COS424 Assignment 1: Email Classification

Eddie D Zhou
edzhou@princeton.edu

Daway Chou-Ren
dchouren@princeton.edu

Abstract

Spam has plagued the inboxes of email users for decades now, but it is still a fruitful machine learning exercise to attempt spam classification. 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 with bag-of-words features achieves the highest accuracy, but when examining different classifiers in a more refined and smaller feature space, a linear SVM achieves the best performance.

1 Introduction

We first use the given script to obtain bag-of-word features, and using cross-validation, we choose among Naive Bayes, Logistic Regression, and AdaBoost. We re-train the best model on the full training set, and obtain the final test accuracy upon the testing set. Next, we use custom features, extracted from the same dataset, and train the following models – Random Forests, SVM (Gaussian), SVM (Sigmoid), SVM (Linear), Deep Neural Network, and Logistic Regression. Picking the best model on validation score, we re-train said model on the full training set and obtain a testing accuracy again.[1, 2]. All Python and R code is available at our GitHub repo [3].

1.1 Data processing

The TREC 2007 dataset was downloaded from the COS424 Piazza on 2/13/2015. [4] The test set had 5000 emails classified 50/50 as Spam/Haam, and the size 45000 training set was split 50/50.

Email Format

Each individual email was stored as a text file, along with email metadata, with the general format as follows: sender information (address and date/time sent), return path information, receiver information, and message IDs, header information (”From”, ”To”, etc.), and then the actual email body, encoded in plain text with HTML tags.

Data Extraction

Using the `email_process.py` script (modifying the dictionary threshold to 100), we extracted enough word counts for each training and testing email to obtain 15228 features per email.

The custom features are as follows and stored in this order, where W is the total word count and C is the total character count of an email:[5]

1: hapax legomena / W
2-9 Ratio of punctuation characters / C: , . ! ? ; : ’ ”
11-32 Ratio of special characters / C: ˜@ # $ % ˆ& * ( ) { } [ ] <>/ \- _ + =
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+

Note: neither stemming nor tokenization was used to preserve character counts and punctuation.

1.2 Classification methods

We use three different classifiers from the Sci-Kit-Learn Python libraries for the vanilla feature set: [6]

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 with various R packages: [7, 8]

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 simple accuracy metric – we sum the total number of errors each model incurred on each validation fold, and subtract it from 1.[9] 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. We also obtained some more in-depth metrics rather than just the validation accuracy: namely, precision, recall, F_1-score, and log loss metrics, defined as

$$\text{P} = \frac{TP}{TP + FP}, \quad \text{R} = \frac{TP}{TP + FN}, \quad F_1 = \frac{2\text{PR}}{\text{P} + \text{R}}, \quad \text{log-loss} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{i,j}\log(p_{i,j})$$

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. [10]

2 Results

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 and total number of misclassifications, and the accuracy. While it's clear that the total validation accuracy is extremely high for all three methods, the Logistic Regression model obtains the best performance. 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% test accuracy**. Given this extraordinarily high

	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

test accuracy, we see a ROC curve that spikes up immediately, with a corner very near our perfectly desired $(0, 1)$ point. [11] Due to length restrictions, this ROC curve is included in our Github repo [3].

For the more refined feature space, defined by only 63 features, the is that with much lower space, time, and computational cost, we can achieve comparable performance.

	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 performed the best, with a much higher validation accuracy, precision, recall, and F_1 score. The ROC curves confirm the numbers, regarding the linear

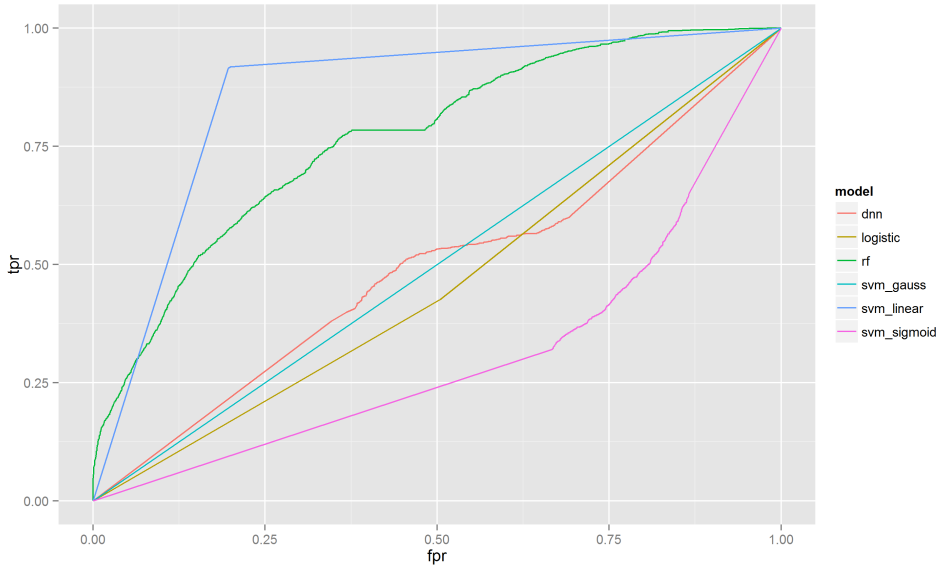

Figure 1: ROC curves for each classifier on the custom features validation set

SVM performance. Re-training on the full training set, and evaluating on test gives **88.98% test accuracy**, which is very comparable to the vanilla feature set given the lower costs.

3 Discussion

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. 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. This leads us to believe that our logistic regression model mainly had trouble with examples where the feature values (word counts) 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. [12]

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 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. 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. 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).[13, 14, 15]

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.

References

[1] Guzella TS, Caminhas WM (2009) A review of machine learning approaches to spam filtering. Expert Systems with Applications 36: 10206–10222.

[2] Hastie T, Tibshirani R, Friedman J, Hastie T, Friedman J, et al. (2009) The elements of statistical learning, volume 2. Springer.

[3] Zhou E, Chou-Ren D (2015). cos424-hw1. `https://github.com/edz504/cos424-hw1`.

[4] University P (2007). Trec 2007 data. URL `isun.cs.princeton.edu/trec07p_data.tar.gz`.

[5] Abbasi A, Chen H (2008) Writeprints: A stylometric approach to identity-level identification and similarity detection in cyberspace. ACM Transactions on Information Systems (TOIS) 26: 7.

[6] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, et al. (2011) Scikit-learn: Machine learning in Python. Journal of Machine Learning Research 12: 2825–2830.

[7] Liaw A, Wiener M (2002) Classification and regression by randomforest. R News 2: 18-22.

[8] Meyer D, Dimitriadou E, Hornik K, Weingessel A, Leisch F, et al. (2014) Package e1071 .

[9] Kohavi R, et al. (1995) A study of cross-validation and bootstrap for accuracy estimation and model selection. In: Ijcai. volume 14, pp. 1137–1145.

[10] Powers DM (2011) Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation .

[11] Wickham H (2009) ggplot2: elegant graphics for data analysis. Springer New York. URL `http://had.co.nz/ggplot2/book`.

[12] Zhu J, Ahmed A, Xing EP (2009) Medlda: maximum margin supervised topic models for regression and classification. In: Proceedings of the 26th Annual International Conference on Machine Learning. ACM, pp. 1257–1264.

[13] Matthews RA, Merriam TV (1993) Neural computation in stylometry i: An application to the works of shakespeare and fletcher. Literary and Linguistic Computing 8: 203–209.

[14] Merriam TV, Matthews RA (1994) Neural computation in stylometry ii: An application to the works of shakespeare and marlowe. Literary and Linguistic Computing 9: 1–6.

[15] Holmes DI, Forsyth RS (1995) The federalist revisited: New directions in authorship attribution. Literary and Linguistic Computing 10: 111–127.