SERbrainware at TREC 2001

Pál Ruján
SER Technology Deutschland GmbH
Sandweg 236, D-26135 Oldenburg, Germany
e-mails:firstName.lastName@ser.de

Introduction

SER Technology Deutschland GmbH is the technological arm of SER Systems Inc, Herndon, Virginia. Our company focuses on knowledge-enabled software products, mostly related to document management. At the core of many of our products is a knowledge management engine called SERbrainware, which is being developed by our group in Oldenburg since 1999. This engine contains, among others, a text classifier and an associative access module. Both were used in preparing our entries.

This is our first TREC. In order to get acquainted with the usual procedures, evaluation criteria, etc., we decided to participate first in the filtering track. Due to the fact that we had a rather restricted amount of time - two weeks - at our disposition, we used the commercially available engine version 2.40 without any special add-ons.

Data analysis and system characteristics

As always when facing a new problem, one must first analyze the data. As noted also other participants, the 'new' REUTERS data - although rather well prepared in comparison with other data we dealt with in commercial applications - had some problems related to missing text and identical or almost identical texts being classified differently. In general, the documents belong to more than one class, with the classification scheme used for the filtering track being mostly flat with a few exceptions. Although permitted, we did not take advantage of the hierarchy. In our experience, news companies tend to define their own categories through a complete list of boolean expressions whose presence triggers then the class label. The problem with this approach is that it might be strongly biased and actually imply a 'higher order' clarification scheme, masked by the primary statistics of the text. As an example, consider an example set where in one class one has mostly English, on the other class mostly Berman newswire items. Any statistics based classifier will learn to distinguish the two classes based on this *primary* statistics about different languages. However, the task was actually to distinguish between America football vs. Europe soccer items. We are only beginning to work on text classifiers able to deal with such subtle situations.

Using the visualization tools provided by the engine, we could establish that in many cases the subclass structure did indeed correspond to the main clusters seen in three-dimensional projections. We did not use the 'cleaning' tools to improve the learning sets, hence we used all provided training examples. In addition, we did use exclusively the text, no titles or dateline entries of the news items were included. In retrospective, this was a mistake: some other groups at TREC 10 did very well using exactly those informations. Our software did not take advantage of any extra dictionaries, thesauri, or any external information other than the training sets. The runs were made in our usual work environment: PC's with Athlon 800 MHz processors running under Linux OS.

Understanding the tasks

We considered the routing/batch task to be basically a text classification task and used the standard tools of machine learning to do so. Our classifier was run on multiclass mode, not in the one class against the rest mode (which we call one-from-all $\rightarrow$ OFA mode). In multiclass mode our classifier constructs a maximal margin Voronoi tiling in the **space of classes**, which has class-error-correcting abilities. Although this approach works in general very well in text categorization tasks, it turned out to be difficult to tune for optimizing the linear and F-ratio utility functions. This fact will be taken into account for our next software generation. Another possibility - which in our interpretation was excluded by the TREC rules - is to use not only the given set of training documents but also statistics gathered from the full Reuters database. Such an optimization for a given data set is called **transduction** and profits from the extra information gathered about the statistics of unlabeled documents. A simple approximation of the transduction protocol implies that we add to the learning set documents classified with a high degree of precision to belong into one or more classes, followed by a relearning step. In our case this is mostly due to the extension of the document vector space by related class terms not contained in the original training set.

The adaptive filtering track seems particularly challenging also from a theoretical point of view. Consider a situation where there are K classes there and there are a possibly large number of items whose class is 'NOT in this set of classes'. Given some initial class information (or none!), a chronologically sorted stream of news items must be processed according to the following 'card playing' rules. Given an actual item, one can read the text but not its label(s). The actual item is like a card placed face-down.

1. Decide to 'turn-up' an item or not. If an item is missed, recall is diminished by d.

2. To turn a card face up, one must guess first one or more labels.
3. If a label is correct, one gets a reward r. If it is not, one gets a penalty p.
4. All information contained on turned-up cards can be used afterwards.
5. There is a limit of the maximal penalty P one can accumulate (per class). Below it this class is 'lost' completely.

What should be the optimal strategy to win at this game? Assume we have a trainable classifier algorithm, for simplicity let us assume that we use on for each class (OFA-structure). From previous experiments we might know the classifier's learning curve, that is how the average generalization error depends on the number of training examples.

This game is quite interesting, because it is similar to the typical problems a company would face in the market place. You must invest in research Let us now enumerate what kind of information would be desirable:

- The size-horizon: it is necessary to estimate how many items one will have to process. Let denote this number by S.
- The priors: from a total of S items, how many k items belong to class $k \in K$?
- The time-correlations, if any, between occurrences of documents belonging to class k, $k \in K$.

There are three aspects which has to be taken into account in order to make a rational (or almost Bayesian) decision about whether to turn up a new item or not. First, we must get some probability estimate from the classifiers concerning the probabilities that this item belongs to one of the classes of interest. A similar kind of information could be obtained from a prediction based on the time-series of successful hits for each class. There two distinct reasons to turn up a card: either the estimated risk $1 - p_c$ for the suggested label c is smaller than r/p, or $p_c = 0.5$ in which case we could get a very good candidate for a new training document - but must pay accordingly some extra price. The situation is made complex by the existence of a maximal penalty, the fact that the priors of different classes are unequal and because in the beginning there is an additional risk associated with a small amount of available data. It is hoped a full theory will emerge before the submission of TREC 2002 runs. It is perhaps interesting to remark that the filtering track could be seen also as a nice model for a commercial firm, where the costs of research and development are restricted to certain maximal losses and the profit generated by it should in the long run be optimized.

Methods

We used basically two methods: the classifier (serCLST10* runs) and an additional 'experimental' run using the associative access module in a nearest neighbor setting (serASSAT10* runs). The classifier requires disjoint classes. Therefore, some training examples are automatically discarded by the engine when the normalized overlap between two items belonging to different classes was bigger than 0.9. When an unknown document is processed, the classifier returns the confidences that the document belongs to every one of the learned classes. The confidences are real numbers between zero and one. With an appropriate normalization they can be transformed into estimated probabilities. The search engine also returns a score between 0 and 1 for ranking the documents by the similarity to a given query. Therefore, the handling of the two modules was quite similar but used different values for the thresholds.

The associative access module is basically tuned for precision: it has a more sophisticated and precise text representation than the classifier. For performance reasons we restricted the queries to be maximally 1K long (after filtering). This seems not to be a very important restriction for the REUTERS news, since by internal company rules the beginning of each item functions also as a kind of summary of the whole document. The actual learning items (their text content) were stored in the associative cache together with their class and ID information. Each unknown document was posed as a query to the search engine, which then returned a list of the best 100 hits. In general, the class of the best hit was accepted if the score was larger than a certain threshold. In special cases to be described later also the second and third class were accepted (the average number of classes per document is about 1.7). This method has absolutely no problems with overlapping classes: however, its classification capabilities are limited when compared to the classifier.

The batch and routing tracks

It is easy to show that the linear utility function whose optimization has been proposed does not scale correctly with size (or time). Therefore, we used default settings of our software, derived from many practical situations we solved before. This means that for the classifier the threshold was set at 0.8 confidence for the best class and at 0.9 for the second best class when the gap to the next (third best class) was greater than 0.05. Note that all learned documents have confidence 1.0. Classes R42 and R57 were ignored, because they overlapped with R40 and R65, respectively. For the search-engine/nearest neighbor classification we used a threshold of 0.4, which is the default score for our Internet application SERglobalBrain. Here we accepted the second and third best class only if they had practically the same score as the best hit. Note that in principle we could have many good hits belonging to the same class, because we compute the similarity to each training item and not to a class 'center'.

Obviously, the classifier performed much better here than the search-method. Since all our decisions were based on scores (confidences) the routing was just a special case of the batch track. We found these two tracks being a test on classification performance, which in turn, depends crucially on the document representation. We do not know how REUTERS defined its classes: it would be an interesting task to try to find this out from the provided data. Anyway - at least at the time of processing - we can be sure that REUTERS did not used advanced software letting employees immediately know how a certain actual news item was classified by another colleague. We guess that some kind of full text engine with boolean SQL expressions might have been used for guidance.

The adaptive track

The adaptive track seems much more interesting from both a theoretical and practical point of view. Due to the time constraints, however, we have directed our efforts to two experiments: we used the classifier for a very 'precise' run, not allowing for negative utility scores at all. In retrospect, this was a poor decision. The search engine run was directed towards recall. Here are the two algorithms we used:

Run serCLST10af

We start with two documents per class, number of classes is constant. At each iteration there are maximally 10 documents per class in the training-queues.

1. Initialize queues, learn training set (2 documents per class)
2. For each new document perform classification and get list of confidences
   - if best confidence < 0.8 ignore
   - else: ask for confirmation. If class is OK and confidence < 0.85 $\rightarrow$ add to *best* class. If class is false $\rightarrow$ add to *worst* class
3. Add to best class consists of the following procedure:
   Put the new document with label good at the bottom of the queue. If the queue is full, remove the top document. If a 'bad' marked document exist, remove this instead. Relearn the training set.
4. Add to worst class is similar, except that the new document is marked as bad.

Run serASSAT10ad

We start with two documents per class, number of classes is not constant. As explained below, for each class we might build an additional anti-class. The number of documents per class is not controlled.

1. Initialize and learn training set (2 documents per class)
2. For each new document: use document as query, get ranked list of training examples. Let *score* be one of the best three scores.
   - If score $< 0.4 \rightarrow$ ignore.
   - If the best score > 0.4 and the corresponding class is an anti-class, ignore.
   - If $0.4 \leq$ score ≤ 0.55 AND class is correct, add document to class if not already done so. If the class is not correct, count one error but do nothing.
   - If score > 0.55 AND class is incorrect, add document to anti-class set. If class is correct, count a good hit but otherwise do nothing.
3. After each **day** reload the actual set of training documents into associative cache.

Our results in both the routing and adaptive track were on the low-middle range of the spectrum. We did learn a lot from this competition and we are looking forward hope to perform much better next time!