

3 Discussion

The difference in models considered in the validation phase between the vanilla and custom feature sets (as well as the choice to partition out a set validation set rather than employ cross-validation



in the custom feature training) was due largely to the desire for experimentation. The two final test accuracies are not directly compared – in any case, given the extraordinarily high validation and test accuracies in the models trained on the vanilla feature set, there is not a large difference in performance. This leads us to believe that even if we were to spend the huge amount of time necessary to train the linear support vector machine in 15228-space, the gain in accuracy over the more parsimonious and efficient logistic regression would be negligible.

3.1 Vanilla

Given the extremely high accuracy of the logistic regression on the vanilla feature datasets, it would be interesting to examine the incorrectly classified examples from that testing dataset. We split the testing set into two parts – the first being the 12 samples that were incorrectly classified, and the second being the 4988 samples that were correctly classified. Taking the average over each features, grouped by the set, we then examine the differences between each of these mean feature arrays – for how many of the features (if any) is there a significant difference in the values for the incorrectly classified data vs. the correctly classified data? The analysis gives 12705 features where the incorrect samples had a greater mean value, and 1446 features where the correct sample average was higher. The average of these differences was only 0.021 for when in the incorrect samples' favor, but a much larger 1.559 in the other direction. Almost all of the features have larger values in the wrongly classified set than in the correctly classified set. Moreover, for the few features that are larger in the correctly classified set, the difference is 74 times smaller. This leads us to believe that our logistic regression model mainly had trouble with examples where the feature values were relatively large.

We can further examine the classifier's specifics regarding the incorrectly and correctly classified examples by looking at the decision function values. The absolute value of the classifier's decision function values on the incorrect samples was 3.915, but on the correct samples, it was 17.188. It makes sense that the decision function scores are high for the correctly classified samples – in other words, the logistic model is very certain of the outputted classification on these samples. On the other hand, the magnitude of the decision function values on the incorrectly classified samples is much smaller, reflecting the uncertainty the model has in classifying them.

3.2 Custom Features

To analyze which custom features were most distinguishing, we took the difference of average feature values for the sets of emails classified as spam and not spam, divided by the average variance, and sorted in decreasing order. The corresponding feature indices (0-based), were as follows: `[26, 29, 9, 13, 12, 15, 3, 11, 37, 43, 30, 2, 8, 61, 62, 38, 23, 53, 21, 18, 17, 22, 31, 4, 7, 28, 27, 52, 55, 50, 5, 10, 24, 35, 49, 46, 1, 60, 57, 14, 58, 45, 44, 48, 34, 51, 54, 42, 0, 6, 36, 41, 56, 33, 19, 40, 20, 47, 16, 59, 32, 25, 39]`. The most distinguishing features turned out to be mostly special character frequencies, such as '/', '_', '"', '$', '#', '^', ',', and '@'. Of the ten most distinguishing features, the first eight were special characters, and the last two were the total number of characters and the day of the week the email was sent.

The influence of special character frequencies makes sense since these are most likely very infrequently used in non-spam messages. Even a relatively low usage in spam messages, perhaps because they are automated may give them away. We also expect time features like the day of the week to prove useful since spammers might use automated tools to batch send emails at certain times.

4 Conclusion and Extensions

In summary, using a slightly modified version of the bag-of-words features given to us alongside a logistic regression allows us to obtain superb testing accuracy of 99.762%. With a much smaller, custom-made feature space, the validation phase gives us a linear support vector machine, which produces a testing accuracy of 88.98% at a much smaller space cost.

A natural extension would be to train the linear SVM on the larger vanilla-feature dataset, but as mentioned earlier, the potential gains in accuracy would be negligible at best given the already superb (and presumably quicker) performance of other models on that data.

Another extension that we believe could be fruitful is hyperparameter tuning for the deep neural network – since these methods are highly parameter-dependent, tuning of the number of epochs, layers, etc. could result in performance that outperforms the linear SVM. This would be highly desirable, given the shorter training time for the DNN over the SVM.

From the testing accuracies of almost 90% using our much smaller feature set, it seems possible that a particularly well-refined feature set could generate results as good as the bag-of-words. Some ideas we would explore if given more time would be to full use the metadata associated with each email to generate features. Given the high distinguishing power of special characters, we would like to do character and token analysis on the email addresses of senders as well as on the subject lines. We would also like to target html tags by doing an analysis of what tags are used by spammers to catch readers' attention. Extending our set of syntactic and lexical features to include things like hapax dislegomena, types of capitalization used, letter n-grams, word n-grams, and even part of speech tags may also prove useful, though in the case of n-grams and part of speech tags, we would almost certainly do variance analysis to see which specific features would be useful since it seemed most of our character and word ratio features were not very distinguishing. We are also interested by the possibility of analyzing the structure of emails themselves. The number of paragraphs, length of paragraphs, and even the type and amount of whitespace used in between them may prove useful in email classification.

Another interesting approach may be to break up the set of non-spam emails through clustering. Emails sent to co-workers and bosses will be different both stylistically and structurally than emails sent to friends. It is possible that spam emails closely resemble one type of email. It may be useful to identify clusters of emails that are more similar to spam than others and extract features that have high distinguishing power for these clusters.