Incrementality, Half-life, and Threshold Optimization for Adaptive Document Filtering

Avi Arampatzis* **Jean Beney†** **C.H.A. Koster** **T.P. van der Weide**

Proceedings of the Ninth Text REtrieval Conference (TREC-9),
Gaithersburg, Maryland, November 13–16, 2001.
Department of Commerce, National Institute of Standards and Technology (NIST).

1 Introduction

This paper describes the participation by researchers from KUN (the Computing Science Department of the Katholieke Universiteit Nijmegen, The Netherlands) in the TREC-9 Filtering Track. As first-time TREC participants, our group participated in all three subtasks — adaptive, batch, and routing — while concentrating mainly on adaptive tasks. We have made use of two different systems:

- FILTERIT, for the adaptive and batch-adaptive[1] tasks: a pure adaptive filtering system developed in the context of our TREC-9 participation. It is based on the Rocchio algorithm.
- LCS, for the routing and batch filtering tasks: a multi-classification system based on the Winnow algorithm.

In adaptive filtering, our contribution has been threefold. Firstly, we have investigated the value of retrieved documents as training examples in relation to their time of retrieval. For this purpose we have introduced the notion of the half-life of a training document. Secondly, we have introduced a novel statistical threshold selection technique for optimizing linear utility functions. The method can be also applied for optimizing other effectiveness measures as well, however, the resulting equation may have to be solved numerically. Thirdly and most importantly for adaptive long-term tasks, we have developed a system that allows incremental adaptivity. We have tried to minimize the computational and memory requirements of our system without sacrificing its accuracy. In the batch and routing tasks, we have experimented with the use of the Winnow algorithm, including a couple of small improvements.

From the two topic-sets given, we have experimented only with the 63 OHSUMED queries. We did not submit any runs on the 4904 MeSH topics; these were simply too many to be processed by our present systems in a reasonable time and space. All experiments were done using a keyword-based representation of documents and queries, with traditional stemming and stoplisting, although our long-term intention is to use phrase representations [2], and apply more sophisticated term selection methods [3]. Table 1 summarizes our official TREC-9 runs.

Next, we will briefly describe the pre-processing applied to the data. The FILTERIT and LCS systems are described in Sections 3 and 4, respectively. In Section 5 we give an overall view to how our systems performed in relation to other participants.

2 Stream pre-processing

We used only the title and abstract fields (.T and .W tags) of the OHSUMED documents; their MeSH-headings were discarded. The pre-processing of the documents and topics was minimal and quite traditional. It consisted of (in the order of application): replacement of all non-letters by spaces, deletion of all one-letter words, lowercasing, stoplisting[2], stemming[3], deletion of all one-letter stems, and DF-stoplisting (removal of the top-100 stems with the highest document frequencies in `ohsumed.87`).

In summary, our pre-processing was quick-and-dirty. There was no special treatment of proper names, all numbers were lost, and we made no use of multi-word terms such as phrases or word clusters. Moreover, we used no external resources such as online dictionaries or thesauri.

*Computing Science Institute, Faculty of Mathematics and Informatics, University of Nijmegen, Postbus 9010, 6500 GL Nijmegen, The Netherlands,
tel: +31 6 51 408838, fax: +31 24 3553450,
e-mail: `avgerino@cs.kun.nl`, `http://www.cs.kun.nl/~avgerino`
†On sabbatical leave from INSA de Lyon.

[1] We see the batch-adaptive task as an adaptive rather than a batch filtering task.

[2] We used the standard stoplist of the SMART system, `english.stop`, available from: `ftp://ftp.cs.cornell.edu/pub/smart/`

[3] We used the Porter stemmer of the `Lingua::Stem` (version 0.30) library extension to PERL

Task	Topics	Optimized for	System	Run-tag
adaptive	OHSUMED	T9U	FILTERIT	KUNa1T9U
adaptive	OHSUMED	T9U	FILTERIT	KUNa2T9U
adaptive	OHSUMED	T9P	FILTERIT	KUNa1T9P
adaptive	OHSUMED	T9P	FILTERIT	KUNa2T9P
batch-adaptive	OHSUMED	T9U	FILTERIT	KUNbaT9U
batch	OHSUMED	T9U	LCS	KUNb
routing	OHSUMED	—	LCS	KUNr1
routing	OHSUMED	—	LCS	KUNr2

Table 1: TREC-9 filtering runs submitted by KUN.

3 The FILTERIT System

The FILTERIT system, which we used for performing the adaptive and batch-adaptive tasks, has been developed in the context of our TREC-9 participation. It is a pure adaptive filtering system based on Rocchio's method [10]. Rocchio's method performs well in a situation where only a few training documents are available, see e.g. [9], and this is exactly the case in the adaptive task. In such a situation, the initial query becomes important and the method can moreover deal in a suitable way with the topic descriptions.

We have modified the formula traditionally used for relevance feedback in order to allow for weighing of training documents according to their time-stamps. Moreover, the implementation of the algorithm we will present, allows very accurate incremental training of classifiers, without using any document buffers, so its memory and computational power requirements are low. In order to limit further the memory requirements of our system per topic, we also use a form of on-the-fly term selection.

Our system adapts queries and thresholds independently for each topic, meaning that the filtering model for a topic is updated after the retrieval of every single document for that topic. In the runs optimized for the T9P measure, threshold adaptations are even triggered independently of document retrievals.

For optimizing the filtering thresholds, we have introduced a new statistical technique which takes into account the relative density of relevant to non-relevant documents seen in the stream, and their score distributions. Most of the quantities that our technique requires can be updated incrementally, but a small document buffer seems unavoidable.

3.1 Incremental Query Training

The version of Rocchio's method traditionally used for relevance feedback is

$$Q = \alpha\, Q_0 + \beta \frac{1}{|\mathcal{R}|} \sum_{D \in \mathcal{R}} D - \gamma \frac{1}{|\mathcal{N}|} \sum_{D \in \mathcal{N}} D \;, \qquad (1)$$

where Q_0 the initial query, $\mathcal{R}$ and $\mathcal{N}$ the sets of relevant and non-relevant documents respectively, and $|.|$ denotes the number of elements in a set. The parameters α, β, and γ control the relative contribution of the initial query, and that of the relevant and non-relevant documents to the new query Q. All components which end up with negative weights in Q are removed.

The initial query and the documents are usually represented by vectors weighted in a *tf.idf* fashion[4]. While the *tf* components are usually independent of corpus statistics, the *idf* components depend on the collection. Since in filtering the whole collection is not available in advance, the *idf* components should be updated over time (*incremental idf*). Therefore, it would be more suitable for filtering to keep these quantities separately. As a result, queries and documents in our system are only *tf*-weighted, e.g., a document D_i is represented by

$$D_i = [tf_{i1}, \ldots, tf_{iK}] \;, \qquad (2)$$

where K the total number of terms known by the system at one point in time. Any document or query is a sparse array since it contains far less non-zero components than K, so they are implemented by *hash arrays*.

Since all vectors are only *tf*-weighted, we have moved the impact of *idf*s into the similarity function, which for a query Q and a document D has been defined as:

$$S(Q, D) = Q\, IDF\, D^T \;, \qquad (3)$$

where IDF is the diagonal matrix $diag(idf_1, \ldots, idf_K)$, and X^T denotes the transposed array of X. Such an implementation allows, at any time, the usage of the latest *idf* values.

[4] *tf.idf* denotes here the *family* of weighting schemes which weigh a term proportionally to its frequency in a document or query and inversely proportional to its frequency across the collection. In practice, *tf* and *idf* are implemented by some monotonically increasing functions of the corresponding frequencies. We consider the decision to use a *tf.idf*-type weighting scheme as an *architectural* choice, while the exact form of the functions is an *implementational* choice. We give our implementational choice in Section 3.3.

Now, formula 1 can be calculated incrementally by simply re-writing it as

$$Q_n = \alpha\, Q_0 + \beta \frac{1}{R_n} B_n - \gamma \frac{1}{N_n} C_n \ , \tag{4}$$

where B_n, C_n are the accumulated sums of the term frequency vectors of relevant and non-relevant documents respectively, and R_n, N_n are the numbers of documents in each category[5]. When document D_n is retrieved, Q_n is calculated in two steps. First, all time-dependent quantities (everything on the right side of the formula which has the subscript n) in the last formulation are updated. Then, the query Q_n is calculated using the updated quantities.

Summarizing, the architecture we have just described allows the most accurate incremental training with Rocchio. No training documents have to be discarded, as would have been necessary in a *sliding window* adaptive system. Moreover, no document buffers are necessary, except B_n and C_n in which all training documents are accumulated. In order to achieve all these, the only requirement is that *tf*s are *static* in the sense that they can be calculated only once when a document arrives.

Of course, there is another minor concession we make here, that is to allow counting registers of infinite width (the values of the components of B_n, C_n, and the variables R_n, N_n can grow up to infinity). Double precision arithmetic approximates this assumption well. In any case, when a number approaches the maximum width, all quantities can be divided by a constant without invalidating the model.

3.2 Convergence, Responsiveness, and Decay

The goal of the incremental training we have described so far is to gradually converge to a perfect classifier. All training documents, irrespective of their time of retrieval, are taken into account with equal importance in constructing the classifier. Systems that implement this kind of converging adaptivity we shall call *asymptotically adaptive*. The use of an asymptotically adaptive system for filtering implicitly assumes that topics are *stable*, i.e. there are no *topic drifts*.

If there are topic drifts, the position of the perfect classifier moves in the document-space. Therefore, it is beneficial for a filtering system to be capable of tracking a topic rather than converging. This capability can be achieved by weighing more heavily training documents that are retrieved recently. We call such systems *locally adaptive*. The choice between local adaptivity and asymptotic adaptivity should be made depending on whether *convergence* or *responsiveness* is more important. More about various forms of adaptivity for filtering systems and the nature of topics in filtering can be found in [1].

In TREC-9, topics are assumed to be stable, suggesting that an asymptotic behaviour would be more proper. However, the OHSUMED collection consists of documents collected in a period of five years and it is likely that for a topic the content of its relevant documents changes over the years, e.g., think of new treatments developed for the same sickness. The effect of such *document content drifts* is equivalent to *user interest drifts* in the sense that the idea of *relevance* changes.

In order to weigh training documents differently, we replace the average vectors in the Rocchio formula of Eq. 1 with *weighted averages*. This does not invalidate the motivation of the formula. For instance, the average vector of relevant documents becomes

$$\frac{1}{|\mathcal{R}|} \sum_{D \in \mathcal{R}} D = \frac{1}{\sum_{i: D_i \in \mathcal{R}} l_i} \sum_{i: D_i \in \mathcal{R}} l_i D_i \ , \tag{5}$$

where l_i represents the weight with which the document D_i contributes to the average.

A heavier weighting of recently retrieved training documents may be implemented by a *decay* operation with *half life* h, i.e. the age that a document must be before it is half as influential as a fresh one in updating the query. If a document D_i is retrieved at time t_i, and the current time is t_n, we set

$$l_i = 0.5^{(t_n - t_i)/h} \ , \tag{6}$$

where t_n, t_i, and h are measured in the same units, e.g., months.

Whether the initial query Q_0 should decay or not depends on the nature of a topic. For a drifting user interest, Q_0 should decay. For a stable interest with document content drifts (as we argued to be true for TREC-9), any of the two choices can be motivated (it rather depends on how Q_0 is formulated). For our official TREC-9 runs, we chose to decay Q_0.

The decay operation can be performed incrementally. When D_n is retrieved, and assuming that it is found to be relevant, then it is easy to show that average vectors, e.g., the one of relevant documents, can be updated as:

$$\frac{1}{R_n} B_n = \frac{1}{l R_{n-1} + 1} \left(l B_{n-1} + D_n \right) \ , \quad l = 0.5^{(g/h)} \ ,$$

where $g = t_n - t_{n-1}$ stands for the elapsed time since the previous query update (i.e., since Q_{n-1} was calculated). Therefore, when a document is retrieved, all

[5] The convention we use for the subscript n is: n is the total number of training documents available (relevant and non-relevant). Training documents are the ones given at the time of bootstrapping (as for the batch-adaptive task), and all retrieved ones during filtering since their relevance judgment can be seen. Thus, Q_n is the classifier built using n training documents. If r of them are relevant, then $R_n = r$ and $N_n = n - r$, and B_n, C_n contain the sum of r and $n - r$ document vectors respectively.

time-dependent quantities of equation 4 are multiplied by the current decay factor l before they are updated with the new document. To maintain correct decaying weights, even the quantities which are not going to be updated have to be multiplied, e.g. even if D_n is relevant, $N_n = lN_{n-1}$ and $C_n = lC_{n-1}$.

In TREC-9, time is estimated on the number of documents seen in the stream. It is given that the stream produces, on average, around 6,000 documents per month. Therefore, for a half life of m months, we set $h = 6,000m$, and g is simply the number of documents filtered since the previous query update.

3.3 Term Weighting

For term weighting, we "borrowed" the *Ltu* formula from [11]. In the *Ltu* weighting scheme, L is the term frequency factor, t is the inverted document frequency factor, and u is the length normalization of the document or query.

The t factors were initialized from `ohsumed.87`. Then we used *incremental idf*: upon the arrival of a new document and before any other calculation is performed, all quantities that contribute to the t factors are updated

The application of the *Ltu* formula in adaptive filtering presents a small problem. The average number of unique terms per document changes over time, therefore, the term weights of past documents should be re-calculated as well. We chose to calculate this average document length on `ohsumed.87` and assume that it will not change in the future. This allows to calculate the u factor once and for all, when a document arrives. The assumption that the average document length will remain the same in the future is not far from reality for the OHSUMED collection, since there is no special reason why medical researchers should write abstracts of different lengths over time.

Summarizing and using our notation, the exact form of the term weighting we used is:

$$tf = L \times u' \ , \quad idf = t \ , \tag{7}$$

where u' is the same as u but with the average document length fixed on its `ohsumed.87` value (that was 40.8 after the pre-processing). This form presents *static tf* components, in the sense that they are calculated once when a document arrives without the need to re-calculate them in the future, and *dynamic idf*s. These features allow for incremental training, as we have shown in Section 3.1.

3.4 On-the-fly Term Selection

It is empirically known that as the size of a corpus grows, the number of unique words seen grows with the square-root of the number of documents. In case of multi-word terms (phrases), the number of such enriched terms grows even faster. Therefore, the number of components of B_n and C_n vectors grows, at least, with the square-root of the number of retrieved documents n. To limit the size of these vectors we use term selection.

In fact, term selection is more critical for the threshold optimization technique we will describe in Section 3.5. The incremental application of the optimization technique requires matrices as large as the *square* of the size of B_n or C_n, consequently the memory requirements may explode soon if no term selection is used (see Section 3.5.3).

Term selection was applied for each topic independently, before every incremental update of the corresponding query. Our *on-the-fly* term selection consists of the following steps. First, a query is constructed using information only from relevant instances and the current *IDF* matrix:

$$Q_{n,\mathrm{rel}} = \left(\alpha\, Q_0 + \beta \frac{1}{R_n} B_n \right) \mathit{IDF} \ . \tag{8}$$

Then, we rank all terms of $Q_{n,\mathrm{rel}}$ according to their weight, and select only the top-k ones and the terms occurring in Q_0. The rest of the terms are discarded and removed from all quantities kept by the system for the topic (e.g., B_n and C_n). Then, Q_n is calculated using the reduced data.

This technique limits the memory required for filtering a topic. However, the size of the *IDF* matrix still grows by the time, as previously unseen terms occur in documents of the stream. We consider *IDF* as *stream data* rather than *topic data*, since it is the same for all topics being filtered at any point in time. Therefore, we do not limit its size.

3.5 The Score-Distributional Threshold Optimization

Let us assume that for some topic a training stream of n documents is available, of which r are relevant. After the filtering the stream with some query and a threshold, each document may be classified under one of the four categories shown in the contingency table:

	relevant	non-relevant
retrieved	R_+	N_+
non-retrieved	R_-	N_-
total	r	$n-r$

The variables R_+, N_+, R_-, and N_- refer to the number of documents in each category. Given any evaluation measure M and a query, a filtering threshold θ can be selected so as to optimize the measure on the training stream. In this Section, we outline a threshold optimization technique which can be applied for any evaluation measure of the form $M(R_+, N_+, R_-, N_-)$, i.e. M is any function of the document counts in each category.

The idea is to describe a dataset of document scores with their probability density function. There are two such functions, one for relevant and one for non-relevant document scores. Then, these functions are multiplied by the corresponding numbers of document scores used for their estimation, so that the area below each curve amounts to the number of documents. Now, each variable of the contigency table can be expressed as a function of θ by integrating a range on the corrensponding curve, e.g. $R_+(\theta) = \int_\theta^{+\infty} r P_\mathrm{r}(x)\, dx$ where P_r is the probability density of relevant document scores. The threshold which optimizes M is a solution of

$$\frac{dM\left(R_+(\theta), N_+(\theta), R_-(\theta), N_-(\theta)\right)}{d\theta} = 0\ . \qquad (9)$$

Depending on the exact form of function M, equation 9 may not have analytical solutions and it should be solved numerically.

3.5.1 Score Distributions

In [4] we prove that a Gaussian limit appears for the distribution P_r of relevant document scores. Furthermore, we show that the distribution approaches the Gaussian quickly, such that corrections go to zero as $1/K_Q$, where K_Q the length of the query. Empirically, Gaussian shapes form at around $K_Q = 250$.

For non-relevant documents, we show in the same study that a Gaussian limit is not likely, and if it appears, then only at a very slow rate with K_Q. Empirically, we have never seen Gaussian shapes even for all dimensions resulted from massive expansion of queries. Our empirical data, however, point out that the right tail of the distribution can be very well approximated with an exponential.

Figure 1 shows the empirical score distributions for TREC topic 352 on the Financial Times collection. We collected these data as follows. First, we trained a classifier using all relevant documents and an equal number of the top-scoring non-relevant using the *query zone*[6]. Then, we calculated the scores of relevant and non-relevant documents for the classifier. The middle plot shows the empirical distribution of relevant document scores, and the corresponding Gaussian multiplied by the number of scores. The left plot shows the empirical distribution of the top-100 non-relevant scores, and exponential curves of the form $c_1 e^{-c_2 x}$ fitted on the top 100, 50, 25, and 10 scores. It seems that at least 50 or more scores are needed for an accurate threshold estimate. The right plot shows the optimal T9U threshold. As we will prove in Section 3.5.2, the optimal T9U threshold is the intersection of the probability densities of relevant and non-relevant document scores, weighted as $2r$ and $n-r$ respectively.

[6]For the query-zoning method, see Section 3.6.2 or [12].

3.5.2 Optimizing Linear Utility Functions

Let U any linear utility function of the form

$$U_{(\lambda_1,\lambda_2,\lambda_3,\lambda_4)} = \lambda_1 R_+ + \lambda_2 N_+ + \lambda_3 R_- + \lambda_4 N_-\ , \qquad (10)$$

where $\lambda_1, \lambda_2, \lambda_3, \lambda_4$ denote the gain or cost associated with each document that falls in the corresponding category. The linearity of such measures allows for analytical solutions of Eq. 9, since the integrals describing the document counts cancel out with the derivative of the measure. Consequently, and after a few calculations, Eq. 9 becomes

$$\lambda \rho P_\mathrm{r}(\theta) = P_\mathrm{nr}(\theta), \quad \lambda = \frac{\lambda_3 - \lambda_1}{\lambda_2 - \lambda_4}\ , \quad \rho = \frac{r}{n-r}\ . \qquad (11)$$

ρ is the *relative density* of relevant to the non-relevant documents in the training stream. For T9U, $\lambda = 2$. P_r and P_nr are the density functions of the probability distributions of relevant and non-relevant document scores.

Let P_r be a Gaussian density with mean μ_r and standard deviation σ_r, and P_nr an exponential density of the form $c_1 e^{-c_2 x}$ estimated on the right tail of the distribution of the non-relevant scores. Then, the solution of Eq. 11, i.e. the optimal threshold, is:

$$\theta = \begin{cases} (b - \sqrt{\Delta})/a & \text{if } \Delta \geq 0 \\ +\infty & \text{if } \Delta < 0 \end{cases}\ , \quad \Delta = b^2 - ac\ ,$$

$$a = \frac{1}{\sigma_r^2}\ , \quad b = \frac{\mu_r}{\sigma_r^2} + c_2\ , \quad c = \frac{\mu_r^2}{\sigma_r^2} - 2\ln\left(\frac{\lambda\rho}{c_1\sqrt{2\pi\sigma_r^2}}\right). \qquad (12)$$

Note that since the exponential corresponds to the top non-relevant scores, it does not extend accurately to low scores. Consequently, the method gives more accurate results when there is no contribution of N_- into the utility score, i.e. for $\lambda_4 = 0$, which holds for T9U.

3.5.3 Incremental Mean and Deviation

The method we have described for threshold optimization uses the mean of the relevant document scores and their standard deviation. In general, means and deviations can be calculated incrementally. However, in the case of filtering every update of the query causes the scores of the previous training documents to change. The choice that we have made in Section 3.1 to have query and documents only *tf*-weighted and keep *idf*s separately, allows for incrementality also here.

For query Q_n, the average score of r relevant documents $D_1, \ldots, D_r$ with scores $s_1, \ldots, s_r$ can be written as

$$\mu_\mathrm{r} = \frac{1}{r}(s_1 + \cdots + s_r)$$

$$= \frac{1}{r}\left(Q_n\, IDF\, D_1^T + \cdots + Q_n\, IDF\, D_r^T\right)$$

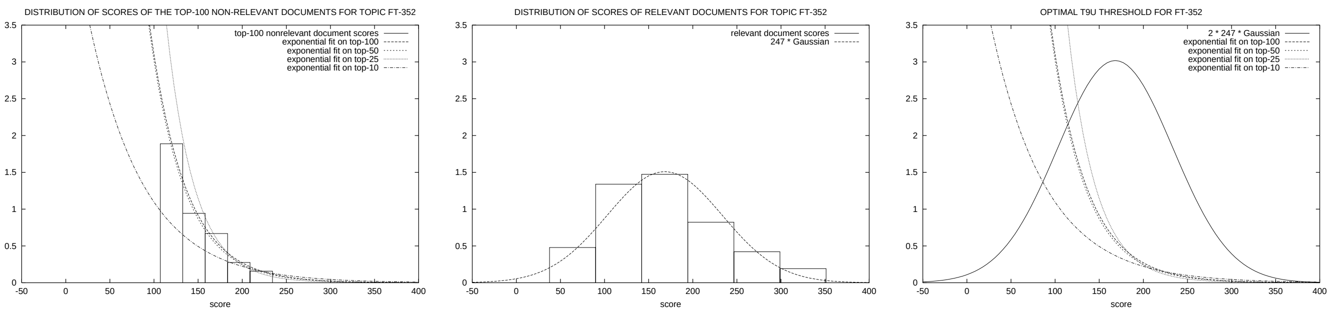

Figure 1: Score distributions and the optimal T9U threshold.

$$= \frac{1}{r}\, Q_n\, IDF \sum_{i=1}^{r} D_i^T = \frac{1}{r}\, Q_n\, IDF\, B_n^T \ . \qquad (13)$$

Obviously, the individual document scores $s_1, \ldots, s_r$ are not needed, but only the accumulated sum of the relevant document vectors B_n. Using B_n, the current query Q_n, and the most recent *IDF* matrix, the mean score can be calculated accurately and incrementally. This way of keeping an average score accurate has been seen before in [6]; we have merely re-formulated it, using matrices, for compactness.

The standard deviation may be obtained from the formula $\sigma_{\rm r}^2 = \mu_{\rm r}^{(2)} - \mu_{\rm r}^2$, where $\mu_{\rm r}^{(2)}$ is the mean of the squares of the relevant document scores. The proof of the incremental formula for $\mu_{\rm r}^{(2)}$ is more complex; here we give only the final formula:

$$\mu_{\rm r}^{(2)} = \frac{1}{r}\, (Q_n\, IDF)\, B_{{\rm dyad},n}\, (Q_n\, IDF)^T \ ,$$

$$B_{{\rm dyad},n} = \sum_{i=1}^{r} D_i^T D_i \ . \qquad (14)$$

Consequently, a $K \times K$ matrix $B_{{\rm dyad},n}$ is required for keeping the deviation of the scores accurate, however, this matrix can be updated incrementally. K grows with a square-root, so $B_{{\rm dyad},n}$ grows linearly in time. This makes term selection indispensable (see Section 3.4).

3.5.4 Optimizing T9P

The S-D threshold optimization we have introduced in Section 3.5 can be applied to optimize T9P. However, in this case Eq. 9 does not have analytical solutions, therefore it has to be solved numerically. Regrettably, we did not bother to do that.

The technique we used lowers the threshold after every "quiet" month with respect to how many documents are missing according to the pro-rata adjusted minD value. It goes as follows:

1. right after a query update, start collecting the document scores in the range $[\mu_{\rm nr}, \theta]$, where $\mu_{\rm nr}$ is the mean score of the N^+-documents and θ the optimal S-D threshold for $U = R_+ - N_+$.

2. if after one month of documents nothing is retrieved, calculate how many should have been retrieved by the current time (pro-rata).

3. check how many are missing:
   m = pro-rata$-$retrieved.

4. if $m > 0$, lower the threshold to s_m, where s_m the top-m score seen below θ.

The method works, in the sense that it retrieves around minD$(= 50)$ documents or more. Moreover, it retrieves the ones that score the highest. It assumes, however, that the distribution of relevant documents in the stream is uniform (or their relative density is approximately constant), in general a false assumption. Another drawback of the method is that it optimizes the threshold for $U = R_+ - N_+$ and not for precision. All of these, we believe, make our submitted T9P runs moderately satisfactory.

After all, we should have at least tried to solve Eq. 9 numerically. Although analytical formulas are mathematically more elegant, in practice, numerical methods are efficient and easy to implement.

3.6 Experiments with FilterIt

The FilterIt system presents two features which we are interested in comparing their effectiveness with other systems: the threshold optimization for linear utility functions (see Section 3.5.2), and the decay of training documents (see Section 3.2). The tuning parameters were numerous, and the runs allowed for submission to TREC-9 were limited to 4 for adaptive and to 2 for batch filtering (including batch-adaptive). Moreover, we submitted one of the two batch filtering runs with the LCS system described in Section 4. These limits do not allow extensive comparisons, and some choices had to be made.

Our strategy in deciding what to submit was as follows. For the two of the four adaptive runs we did not use any of the two features but rather conventional techniques. In this way, we expected to have at least two runs with conventional effectiveness, in case our techniques would have failed. The other two adaptive and the single batch filtering run combine all the new features. All parameters were set at "safe" values, as these were determined by our experiments with the Financial Times (FT) collection. More aggressive settings have yielded better effectiveness on FT, however, we do not believe that these generalize in all collections.

3.6.1 Rocchio Parameters, and Initial Query Elimination

All adaptive runs use $\alpha = \beta = \gamma$ for Rocchio. These tasks start with a query and only 2 relevant training documents. In pilot runs on FT, traditional settings with $\alpha < \beta$ seemed to overfit the classifiers on those 2 relevant documents. Therefore, such small training sets should not be trusted and the initial query Q_0 should be weighted fairly high, e.g., as high as $\alpha = \beta$. As a filter is collecting more and more relevant documents, the contribution of the initial query can gradually be eliminated. Consequently, we moreover multiply Q_0 with $10/(R_n + 10)$ while calculating the new query Q_n. We do not use such an *initial query elimination* for the runs with decay since the initial query decays anyway.

For the batch-adaptive task, α is set at the one-fourth of β. Since larger training sets are given for this task, the danger of overfitting is smaller. When using query zones, [12] have shown that $\beta = \gamma$ is a reasonable setting. This also explains why we set $\beta = \gamma$ also for the adaptive tasks. Thresholding document scores during filtering can been seen as a form of *on-the-fly query zoning*. Any non-relevant documents retrieved in this way are indeed the most interfering with the query. This setting has worked out well for us in our experiments on FT.

3.6.2 Submitted Runs

Table 2 summarizes the runs we submitted, their parameter settings, and the final results obtained.

KUNa1T9U and KUNa1T9P do not use decay, term selection, or the threshold optimization described in this article. The threshold per topic is set at the midpoint of the average scores of relevant and non-relevant documents. In fact, for KUNa1T9U we set thresholds at the one-third of the distance between the non-relevant and the relevant mean score to reflect the fact that the gain of retrieving a relevant document is double than the cost of retrieving a non-relevant one (definition of T9U). Therefore, the thresholds should be lower than the midpoints to retrieve more relevant documents.

KUNa2T9U and KUNa2T9P use a decay for training documents with half life set to 2 years; we have found this value reasonable for filtering medical articles. Term selection cutoff is set at the top-500 terms; a light cutoff because our threshold optimization seems to require at least 250 terms in a classifier (Section 3.5.1), and moreoever, long classifiers are necessary when tracking relevance drifts [1]. Thresholds are S-D optimized, however, not exactly as we have described in this article.

Our S-D method was in an early stage at the time of submition. What we did was to approximate the N_+ document scores with a Gaussian. Repeatedly adapting a query causes the distribution of non-relevant retrieved document scores to look more like a bell-shaped distribution. This is an artifact of re-training, however, and does not correspond to what is really happening below the threshold. Nevertheless, it has worked out resonably, suggesting that a Gaussian approximation may be usable since it still gives some estimation of the *spread* of the non-relevant scores; however, it is of dubious accuracy. We will come back to this in Section 3.6.3.

For KUNbaT9U (batch-adaptive) we basically use the same settings as for KUNa2T9U, except for the Rocchio parameters. Moreover, we apply *document sampling* and *query zoning* [12]. The training stream (`ohsumed.87`) consists of around 54,000 documents, and only a few of them are relevant for a topic. For efficiency reasons we do random sampling with probability 0.1 to reduce the number of non-relevant training documents. Then we apply query zoning to select and use for training only the top-r scoring non-relevant documents, where r is the number of relevant training documents. We calculate the query zone with formula 1 for $\gamma = 0$.

The adaptive runs do not show large differences in effectiveness, mainly because of the modest parameter settings for term selection cutoff, half life value, and the fact that the S-D threshold optimization technique is triggered only when at least 5 relevant and 5 non-relevant training documents are made available. Many topics did not reach these numbers, so they were actually filtered with thresholds set at weighted midpoints.

3.6.3 More Runs

In this Section, we provide the extras runs we made in order to find where some the parameters of FilterIt peak, and determine which techniques actually work. All runs reported here use (unless otherwise is noted): query zoning to select for training only the top-r non-relevant documents, term selection cutoff set at 500, no decay, and thresholds set at weighted midpoints for T9U.

	Runs				
	KUNa1T9U	KUNa2T9U	KUNa1T9P	KUNa2T9P	KUNbaT9U
Task	adaptive				batch-adapt.
Rocchio	$\alpha = \beta = \gamma$				$4\alpha = \beta = \gamma$
Q zoning	—				top-r
Q_0 elimination	$10/(10 + R_n)$	no	$10/(10 + R_n)$	no	no
T-opt. for	T9U	T9U	T9P	T9P	T9U
T-opt. method	$(\mu_{\mathrm{r}} + 2\mu_{\mathrm{nr}})/3$	S-D	$(\mu_{\mathrm{r}} + \mu_{\mathrm{nr}})/2$	S-D	S-D
half life	∞	2 yrs	∞	2 yrs	2 yrs
T.S. cutoff	—	500	—	500	500
Result	+16.8	+17.3	0.258	0.231	+19.4

Table 2: Parameter settings for adaptive and batch-adaptive submitted runs.

Document Sampling and Query Zoning. We have investigated the effect of sampling the non-relevant document space. We have run a batch-adaptive task with 3 different samples. Table 3 presents T9U and F-measure results. All samples are made by selecting randomly one out

sample	T9U	F_1
A (official)	19.5	0.406
B	19.8	0.406
C	19.1	0.403

Table 3: The effect of sampling the non-relevant training document space.

of ten non-relevant training documents from `ohsumed.87`. Then query zoning is applied before training the initial classifier. The results do not show significant differences.

Term Selection. Figure 2 shows the impact of our term selection method (see Section 3.4) for different cutoff values. The runs are batch-adaptive using sample A. The average T9U seems to peak between 500 and 125 terms.

Decay. We have experimented with different half-life values on an adaptive task. Figure 3 shows that the average T9U peaks somewhere between 2 and 8 years of half life. However, further analysis has revealed that effectiveness peaks at considerably different half-life values across topics. An optimization of half-life per topic — if we only had a way to do that — would have resulted in great improvements of the average T9U.

Threshold Optimization. In [13] we give the TREC-9 evaluation table of our submitted batch and batch-adaptive runs. We have made a supplemental batch-adaptive run with the revised S-D threshold optimization as described in this paper, i.e. by fitting an exponential on the top-50 non-relevant training documents[7]. When the non-relevant training document buffer exceeds 50 documents, we sort them according to their scores and discard the lowest scoring one. The results are presented in the last column, labeled as `FilterIt-ba`. They show an improvement in the average T9U from 19.4 to 21.3.

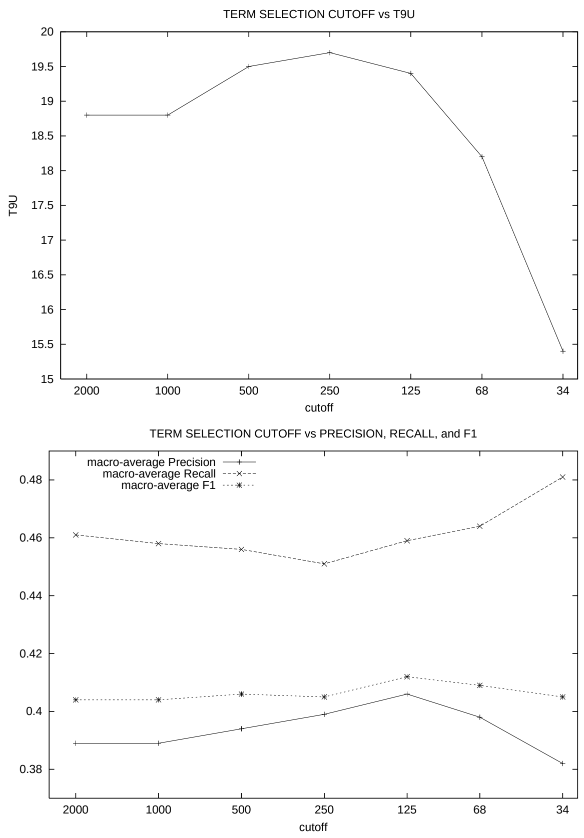

Figure 2: The effect of term selection.

[7]Note that we have not optimized any other parameter according to our post-official runs; we have merely used a better S-D optimization.

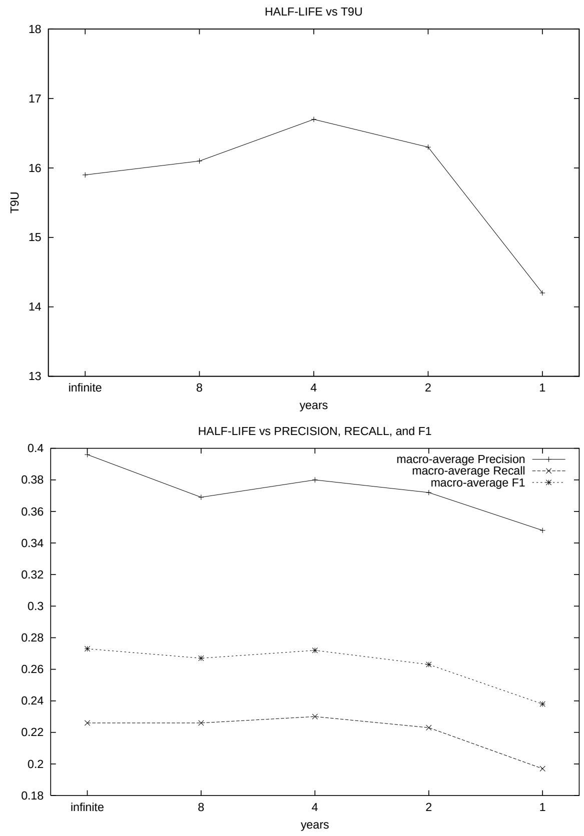

Figure 3: The effect of decay.

One could argue that setting thresholds with the weighted midpoint method works out comparably to the S-D optimization (compare e.g. KUNa1T9U to KUNa2T9U), but this is not the case. In fact, the good performance of the weighted midpoint method has been purely accidental; the same goes for the aforementioned Gaussian fit on non-relevant document scores. The mean score of non-relevant documents $\mu_{\rm nr}$ has been estimated on the top-scoring non-relevant documents. This produces a relatively large $\mu_{\rm nr}$, which in its turn results in tight thresholding. When we have tried to increase the number of non-relevant documents, the weighted midpoint method as well as the Gaussian fit have greatly failed: the more non-relevant documents are used for training, the lower the $\mu_{\rm nr}$, thus lower thresholds. The methods fall too easily into the *selectivity trap* of retrieving too many (mostly non-relevant) documents. The revised S-D optimization as described in this article has proved much more reliable and robust in a range of settings, consistently avoiding such selectivity traps.

4 The LCS System

The routing and batch filtering tasks were carried out by the LCS system[8] [9]. The system is based on the Winnow mistake-driven learning algorithm [8]. The Winnow algorithm has, to our knowledge, not been used before in TREC. It can cope well with large numbers of terms, which is certainly the case here: after pre-processing, the training set had some 52,000 different terms.

4.1 The Winnow Algorithm and Improvements

During training, the *Balanced Winnow* algorithm [8, 7] iteratively computes two weights $w^+_{i,C}$ and $w^-_{i,C}$ for every term i and class (topic) C. These *winnow weights* are used to compute the score $S(D,C)$ of a document D for the class C as:

$$S(D,C) = \sum_{i \in D} (w^+_{i,C} - w^-_{i,C}) * u_{i,D} \ , \qquad (15)$$

where $u_{i,D}$ is the *term strength* (weight) of term i in document D. Classification is achieved by thresholding $S(D,C)$ using a threshold θ.

Winnow is *mistake-driven* in the sense that it adjusts the weights $w^+_{i,C}$ and $w^-_{i,C}$ only if their current value, during an iteration, leads to a misclassification. If a relevant document scores below θ, then the winnow weights for the terms occurring in the document are multiplicatively updated using a promotion factor Alpha. Similarly, for a non-relevant document scoring above θ, the weights are demoted using a demotion factor Beta. The threshold θ is considered fixed, and the learning stops when there are no weight updates during an iteration, or earlier even in order to avoid over-training. Topic descriptions were considered as normal documents, since Winnow provides no special mechanism for dealing with requests.

The implementation of Winnow in LCS is similar to the one described in [7], with two modifications:

1. the document terms $u_{i,D}$ are *ltc* weighted [5], without the vector length normalization factor. Traditionally, $u_{i,D}$ are set either to the frequency of i within D, or to the square-root of the frequency. In experiments on the FT corpus, *ltc* has proved to work definitely better than the former, and slightly better than the latter.

2. Winnow weights were initialized for training as:

$$w^+_{i,C} = \frac{2\theta}{ADS} \ , \quad w^-_{i,C} = \frac{\theta}{ADS} \ ,$$

[8] Esprit project DOcument ROuting (DORO), `http://www.cs.kun.nl/doro`

$$ADS = AVG_D \frac{\sum_{i \in D} u_{i,D}}{size(D)} ,$$

where $size(D)$ is the number of unique terms in document D. This initialization improves Winnow's convergence speed.

The convergence speed of the Winnow algorithm (the number of iterations needed to learn a stable classifier) depends rather critically on the initial values of the weights. In [7], all positive weights are initialized as θ/d, where θ is the threshold and d the average number of "active features" in documents. This choice ignores collection statistics for terms. In our initialization, an average document obtains an initial score equal to θ. Since term strengths are taken into account, fewer iterations are needed.

4.2 Threshold Setting by Cross-evaluation

The Winnow algorithm has a "natural" threshold $\theta = 1.0$ for separating relevant from non-relevant documents, putting equal importance on precision and recall. T9U stresses recall more than precision, however. The S-D threshold optimization, as described in section 3.5, has not (yet) been implemented in the LCS, so the necessary threshold optimization was performed by *cross-evaluation*.

The training set (`ohsumed.87`) was split into n subsets of the same size, which each in turn was used as *optimization test set* while all the other subsets, together with the topic descriptions, were used as *optimization training set*. The scrap of the split was included into the optimization test set. After training Winnow with $n-1$ subsets, the documents of the remaining subset (optimization test set) were ranked according to their scores. Then, by going down the rank, the threshold value that optimized T9U was found. We performed the cross-evaluation for $n = 2$, 3 and 4, and we took the mean of all $(2+3+4=9)$ optimal threshold values.

4.3 Experiments with LCS

4.3.1 Submitted Runs

We set the Winnow parameters to the values that gave the best results on the FT corpus (Table 4). We use the *thick separator* heuristic [7]: instead of a single threshold θ, a *threshold range* $[\theta^- : \theta^+]$ is used. There is a promotion whenever a relevant document obtains a score below θ^+ and a demotion when a non relevant document gets a score over θ^-. This heuristic achieves a better separation between relevant and non-relevant documents. The asymmetry around the standard threshold (1.0) forces the algorithm to perform more promotions than demotions on the early iterations. This compensates for the asymmetry

parameter	value
Alpha	1.1
Beta	0.9
ThresholdRange	on
θ^+	1.3
θ^-	0.9
MaxIters	30

Table 4: Winnow Parameters.

between the numbers of relevant and non-relevant training documents, speeding up convergence.

We have submitted 2 routing runs, KUNr1 and KUNr2. LCS has originally been developed for *mono-classification* tasks, i.e., each document belongs to exactly one class. This means that the relevant training documents for one class are considered as non-relevant training documents for all other classes. That is certainly not the case in filtering, so we had to do separate runs per topic assuming two classes: relevant and non-relevant. The routing results KUNr1 were produced like this.

The approach of separate runs is correct but obviously inefficient. So, we also tried to process all topics at once (KUNr2), hoping that they do not have relevant documents in common, or even if they do, the impact of this dubious approach on effectiveness would not be that great. Luckily, in the given dataset, it was not: the average uninterpolated precision was practically the same. We obtained 0.237 for KUNr1 and 0.234 for KUNr2.

The batch filtering run KUNb was obtained through the thresholding of the rankings of KUNr1. Thresholding was performed by the cross-evaluation method we described in Section 4.2.

4.3.2 More Runs

The KUNb results, obtained with separate thresholds per topic calculated by cross-evaluation, can be compared with those obtained by a simpler method: a uniform threshold for all topics. We can choose as a uniform threshold any value in the threshold range; such a choice should give the same result if the classification is perfect. But two values are special: 1.0 (average document score before training), and 1.1 (the center of the threshold range).

Table 5 shows that the results for $\theta = 1.0$ are worse than those for $\theta = 1.1$. Moreover, a uniform threshold set at 1.1 gives slightly better results than the separate thresholds computed by cross-evaluation. It seems that the cross-evaluation method has failed, mainly because the training sets had relatively small numbers of relevant training documents. Splitting the sets for cross-evaluation, made the things even worse.

Run	T9U
separate θ's via cross-evaluation (KUNb)	5.0
uniform $\theta = 1.0$	-3.5
uniform $\theta = 1.1$	6.0
best possible thresholdings on KUNr1	17.9

Table 5: Different thresholdings on Winnow.

The best possible thresholdings on the rankings of KUNr1 would have obtained an average T9U of 17.9; not very great either, considering that the largest possible average T9U for the given test set is 104.9. This implies that the rankings achieved are not very good.

5 Overall Comparison and Discussion

In [14] we give the TREC-9 evaluation table of our submitted routing runs with the LCS system. The rightmost column, `FilterIt-r`, corresponds to a supplemental routing run with the FILTERIT system. Obviously, FILTERIT gives better rankings than LCS; the corresponding average uninterpolated precision figures are 0.373 and 0.237. Thresholding the rankings of `FilterIt-r` with the optimal S-D thresholds (as these were estimated by the method described in section 3.5.2) we obtained a (non-adaptive) batch run with FILTERIT. Its results are presented under the label `FilterIt-b` in [13]. An average T9U of 14.8 is obtained in contrast to 5.0 obtained by LCS.

Since `FilterIt-r` and `FilterIt-b` are not *post-factum* optimized, it seems that we should have submitted all runs, for all filtering tasks, with the FILTERIT system. The `FilterIt-r` routing run, with an average precision of 0.373, would have ranked us as second best system; the first system scored at 0.385. The `FilterIt-b` batch run, with an average T9U of 14.8, would have ranked us clearly as the best system; the best official batch run scored at 7.5. The official TREC-9 comparison tables of the tasks we have participated can be found in [15]. At an rate, we are very satisfied with the performance of the FILTERIT system in our official runs. We have clearly achieved the best scores in all adaptive and batch-adaptive tasks optimized for T9U. Compare the 17.3 (KUNa2T9U) and 19.4 (KUNbaT9U) to the 10.7 and 13.6 of the second best systems in the corresponding tasks. The official T9P runs are also satisfactory; our best run has achieved 0.258 (KUNa1T9P), a rather comparable effectiveness to the 0.294 of the best system. After all, we have not optimized exactly for T9P, but for some other related utility measure, in order to simplify the calculations (see Section 3.5.4).

Why are the results with the LCS less satisfactory? According to our experience, Winnow performs better than Rocchio when large numbers (hundreds) of relevant training documents are available for each class. This was not the case in the batch and routing tasks of TREC-9 where some topics had very few relevant training documents. This may largely be responsible for Winnow's weak performance. Furthermore, with 30 iterations in the learning phase, there is some evidence of overtraining.

Why are the results with FILTERIT so good? Let us summarize the methods we have used: accurate and incremental adaptivity as soon as a single training document becomes available (in contrast to re-training in batches), local adaptivity (training documents of decaying value in time), on-the-fly term selection (in contrast to just cutting off classifiers), the S-D threshold optimization (note that we are talking about "optimization" rather than "setting"), and initial query elimination. Moreover, all parameter settings (e.g. Rocchio's α, β, γ, term selection cutoff, half life) have either been empirically determined on the Financial Times collection or at least motivated. There is evidence as well that *Ltu* weighting and query zoning have contributed considerably to effectiveness. The FILTERIT system is a typical example of: *the whole is more than the sum of its parts.*

6 Conclusions and Further Research

In this first-time contribution to TREC, we have focussed mainly on the adaptive tasks. Our contribution to adaptive filtering has been threefold:

- We have investigated the value of retrieved documents as training examples in relation to their time of retrieval. For this purpose, we have introduced the notion of the *half-life* of a training document. The approach has presented promising results.
- We have introduced the *Score-Distributional (S-D) threshold optimization* method, capable of optimizing any effectiveness measure defined in terms of the traditional contingency table. The method has found to be very effective, and it can moreover be applied incrementally.
- We have developed a system that allows *incremental adaptivity*, minimizing its computational and memory requirements without sacrificing too much of accuracy.

In overall, we are very satisfied with our adaptive results; we have clearly achieved the best utility scores in all adaptive and batch-adaptive tasks that we have participated.

The results of the batch and routing tasks are less satisfactory, but at least the feasibility of using the Winnow algorithm in these applications has been demonstrated.

Summarizing, our TREC-9 participation has motivated a great deal of research. As a result, we have finalized the S-D threshold optimization in [4], and we have reconsidered the nature of the filtering task in [1]. Our plans for further research include: finding a way of detecting relevance drifts in order to select appropriate half life values, and to revise the term selection method we have introduced in [3].

Acknowledgments

The first (and main) author would like to thank André van Hameren for the fruitful discussions during the development of the FilterIt system.

References

[1] A. Arampatzis and T. P. van der Weide. Document Filtering as an Adaptive and Temporally-dependent Process. Technical Report CSI-R0103, University of Nijmegen, January 2001.
Available from `http://www.cs.kun.nl/~avgerino`.

[2] A. Arampatzis, T. P. van der Weide, C. H. A. Koster, and P. van Bommel. Linguistically Motivated Information Retrieval. In A. Kent, editor, *Encyclopedia of Library and Information Science*. Marcel Dekker, Inc., New York, Basel, 2000. To appear.
Available from `http://www.cs.kun.nl/~avgerino`.

[3] A. Arampatzis, T. P. van der Weide, C. H. A. Koster, and P. van Bommel. Term Selection for Filtering based on Distribution of Terms over Time. In *Proceedings of RIAO'2000 Content-Based Multimedia Information Access*, pages 1221–1237, Collège de France, Paris, France, April 12–14 2000.
Available from `http://www.cs.kun.nl/~avgerino`.

[4] A. Arampatzis and A. van Hameren. The Score-Distributional Threshold Optimization for Adaptive Binary Classification Tasks. Technical Report CSI-R0105, University of Nijmegen, January 2001.
Available from `http://www.cs.kun.nl/~avgerino`.

[5] C. Buckley, G. Salton, and J. Allan. The Effect of Adding Relevance Information in a Relevance Feedback Environment. In W. B. Croft and C. J. van Rijsbergen, editors, *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 292–300, Dublin, Ireland, June 1994. ACM Press.

[6] J. Callan. Learning While Filtering Documents. In W. B. Croft, A. Moffat, C. J. van Rijsbergen, R. Wilkinson, and J. Zobel, editors, *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 224–231, Melbourne, Australia, August 1998. ACM Press, New York.

[7] I. Dagan, Y. Karov, and D. Roth. Mistake-driven Learning in Text Categorization. In *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing*, 1997.

[8] N. Littlestone. Learning Quickly when Irrelevant Attributes Abound: a New Linear-threshold Algorithm. *Machine Learning*, 2:285–318, 1988.

[9] H. Ragas and C. H. A. Koster. Four Text Classification Algorithms Compared on a Dutch Corpus. In W. B. Croft, A. Moffat, C. J. van Rijsbergen, R. Wilkinson, and J. Zobel, editors, *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 369–370, Melbourne, Australia, August 1998. ACM Press, New York.

[10] J. J. Rocchio. Relevance Feedback in Information Retrieval. In *The SMART Retrieval System — Experiments in Automatic Document Processing*, pages 313–323, Englewood Cliffs, NJ, 1971. Prentice Hall, Inc.

[11] A. Singhal. AT&T at TREC-6. In E. M. Voorhees and D. K. Harman, editors, *The Sixth Text REtrieval Conference (TREC-6)*, pages 215–225, Gaithersburg, Maryland, November 19–21 1997. Department of Commerce, National Institute of Standards and Technology (NIST) Special Publication 500-240.

[12] A. Singhal, C. Buckley, and M. Mitra. Learning Routing Queries in a Query Zone. In N. Belkin, D. Narasimhalu, and P. Willett, editors, *Proceedings of the 20st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 25–32. ACM Press, New York, July 1997.

[13] `http://www.cs.kun.nl/~avgerino/Avi_Arampatzis/publications/TREC9/batchT9U.txt` .

[14] `http://www.cs.kun.nl/~avgerino/Avi_Arampatzis/publications/TREC9/routing.txt` .

[15] `http://www.cs.kun.nl/~avgerino/Avi_Arampatzis/publications/TREC9/averages.txt` .