A semi-supervised Spam mail detector

Bernhard Pfahringer

Department of Computer Science, University of Waikato, Hamilton, New Zealand

Abstract. This document describes a novel semi-supervised approach to spam classification, which was successful at the ECML/PKDD 2006 spam classification challenge. A local learning method based on lazy projections was successfully combined with a variant of a standard semi-supervised learning algorithm.

1 Introduction

This ECML/PKDD 2006 spam classification challenge[1] comprised two different setups, which both were setup in specific way encouraging especially the use of semi-supervised learning methods.

The setup for Task A comprised two mailboxes for tuning, one containing 4000 labeled email messages, the other one containing another 2500 labeled emails. Predictions were sought for another set of three un-labeled mailboxes of 2500 emails each, together with a labeled set of 4000 emails. The tuning set and the evaluation set were also distinct, i.e. they could not be fused to form a larger set of examples to learn from.

The setup for Task B problem comprised three much smaller mailboxes for tuning, one containing 100 labeled emails, the other two containing another 400 labeled emails each. Predictions were sought for another set of 15 un-labeled mailboxes of 400 emails each, together with a labeled set of 100 emails. The tuning sets and the evaluation sets again are distinct, i.e. again they could not be fused to form a larger set of examples to learn from.

Furthermore, all messages were already fully pre-processed and represented in SVMlight format, so no specific additional pre-processing was possibly. As proper pre-processing can be essential for learning success, this setup somewhat limited the possibilities, but also ensured a level playing field for all learning approaches. Generally in text classification, and especially in Spam classification, smart preprocessing can greatly simplify the problem, e.g. one can include attributes representing meta-information like sender names and servers, or other header-fields, or make a distinction between text coming from the subject and text coming from the message body and so on. None of that was possible for the fully pre-processed data given here.

2 Initial experiments

Initial experiments with standard (i.e. non semi-supervised) learning algorithms like multinomial Naive Bayes or support vector machines looked very promising

in cross-validation estimation on the labeled 4000 messages tuning box, but results did not transfer well over to the 2500 messages tuning box. Table 1 shows these somewhat surprising and more importantly disappointing results. A possible explanation might be the huge number of available attributes and potentially quite different usage patterns for these attributes in the two mailboxes. Therefore the learning algorithm could be focusing on a particular subset which works well for one mailbox, but not so for the other.

Table 1. Error rates (in percentages) on the tuning data of Task A for standard support vector machine (SMO) and multinomial NaiveBayes (MNB)

Algorithm	Crossvalidation: 4000 emails	Train/Test split 4000/2500 emails
MNB	3.075	51.20
SMO	0.950	20.56

Generally some form of feature subset selection should be able solve such an overfitting problem, but successful feature subset selection can be a very expensive and time-consuming process. Therefore an alternative lazy method was employed: whenever a prediction for some email is needed, transform the whole training set by only selecting those features which are actually present in the test-example (i.e. have a non-zero value). Essentially the training data is projected onto the subset of those features which are actually present in the testing email at hand. Then some model is trained using this transformed training set and that model's prediction is used for the test example in question. This transformation has some interesting properties: it forces the learner to concentrate on the features that are actually present in the test example. The number of such features is considerably smaller than the total number of features available, ranging from a dozen to about 2000, with the majority being in the low hundreds – instead of the more than a hundred thousand attributes in the full training set. Thus a larger range of classifiers becomes feasible, e.g. logistic regression. Still, linear support vector machines usually were among the best performing classifiers. Generally, using this lazy approach, results in terms of both estimated accuracies and estimated AUC values seemed to improve over the standard non-lazy approach described above. But even these improved estimates still left some clear room for improvement, which is why a proper semi-supervised approach building on this lazy projection idea was developed next.

3 Semi-supervised learning: a LLGC variant

Semi-supervised learning approaches have been shown in general to be able to improve over standard learning approaches when given plenty of unlabeled data. The given challenge problem does not quite fit the standard assumptions of semi-supervised learning, as actually more labeled than unlabeled data is given, at least in Task A: 4000 vs 2500.

A standard semi-supervised learning algorithm is the so-called LLGC algorithm [4], which tries to balance two potentially conflicting goals: locally, similar examples should have similar class labels, and globally, the predicted labels should agree well with the given training labels. We applied a scalable variant [3] of the standard LLGC algorithm to the challenge data. These experiments resulted in rather modest gains for predictive accuracies, but AUC values usually improved much more. As AUC was the criterion of choice for this challenge, this approach looked most promising and was chosen as the final solution. Here is a short summary of the three main modifications to the original LLGC algorithm:

- Allow different similarity measures: LLGC is based on a so-called affinity matrix capturing all pair-wise similarities over both labeled and unlabeled data. Originally, LLGC uses RBF kernels, but for text a cosine-based similarity measure is a much more appropriate, and is consequently used here.
- Allow pre-labeling of the unlabeled data: LLGC starts with all-zero class-distributions for the unlabeled data. We allow pre-labeling by using class-distributions for unlabeled data that have been computed in some way using the training data. The user can shrink the magnitude of these predictions relative to the hard class-labels of the supplied training set by way of a user-settable parameter. This shrinkage seems to improve results considerably when the number of unlabeled examples is much larger than the number of labeled examples. This is not the case for the current problem, and experiments on the tuning mailboxes returned best results when no shrinkage was employed. The pre-labels were computed by the lazy feature subset selection process described in the previous section together with a linear support vector machine as implemented in Weka.
- Reduce complexity by sparsifying the affinity matrix: the standard LLGC algorithm would need to compute the inverse of a dense $6500x6500$ matrix to solve the mailbox classification problem. To save both space and time, we sparsify the affinity matrix by only including the k nearest neighbours in the matrix. We do make sure that the matrix is still symmetric in a post-processing step after the kNN computation. The sparsification allows for a reduction in computational complexity from $O(n^3)$ to $O(n*k*iter)$ where n is the total number of examples, $k << n$ is the number of nearest neighbours, and *iter* is the number of iterations performed by the iterative relaxation algorithm which is used instead of matrix inversion.

4 The final setup

An extensive grid search over the tuning setup was used to find the following suitable values for all user-settable parameters for Task A:

- k = 100, the number of neighbours
- iter = 10, the number of iterations in the LLGC algorithm
- $\alpha = 0.8$, the mixing proportion of original labels to propagated information

– shrinkage = 1.0 (i.e. effectively no shrinkage), the trade-off factor for hard training labels versus estimated pre-labels.

The alpha parameter controls the balance between local and global consistency inside LLGC. Its range is $[0, 1]$, with higher values favouring local consistency over global consistency.

The three mailboxes of Task A were predicted in isolation of each other. It might be possible to improve the final results by actually fusing them all together into one big semi-supervised problem comprising 11500 examples. We did not attempt to do that as only one 2500 box for tuning was available. Therefore it was not possible to assess which of the two setup would perform better, so we chose the safe option of predicting each of the three mailboxes in isolation.

This final setup might seem to be rather expensive to compute, as we use a lazy approach for pre-labeling, and potentially expensive LLGC computation as a kind of post-processing step. In practise the computation is dominated by the pre-labeling effort, and takes about 90 minutes per mailbox, on an average Linux PC from 2005. It should therefore still be feasible in practise to run a similar setup on a client PC (but not a mail server itself) to rank a batch of emails for a particular user according to their degree of spamness.

A similar grid search over the tuning setup for Task B yielded almost identical values for all user-settable parameters:

– k = 100
– iter = 10
– $\alpha = 0.95$
– shrinkage = 1.0 (i.e. effectively no shrinkage)

It is interesting to note that for Task B an even more extreme *alpha* value (0.95 instead 0.8) was selected by this search, putting even more emphasis on the local neighbourhood of each example. Given the scarcity of labeled data in this setup this is probably a reasonable choice. It is also surprising to see that no shrinkage was chosen, even though Task B is more in line with the standard assumption of semi-supervised learning, having an order of magnitude more unlabeled than labeled examples available for learning.

As tests on the tuning setup showed slightly better AUC values for fusing all data into one big problem instead of predicting each mailbox in isolation, we chose this option for the final prediction task as well (again this is different to the Task A setup). So the final model used all 100 labeled training examples plus 15 times 400 unlabeled examples in one go. Even though this is about the same size as for one mailbox of Task A, computation was still much faster, as the computationally dominant pre-labeling step is a lot cheaper in this setting. We have to compute pre-labels for 6000 instead of 2500 examples, but each feature subset-selected training set has only 100 training instances instead of 4000 such instances, which makes for a big difference given the non-linear behaviour of support vector machine learning.

Therefore we conjecture that such joint processing of a number of relatively small mailboxes together with a carefully selected small training set could be feasible on a mail server for a small to medium-sized work group.

5 Conclusion and further research

This paper has introduced a successful combination of two new ideas for the detection of Spam messages: lazy projection of the training data to single messages to combat overfitting of the training data due to an abundance of features; and an efficient semi-supervised learning algorithm variant that can be viewed as a kind of post-processing step for smoothing out potentially conflicting predictions over similar messages. Pragmatically the most important direction for future work is the issue of pre-labeling. The current approach is effective, but also computationally expensive. An alternative to the lazy projection and training of a full classifier per unlabeled message would be projection of the training data to the set of all attributes being used in one full unlabeled mailbox. The next most expensive computational step after pre-labeling is the formation of the sparse affinity matrix, which is currently done in a naive linear full nearest neighbourhood search. More advanced methods like ball or cover trees should be able to speed up this step. More generally, we would like to investigate meta-approaches that would combine the architecture described in this paper with totally different ones like spam filters based on dynamic markov models [2].

Acknowledgments This work has been funded by a Marsden Grant of the Royal Society of New Zealand.

References

1. S. Bickel. Discovery challenge, 2006. http://www.ecmlpkdd2006.org/challenge.html.
2. A. Bratko, G.V. Cormack, B. Filipic, T.R. Lynam, and B. Zupan. Spam filtering using statistical data compression models, 2006. Journal of Machine Learning Research, in press.
3. B. Pfahringer, C. Leschi, and P. Reutemann. Scaling up semi-supervised learning: an efficient and effective llgc variant, 2006. in preparation.
4. D. Zhou, O. Bousquet, T. Lal, J. Weston, and B. Schölkopf. Learning with local and global consistency, 2003. In 18th Annual Conf. on Neural Information Processing Systems.