Towards Practical PPM Spam Filtering: Experiments for the TREC 2006 Spam Track

Andrej Bratko[1,3], Bogdan Filipič[1] and Blaž Zupan[2]

andrej.bratko@klika.si, bogdan.filipic@ijs.si, blaz.zupan@fri.uni-lj.si

Department of Intelligent Systems[1]
Jozef Stefan Institute
Jamova 39, Ljubljana, Slovenia SI-1000

Faculty of Computer and Information Science[2]
University of Ljubljana
Tržaška cesta 25, Ljubljana, Slovenia SI-1000

Klika, informacijske tehnologije, d.o.o.[3]
Stegne 21c, Ljubljana, Slovenia SI-1000

Abstract

This paper summarizes our participation in the TREC 2006 spam track. We submitted a single filter for the evaluation, based on the Prediction by Partial Matching compression scheme, a method that performed well in the previous TREC evaluation. A major focus of our effort was to improve efficiency of the method, particularly in terms of memory consumption, in order to establish whether compression-based filters are in fact a viable solution for practical applications. Our system exhibited fair performance, despite the fact that the filtering techniques remained virtually unchanged from the previous evaluation. We did not investigate methods for tackling delayed user feedback. A very simple strategy of training on most recent examples was used for the active learning task, and found to work surprisingly well given its simplicity.

1 Introduction

The Text REtrieval Conference (TREC) is an annual event aimed at encouraging and supporting research within the information retrieval community by providing the infrastructure necessary for large-scale evaluation of text retrieval methodologies. Since 2005, a track on spam filtering is included in the TREC framework, with the goal of providing a standard evaluation of current and proposed spam filtering approaches. To facilitate the evaluation effort, a collection of corpora, both public and private, is compiled annually by the track organizers.

Our group from the Jozef Stefan Institute in Ljubljana, Slovenia ("Institut Jožef Stefan" - IJS) participated in TREC 2005, using a spam filter based on statistical data compression algorithms (Bratko and Filipič, 2005). This unconventional filtering approach featured prominently in the results of the TREC 2005 evaluation. Despite the favorable performance, practical implementations of a compression-based spam filter are yet to emerge, with the notable exception of the "bit-entropy" filter included in CRM114 (Yerazunis, Yerazunis). We believe that the main obstacle to adopting such methods in practice is the large amount of memory that is typically required to train such models, and, to a lesser extent, the difficulties in implementing persistent versions of such models.

This paper describes the system submitted to the TREC 2006 spam track by the IJS group. We submitted a single filter configuration for both online filtering and active learning tasks, which featured essentially the same methods that were used for TREC 2005. However, the algorithms were re-implemented in a package that comes much closer to what would be expected of a production quality spam filter in terms of speed and memory efficiency.

The remainder of this paper is structured as follows. In the following section, we describe the filtering methods employed by the IJS filter. We discuss issues related to developing an efficient implementation of the Prediction by Partial Matching compression algorithm in Section 3. In Section 4, we review the results of the TREC evaluation. Finally, we outline some plans for future work in Section 5.

2 Methods

Our online spam filtering submission employed the same compression-based filtering methods that were used by the IJS system at TREC 2005. We did not investigate methods for tackling delayed user feedback and used a very simple strategy for the active learning task: Training on the most recent examples.

2.1 Online Filtering

Our spam filtering approach is based on statistical data compression models. Such models can be used to estimate the probability of a sequences of characters based on previous observations. The system submitted for evaluation at TREC 2006 is based on the Prediction by Partial Matching compression algorithm, a character-level Markov model that is described in the following section.

2.1.1 The Prediction by Partial Matching Algorithm

The Prediction by Partial Matching (PPM) compression algorithm (Cleary and Witten, 1984) is among the best-performing compression schemes for lossless text compression. Essentially, the PPM algorithm is a back-off smoothing method for finite-order Markov models, similar to back-off models used in natural language processing.

It is convenient to assume that an order-k PPM model stores a table of all subsequences up to length k that occur anywhere in the training text. For each such subsequence (or *context*), the frequency counts of characters that immediately follow it is maintained. When predicting the character x_i at position i in a sequence x_1^n, statistics associated with the preceding k characters x_{i-k}^{i-1} (the *context* of x_i) are used for prediction. If this particular context has not appeared in the training text, a shorter suffix of the sequence leading up to position i is looked up, until a context with non-zero statistics is found.

If the target character has appeared in this context in the training text, its relative frequency within the context is used for prediction. This probability is discounted by a small amount to reserve some probability mass, which is distributed among characters *not* seen in the current context, according to a lower-order model, that is, according to statistics associated with a shorter context. The procedure is applied recursively until all characters receive a non-zero probability. If necessary, a default model of order -1 is used, which always predicts a uniform distribution among all possible characters.

2.1.2 Using the PPM Algorithm for Spam Filtering

For classification tasks, we build one PPM model from the training data of each class, and use the trained models to evaluate the sequence of characters contained in the target document. The classification outcome is determined by the model that assigns the greatest probability to the target document (the document probability is simply the product of the probabilities of all characters contained in the document).

Our filter uses an order-6 PPM-D model (Howard, 1993) in combination with the exclusion principle (Sayood, 2000). The models of spam and legitimate email are used adaptively, that is, the models are updated after evaluating each character probability as if the sequence of characters leading up to the current position was part of the training data. Such models were found to perform well in previous experiments (Bratko et al., 2006). To produce scores that are comparable among different-length documents, the log odds of the message being spam is divided by the message length to produce the final “spamminess score”.

2.2 Active Learning

We used a very basic strategy for the active learning task: We decided to train on the most recent messages in the email stream first, in the hope that these recent examples better represent future data. The motivation for this approach is based on the study by Cormack and Bratko (2006), which indicates a that large amount of locality is present in the chronological order of messages.

3 System Description

Our system was designed as a client-server application. The server maintains the compression models required for classification in main memory and updates the models incrementally with each new training example. The system performs very limited message preprocessing, which primarily includes MIME-decoding and stripping messages of all non-textual message parts.

3.1 PPM Implementation

Most of our efforts for TREC 2006 were devoted to implementing an efficient compression-based spam filter, since efficiency was the main shortcoming of our previous system. In this section, we describe some technical details of this effort.

3.1.1 Optimizing Memory for PPM Models

Our PPM implementation is based on the suffix tree data structure (Weiner, 1973). The suffix tree is a tree data structure, used to represent all suffixes of a given sequence. The data structure requires no more than $2n$ tree nodes for this purpose, where n is the length of the indexed string. In practice, the size of suffix trees required for our PPM model is much smaller, primarily due to the fact that we only need to store suffixes of length not greater than $k + 1$, where k is the order of the PPM model.

In our implementation, sibling nodes at each level of the suffix tree are stored consecutively as sorted arrays. Our implementation requires 7 bytes of memory per node. Leaves of the suffix tree may store the corresponding suffixes, or contain pointers back into the training sequences to the same effect. The former approach is slightly faster, while the latter is better suited for higher-order models and for tasks that require indexing of strings or files. Memory buffers in which the model is stored take one of $|\Sigma|$ different sizes, where $|\Sigma|$ is the cardinality of the alphabet (i.e. the maximum arity of internal nodes).[1] We implemented custom memory allocators for our suffix tree implementation, which eliminate allocation overhead by taking advantage of the fact that all possible buffer sizes are known in advance.

Statistics required by the PPM algorithm must be stored at every node of the suffix tree. For a binary classification model, an additional 8 bytes are needed to store the statistical counters required by PPM at each node. These counters could trivially be quantized to a fraction of this size to further reduce memory usage. However, incremental training would not always be reversible in this case, since the counters may require scaling to prevent overflows. At 15 bytes per node, memory usage of our PPM filter peaks at 410 MB after training on the full TREC 2005 public corpus (92,189 messages; MIME decoded), which is comfortably within limits for the TREC evaluation.

3.1.2 Persistent Models

The memory requirements of our PPM model may still be unacceptable for very large datasets or for environments where memory is limited (e.g. for desktop use). To this end, our suffix tree implementation may also use file-based allocators. Such allocators return temporary copies of the actual buffers and release the cached copies when they are no longer needed (possibly writing any modifications back to file storage). The memory requirements of file-based models are minimal, since no more than k paths of the whole suffix tree are kept in memory at any time, where k is the order of the PPM model.

[1] Typically, $|\Sigma| = 256$, corresponding to all 1-byte symbol codes.

3.1.3 Filtering Speed

The training and classification time complexity of our PPM implementation is $O(kn)$, where n is the length of the target document, in characters, and k is the order of the PPM model. Some optimizations further reduce this complexity in practice, following Ukkonen's online suffix tree construction algorithm (Ukkonen, 1995) (specifically, nodes before the "*active point*" of Ukkonen's boundary path are skipped). Unfortunately, Ukkonen's linear time suffix tree construction algorithm cannot be fully applied in our setting, since statistics must be updated also for nodes that are not visited by Ukkonen's algorithm (specifically, this applies to nodes beyond the "*end point*" of Ukkonen's boundary path). Our PPM-based filter requires just under 1.5 hours to complete an online filtering run on the TREC 2005 public corpus (92,189 messages; includes MIME decoding) on a 2.0 GHz Intel Pentium M processor.

3.2 The Probabilistic Sequence Modeling Library

The PPM algorithm submitted for the TREC 2006 evaluation is part of a larger sequence modeling toolkit developed and maintained by the authors (the Probabilistic Sequence Modeling Library - PSMLib). A wrapper around this software is available online as a C++ library.[2] A Python extension module which exposes the same functionality to Python is also available.[3]

4 Results

In this section, we comment on the results of the TREC evaluation that are most relevant to our system. We also present measurements related to the efficiency of our filter in terms of filtering speed and memory consumption. The full official results of the TREC evaluation are reported in the spam track results section of the TREC 2006 proceedings.[4]

4.1 Online Filtering

The performance of our system on the online filtering task was satisfactory. Figure 1 shows the results of the online filtering evaluation with immediate feedback on the TREC public corpus (admittedly, this is a trial in which our filter performed relatively well compared to most other experiments). Overall, the system was among the better performers in all tasks involving immediate feedback, but appears to be more sensitive to delayed feedback than the majority of the other systems. It is not clear whether other participants employed methods to harness the information contained in unlabeled examples, or whether the PPM method is more sensitive to delayed feedback in general.

[2]Available at `http://ai.ijs.si/andrej/psmslib.html`
[3]Available at `http://ai.ijs.si/andrej/psmslib.html`
[4]The TREC proceedings are available online at `http://trec.nist.gov/pubs.html`

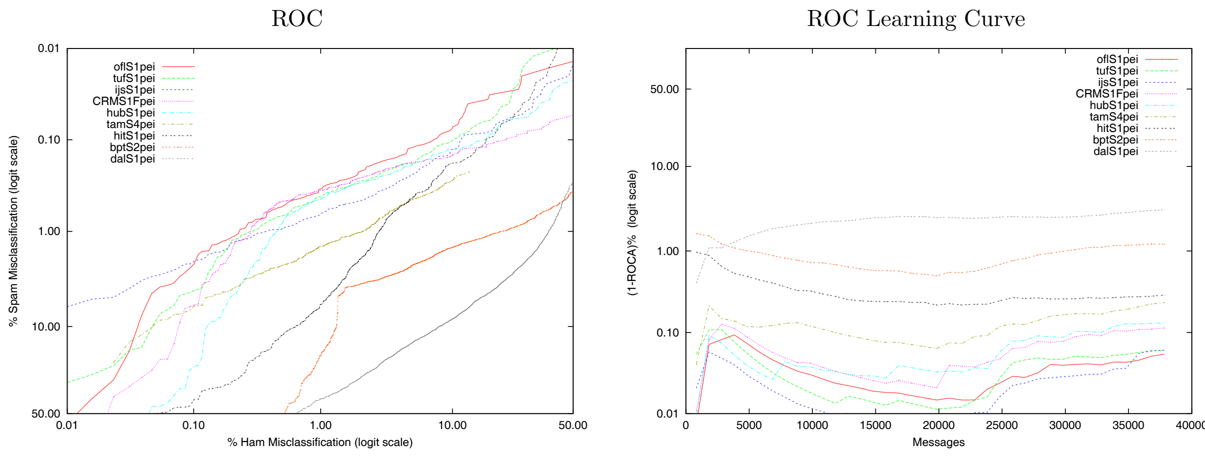

Figure 1: Online filtering performance on the TREC public English corpus (figure reproduced from (Cormack, 2006)). AUC curves (left) and ROC learning curves (right) of the best performing filters of all nine groups that participated in TREC 2006 are shown.

4.2 Active Learning

Our simple active learning strategy worked surprisingly well, and the official results show no indication that our approach was inferior to any of the more sophisticated methods that were used by other participants. Figure 2 shows how filtering performance on the two public corpora relates to the number of training examples when using random sampling and our “most recent first” strategy.

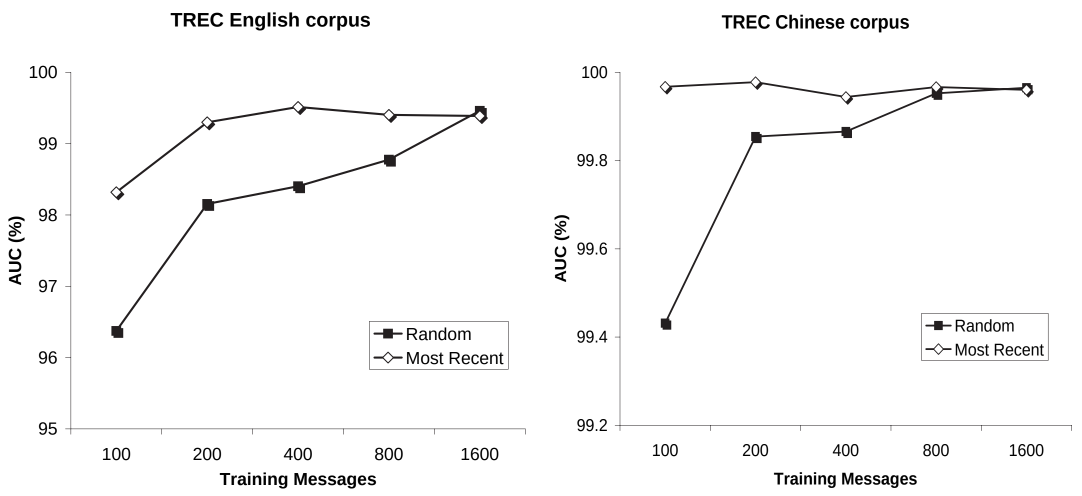

Figure 2: Active learning results on the TREC public English (left) and Chinese (right) corpora. The area under the ROC curve (%) is plotted as a function of the number of training messages for the “most recent first” active learning strategy, and random sampling with an equal number of training examples.

Using the most recent messages for training is clearly beneficial for up to 1600 training messages. On the Chinese corpus, performance levels off at this point, even when additional training data is provided. Similarly, performance cannot be improved significantly by adding training data for the English corpus, however, choosing the most recent examples hurts performance at certain sampling points (not shown in the graph). This is possibly due to an anomaly in the English corpus that is also reflected in the results of the online filtering runs in Figure 1, in which performance of all filters drops after processing the first 25,000 messages.

4.3 Efficiency

Table 1 compares the efficiency of memory-based and disk-based models for online filtering on the SpamAssassin corpus. Although the disk-based version of the filter is more than 30 times slower, the filtering rate of 3.2 messages per second appears to be satisfactory for desktop use. We found that the efficiency of disk-based models drops considerably as the size of the model grows. A better caching mechanism may help improve performance for such large model files.

Storage	*Total time*	*CPU time*	*Peak mem.*	*Peak disk*	*Msg/sec*	*KB/sec*
Memory	56 sec.	55 sec.	43.6 MB	0.0 MB	107.7	242.1
Disk	1892 sec.	1393 sec.	9.0 MB	41.6 MB	3.2	7.2

Table 1: Efficiency of memory-based and file-based adaptive order-6 PPM-D models in online filtering experiments on the SpamAssassin corpus. The corpus contains 6034 messages with a total size of 13,883,063 bytes (messages are truncated to 2500 bytes). Each message is first classified, then trained on. The experiment was performed on a 2.0 GHz Intel Pentium-M processor with 2 GB of main memory and a 7200 RPM portable disk drive with 8 MB of disk cache.

5 Conclusion

Our system exhibited fair performance in the TREC 2006 evaluation, despite the fact that the filtering techniques employed by the IJS system remained virtually unchanged from the previous TREC evaluation. We are satisfied with this result, bearing in mind that the efficiency of our system was improved substantially, bringing compression-based spam filters a step closer to becoming a viable solution for real world spam filtering applications. A very crude estimate is that memory usage was reduced to about a quarter of what was required by our previous system, while filtering speed was improved about two-fold. Further improvements, particularly in terms of memory consumption, would be possible if we were willing to sacrifice some of the flexibility offered by the system (such as, for example, decremental learning).

Unfortunately, the development effort required for technical improvements of our system hindered our research towards possible enhancements in terms of filtering performance. We wish to remedy the lack of progress in this area at TREC 2007.

References

Bratko, A., G. V. Cormack, B. Filipič, T. R. Lynam, and B. Zupan (2006). Spam filtering using statistical data compression models. *Journal of Machine Learning Research 7*(Dec), 2673–2698.

Bratko, A. and B. Filipič (2005, November). Spam filtering using character-level markov models: Experiments for the TREC 2005 Spam Track. In *Proc. 14th Text REtrieval Conference (TREC 2005)*, Gaithersburg, MD.

Cleary, J. G. and I. H. Witten (1984, April). Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications COM-32*(4), 396–402.

Cormack, G. V. (2006, November). TREC 2006 Spam Track overview. In *Proc. 15th Text REtrieval Conference (TREC 2006)*, Gaithersburg, MD.

Cormack, G. V. and A. Bratko (2006, July). Batch and online spam filter comparison. In *Conference on Email and Anti-Spam, CEAS 2006*, Mountain View, CA.

Howard, P. G. (1993). *The Design and Analysis of Efficient Lossless Data Compression Systems.* Ph. D. thesis, Brown University, Providence, Rhode Island.

Sayood, K. (2000). *Introduction to Data Compression*, Chapter 6.2.4. Elsevier.

Ukkonen, E. (1995). On-line construction of suffix trees. *Algorithmica 14*(3), 249–260.

Weiner, P. (1973). Linear pattern matching algorithms. In *14th Annual Symposium on Switching and Automata Theory*, pp. 1–11. IEEE.

Yerazunis, W. *CRM114 Revealed Or How I learned To Stop Worrying and Trust My Automatic Monitoring Systems.*