CLARIT Experiments in Batch Filtering:
Term Selection and Threshold Optimization in IR and SVM Filters

David A. Evans, James Shanahan, Norbert Roma, Jeffrey Bennett, Victor Sheftel, Emilia Stoica, Jesse Montgomery, David A. Hull, Waibhav Tembe

Clairvoyance Corporation, Pittsburgh, PA

1. Introduction

The Clairvoyance team participated in the Filtering Track, submitting two runs in the Batch Filtering category. While we have been exploring the question of both topic modeling and ensemble filter construction (as in our previous TREC filtering experiments [5]), we had one distinct objective this year, to explore the viability of monolithic filters in classification-like tasks. This is appropriate to our work, in part, because monolithic filters are a crucial starting point for ensemble filtering, and it is possible for them to contribute substantially in the ensemble approach. Our primary goal in experiments this year, thus, was to explore two issues in monolithic filter construction: (1) term count selection and (2) filter threshold optimization.

In fact, our pre-TREC experiments were conducted in a brief period and we were unable to complete all the tests we had planned. Our official submissions reflect essentially our first, baseline results. They are overall poor in comparison to other results reported this year.

However, an additional focus of our work relates to the general problem of exploiting training data, in particular, where there are only a few positive-example documents for a topic. We regard such cases as more realistic (e.g., in commercial settings) than the categorization-oriented tasks we have seen in TREC filtering in the past, e.g., based on the Reuters collection. Thus, in a series of follow-up experiments, we explored the strengths and limitations of classifier-based approaches (using kernel methods) and CLARIT-IR-based ones on the fifty TREC-2002 "Assessor" Topics.

In our CLARIT-IR-based experiments, we aimed to establish a more accurate baseline than the one reflected in our official submissions. We also sought to vary the term-extraction techniques we used, to optimize performance on a topic-by-topic basis.

In our kernel-based (SVM) experiments, we used non-mathematical (non-QP) based approaches to learning SVMs. We used both NLP-based features and simple white-space-delimited ones; and we developed a preliminary approach to thresholding the classifier margin.

In the following sections we first describe our official submitted runs and results and then present in greater detail the post-TREC experiments that we conducted.

2. Official Batch Filtering Runs

Our official batch filtering runs reflected a straight-forward extraction of term vectors from positive training documents, the setting of thresholds based on calibration of the term vectors over the training data, and the use of the term vectors to score (retrieve/rank) documents in the test collection.

2.1. Preparing Filters and Testing

As a general approach to handling the available training data, we divided the training corpus equally into two parts, one half of which we used for constructing filters (i.e., extracting terms, assigning weights, and determining the optimal term profile cutoff), the other half of which we used for validation (including score-threshold setting). We constructed monolithic filters for each topic automatically, based on the positive examples of each topic. In fact, we used a slightly modified version of the training database: we added two additional, identical positive example "documents" for each topic. These were created by the system from the topic's title and its short and long descriptions, with all meta-language ("Retrieve documents which…," "Find documents that…") automatically removed. We chose to add these artificial documents to increase the number of training documents and to emphasize terms that we anticipated would be especially useful in the filter.

Terms for all topics in this combination of positive examples and artificial documents were generated by CLARIT NLP (yielding morphologically normalized single words, phrases, and sub-phrases), then weighted, ranked, and selected for extraction (to represent the term profile in the filter) using our thesaurus extraction method "Prob2" (as given in Figure 1). We used Prob2 as our default and only term-extraction method based, in part, on our observations of Prob2's overall robust performance compared to other term-extraction methods in our TREC 2001 experiments. While keeping the method of selecting terms for topics constant, we

experimented with optimizing the number of terms for a given topic.

We investigated several techniques for determining how many terms to include in the filter for a given topic. We settled on a method based on the 2nd derivative of a topic's term weight profile (w''). In short, this approach examines a set of terms ranked according to term weight, and disregards all terms occurring after the point where the term weight profile begins to level off. This point is determined by the condition $0 > w'' > \varepsilon$. (In our case, we set ε to 0.01.) In this way, all terms that did not show evidence of being particularly characteristic of a topic (according to their rank in the term weight profile) were disregarded. We applied this method of term count selection to construct filters for each topic. We imposed the additional condition that no topic filter use fewer than five terms. For each of our submitted runs, the average number of terms in a filter was 26, and the maximum was 104.

Once we established which terms (and how many) to use in constructing a filter, it remained for us to determine the threshold for each topic. This was one of the chief issues we wanted to explore, and so we took a different approach for each of our submissions. In our runs CCT11BFC and CCT11BFD, the filter was applied to the entire training corpus. That is, the threshold was optimized over both the first half of the corpus, which we used for constructing the filter, as well as the second half, which had thus far been unused. For CCT11BFC, the filter's threshold was set using the beta-gamma method on normalized linear utility, T11SU, to decrease the likelihood that we would over-fit the training data. (Cf. [13;14] for discussion of the beta-gamma threshold setting method.) For this run, beta-gamma values were 0.1 and 0.4, respectively. To further decrease the possibility of over-fitting the training data, we employed an additional "global threshold multiplier" that relaxed the optimal threshold a bit further. For run CCT11BFC, this global multiplier was set to 0.95.

Our other submission, CCT11BFD, was identical to CCT11BFC, with two exceptions. For this run we employed no beta-gamma regulation at all, and instead lowered the global threshold multiplier to 0.85. Finally, all filters for both CCT11BFC and CCT11BFD were run on the full testing set.

2.2. Official Test Results

Table 1 presents a summary of various batch filtering runs in terms of normalized linear utility (T11SU) and F-Beta. Row one of this table gives the median of all submitted runs from all groups for TREC-2002 batch filtering. The second and third rows summarize the results for our two submitted runs. The remaining rows show results for other unofficial runs we completed, including an Adatron SVM [1;6;11] run.

$$Prob2(t) = \log(R_t + 1)\ x\ \left(\log(\frac{N - R + 2}{N_t - R_t + 1} - 1) - \log(\frac{R + 1}{R_t} - 1) \right)$$

$$Rocchio(t) = IDF(t)\ x\ \frac{\sum_{D \in DocSet} NTF_D(t)}{R}$$

$$RocchioFQ(t) = IDF(t)\ x\ \frac{\sum_{D \in DocSet} TF_D(t)}{R}$$

$$GL2(t) = \frac{4 x R_t x N_t}{(R + N_t)^2}$$

N is the number of documents in the (reference) corpus; N_t is the number of documents in the (reference) corpus that contain term t; R is the number of documents (for training or feedback) that are relevant to the topic; R_t is the number of documents (for training or feedback) that are relevant to the topic and contain term t; TF is the (raw) frequency of term t in a document; and NTF is the normalized frequency of term t in a document.

Figure 1. Term-extraction formulae

Run Description		T11SU	F-Beta
Median for all Submitted Runs		0.316	0.129
Submitted Results	CCT11BFC	**0.186**	**0.147**
	CCT11BFD	0.184	0.145
Unofficial Results	CCT11BFA	0.147	0.130
	CCT11BFB	0.165	0.129
	Adatron	0.328	0.035

Table 1. Results of official and pre-TREC batch experiments (on all 100 topics)

2.3. Observations on Official Runs

On the whole, our official results were unsatisfactory. Our failure to perform well may have been due to several factors. We may have selected terms poorly for various topics, and therefore had a poor characterization of these topics. It was also possible that we chose appropriate terms, but too many or too few of them. Finally, we may have correctly chosen our terms and term counts, and still have performed poorly on various topics due to poor thresholding. We did several analyses to see which of these factors actually was responsible for our weak performance.

As we investigated topics where we performed poorly, we saw little indication that we had grossly erred in our method of choosing which terms to extract from positive examples. Thus, our decision to use a single feature-extraction method (Prob2) that had performed well and robustly in the past does not seem to have harmed our effort significantly.

Additionally, there was little evidence that our term count optimization was faulty. We conducted post-submission experiments where we added terms to (poorly performing) topic profiles in which we had originally used relatively few terms. We likewise did experiments where we removed terms from topic profiles in which we had originally used many terms. In neither case did we see a dramatic change in the performance of filters upon the addition or removal of terms from the profile.

We did see, however, that setting filter thresholds improperly had a remarkable impact upon a filter's performance. In particular, we observed that for many of the topics where we performed poorly, we had set the filter threshold much too low, thereby allowing for the retrieval of many non-relevant documents. Results of our post-submission experiments indicate that the negative effects of poor thresholding far outweigh the positive effects of good term and term count selection.

We saw this principle at work, for example, in Topic 144, *Mountain Climbing Deaths*, which was one of the topics on which we performed extremely poorly. Upon examining the actual terms (and number of terms) in the profile, we could see that they were a fair characterization of the topic. Our recall figure was quite high (0.965—we retrieved 55 of 57 total relevant), and initial precision was quite high: roughly the first third of the documents we retrieved were relevant—a good indication that our terms and term weights were on target. The explanation for such poor performance, then, can be found in our set precision figure (0.026): we retrieved far too many non-relevant documents (2048 out of 2104).

Furthermore, we observed the positive effect that conservative thresholding can have in overcoming the lesser negative effects of poorly chosen terms or term counts. We saw this in some topics where we performed quite well in comparison to the TREC median (T11SU), even though many of the highly ranked documents were not relevant. Our good performance (relative to the median) is likely the result of choosing terms that were at least adequate, and, especially, having a threshold that was conservative enough to prevent over-delivery of non-relevant documents. Topic 122, *Symptoms Parkinson's Disease*, was one topic where we observed this behavior.

The results of our analyses, then, clearly demonstrate that having good terms and term counts is outweighed by setting an improper threshold. On the other hand, accurately choosing a proper threshold helps even in instances where term and term count selection are not especially good. The greater danger lies in using a threshold that is too relaxed rather than setting too conservative a threshold. Thus, our decision to override and *lower* the threshold for each topic set automatically on the training data was the principal cause of our poor overall performance.

3. Post-TREC Experiments: IR-Based Filters

We were naturally interested in assessing the problem of threshold setting in our post-TREC follow-up experiments. In particular, we wanted to establish our baseline performance in threshold setting and to look more closely at the problem of term selection.

We confined our evaluation to the first fifty ("Assessor") topics, because they proved to be the most valuable (and valid) ones in the test suite, and because these topics also seem more realistic than the artificially generated "Intersection" topics. In our subsequent analysis, we report both our post-TREC results and official TREC results on Assessor topics only.

3.1. Revised (Corrected) Term/Threshold Selection

In our first post-TREC experiment, we repeated our basic TREC runs with "normal" threshold setting. That is, we did not force the threshold (set on the training data) to be more relaxed when running on the test collection. This experiment used only the simplest approach to term selection (based on Prob2 extraction and 2nd-derivative term-count selection), threshold calibration on the full training database (using beta-gamma threshold setting), and direct ranking of the test collection. We called this run *Prob2-2D*.

In our second post-TREC experiment, we first split the training data into halves and used one half (including approximately half the positive examples) for candidate term selection and the other half for validation. In this approach, we were interested in trying several different term-extraction methods and predicting which method would give the best terms for each topic. Thus, we used each method (and 2nd-derivative term-count selection) on the positive training documents for a topic in the first half of the training corpus to create a term vector for each topic, and then tested the performance of each vector against the second half of the training corpus. Based on which vector gave the best performance (T11SU score), we chose the term-extraction method used to create that vector as the "best" for that topic. We then repeated the procedure in our first post-TREC experiment, but with the term-extraction method set to the "best" method for each topic. We called this run *Opt-2D*.

The steps in our process are given in Figure 2. Note that the split of the training data into halves (or any other arbitrary proportion) to yield a sub-corpus for topic modeling (term extraction) and a sub-corpus for validation (testing a model), is based on a pseudo-random assignment of documents to one or the other portion. This means that, in the case of some topics, there might be very few positive examples of a topic in any one of the training sub-corpora. A paucity of data can lead to poor training, of course, but we decided not to intervene to insure optimal splits in training data precisely because we wanted to assess, as well, the robustness of our generalized topic-modeling process.

1. Split the training set. First, sort (scramble) the document ids. (For a database with 10 docs, such "scrambling" might produce: 0 2 4 6 8 1 3 5 7 9.) Next, apply the desired Training/Validation split. (For the TREC experiments, the split is 50/50, based on choosing every other document for a split.) Pick one split for Training, one for Validation. (In the TREC experiments, the 2nd half ("odd" documents) was chosen for Training/term-extraction and the 1st half ("even" documents) was chosen for validation/optimization.) The pre-scrambling makes it possible to select any subset with reduced bias. (If one chose a 60/20/20 split for the 10-document collection, above, the system would deliver the subsets "0 3 6 9 1.4", "7 2," and "5 8".)

2. Choose extraction method. If the extraction method is fixed (e.g., Prob2), skip this step and go to Step 3. For each candidate extraction method (e.g., Prob2, Rocchio, RocchioFQ, and GL2), create a filter using the Training half of the training corpus. Optimize the term count by applying the 2nd derivative method. Choose Max(MinimumTermCount, 2ndDerivTermCount). (The MinimumTermCount used in post-TREC experiments is 10.) Truncate the term vector to the specified term count. Set the threshold using beta-gamma optimization (β=0.1,γ=0.4) over the entire training set. Retrieve over the Validation half of the training set (using the optimized threshold) and compute utility (T11SU). Choose the extraction method with the highest score.

3. Extract final filter. Extract terms from the entire training set using the chosen method. Optimize the term count (using 2nd derivative, as described above), subject to the MinimumTermCount. Truncate the vector to that count. Set the threshold on the entire training set using beta-gamma optimization.

Figure 2. Procedure for creating a CLARIT filter profile

The formulae for our term-extraction methods—Prob2, Rocchio, RocchioFQ, and GlobalLocal2 (GL2)—are given in Figure 1. In both experiments, we used the full training corpus as the reference corpus. Processing time for these filters averaged 17 seconds per topic for training and testing *combined*. Filter length for Prob2-2D averaged 33.34 terms and for Opt-2D 15.76.

3.2. Post-TREC Experiment Results

As can be seen from the results in Table 2, both Prob2-2D and Opt-2D clearly out-perform our submitted (official) runs. (The values in Table 2 for our official runs reflect our performance on the Assessor topics only.) Compared to the median reported for the group on Assessor topics, both Prob2-2D and Opt-2D have lower T11SU scores. However, in terms of F-Beta, both post-TREC runs show rather impressive performance.

In our Opt-2D runs, the Prob2 extraction method was chosen only 6 times, whereas Rocchio was chosen 30 times, RocchioFQ 8 times, and GL2 6 times. In terms of individual-topic results, Opt-2D gave significantly better performance (>0.10 absolute difference in score) than Prob2-2D on 10 topics for T11SU and 9 topics for F-Beta. In contrast, Opt-2D was significantly worse on 11 topics for T11SU and on 7 for F-Beta. In the aggregate, however, the effect of term-extraction method optimization appears to be negligible.

Run Description		T11SU	F-Beta
Median for all Submitted Runs		0.377	0.234
Submitted Results	CCT11BFC	0.243	0.259
	CCT11BFD	0.243	0.259
Post-TREC Experiments	Prob2-2D	0.309	0.323
	Opt-2D	**0.315**	**0.326**

Table 2. Results of post-TREC-batch experiments

We note, however, that our choice of an optimum method was based on the performance of a candidate filter on half the training corpus. In those cases where we had poor training splits, our choice was not well informed. Clearly, this is an area for further work.

The overall strong performance on F-Beta for both runs confirms our hypothesis that the basic method we have used is robust and practical. It also confirms that the poor results in our official runs were due to improper threshold setting, in particular, our decision to relax the threshold values that were determined for filters on the training corpus.

4. Post-TREC Experiments: Kernel-Based Filters

In addition to our IR-based runs, we decided to expand our evaluation of kernel techniques for batch filtering in a series of post-TREC experiments. The essential questions we focused on include how well kernel methods perform on topics with limited training data and how flexible the learned thresholds can be when data is sparse. We explored both our kernel-Adatron and a new version of an SMO algorithm.

4.1. General Note on Kernel (SVM) Methods

Support vector machines (SVM) are a general purpose machine learning approach [2;12], with our interest being principally in learning classification models from labeled data. Our batch filtering SVM study was limited to learning a binary SVM classifier for each topic (positive and negative class). This corresponds to searching for (or learning) a hyperplane that provides maximum separation between the positive and negative training examples. Since text classification problems are of high dimensionality (which are generally linearly separable), it is sufficient to search

for this hyperplane in the term/word space, thus avoiding the use of more complex feature spaces that can be induced easily using kernel (non-linear similarity) functions. Hyperplane selection is based upon ideas from statistical learning theory [12], where the hyperplane that is furthest away (has maximum margin) from all training data and that provides (tolerable) class separation is chosen. Large margin separation has been theoretically shown to lead to improved generalization.

More formally, SVM models or classifiers denote a separating hyperplane between two classes, whereby datapoints falling on one side of the hyperplane denote one class and datapoints falling on the other denote the other class. In linear kernel-based SVMs, hyperplanes are typically represented in *primal form* as follows (where <.,.> denotes inner/dot product):

$$Class\ (X) = Sign\left(\langle W, X\rangle + b\right)$$

where W is a weight vector, and b is the bias or threshold. See Figure 3 for a graphic depiction of a hyperplane for a linearly separable dataset. An alternative and more general representation of a hyperplane that is commonly used in SVMs is the following *dual representation*:

$$Class(X) = Sign\left(\sum_{i=1}^{L} \alpha_i y_i \langle X_i, X\rangle + b\right)$$

Here, the alphas (α_i) denote the Lagrange multiplier associated with each example. This representation permits the learning of such classifiers using well-known optimization techniques such as quadratic programming. After learning, only a small percentage of the training data will have non-zero Lagrange multipliers. These examples are known as the *support vectors*. For dot-product (linear) kernels the dual representation of a hyperplane can be mapped to the primal form, thus, yielding a computationally more efficient model, akin to the more traditional information retrieval model. The above dual representation of a hyperplane can be further generalized by considering different forms of the similarity function or kernels such as polynomial, LSI Kernels [4], and String Kernels [9]. For our current purposes of text classification, linear kernels were deemed to be sufficient.

4.2. Learning SVMs

Support vector machines are commonly trained using either mathematical programming (MP) approaches such as quadratic programming or by strategies that avoid the use of the MP techniques. The latter techniques have the added attraction of being easier to implement, while providing similar levels of performance as their MP counterparts. For our experiments, we implemented and evaluated two non-MP based approaches: the kernel-Adatron (KA) algorithm [1;6;11] and variations of the sequential minimal optimization (SMO) algorithm [10;7]. Both of the algorithms are outlined briefly below.

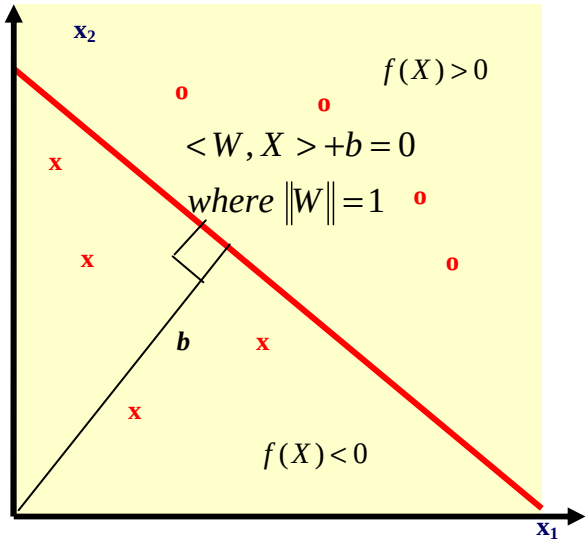

Figure 3. A linearly separable dataset in a two-dimensional space for a two-class problem (where the "o"s correspond to one class, and the "x"s denote the other)

Kernel-Adatron Learning Algorithm. One of the simplest strategies for learning a support vector machine is to update the Lagrange multipliers, α, associated with each example iteratively. This approach has been taken in the kernel-Adatron algorithm proposed by various researchers (cf. [6] and [11]). The Adatron was originally proposed by Anlauf and Biehl [1] in the field of statistical mechanics. It is an on-line learning algorithm for learning perceptrons. In [1], it was proved that the Adatron converges to a maximum margin solution; that is, the discovered hyperplane is a fixed point of the adaptive algorithm for linearly separable data. In [6] and [11], the Adatron algorithm was extended to learn the dual representation of a separating hyperplane in which the dot product is replaced with the more general kernel, thereby expanding the domain of application of the Adatron to non-linear problems. (A simplified version of the pseudo-code for the kernel-Adatron algorithm is presented in Figure 4.) We limited our implementation to training hard-margin SVMs.

SMO Learning Algorithm. The Sequential Minimal Optimization (or SMO) algorithm is an alternative method for training SVMs [10]. Traditionally, training an SVM required the solution of a very large quadratic programming (QP) optimization problem. SMO breaks this large QP problem into a series of the smallest possible QP problems, where only two Lagrange multipliers, α_l, are optimized at each iteration. Since only two parameters are considered at a time, while all others are fixed, it is possible to derive an analytical solution as opposed to the numerical methods used in MP solutions. This avoids using a time-consuming numerical QP optimization as an inner loop in the algorithm. On each iteration, SMO chooses two Lagrange multipliers to optimize jointly (typically the

1. Given Training data S where each example *i* is of the form $(x_{i,1},\ldots, x_{i,n},y_i)$, and a learning rate η
2. Set α vector to zeros; (could use b_o=0)
3. For i = 1 to |Train|

$$z_i = \sum_{i=1}^{|Train|} \alpha_i y_i \langle X_i, X \rangle$$

4. For i = 1 to |Train|
    1. Let $\delta_i = \eta(1-y_i z_i)$ be the proposed change to the multiplier α_i
    2. If $((\alpha_i + \delta_i) \le 0)$ set αi to 0 else $\alpha_i := \alpha_i + \delta_i$
5. (if b used, b=0.5 (min(z_i+) + max(z_i-)) where z_i+ denotes those patterns *i* with class label +1 and z_i- denotes those patterns *i* with class label -1)
6. If maximum number of presentations of the pattern set (epoch) has been exceeded OR (min(z_i+) - max(z_i-)) == 2.0
   then stop otherwise goto step 1

Figure 4. Partial pseudo-code for Kernel-Adatron algorithm

1. Given Training data S where each example *i* is of the form $(X_1,y_1,\ldots, X_n,y_n)$
2. Set α vector to zeros; b_o=0 (the bias term);
3. ExamineAll = true
4. Compute error vector E; $E_i = -y_i$
5. If ExamineAll then //loop thru all examples
   For (int i =0; i < TrainDB.count; i++)
   Set i2 to I; Use heuristics to find a partner i1
   Try to optimize(alpha[i1], alpha[i2])
   Else //loop thru all examples with non-bounded alphas
   For (int i =0; i < TrainDB.count; i++)
   If alpha(i) > 0 and alpha(i) < C
   Set i2 to I; Use heuristics to find a partner i1
   Try to optimize(alpha[i1], alpha[i2])
   If (ExamineAll), then ExamineAll= false
   Else if (NumberOfUpdates == 0), then ExamineAll =1
6. if more alpha updates are possible (i.e., ExamineAll or NumberOfUpdates > 0) Goto step 5

Figure 5. Partial pseudo-code for SMO algorithm

Decision Variable	Explored Values
Learning Algorithm	Adatron, SMO, SVM^{Light}
C (Upper bound for Lagrange multipliers)	3, 10
Learning Rate (Adatron)	0.75
Tolerance	0.001
Type of kernel	Linear
Class Ratio	1:4, 1:10 , use all training data
Sampling Strategy	Random
Term type	NLP; single words
Term Ranking Algorithm	• Use all terms • Mutual Information: Use k terms that have highest MI for each topic
Number of terms k	k = 1,000, 10,000, All
Term weighting	Normalized TF∗IDF

Table 3. Decision variables and explored values for current experiments using SVM text classifiers

examples that have the largest polar error), finds the optimal values for these multipliers analytically, and updates the SVM to reflect the new optimal values. (A simplified version of the pseudo-code for the SMO algorithm is presented in Figure 5.) For our experiments, we implemented two variants of the SMO algorithms proposed by Keerthi et al. [7] that provide better heuristics for determining which pair of Lagrange multipliers to update next and that provide better stopping criteria. For our current study, two variations of the SMO algorithm were implemented and evaluated: SMOK1 and SMOK2, corresponding to modification 1 and modification 2, respectively, as proposed in [7].

4.3. Preprocessing

We examined two representations of documents: one using CLARIT NLP-based (single or multi-word) terms and the other using single white-space-delimited words. The latter approach involved the following steps: replace all numbers and punctuation by spaces; eliminate stopwords such as *articles* and *prepositions,* etc. In both preprocessing approaches each term is associated with a TF∗IDF weight, where *TF* denotes the frequency of a term in a document, and *IDF* is calculated based on the distribution of the term in the training corpus. The TF∗IDF weights were then normalized leading to documents vectors of unit length.

For some of our experiments we chose a subset of terms in the term-space of the training corpus. In such cases, we ranked terms based upon their mutual information with the class label and the *k* terms with highest mutual information were selected to represent each document. The mutual information $MI(x_i, c)$ between a feature, x_i, and a category or topic, c, is defined as follows:

$$MI(x_i, c) = \sum_{x_i \in \{0,1\}} \sum_{c \in \{0,1\}} P(x_i, c) \log \frac{P(x_i, c)}{P(x_i)\,P(c)}$$

Following feature selection, the document vectors were again normalized to unit length.

A learning step follows where for each topic/class/ category a topic-specific binary classifier is learned from the training data that models the topic (positive class) and the *not*-topic (or negative class). While it is possible to learn a topic SVM classifier by using all available training data, it is computationally attractive to reduce the number of training data, especially the number of negative examples. We currently achieve this through random sampling of the negative class, though all explicitly labeled negative documents are used. Typically, given n positive training examples for a topic, we chose $m * n$ negative documents. We explore different values of m in our experiments.

4.4. Experiment Results using SVMs

For each of our experiments, we trained a linear TF∗IDF kernel-based SVM (i.e., linear kernel, where each term is weighted using TF∗IDF) for each topic using the training data. For most machine learning processes, with SVM-based approaches being no exception, there are many parameters and decisions that need to be made in order to generate a model that performs well on unseen data. Some of these are domain specific (e.g., text vs. images), while others are algorithm specific (e.g., the upper bound for Lagrange multipliers, C). The domain-specific decision variables for text include the following: (1) the number of terms used to represent each topic; (2) the number of on-topic training documents; (3) the ratio of positive to negative documents; (4) the sampling strategy for the negative class; and (5) the representation of a document using single words or NLP-based terms. In an ideal setting one could potentially chose the optimal configuration for a topic using, for example, n-fold cross validation. However, due to time limitations, we were unable to carry out such experiments in our post-TREC work. Instead, we report results where the different experiment variables are set to equivalent values across all topics for a particular experiment. The decision variables and explored values for our experiments are presented in Table 3.

The results of the more interesting experiments are presented in Table 4, where each row denotes one experiment on 50 Assessor topics. We report the T11SU and F-Beta measures for each experiment. For our some of our experiments we used different document sampling strategies. One was based upon a sampling of positive to negative documents (denoted as a ratio in Table 3). The other was based on using all labeled documents and fixed sample size of all unlabeled documents in the training size (denoted as an integer in the *Class Ratio* column in Table 4). Experiments on the 50 Intersection topics were not carried out, apart from one experiment, which was performed with the kernel-Adatron algorithm using all NLP-based terms and a 15,000 sample of the training set, yielding a T11SU performance of 0.328 (and 0.342 on the fifty Assessor topics).

For each experiment, training a battery of 50 binary classifiers (one classifier corresponding to each Assessor topic) took approximately twenty minutes (or approximately 24 seconds per topic), while evaluation took approximately two to three hours. The CC SVM toolkit is developed in Java and experiments were carried out under Linux and Windows XP on a 866-MHz Pentium III computer with 1 gigabyte of RAM.

Among the results, the best overall performance was given by an SMOK2 run (*SMOK2-θ.45*) representing one of our first experiments in thresholding the margin scores given by the SVM. In particular, we found the margin that gave optimal T11U utility on a resample of the training corpus, $Margin_{MaxU}$, and used the following formula to compute the new threshold, $\theta_{Opt,}$,, where ThresholdDiscount was set to 0.45 for this run:

$$\theta_{Opt} = ThresholdDiscount * (Margin_{MaxU} + 1) - 1$$

Algorithm	C	Class Ratio	Term Type	# Terms	T11SU	F-Beta
SMOK1	3	4	SingleWds	1000	0.347	0.144
SMOK1	3	10	SingleWds	1000	0.356	0.129
SMOK1	3	4	SingleWds	10000	0.367	0.173
SMOK1	3	10	SingleWds	10000	0.368	0.170
SMOK2	10	14125	NLP	All	0.356	0.077
SMOK2	10	14125	SingleWds	All	0.373	0.142
Adatron	N/A	14125	NLP	All	0.341	0.053
Adatron	N/A	14125	SingleWds	All	0.366	0.149
SMOK2	10	7605	SingleWds	All	0.376	0.147
Adatron	N/A	7605	SingleWds	All	0.363	0.154
SMOK2-θ.45	10	7605	SingleWds	All	**0.408**	**0.271**

Table 4. Results of SVM experiments on 50 Assessor topics

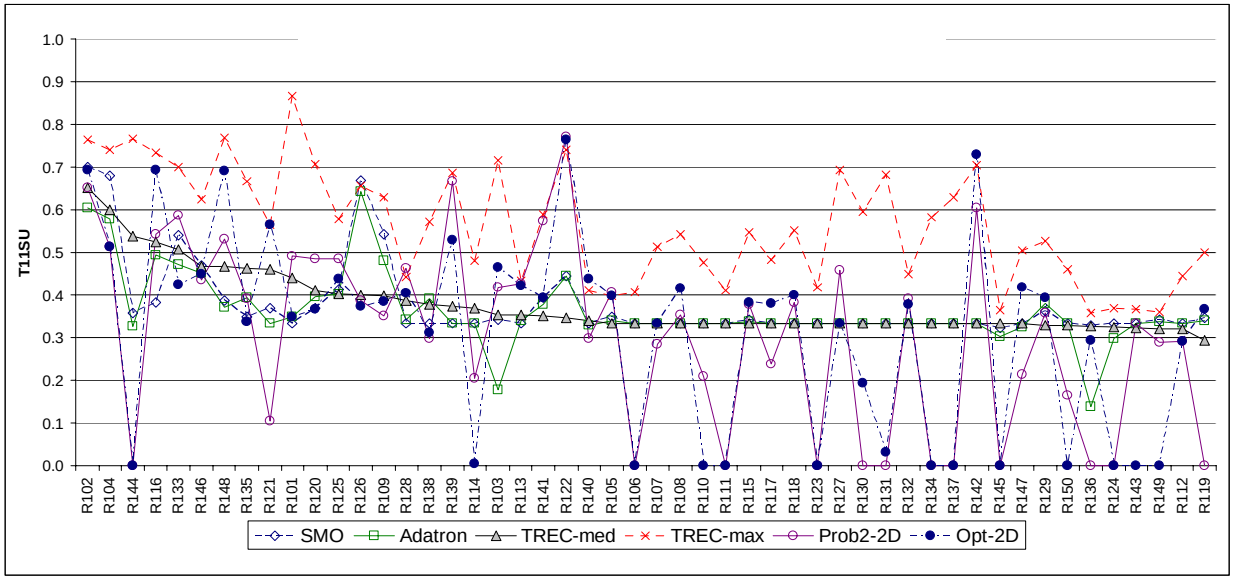

Figure 6. T11SU comparative results for SVM-and IR-filters on topics ranked by TREC median performance

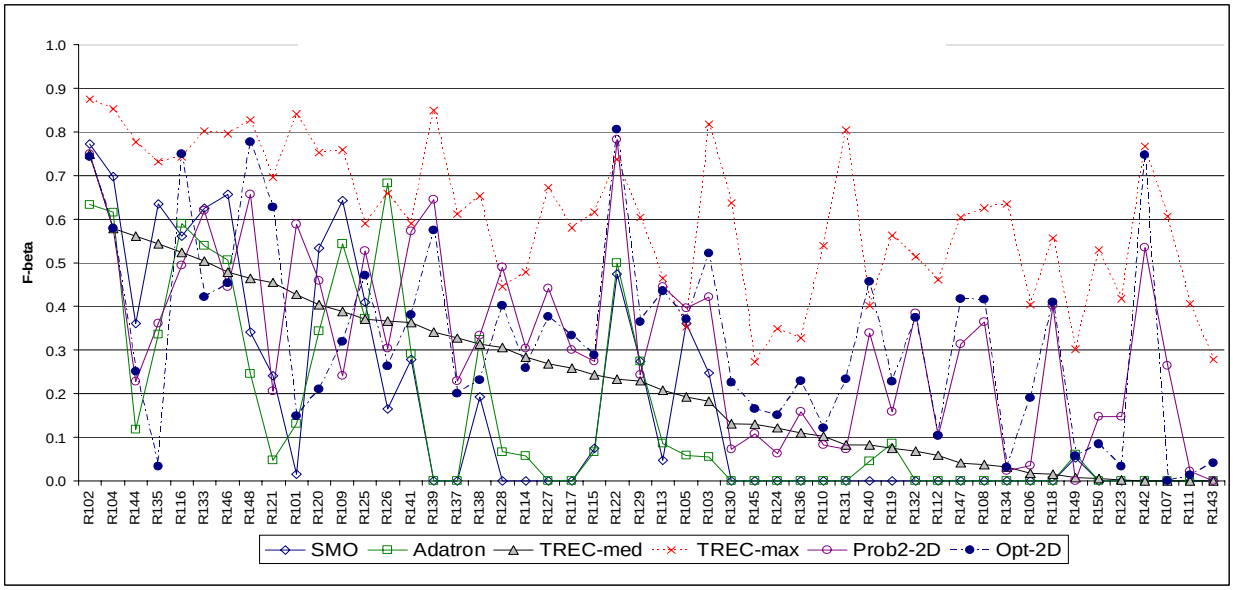

Figure 7. F-Beta comparative results for SVM-and IR-filters on topics ranked by TREC median performance

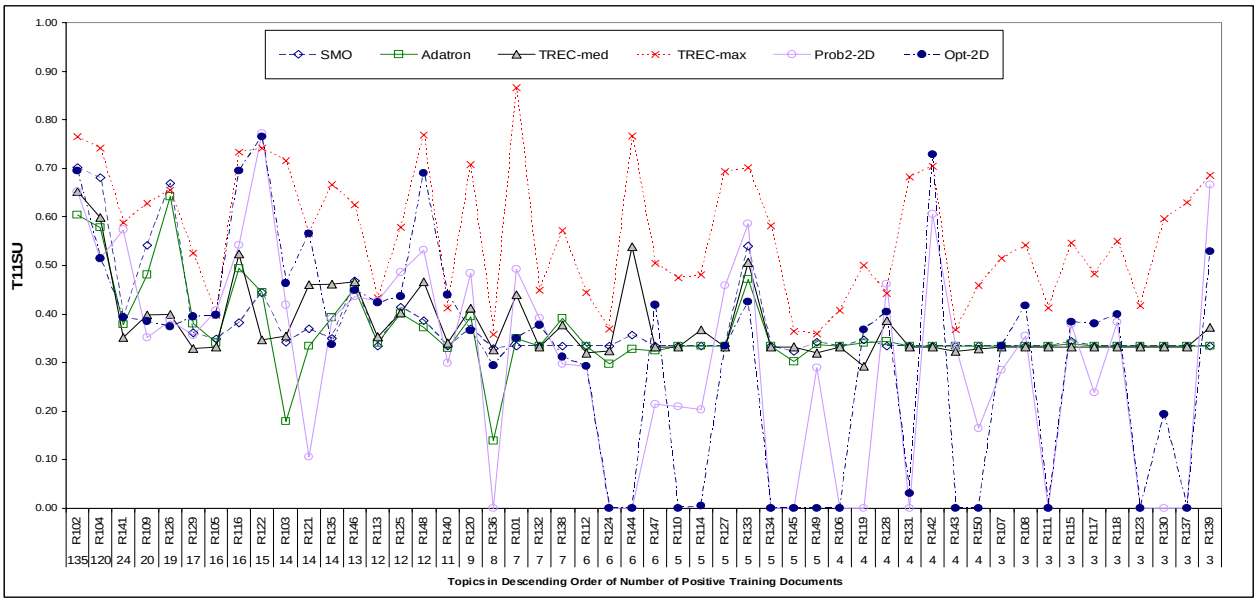

Figure 8. Comparison of SVM-based and IR-based filters: T11SU scores x Topic x Training Data

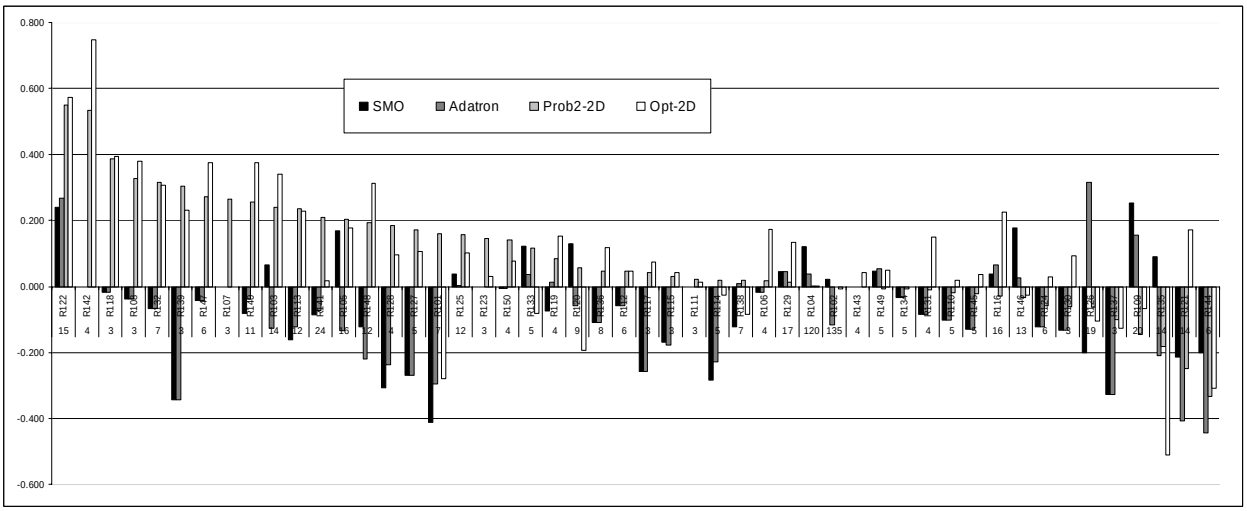

Figure 9. Difference from TREC med F-Beta-scores x Topic x Training Data, ranked by Prob2-2D–TREC med

4.5. Observations on SVM Filters

Overall, the performance of the learnt SVM classifiers is good compared to the submission results for other groups. The T11SU utility measure is average, while the F-Beta measure is low, apart from the SMOK2-θ.45 run with its more reasonable performance of 0.271. The lack of higher performance for the SVMs is partly due to the experimental setup, which did not employ cross-validation; i.e., for each experiment, each topic SVM was trained using the same parameter settings, thereby limiting potential performance of the learnt SVMs. Allowing the determination of a customized setting (potentially optimal) for each topic should lead to improved performance. In addition, our preliminary work on thresholding the margin value of the SVM output has given very encouraging results. This is consistent with results form other groups for this particular dataset [3]. However, for some other datasets in the past, this did not improve performance [8].

Given the results of our limited experiments, we can make the following observations:

- Using a simple tokenizing-based representation of a document (in lieu of NLP) actually boosts performance.
- Sampling negative class documents does degrade evaluation performance, while it improves the efficiency of classification and learning.
- Using all available terms gives the best performance, though sampling terms does not

degrade performance substantially, while it improves the efficiency of classification and learning.

- The kernel-Adatron algorithm gives a very reasonable performance, though it is a much simpler algorithm that the examined SMO variations.
- Using cross-validation for customizing the parameters of learning should improve the quality of the learnt SVM classifiers.
- Our simple thresholding results are encouraging. Using a principled approach to thresholding (such as beta-gamma or other distribution based approaches) may prove practical and effective.

5. Concluding Thoughts

As the additional analyses in Figures 6–9 show, our experiments on the TREC Assessor topics underscore the comparative strengths and differences in the two filter types we have developed. For IR-based filters, we see responsiveness and delivery of relevant documents even when there are limited training data. They also were very fast to train and run and generally required only a handful of features (cf. Figure 10). The IR-based filters failed to return relevant documents in only one case out of fifty topics. However, a measure that rewards the delivery of *no* documents, such as T11SU, penalizes the IR-based approach. In contrast, we see consistent positive (or neutral) performance from SVM-based filters, giving high precision, if under-delivery, on unseen data. However, for many of the SVM runs (on on more than twenty topics) there were no documents returned at all.

The challenge in many practical (commercial) applications is limited training data and the need to optimize performance in virtually real time.
Some of the best methods for classifier training, such as kernel-based approaches, require significant amounts of data and may depend on sensitive parameter tuning. However, we see in our own experiments that kernel methods can give high precision and accuracy. If we can overcome their high-precision bias using thresholding or uneven-margin-based learning and adapt them to sparse data, they may become an attractive solution. We also see the robustness and generally good performance of IR-based approaches. Perhaps the ideal application will combine features of both and optimize the choice of classifier—IR or SVM—for each topic on a case by case basis

convict child rapist (40.4)	child rapist (39.6)
convict child rapist marc dutroux (39)	eefje (38.1)
rapist marc dutroux lambrecks (37.6)	marchal (38.1)
child rapist marc dutroux (37.6)	lambrecks (37.6)
eefje lambrecks (37)	dardenne (37.3)

Figure 10. CLARIT terms/weights for topic 103

References

[1] Anlauf JK, Biehl M, The adatron: an adaptive perceptron algorithm. *Europhys. Letters*, 10, 1989, 687–692.

[2] Boser BE, Guyon IM, Vapnik VN, A training algorithm for optimal margin classifiers. In D. Haussler, editor, *5th Annual ACM Workshop on COLT*. Pittsburgh, PA: ACM Press, 1992, 144–152.

[3] Cancedda N, Cesa-Bianchi N, Conconi A, Gentile C, Goutte C, Li Y, Renders JM, Shawe-Taylor J, Vinokourov A, Kernel Methods for Document Filtering. *TREC 2002 Notebook Papers*, 2002.

[4] Cristianini N, Lodhi H, Shawe-TaylorJ, Latent Semantic Kernels. *Journal of Intelligent Information Systems* (JJIS) Vol. 18, No. 2, 2002.

[5] Evans DA, Shanahan JG, Tong X, Roma N, Stoica E, Sheftel V, Montogomery J, Bennett J, Fujita S, Grefenstette G, Topic-Specific Optimization and Structuring. A Report on CLARIT TREC-2001 Experiments. In EM Voorhees and DK Harman (Editors)*, The Tenth Text REtrieval Conference (TREC-2001).* NIST Special Publication 500-250. Washington, DC: U.S. Government Printing Office, 2002, 132–141.

[6] Frieß T-T, Cristianini N, Campbell C, The kernel adatron algorithm: A fast and simple learning procedure for support vector machines. *15th Intl. Conf. Machine Learning*. Morgan Kaufmann Publishers, 1998.

[7] Keerthi SS, Shevade SK, Bhattacharyya C, Murthy KRK, *Improvements to Platt's SMO algorithm for SVM classifier design*. Technical report, Dept of CSA, IISc, Bangalore, India, 1999.

[8] Lee K-S, Oh J-H, Huang JX, Kim J-H, Choi K-S, TREC-9 Experiments at KAIST: QA, CLIR and Batch Filtering. *TREC Proceedings 2000,* 300*ff.*

[9] Lodhi H, Shawe-Taylor J, Christianini N, Watkins C, Text classication using string kernels. *Advances in Neural Information Processing Systems,* 13, 2001.

[10] Platt J, Fast Training of Support Vector Machines using Sequential Minimal Optimization, in *Advances in Kernel Methods - Support Vector Learning*, Schölkopf B, Burges C, Smola A, eds., MIT Press, 1998.

[11] Santamaria J, Pantaleon C, Principe JC, Minimising BER in DFE with the Adatron Algorithm, *Neural Networks for Signal Processing XI (NNSP 2001)*, Falmouth, MA, 2001, 423–432.

[12] Vapnik V, *The Nature of Statistical Learning Theory.* New York, NY: Springer, 1995.

[13] Zhai C, Jansen P, Stoica E, Grot N, Evans DA, Threshold Calibration in CLARIT Adaptive Filtering. In EM Voorhees and DK Harman (Editors)*, The Seventh Text REtrieval Conference (TREC-7).* NIST Special Publication 500-242. Washington, DC: U.S. Government Printing Office, 1999, 149–156.

[14] Zhai C, Jansen P, Roma N, Stoica E, Evans DA., Optimization in CLARIT TREC-8 Adaptive Filtering. In EM Voorhees and DK Harman (Editors)*, The Eighth Text REtrieval Conference (TREC-8).* NIST Special Publication 500-246. Washington, DC: U.S. Government Printing Office, 2000, 253–258.