Topic-Specific Optimization and Structuring

A Report on CLARIT TREC-2001 Experiments

David A. Evans, James Shanahan, Xiang Tong, Norbert Roma, Emilia Stoica, Victor Sheftel, Jesse Montgomery, Jeffrey Bennett, Sumio Fujita*, Gregory Grefenstette

Clairvoyance Corporation, Pittsburgh, PA & *Justsystem Corporation, Tokushima, Japan

1. Introduction

The Clairvoyance team participated in the Filtering Track, submitting the maximum number of runs in each of the filtering categories: Adaptive, Batch, and Routing. We had two distinct goals this year: (1) to establish the generalizability of our approach to adaptive filtering and (2) to experiment with relatively more "radical" approaches to batch filtering using ensembles of filters. Our routing runs served principally to establish an internal basis for comparisons in performance to adaptive and batch efforts and are not discussed in this report.

2. Adaptive Filtering

In previous TREC work (TREC 7 & 8), we developed an approach to adaptive filtering that proved to be robust and reasonably effective, as evidenced by the relatively strong performance of our systems [1,2]. For TREC 2001 we sought to assess the generalizability of the approach, given especially the differences this year in (a) the amount and nature of the training data and (b) the inherently "classification"-oriented (vs. "query"-oriented) task. Indeed, additional differences, such as the large numbers of expected "hits" in the test set, contributed to the special character of this year's task.

The CLARIT Filtering system is based on core CLARIT retrieval technology. In brief, the CLARIT approach uses text structures such as noun phrases, sub-phrases, and morphologically normalized words, as features or terms to represent text (or passage) or topic (query) content. Terms, in turn, are weighted based on document and corpus statistics (such as IDF and TF), and additionally can have independent coefficients to adjust weights according to processing requirements (such as updates). Information objects are modeled as vectors in a high-dimensional vector space; the Euclidean inner product gives the distance (or closeness) measure (document score) in the space. The system also has a variety of thesaurus (term-extraction) algorithms; these are used to identify characterizing terms for a document or set of documents (e.g., the set of "relevant" documents associated with a topic).

In addition to core processing, our adaptive-filtering system has several parameters, including (i) the number and type of features used to create a topic profile, (ii) the score/threshold setting, (iii) the frequency of setting updates (driven by feedback), (iv) the selection and number of (new) features added at updates, (v) the resetting of score thresholds, and (vi) the number of documents retained over time as a basis for modeling the topic (historical reference statistics and aging).

For this year's TREC adaptive filtering task, we used the same system that was used for our TREC-8 experiments [2]. However, for threshold setting and updating we further experimented with our *beta-gamma* adaptive threshold-regulation method. The method selects a threshold by interpolating between an "optimal" threshold and "zero" threshold for a specified utility function. This method can be applied both to training or sample document sets, as well as to documents that have been returned and judged during actual filtering.

The optimal threshold is the threshold that yields the highest utility over the training or accumulated reference data. Operationally, this threshold is determined by using the topic profile as a query over the reference (judged) documents to score and rank them based on their features (terms). Additionally, based on the utility function for the filter, a cumulative utility score is calculated at each rank point in ascending order. Typically, the cumulative utility score at each rank point manifests a well-behaved trend: it ascends, reaches a peak value, and descends again, eventually turning negative (as the remaining documents are mostly non-relevant). The feature score on the document at the lowest rank point where the cumulative utility score reaches its maximum is taken as the optimal threshold. The zero threshold is determined by the score on the document at the highest rank point below the optimal threshold that has a cumulative utility of zero or less.

The two parameters, *beta* and *gamma,* are used to determine the feature score—between the optimal threshold and the zero threshold—that will be used as the actual threshold for the filter. Beta attempts to account for the inherent or systematic (i.e. sampling) bias in optimal threshold calculation. Gamma makes the thresholding algorithm sensitive to the number of documents processed. The inverse (1/gamma) expresses the number of documents needed to gain reasonable confidence in the value of the score threshold (apart from the bias already accounted for through beta). The parameters are

$$\theta = \alpha\theta_{zero} + (1-\alpha)\theta_{optimal}$$

$$\alpha = \beta + (1-\beta)e^{-n\gamma}$$

where

θ is the filter threshold

n is the number of positive examples

Figure 1. Beta-Gamma Threshold Regulation

determined empirically. See Figure 1 for the formulae that use beta and gamma to calculate a filter-threshold score value.

2.1. Pre-Test Experiments and Calibration

We chose to recalibrate all parameter settings by running the system again on the tasks for TREC-8 and TREC-9 Filtering—the latter task to better approximate the classification-oriented features of the Reuters data. (In particular, we did no adaptive-filtering calibrations on any Reuters data (1987 or 1996), given our desire to assess unbiased system performance.) Our results for these preliminary tasks were quite good (actually better than any of the reported results in TREC 8 and equal to the best results in TREC 9).

2.2. Test Configuration

Our approach to testing included the following elements.

- Preprocessing: All documents, including testing documents, training documents, and topic descriptions, were pre-indexed using all single nouns, single words occurring in noun phrases, and two-word noun phrases, as recognized by the CLARIT parser.
- IDF Statistics: The IDF statistics were collected from all the training data. We did not update initial term statistics in the process of filtering as our past experiments indicate that doing so does not seem to have as much impact on the overall filtering performance as other factors.
- Term Weighting and Scoring: We used the BM25 TF formula for TF-IDF weighting; the average document length was set to 1,000. We used dot product scoring for matching documents with profiles.
- Initial Profile Term Vector: An initial profile vector was built from the original topic description and trained using the two training examples for each topic by adding 20 terms to the original profiles with coefficient of 0.1. We used a delivery-ratio estimation method to set the initial score threshold and set both *gamma* and *beta* to 0.1 for use in processing test data.
- Term Vector Updating: We used Rocchio term vector learning, but only positive examples were used to expand the profile. A centroid vector accumulator was updated whenever a profile accepted a relevant document. The top *K* terms with the highest scores were selected from the centroid and added to the original profile with a uniform coefficient. The vector was updated when a specified number of documents had been delivered since the last update or when the profile had not been updated for a time interval measured by 3,000 documents in the test stream.
- Threshold Updating: We used the same beta-gamma threshold-regulation algorithm as in TREC-8. To emphasize recent documents, we discarded any documents in the cached set of scored documents for each profile that were older than 30,000 documents, provided the cached set did not fall below a minimum of 1,000 documents. The cached documents included both true- and false-positive examples. At any point when a false-positive document scored below a "reference threshold"—equal to half of the then-current real threshold for the profile—the document was discarded.

Run	Ts Added	Coeff.	Beta	Gamma	Interval
CL01afa	20	0.10	0.10	0.10	2
CL01afb	200	0.25	0.25	0.05	2
CL01afc	200	0.25	0.25	0.05	4
CL01afd	20	0.10	0.10	0.10	4

Table 1. Configurations for Adaptive Filtering Runs

	CLT10AFA	CLT10AFB	CLT10AFC	CLT10AFD
T10U	163.7	160.4	172.9	221.8
T10SU	0.054	0.051	0.05	0.051
T10F	0.081	0.075	0.07	0.078

Table 2. Official Adaptive Filtering Test Results

	CLT10AFA	CLT10AFB	CLT10AFC	CLT10AFD
Best	3	1	1	2
> Median	13	17	16	12
Median	10	9	10	14
< Median	56	55	55	54
Worst	2	2	2	2

Table 3. Comparative T10SU Results: Number of Topics Scoring at Various Ranks

For the official TREC-2001 submission, we used the best parameter settings we discovered in our preliminary experiments (on TREC-8 and TREC-9 data). In particular, we varied only two parameters—the number of terms added at each update and slight differences in threshold convergence rates—to create four different submissions, with configurations as given in Table 1, optimized for linear utility T10U.

2.3. Test Results

Table 2 gives official results for our submitted runs and Table 3 gives comparative results. Our four official runs have similar performance. Our results were good from the point of view of "conservative" filtering (and delivery of information); we achieved an average utility of 222 for our best run, with only 30 topics scoring slight negative utility (the average of these being –4.37 and the maximum –12). However, in the context of the TREC task and the Reuters data, this is poor-to-mediocre performance.

2.4. Observations

It seems clear in retrospect that the principal problem in the system was the setting of rather high thresholds (scores), resulting in the delivery of too few documents, especially in the first stages of filtering.

In our system, the initial threshold setting is determined, in part, by the expected "delivery ratio" or density of relevant documents expected in the stream of data to be processed. In particular, before any filtering can occur, a score threshold must be established based on the available information about the topic. Two example documents alone do not constitute a sufficiently representative sample of documents for effective beta-gamma regulation. Instead, we employ a reference

collection (in this case, the Reuters 1996 training documents) as a target corpus. In practice, we use the topic profile (based on terms extracted from the topic description and the two example documents) as a query to score and rank the reference documents. Note that we examine none of the documents in the reference corpus in this process and neither make nor require any information about the relevance of individual documents. We merely use the documents of the collection as an empirical test of the scoring potential of the topic profile. After scoring, we identify the rank point that corresponds to the expected ratio of relevant documents for the collection. The score on that document is used as the initial score threshold for the filter. As a concrete example, if we project the delivery ratio to be 1-in-1,000 and we have 10,000 documents in the reference collection, we would use the score on the document at rank 10 as the initial score for the filter threshold.

Given that we calibrated on TREC-8 and TREC-9 tasks, where observed delivery ratios average approximately 1-in-10,000 (TREC-9 = 0.000173 and TREC-8 = 0.00019), we began the TREC-2001 task with default assumptions of delivery that were far out of line with the actual density of topics in the Reuters 1996 Test Collection. In fact, the average density of topics in Reuters is approximately 1-in-100 (0.0125), nearly two orders of magnitude greater than in the collections we have seen in previous TREC tasks.

This discrepancy between our initial expectations (and the only ones that we might legitimately make) and the actual topic density in Reuters is an immediate source of error in our processing. It might underscore one criticism of the Reuters collection—or at least the use of Reuters subject categories in that collection—as a test bed for adaptive filtering, namely, that such "topics" with such high densities are poor representatives of real-world adaptive filtering tasks.

This problem in delivery-ratio expectations can also be regarded as an indication of a flaw in the user model we (as a group) have adopted for TREC adaptive filtering. In that model, we assume that a simple utility function—balancing the value of true versus false positives, and possibly taking into account false negatives—can represent the target outcome of a process. It is clear, however, that some expectations of delivery are also critical and are very likely a part of any user's set of expectations on filter performance.

Note, it is possible to criticize a system that requires a delivery-ratio setting to perform well in contrast to one that does not. Any system that can perform well without such a setting is to be preferred to one that cannot—*ceteris paribus*, by Occam's Razor alone. However, it is not clear that any of the more successful adaptive filtering systems that participated in TREC 2001 experiments are such systems. In fact, these better systems seem to have modeled the delivery ratio quite accurately. One wonders how such a model might have been developed on the basis of the topic statement and two sample documents alone. Of course, it is possible that such systems were simply initialized with expectations of 1- or 2-in-100 documents as candidate density. If so, these were lucky choices, indeed. And, of course, if these were just good guesses, it still remains to determine how such good guessing might be ensured, in principal, in filtering over other streams/collections, such as the ones we saw in TREC-8 and TREC-9 tasks, or such as occur in real-world applications, where information about expected density of a topic in the possibly many data streams that are accessed is not available.

	CLT10F01	CLT10F02	CLT10F03	CLT10F04
Del-Ratio	0.025	0.025	0.025	0.025
Beta	0.15	0.15	0.15	0.15
Gamma	0.01	0.01	0.005	0.005
Update	9	9	9	9
Mu	1.0	0.9	1.0	0.9
T10U	1716.8	1678.9	1714.9	1679.6
T10SU	0.1003	0.0843	0.0983	0.0813
T10F	0.1980	0.2097	0.2057	0.2148

Table 4. Results of Post-TREC Adaptive-Filtering Experiments

2.5. Follow-Up Experiments

Recognizing that our system suffered from the inappropriate expectations of density we used, we decided to re-run the experiments with explicitly different delivery-ratio settings. In particular, we wanted to assess the inherent strength (or weakness) of the system without the artificial constraint imposed by inappropriate delivery-ratio assumptions.

In a set of follow-up experiments, we re-set a variety of parameters to accommodate the special conditions of Reuters topics. We used a delivery-ratio expectation of 2.5% (0.025) to model the relatively frequent occurrence of topics. This was designed to insure that we would commence filtering with a lower expected score threshold. But given the extraordinarily high ratios of relevant documents for many topics, we might well find the lowered thresholds to be still too high. We hypothesize that, when we expect high density of a topic in a stream, we should expect any small number of sample documents (e.g., 2) to be extremely under-representative of the topic and to create a high-score bias. This is because features (terms) extracted from such non-representative documents will emphasize the distinct characteristics of those documents and will tend to select and score highly only the small subset of similar documents that share their biases. In such cases, we should depress the lower-bound score further, at least until we have achieve a feedback sample of sufficient size to insure that topic-representation biases are minimized.

As a test of this hypothesis we used lowered beta and gamma values to retard the convergence on a stable, high (optimal) threshold score. (Note that a beta = 0 would essentially deliver any document that matched on any of the features in the profile.) We also delayed the profile updates until we had accumulated sufficient judgments to yield nine true positives (along with any false positives that also were delivered in the interval). And, finally, we introduced a new parameter, *mu*, to serve as a coefficient on the filter threshold. For $0 < mu < 1$, this effectively further lowers the threshold to allow more documents to be delivered for judgment.

The results of these follow-up experiments, given in Table 4 for the Runs labeled CLT10F01–04, demonstrate immediate, dramatic improvements. Compared to the official runs (Table 2), the improvement in performance is nearly 100% for T10SU, 250%+ for T10F, and approaching 800% for T10U. Note that these results do not reflect the effect of different term selection (or numbers of terms selected), rather derive only from (1) assuming a more appropriate delivery ratio, (2) lowering the rate of convergence on an "optimal" utility point, (3) postponing updates, and (4) further reducing the threshold.

Still, the results are sub-optimal and not at the level of the best-performing systems. We suspect that several factors are interacting to limit performance, including the fact that our core process is geared to retrieval performance and not classification. Thus, we did not model negative or border cases explicitly in developing topic profiles. In addition, the Reuters topics are quite vague and in some cases diffuse, in the sense of having a variety of sub-topics. We believe that such cases are best treated with complex filters, not simple ones, capable of modeling the topic structure directly. We offer more specific thoughts on this point in the following section, in our discussion of topic-specific optimization strategies in batch filtering.

3. Batch Filtering

Traditional information retrieval approaches to batch filtering have tended to represent a category or topic using a single or monolithic filter (model) that is extracted from positive examples of the category. However, both empirical and theoretical studies in other fields such as machine learning have shown that using multiple models or ensembles of models can lead to improved performance given some weak assumptions about the constituent models [3,4,5,6]. Hansen and Salamon [3] proved that, given an ensemble of models in which the error rate of each constituent model is better than random and where each constituent model makes errors completely independent of any other, the expected ensemble error decreases monotonically with the number of constituent models. As examples of these theoretical claims, empirical studies in the field of machine learning have shown that, when weak or unstable learning algorithms, such as C4.5, are used in conjunction with ensemble techniques, the performance of these approaches can be improved significantly [4,8].

The improved performance gained from using ensemble approaches can be attributed to avoiding risks that arise from using a single model. These risks can be statistical in nature, where more than one statistical solution exists (stability). They can be algorithmic in nature, e.g., with high risk of getting stuck in local minima models. They can be representational in nature, e.g., when the space of representable models is infinite. In addition, some concepts can be very diverse and can be more accurately modeled using multiple models. Though the use of ensemble models is a relatively new, active, and very promising field of research in machine learning, very little work in information retrieval has incorporated the notion of ensemble models.

Our TREC-2001 experiments were designed explicitly to explore some of the issues in the use of ensemble filters for batch filtering. The arguments for using complex (non-unitary) filters are intuitively compelling. We recognize (a) that no single term-selection method works uniformly well for all topics and (b) that some topics are best modeled as "dispersed"—not based on a single set of features, but possibly a family of distinct sub-features. This would seem to suggest that multiple representations (hence, multiple filters) are needed. Thus, we created an approach that optimized filters on a topic-by-topic basis according to feature extraction method and filter structure. In particular, in this heterogeneous approach, filters for each topic were unique: each topic's features were derived by one of five different feature extraction techniques and each was modeled by either (i) a single (monolithic) filter, or (ii) a family of four, parallel (multiplexed) filters, or (iii) a set of *n* (cascaded) filters sequenced so that each filter after the first considered only the fallout (below-threshold-scoring) documents of the preceding filter.

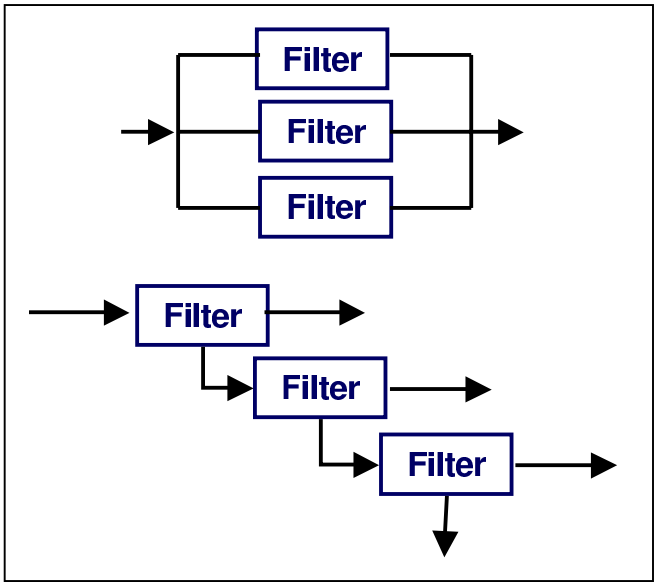

Figure 2. Schematic Representation of a Multiplex (Parallel) and Cascade (Sequential) Filter

3.1 General Description of Ensemble Batch Filtering

Ensemble filtering explores the general idea of constructing many weak or focused filters and combining these into a single highly accurate filter (using, for example, voting) in order to filter or classify an unlabeled document. Ensemble filters can be constructed and combined using various techniques that have been proposed and empirically demonstrated in the fields of machine learning and statistics. Construction approaches vary widely but can generally be placed into three broad categories: data-related methods (such as bagging and boosting); representation-based methods (such as constructive induction and alternative representations of the output space, such as error correcting output codes); and approaches that differ based upon the hypothesis search strategies employed. When it comes to aggregating the constituent filters of an ensemble, various strategies can be used, such as voting strategies as in multiplexing, a cascade (or waterfall) aggregation strategy, or aggregation strategies that are learned, as in stacked generalization [9].

For TREC 2001, we limited our exploration of ensemble filters to multiplex and cascade filters, illustrated schematically in Figure 2. Due to time and system limitations, we used simplified versions of the bagging and boosting algorithms, both of which generate component filters based upon different training data sampling procedures, to construct multiplex and cascade filters respectively.

A *multiplex filter* is a filter made up of constituent filters F_i, where the multiplex filter accepts the unlabeled document (and classifies it as positive) based on some interpretation of the independent scoring of each constituent filter F_i. In fact, there are many possible, alternative methods for interpreting the score of a set of filters, ranging from some simple combination of binary outcomes (e.g., the sum of the "votes" of each filter) to a weighted, possibly non-independent scoring based on the interaction of filters. In our experiments, we chose the simplest approach and accepted a document if the document was accepted by any one of the constituent filters.

On the other hand, a *cascade filter* is an ensemble filter that consists of an ordered list of filters $\{F_1, \ldots, F_n\}$, where each filter, F_i, consists of two outputs: one corresponding to the positive class and the other corresponding to the negative or fallout class. Each constituent filter F_i is linked to the fallout class of the filter F_{i-1}. An unlabeled document is processed by each filter F_i in left-to-right fashion. Should any filter accept the document, processing terminates and the unlabeled document is accepted by the ensemble filter. Otherwise, the subsequent filter F_{i+1} processes the unlabeled document in a similar fashion. This process repeats until either some constituent filter has accepted the document or no filter has.

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

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

$$Prob1(t) = \log\left(\frac{N - R + 2}{N_t - R_t + 1} - 1\right) - \log\left(\frac{R + 1}{R_t} - 1\right)$$

$$Prob2(t) = \log(R_t + 1) * Prob1(t)$$

Where,

$$IDF(t) = 1 + \log \frac{N}{N_t}$$

$TF_d(t)$ = Frequency of term t or how many times the term appears in document d

N = Number of documents in the reference corpus

N_t = Number of documents in the reference corpus that contain term t

R = Number of relevant documents

R_t = Number of relevant documents containing term t

Figure 3. Term-Extraction Formulae

Method X Terms	CC	Roc	RocFQ	Prob1	Prob2
10	X	X	X	X	X
25	X	X	X	X	X
30	X				X
50	X	X	X	X	X
80	X	X	X	X	X
100	X	X	X	X	X
120	X	X	X	X	X
150	X	X	X	X	X
180	X				X
200	X	X	X	X	X
260	X				X
300	X	X	X	X	X
340	X				X
400	X	X	X	X	X
450	X				X
500	X				X

Table 5. Term Count and Extraction Method Combinations

In the case of ensemble filters, we used two different construction approaches for our TREC-2001 submitted runs (both of which are outlined below): one based upon the iterative modeling of fallout examples, which is a simplified version of boosting; the other based upon cross-validation, which is a simplified version of bagging. We used n-fold cross-validation [7] to choose the construction and aggregation method and make other representational decisions, such as which of several term-extraction methods and term counts to use.

3.1.1. Cross-Validation to Construct Monolithic Filters

For ease of presentation, prior to describing ensemble filter construction algorithms, we review how cross-validation was used to construct monolithic filters. Monolithic filters served as our baseline submission. The presentation is made more concrete by using the TREC-2001 filtering problem—the Reuters 1996 dataset.

For all submissions, the training corpus was divided into four folds. More specifically, the Reuters 1996 training corpus of twelve days was partitioned into four subsets denoted by *Q1, Q2, Q3,* and *Q4*, where each quarter consisted of a non-overlapping sequential sampling of a subset of the full training dataset.

Monolithic filters were constructed using either (a) the topic descriptions alone, which we subsequently refer to as "topic filters," or (b) the training corpus. To construct topic filters, we extracted filter terms from the topic descriptions and set thresholds using a beta-gamma optimization on three quarters of the data, while the unseen quarter was used as a blind test. This led to a utility measure for the test quarter. This experiment was repeated for each quarter, thereby generating four utility measures U_1, U_2, U_3, U_4. The average was taken of these four utility measures resulting in a utility measure, *AvgTopicU*, for the corresponding topic filter.

On the other hand, when constructing monolithic filters from training examples, we examined the results of various thesaurus extraction methods and corresponding term counts and chose an optimal filter representation based upon its cross validation performance. See Table 5

for a list of all combinations of thesaurus extraction methods and term counts that were examined for our TREC-2001 submissions. Note that the label "CC" in the Table stands for "CLARIT Classic," a proprietary term-extraction method. Our implementations of the methods we refer to as "Rocchio" ("Roc"), "RocchioFQ" ("RocFQ"), "Prob1," and "Prob2," are given in Figure 3.

In practice, given a training dataset that is partitioned into four folds, *Q1, Q2, Q3*, and *Q4* (for the purposes of our experiments), this optimization procedure translates into taking each combination of thesaurus extraction method *E* and number of terms *N* and performing the following steps:

For each topic *T*

1. Repeat steps 2 to 6 for all combinations {*E, N*} listed in Table 5, thereby generating an average utility for each combination.
2. Do thesaurus extraction on *Q1+Q2* with the {*E, N*} combination.
3. Optimize the threshold for *T* using beta-gamma threshold optimization over *Q3*.
4. Do a blind test on *Q4* generating a utility value U_4.
5. Repeat steps 1 to 4 for alternative combinations of *Q1, Q2, Q3, Q4*, insuring that each database subset is used as a blind test at most once. This leads to a utility value for each database subset of U_1, U_2, U_3, U_4.
6. Set average utility for this combination of {*E, N*} *AvgMonoU* to *Average*(U_1, U_2, U_3, U_4).
7. Select the combination {*E, N*} that provides the highest average utility as the optimal means of generating a monolithic filter for this topic *T*.

A utility measure for each fold of the dataset can be generated using various combinations of extraction folds and optimization folds, however, for our experiments, we limited our exploration to the following: {*Q1=3, Q2=4, Q3=2, Q4=1; Q1=1, Q2=3, Q3=4, Q4=2; Q1=2, Q2=4, Q3=1, Q4=3; Q1=1, Q2=2, Q3=3, Q4=4*}, where *1, 2, 3,* and *4* denote the folds in the training dataset and Q_i denote the variables used in the above algorithm. In deciding between modeling a topic using a monolithic filter or a topic filter, we choose the filter with the highest average utility scores on the four-fold datasets (i.e., *AvgMonoU* and *AvgTopicU*). Prior to running the selected filters on the eleven months of test data, the system retrains each filter using the entire twelve days of training, where the filter thresholds are set using the beta-gamma method on linear utility, T10U, over the entire training dataset.

3.1.2. Ensemble Filter Construction Algorithms

For our TREC-2001 submissions we developed one construction algorithm for each of the two ensemble-filter types used. Since the construction algorithm for multiplex filters is closely related to that for monolithic filter construction (described above), we begin by presenting multiplex filter construction. This algorithm is a simplified version of bagging, whereby each filter is constructed from a sampled subset of the training data based upon an n-fold partitioning of the data. This is in contrast to the more commonly used approach for bagging, where each filter is constructed from a randomly generated dataset. In this case each filter's training dataset is generated by randomly drawing, with replacement, a specified number of examples from the training dataset (typically equal to the size of the training data). We adapted the simpler strategy based on n-folds due to time and system limitations.

For our current experiments, each topic was modeled using a multiplex filter consisting of four component filters (unless there was insufficient training data for the topic), where each filter was constructed using steps 2 to 6 in the algorithm for constructing monolithic filters (above), i.e., one filter corresponding to each fold in the training data. Unlike the monolithic run (where the final monolithic filters were trained on the entire training dataset), the component filters in the multiplex filter were trained on two quarters of the training data, while the threshold was optimized using the beta-gamma method on the third quarter.

Figure 4 gives a screen shot of a multiplex filter that was constructed for topic 39 using the CLARIT AW Toolkit. It presents a multiplex filter consisting of four component filters (each depicted as a node) that were constructed using four different subsets of the training dataset. The folds that were used to generate these subsets are depicted as nodes in the top portion of the screen shot.

On the other hand, the construction algorithm for cascade filters in our TREC2001 submissions is a simplified version of boosting (a version of boosting based upon sampling). The focus of these methods is to produce a series of filters. The training set used for each filter in the series is chosen based on the performance of earlier filters in the series. In boosting, examples that are incorrectly classified by previous filters in the series are chosen more often to train subsequent filters than examples that were correctly classified. Thus boosting attempts to generate filters that are better able to predict examples for which the current ensemble's performance is poor.

For our experiments, we adapted a simplified, rather radical, approach to boosting, whereby each example that was correctly modeled using the current ensemble was not used in the construction of subsequent filters. As a result of this simplification, different stopping criteria were required to ensure termination of the algorithm. These are presented subsequently.

The approach to constructing a cascade filter for a topic *T* involves a number of steps and assumes as input three datasets *D1, D2,* and *D3*, which are respectively used for thesaurus extraction, threshold optimization, and blind testing. These datasets could correspond to the following folds in the training data: *Q1+Q2, Q3,* and *Q4* respectively. In this case the training dataset is split into four subsets/folds. The algorithm consists of two threads: the extraction thread and the threshold setting or optimization thread. Each thread results in the construction of its own cascade filter, namely, $C_{Extraction}$ and C_{Opt}. The algorithm is presented in stepwise fashion in Figure 6, where the left and right hand sides of the figure depict the extraction thread and optimization thread respectively. The algorithm is iterative in nature, whereby the first filter in the cascade, $C1_{Extraction}$, is constructed using the positive topic

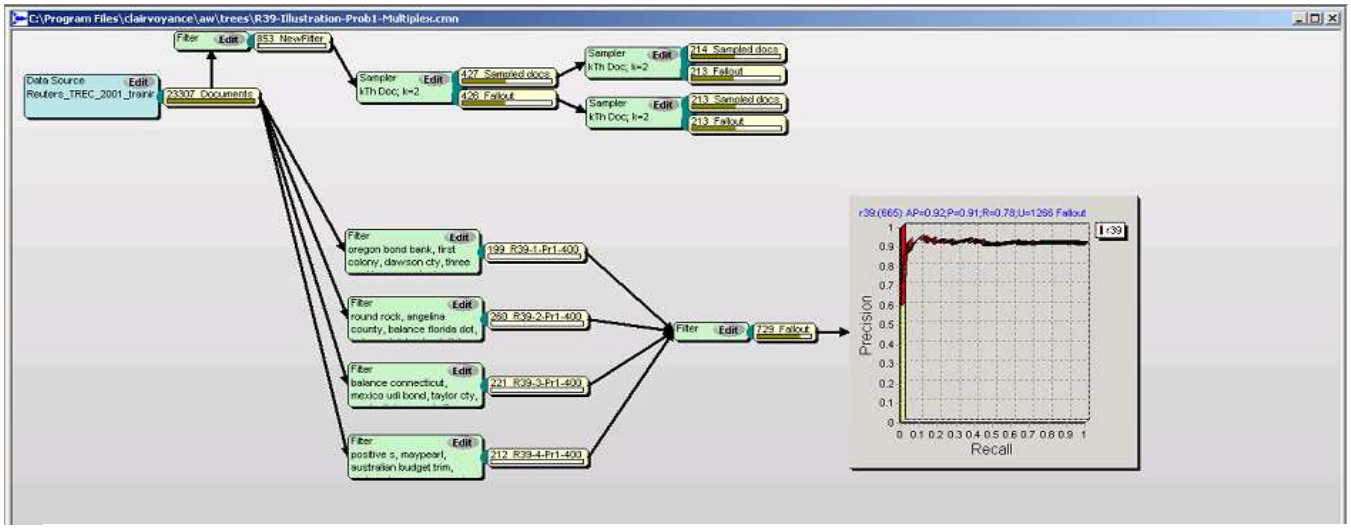

Figure 4. Example of a Multiplex Filter for Topic 39, with Performance Illustrated on the Training Corpus

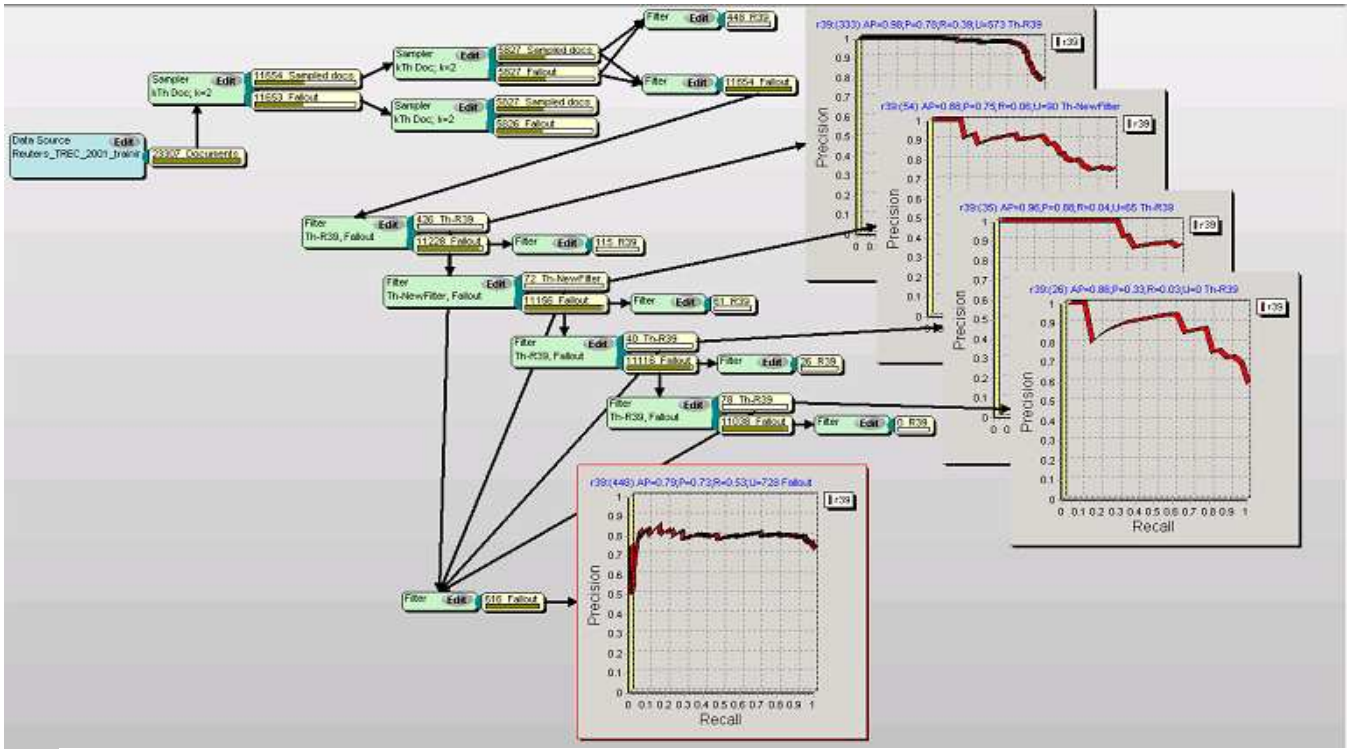

Figure 5. Example of a Cascade Filter for Topic 39, with Performance Illustrated on the Training Corpus

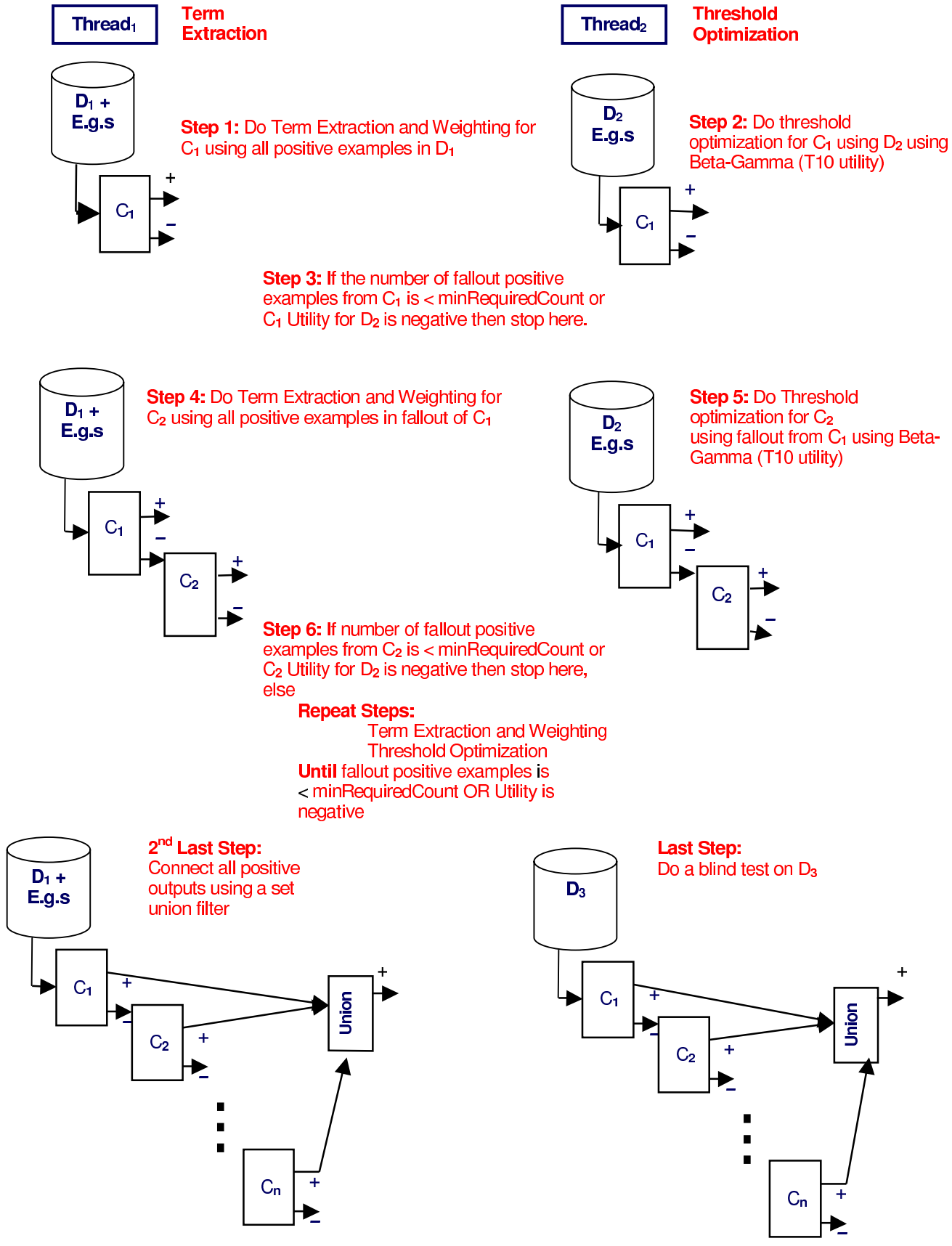

Figure 6. Schematic Representation of the Cascade-Filter Creation Procedure

examples in the extraction dataset *D1* (Step 1 in Figure 6). This cascade corresponds to the extraction cascade $C_{Extraction}$. In order to set the threshold for $C1_{Extraction}$, a second cascade filter (i.e., the optimization cascade) is constructed. The first constituent filter in this cascade is simply a copy of $C1_{Extraction}$ and is denoted as $C1_{Opt}$. To avoid clutter in Figure 6, the *Extraction* and *Opt* suffixes are dropped from the component filters names. The threshold for $C1_{Opt}$ is set using the beta-gamma method based upon the linear utility over the optimization dataset *D2* (Step 2 in Figure 6). The threshold of the $C1_{Extraction}$ filter is set to the optimized threshold of $C1_{Opt}$. Subsequently, the fallout documents from *C1* (i.e., positive examples from *D1* that are rejected by $C1_{Extraction}$) are used to construct the second filter $C2_{Extraction}$ in the cascade, provided various continuation conditions are met. These include: the number of documents in the fallout of $C1_{Extraction}$ is greater than a minimum number of documents required to construct a filter; the linear utility of the $C1_{Opt}$ over the optimization dataset is positive. The above steps of constituent filter extraction and threshold optimization (on the fallout of each preceding filter) are repeated as long as the continuation conditions are satisfied. Once any continuation conditions fail, all the positive outputs of the constituent filters of the extraction cascade $C_{Extraction}$ are connected to a union filter (2^{nd}-Last Step in Figure 6). Subsequently this cascade filter is applied to the D3 dataset and a utility measure is obtained (Last Step in Figure 6).

Figure 5 presents a screen shot of the extraction cascade for topic 39 of the Reuters Corpus. Each filter in the cascade, in this figure, is associated with a precision-recall curve.

The above procedure is repeated such that each quarter of the training dataset serves as a blind test (D3) at most once. This results in a utility measure for each quarter. An average of these four values is then taken, resulting in an average utility, *AvgCascU,* for this cascade filter. This value serves as a comparison to other approaches used for modeling a topic. Should the cascade filter be chosen (based upon average utility) to model a topic, the cascade filter is first re-trained using three quarters of the data for extraction (*D1*), while the fourth quarter is used for threshold optimization (*D2*) and a blind test is carried out on the test dataset.

3.2. Test Configuration

We submitted two batch filtering runs: one where each topic was represented by a single or monolithic filter (*CLT10BFA*); the other, which we call the "rainbow run" (*CLT10BFB*), where each topic was represented by using one (the best) of either a topic, monolithic, multiplex, or cascade filter. Note that for both submission runs, beta was set to 0.1 and gamma was set to 0.05. The minimum number of documents required for filter construction in ensemble filters was set to five. In both cases, we chose the final representation for a topic based on which of the alternative choices yielded the highest average cross-validation utility.

In the construction of cascaded filters, the representation of each constituent filter was globally set to the optimally determined representation for monolithic filters. Since the average cross-validation utility for both monolithic and multiplex filters corresponds to the same value, a further evaluation criterion was necessary. To decide between these two approaches, we re-trained a corresponding monolithic filter using the entire twelve days of training. The beta-gamma method was used to optimize the threshold of the resulting monolithic filter. This re-trained filter was subsequently used for retrieval over the twelve days of training data and a corresponding global utility was calculated. Similarly, the extracted multiplex filter was used to filter documents from the twelve days of training and a corresponding global utility calculated. The approach with the highest global utility was chosen as the approach to model the associated topic.

As a benchmark for our ensemble approaches, we carried out a batch filtering experiment using our traditional information retrieval system in conjunction with an optimization strategy for identifying the term extraction method and term count similar to that outlined above. This experiment modeled each topic using a monolithic filter.

3.3. Test Results

Table 6 presents a summary of various batch filtering runs in terms of the linear utility (T10U) and normalized linear utility (T10SU). Row one of this table presents the median of all submitted runs (from all groups) for TREC-2001 batch filtering. The second and third rows summarize the results for our two submitted runs. Our official heterogeneous or rainbow submission had an average normalized utility of 0.152 and F-utility of 3371. The *CLT10BFC* run represents the result of our benchmark run, which was inadvertently not submitted.

In the context of this year's task, our Batch results are weak. In follow-up analyses, we determined that some of the performance shortfalls were due to processing errors under our control. For example, the test data had become corrupted in our translation of the NIST sources into our processing format; this led to our losing actual test documents, which naturally limited our results in some cases. In another more serious case, we inadvertently failed to reset a critical system parameter—one that sets an upper bound on the number of documents that are considered in any set of documents to be compared to a topic profile—when the system moved from training to testing data. In effect, we only considered about one-third of the test documents that should have been considered as candidates for matching each topic. This particular problem affected both our routing-based baseline run and our heterogeneous run. When we corrected these problems and re-ran over the correct version of the testing data, we saw immediate improvement in both runs. In particular the heterogeneous approach improved by about 60% (from 0.152 to 0.239 normalized utility), making it essentially indistinguishable from our routing-based results. The *CLT10BFA2*, *CLT10BFB2,* and *CLT10BFC2* runs correspond to re-runs of our official submissions and our benchmark run, respectively, where both the dataset errors and the critical retrieval parameter have been corrected. Here, our benchmark run yields a performance of 0.257.

The remainder of Table 6 relates to some post-TREC experiments that addressed various problems that occurred during the preparation of our final submissions.

Run Description		T10SU	T10U
Median for all Submitted Runs		0.256	N/A
Submitted Results	CLT10BFA	0.147	N/A
	CLT10BFB	0.152	3371
Post-TREC Runs	CLT10BFA2	0.237	5834
	CLT10BFB2	0.210	4925
	CLT10BFC	0.234	5453
	CLT10BFC2	0.257	5895
	GlobalCascade	0.220	5323
	LocalCascade	0.195	4882
	Multiplex	0.225	5665

Table 6. Results of Post-TREC Batch Experiments

These experiments were all conducted on the corrected dataset and with properly set retrieval parameters. *GlobalCascade* relates to a run where each topic is represented using a cascade filter where the extraction method and associated term count for each constituent filter in the cascade has a fixed setting. In particular, all constituent filters in the cascade are given the same settings as were determined to be optimal for the corresponding monolithic filter for the topic (in *CLT10BFA2*). *LocalCascade* denotes an experiment where each topic is represented using a cascade filter. In this case the extraction method and associated term count for each constituent filter in the cascade is optimized locally. *Multiplex* relates to an experiment where each filter is represented using a multiplex filter.

3.4. Observations

Our work in batch filtering this year represented an initial attempt at the construction of ensemble filters. Due to system limitations and time constraints, our experiments were accomplished using simplified versions of bagging and boosting algorithms for the construction of multiplex and cascade filters respectively. Even though the performance of the current system is only comparable to the TREC median, performance should improve with full implementations of bagging and boosting algorithms along with more comprehensive experimentation. Current work is addressing both of these issues.

Though our proposed approaches to ensemble filters model subtopic structure, albeit in a limited fashion, a more natural means of identifying and thereby representing topic structure can be achieved using clustering. This forms a very important part to our current work in this area.

Our analysis suggests that ensemble filters perform better than monolithic filters for certain classes of topics. To take advantage of the potential boost in performance that would come from topic structuring, any operational system would have to be able to predict whether a particular filtering task (topic) is best modeled via a monolithic or an ensemble filter. Our hypothesis is that, if there is sufficient training data (or relevance judgments in a running filter), then multiplex filters will outperform monolithic ones. In such cases, then, the task becomes choosing between multiplex and cascade approaches.

Our hypothesis is that cascading is optimal when the topic is vague or diffuse. Thus, to make a principled decision about which approach to select, we need a means of diagnosing topic structure. We have begun work on the development of a simple method to accomplish this.

In our report at the TREC 2001 Meeting, we described retrospective results that simulated such an ideal choice—essentially, the best performing method for each topic as seen in the homogeneous runs represented by our baseline monolithic run (*CLT10BFB2*), our multiplex run (*Multiplex*), and our cascade run (*GlobalCascade*)—which we called "MadMax." A striking feature of the MadMax simulation run was the remarkable performance of cascade filters in the case of twelve topics, significantly exceeding the reported official TREC maximum results. *We feel obligated to report now that, in work since the time of the Meeting, we have not been able to duplicate the extraordinary performance of the cascade runs and now believe we had corrupted data in reporting those results.* In repeated experiments with cascade filters—albeit with the limited conditions we employed in our original runs—we have achieved individual topic performance that equals or exceeds the reported TREC maximum on six topics; but only one of these is significantly better than the maximum.

We continue to believe that successful, robust filtering (especially in distinction to classification) will require topic-specific optimizations, including topic structuring. We have only just begun to explore this problem. We will continue to develop topic-structuring techniques and apply them in future TREC experiments.

References

[1] 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.

[2] 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.

[3] Hansen L, and Salamon P. 1990. *Neural network ensembles*. IEEE Transactions on PAMI 12:993–1001

[4] Quinlan JR, Bagging, boosting, and C4.5. In *Proc.Fourteenth National Conference on Artificial Intelligence*, 1996.

[5] Schapire RE, The strength of weak learnability, *Machine Learning*, 5(2):197–227, 1990.

[6] Breiman, L, *Bagging Predictors*, Machine Learning, 24(2):123-140, 1996.

[7] Stone, M (1974), "*Cross-validatory choice and assessment of statistical predictions*", Journal of RSS B 36, 111–147.

[8] Schapire RE and Singer Y, BoosTexter: A boosting-based system for text categorization. *Machine Learning*, 39(2/3):135–168, 2000.

[9] Wolpert, DH *Stacked generalization*. Neural Networks 5 (1992) 241–259.