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 of "the use of electronic messaging systems to send unsolicited messages (spam), especially advertising, as well as sending messages repeatedly on the same site." In this assignment, we address the problem of classifying emails into spam or ham (not spam) using two different feature sets, and a variety of different classifiers. The datasets used are the training and testing partitions of the TREC 2007 spam track overview 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

According to a report on workplace productivity by McKinsey, workers spend up to 28% of their time checking their email. While most companies have a rigorous spam filter or secure email gateway in place (such as Proofpoint or Gmail), it is still a fruitful exercise in machine learning to examine classifier and feature extraction in the spam space.

We first use the given "vanilla" feature extraction strict, modified to threshold 100, 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 with more meaning than simple bag-of-words features. 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 and testing split. 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 emai to form a dataframe with the 45000 examples, and 15228 features per email.

For the custom features, we)insert stuff from Daway's explanation here)

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

$$\text{precision} = \frac{TP}{TP + FP}$$

$$\text{recall} = \frac{TP}{TP + FN}$$

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

$$\text{log-loss} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{i,j}\log(p_{i,j})$$

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

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).

	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

2.2 Computational speed

The variability in the time for training and testing these linear classifiers was substantial (Table 1). In particular, we found that the KNN classifier, which does not perform training, takes the largest amount of time because of the all-by-all comparison that occurs during test phase. AdaBoost takes the second longest, but here the time is spent on training the weak classifiers and the weights of the linear combination of those weak classifiers. The fastest classifiers include the NB classifier, the perceptron, and the hinge loss classifier, followed by the linear SVM and then the decision tree and random forest classifiers.

2.3 Feature selection

These results highlight the benefits for some of the methods of reducing the number of features before training the classifiers. In particular, we found that using feature selection improved the precision for SVMS, AB, and RF classifiers, and the recall for KNN, LR, NB, SVMS, and RF classifiers. The largest improvement was for the SVMS and RF classifiers. The effect on the RF classifier might be mitigated by increasing the number of trees in the random forest for larger numbers of features, although this would slow down the training time proportionally. Across all methods, feature selection substantially improved the average wall clock time, e.g., improving the time of AdaBoost by 87.5%.

3 Discussion and Conclusion

In this work, we compared ten different classifiers to predict the newsgroup for a particular newsgroup post using bag-of-words features. We found that, considering precision, recall, and time, the decision tree and random forest classifiers showed superior performance on this task. The effect of feature selection was mostly on the time, although the improvement in performance was substantial for the random forest classifier on this task.

There are a number of directions to go that would improve these results. First, we could expand our data set using available data from these and other related newsgroups. Second, we could consider more sophisticated features for the newsgroup posts than dictionary word counts; bi-grams, post length, or punctuation may be useful in this classification task. Third, we could use the most promising models in a more problem-tailored way. In particular, because the random forest classifier showed such promise in this task, we could consider applying it to this problem using multi class class labels instead of one-versus-rest class labels, and reducing the dimension of the feature space using supervised latent Dirichlet allocation based methods [**?**, **?**].