Unbiased S-D Threshold Optimization, Initial Query Degradation, Decay, and Incrementality, for Adaptive Document Filtering

Avi Arampatzis

Information Retrieval and Information Systems, University of Nijmegen,
Postbus 9010, 6500 GL Nijmegen, The Netherlands.
avgerino@cs.kun.nl, http://www.cs.kun.nl/∼avgerino

Proceedings of the Tenth Text REtrieval Conference (TREC-10).

Abstract

We develop further the *S-D threshold optimization* method. Specifically, we deal with the bias problem introduced by receiving relevance judgements only for documents retrieved. The new approach estimates the parameters of the exponential–Gaussian score density model without using any relevance judgements. The standard *expectation maximization (EM)* method for resolving mixtures of distributions is used. In order to limit the number of documents that need to be buffered, we apply *nonuniform document sampling*, emphasizing the right tail (high scores) of the total score distribution.

For learning filtering profiles, we present a version of Rocchio's method which is suitable and efficient for adaptive filtering. Its main new features are the *initial query degradation* and *decay*, while it is fully *incremental* in query updates and in calculating document score statistics. Initial query degradation eliminates gradually the contribution of the initial query as the number of relevant training documents increases. Decay considers relevant instances (documents and/or initial query) of the near past more heavily than those of the early past. This is achieved by the use of *half-life*, i.e. the age that a training instance must be before it is half as influential as a fresh one in training/updating a profile. All these new enhancements are consistent with the initial motivation of Rocchio's formula.

We, moreover, use a form of *term selection* for all tasks (which in adaptive tasks is applied repeatedly), and *query zoning* for batch filtering and routing.

1 Introduction

This paper describes the participation in the TREC-10 Filtering Track by researchers from the Katholieke Universiteit Nijmegen (KUN). We participated in all three subtasks: adaptive filtering, batch filtering, and routing. The description of the tasks and evaluation measures can be found in [10]. We have mainly used the FilterIt system for all but one routing run which was made by the LCS system. Table 1 summarizes the runs we submitted.

Task	Optimized for	System	Run-tag
adaptive	T10SU	FilterIt	KUNaU
adaptive	F05	FilterIt	KUNaF
batch	T10SU	FilterIt	KUNbU
batch	F05	FilterIt	KUNbF
routing	—	FilterIt	KUNr1
routing	—	LCS	KUNr2

Table 1: TREC-10 filtering runs submitted by KUN.

FilterIt was developed for our TREC-9 participation [2]. It was initially designed as a pure adaptive filtering system, based on a variant of Rocchio's relevance feedback formula which is more suitable for adaptive tasks. It has recently been extended to provide mechanisms for batch training, non-adaptive filtering, and routing. LCS was developed in the context of the Esprit project DOcument ROuting (DORO)[1] [8]. It is based on the Winnow mistake-

[1] http://www.cs.kun.nl/doro

driven learning algorithm [6]. Both systems are described in length in [2]; here we will concentrate on the changes made in FilterIt.

In the next section, the preprocessing applied to documents and topics is described. Section 3 expands on incremental profile training. It is shown how the initial query degradation and decay are integrated into Rocchio's method. Many technical details are given which may be proved useful for developing incremental and effective filtering systems. Section 4 deals with optimizing filtering thresholds. Finally, results are summarized in Section 5, and conclusions are drawn in Section 6.

2 Preprocessing

All tasks were performed using a keyword-based representation of documents and queries, with traditional stemming and stoplisting. There was no special treatment of proper names, all numbers were eliminated, and we made no use of multi-word terms such as phrases or word clusters. We did not use any external resources such as online dictionaries or thesauri. In summary, the pre-processing was quick-and-dirty. It reduces dramatically the number of indexing terms, however, it worked out well with the OHSUMED collection in the TREC-9 Filtering Track where we obtained very good results. This year, however, a programming bug in the preprocessor introduced a serious disadvantage in performing the adaptive tasks: all stems of the test stream which did not occur in the training stream were discarded. As a result, filtering profiles could not be expanded with new terms during adaptations. The impact of this error on the routing and batch filtering effectiveness, however, was negligible.

3 Incremental Profile Training

The query training of FilterIt is derived from Rocchio's method [11]. Our version of the formula presents the following features:

1. It introduces *initial query degradation*. The initial query is considered as carrying a worth of a certain number of relevant documents. As a result, the contribution of an initial query in training a classifier decreases with the number of relevant training documents.
2. It incorporates the notion of the *half life* of a training document, i.e. the age that a document must be before it is half as influential as a fresh one in training a classifier.
3. It allows *accurate incremental training* without using any document buffers, resulting in low memory and computational power requirements.

Table 2 shows all the quantities used for incremental training. They are grouped (from top to bottom) into user-supplied parameters, document stream variables, filter variables, and system-wide parameters.

$\mathbf{Q}_0$	the initial user query vector
a	worth of $\mathbf{Q}_0$, in number of relevant docs
h	half life of training docs
N	total number of docs seen
$\mathbf{DF}_N$	diagonal $K \times K$ matrix of df's @ N
$\mathbf{Q}_n$	query vector after n training docs
$\mathbf{B}_n$	linear combination of relevant docs
$\mathbf{C}_n$	linear combination of irrelevant docs
a_n	worth of $\mathbf{Q}_0$ at time t_n
b_n	accumulated worth of relevant docs
c_n	accumulated worth of irrelevant docs
β'	relevant to irrelevant feedback ratio
k	term selection cutoff

Table 2: Parameters and variables.

The user-supplied parameters are three: the initial query $\mathbf{Q_0}$, its assumed worth a measured in number of relevant documents, and the half life h of training documents measured in actual time. This year, we used $a = 2$, a low value in comparison to last year's tasks where our formula behaved more like $a = 10$. The main reason for this was that we did not find the TREC-10 queries (consisting mainly of one or two keywords) too informative. Furthermore, we used a mild decay (despite the fast-changing nature of a news stream) setting the half life h to 6 months for the adaptive tasks, and no decay for batch and routing (due to the limited timespan of the training data). More about document decay and half life, and a discussion on *convergence* or *responsiveness* of classifiers can be found in [3] and [1].

Training documents $\mathbf{D}_i$, $i = 1, 2, \ldots$ (relevant and non-relevant) are the pre-classified documents given at the time of bootstrapping, and all retrieved ones during filtering since their relevance judgment is given. The index i denotes the position of a training document in time sequence of all training documents, e.g. $\mathbf{D}_1$ is the oldest. $\mathbf{Q}_0$ and $\mathbf{D}_i$ are *not* idf weighted, but only tf and length-normalized. The precise weighting scheme we currently use for documents is the *Lu* [12].

Stream variables are the total number N of documents seen by some point in time, and the $K \times K$ diagonal matrix $\mathbf{DF}_N = \mathbf{diag}(df_1, \ldots, df_K)$ of the document frequencies of a total K terms contained into the N documents. Each new document that arrives increments N by one, and updates $\mathbf{DF}_N$ (i.e. incremental df).

The filter variables are initialized as

$$\mathbf{B}_0 = \mathbf{C}_0 = [0, 0, \ldots, 0], \quad b_0 = c_0 = 0, \quad a_0 = a. \tag{1}$$

Now, let us assume that the system retrieves the nth document $\mathbf{D}_n$ at time t_n, and that the user feedback says that it is relevant. Then the filter variables are updated as

$$\begin{aligned} \mathbf{B}_n &= l_n \mathbf{B}_{n-1} + \mathbf{D}_n, \\ \mathbf{C}_n &= l_n \mathbf{C}_{n-1}, \\ a_n &= l_n a_{n-1}, \\ b_n &= l_n b_{n-1} + 1, \\ c_n &= l_n c_{n-1}, \end{aligned} \tag{2}$$

where l_n is the *decay factor* calculated as

$$l_n = 0.5^{(t_n - t_{n-1})/h}. \tag{3}$$

Similarly, if $\mathbf{D}_n$ is non-relevant, then it is added to $\mathbf{C}_n$ instead of $\mathbf{B}_n$ and c_n is incremented instead of b_n.

The new filtering query $\mathbf{Q}_n$ is then built in 3 steps.

1. The *query zone* is built, i.e. a trained query using only the relevant and discarding all the non-relevant feedback:

$$\mathbf{Q}_{\mathbf{z},n} = \left(\frac{1}{a_n + b_n} (a_n \mathbf{Q}_0 + \mathbf{B}_n) \right) \mathbf{idf}(\mathbf{DF}_N). \tag{4}$$

   (We use the same formula in batch filtering and routing to select which non-relevant documents are going to be used for training: currently, the top-500 non-relevant as ranked by Eq. 4.)

2. All terms in $\mathbf{Q}_{\mathbf{z},n}$ are ranked according to their weight, and only the top-k terms are selected. All the rest of the terms are removed from vectors $\mathbf{Q}_{\mathbf{z},n}, \mathbf{Q}_0, \mathbf{B}_n$, and $\mathbf{C}_n$. This year, we used 250 terms for routing and batch filtering, and 100 for adaptive filtering.

3. The new filtering query is calculated as

$$\mathbf{Q}_n = \beta' \, \mathbf{Q}_{\mathbf{z},n} - \frac{1}{c_n} \mathbf{C}_n \, \mathbf{idf}(\mathbf{DF}_N). \tag{5}$$

The function $\mathbf{idf}(.)$ returns the diagonal matrix of the idf components. We should remind that $\mathbf{Q}_0$ and $\mathbf{D}_i$ are not idf weighted. The idf components are currently calculated with the t formula [12].

Note that for $h = +\infty$ (no document decay), $\beta' = 1$, and $a = 0$ (no initial query), the procedure we have just described calculates the original Rocchio formula. A relevance feedback setting with the traditional parameters α, β, and γ can be simulated using $a_n = b_n \, \alpha/\beta$ and $\beta' = (\alpha + \beta)/\gamma$. In short, our version of the formula can behave like most variants seen in the literature. Additionally, it allows accurate incrementality, it can consider the initial query as being an equivalent of some number of relevant training documents, and it incorporates the notion of decaying over time training documents and initial query. Moreover, it does not invalidate the initial motivation of Rocchio's formula. For example,

$$\frac{1}{c_n} \mathbf{C}_n = \frac{1}{\sum_i d_i} \sum_i d_i \mathbf{D}_i, \tag{6}$$

where $\mathbf{D}_i$ is a non-relevant document, and

$$d_i = \prod_{j=i+1}^{n} l_j = 0.5^{(t_n - t_i)/h}, \tag{7}$$

is still the (weighted) average vector of non-relevant documents.

3.1 Incremental Score Statistics

The incremental training we have just described allows us to calculate (weighted) score statistics, e.g. mean score and variance, incrementally without using any document buffers. Score statistics are necessary for thresholding as we will see in Section 4.

If the dot-product $*$ is used as a scoring function, the mean relevant document score $\mu_{\mathrm{rel},n}$ at t_n is simply:

$$\mu_{\mathrm{rel},n} = \frac{1}{b_n} (\mathbf{B}_n * \mathbf{Q}_n). \tag{8}$$

The variance $\sigma^2_{\mathrm{rel},n}$ can be calculated via

$$\sigma^2_{\mathrm{rel},n} = \mu^{(2)}_{\mathrm{rel},n} - \mu^2_{\mathrm{rel},n} \,. \tag{9}$$

The mean of the squared scores $\mu^{(2)}_{\mathrm{rel},n}$ is given by

$$\mu^{(2)}_{\mathrm{rel},n} = \frac{1}{b_n} \left((\mathbf{Q}_n \, \mathbf{B}_{\mathrm{dyad},n}) * \mathbf{Q}_n \right) \,, \tag{10}$$

where

$$\mathbf{B}_{\mathrm{dyad},n} = \sum_i d_i \, \mathbf{D}_i^{\mathrm{T}} \, \mathbf{D}_i \tag{11}$$

is 2-dimensional matrix. $\mathbf{D}_i^{\mathrm{T}}$ denotes the adjacent of $\mathbf{D}_i$. $\mathbf{B}_{\mathrm{dyad},n}$ can be updated incrementally as

$$\mathbf{B}_{\mathrm{dyad},n} = l_i \, \mathbf{B}_{\mathrm{dyad},n-1} + \mathbf{D}_n^{\mathrm{T}} \, \mathbf{D}_n \,. \tag{12}$$

The derivations of Formulae 8 and 10 can be found in [1].

4 Threshold Optimization

Thresholds in FilterIt are empirically optimized for batch filtering, and S-D (score-distributional) optimized for adaptive tasks.

4.1 The Empirical Method

The *empirical* technique for optimizing a threshold on training documents consists of the following steps: rank the documents according to their scores, calculate the effectiveness measure of interest at every position of the rank, find the position where the measure is optimal, and set the threshold somewhere between the score of the optimal position and the score of the next lower position. The technique works out well given sufficient training data. Our batch filtering runs, KUNbU and KUNbF, use this technique for optimizing thresholds.

The main drawback of the empirical technique becomes apparent when adaptivity is required, namely, it cannot be applied incrementally. A large document buffer should be carried along during filtering, and the scores of all its documents must be recalculated after every query update.

4.2 The S-D Method

The S-D method [2, 4] eliminates the need for a document buffer by using the statistical properties of scores rather than their actual values. Statistical properties like mean score and variance can be updated incrementally, as we have shown in Section 3.1.

The idea behind the S-D threshold optimization is the following. If relevant and non-relevant document scores are modelled separately using their probability densities, then the total score density is a (weighted) mixture of the individual score densities. Having determined the individual score densities and the mixing parameter, all measures that satisfy the *probability thresholding principle* (PTP) [5] can be optimized. The optimization of non-PTP measures requires moreover the knowledge of the number of documents to be filtered, but this is usually an unknown quantity. In such cases, the method can be applied by optimizing the measure only for the near future, e.g. for a certain assumed number of incoming documents.

The procedure has as follows. Let M be an effectiveness measure, a function of the four variables R_+, N_+, R_-, N_- (relevant retrieved, non-relevant retrieved, relevant not retrieved, etc.) of the traditional contingency table. The measure is calculated at m levels $i = 1, \ldots, m$, of decreasing score s_i as

$$M_i = M(R_+(s_i), N_+(s_i), R_-(s_i), N_-(s_i)) \,, \tag{13}$$

where, e.g., $R_+(s_i)$ gives the number of relevant documents that would have been retrieved for a threshold equal to s_i. That is

$$\begin{aligned} R_+(s_i) &= r \int_{s_i}^{s_0} P_{\mathrm{rel}}(x) \, dx & (14) \\ &\approx R_+(s_{i-1}) + r \, (s_{i-1} - s_i) P_{\mathrm{rel}}(s_i) \,, & (15) \end{aligned}$$

where P_{rel} is the probability density function of relevant scores, r is the number of relevant documents, and s_0 is the maximum possible score. The other three variables of the contingency table parameterized by the score can be similarly calculated.

Having calculated the M_i at all levels, the procedure goes on as in the empirical method. Note that Eq. 15 calculates numerically and incrementally a series of integrals. The method is simple to implement and efficient.

4.3 Score Distributions

The S-D optimization requires modelling of the score distributions of relevant and non-relevant documents. In [4] we introduced a numerical method for calculating the probability density of the score distribution of an arbitrary set of documents. The method needs as input the query and what we call *term probability* for each query term. The term probability of a term is simply the fraction of the documents in the set that it occurs in. Thus, the method has the desirable property of depending only on quantities which can be updated incrementally, and it does not need the actual documents. Nevertheless, it is computationally expensive.

In the aforementioned study, we also investigated whether the score distributions can be approximated with known distributions. Assuming that each score is a linear combination of the query weights (e.g. a dot-product), and that relevant documents cluster around some point in the document space with some hyper-ellipsoidal density (e.g. a hyper-Gaussian centered on the point), we proved that the relevant score distribution has a Gaussian Central Limit in a large number of dimensions (query terms). Moreover, we showed that the Gaussian limit appears fast. How fast depends also on the quality of a query; the better the query, the fewer the terms necessary for a Gaussian approximation. Practically, on the OHSUMED collection and for the 63 OHSU queries (TREC-9's data) trained with Rocchio on the 1st year of data, the relevant score distributions can be very well fitted with Gaussians at around 250 query terms.

In the case of the distribution of non-relevant document scores, we have empirically found in [2, 4] that the right tail (high scores) of the score density can be well fitted with an exponential. Further empirical evidence for the proposed Gaussian–exponential score modeling can also be found in [7].

4.4 The Bias Problem

Adaptive filtering presents a bias problem that arises from the fact that relevance judgements become available only for those documents retrieved. The implication of this for thresholding[2] is that calculating the mean score and variance or term probabilities only on documents retrieved can be very misleading.

[2]Note that the bias problem in filtering does not show up only in thresholding, but also in query training/updating.

An attempt to deal with the bias is seen in [13], where each document score is considered together with a *sampling constraint*, i.e. the current threshold at the time of retrieval of the corresponding document. Then the parameters of the Gaussian–exponential model are estimated by maximum likelihood. Although the method calculates unbiased S-D thresholds, it introduces new complications in updating the query. When the query is updated all sampling constraints change as well, nevertheless, there is currently no way of updating the sampling constraints. Abandoning query updates in exchange for a better threshold does not seem like a good solution for adaptive filtering.

4.5 Unbiased S-D: An EM Approach

We have developed another approach which calculates unbiased thresholds while allowing query updates. Since the problem arises from the fact that the relevance judgements are biased, we fit to the total score distribution a mixture model consisting of an exponential and a Gaussian, *without* using any relevance judgements. A standard approach to determining the mixing parameters and the parameters of the component densities is to use Expectation Maximization (EM) [9]. Recovering the parameters of the Gaussian–exponential score model with EM without relevance judgements has recently been described in [7] in the context of distributed retrieval.

Let $P(x|1)$ and $P(x|2)$ be the exponential and Gaussian densities respectively. The total score density can written as

$$P(x) = \sum_j P(j) P(x|j) \,, \tag{16}$$

where $P(j)$ are the mixing parameters satisfying

$$\sum_j P(j) = 1 \,, \quad 0 \leq P(j) \leq 1 \,. \tag{17}$$

The parameters to be estimated are four: the mean μ and variance σ^2 of the Gaussian, the λ of the exponential $P(x|1) = \lambda \exp(-\lambda x)$, and only the one of the two mixing parameters since the other can be determined from Eq. 17.

EM is an iterative procedure. The update equations for the discussed mixture model are:

$$P_{\text{new}}(1) = \frac{\sum_i P_{\text{old}}(1|x_i)}{\sum_i w_i}\,, \tag{18}$$

$$\lambda_{\text{new}} = \frac{\sum_i P_{\text{old}}(1|x_i)}{\sum_i P_{\text{old}}(1|x_i)\, x_i w_i}\,, \tag{19}$$

$$\mu_{\text{new}} = \frac{\sum_i P_{\text{old}}(2|x_i)\, x_i w_i}{\sum_i P_{\text{old}}(2|x_i)}\,, \tag{20}$$

$$\sigma^2_{\text{new}} = \frac{\sum_i P_{\text{old}}(2|x_i)|x_i - \mu_{\text{new}}|^2 w_i}{\sum_i P_{\text{old}}(2|x_i)}\,, \tag{21}$$

where $P_{\text{old}}(j|x)$ is given by Bayes' theorem

$$P_{\text{old}}(j|x) = \frac{P_{\text{old}}(x|j) P_{\text{old}}(j)}{P_{\text{old}}(x)}\,, \tag{22}$$

and $P_{\text{old}}(x)$ is given by Eq. 16.

In general, when all scores x_i have been obtained unconditionally, $w_i = 1, \forall i$. For thresholding purposes, however, we are interested in the right tail (high scores) of the total density. Moreover, in adaptive tasks, more and more scores are accumulated over time. Consequently, in order to reduce the total number of documents the system has to retain and to focus on the tail of the distribution, we apply *nonuniform sampling* of the documents according to their score. If x the score of a document, the document is sampled with probability $P_s(x)$. $P_s(x)$ should be an increasing function, so that more high than low scoring documents are collected. The sampling function we currently use is

$$P_s(x) = \exp\left(\frac{\log 1000}{x_{\max}}(x - x_{\max})\right)\,, \tag{23}$$

where $x_{\max}$ is the maximum score. This sampling function retains most documents with scores near to $x_{\max}$, and only 1 out of a 1,000 documents with zero score.

The 20,000 (approximately) documents of the training set are first sampled like that for every topic. If after the sampling more than 1,000 documents remain, the buffer is further thinned down to 1,000 documents by uniformly (this time) discarding documents. The initial threshold is calculated on the scores of the remaining documents using EM, but now the scores x_i must be weighted as

$$w_i = \frac{1}{P_s(x_i)}\,. \tag{24}$$

As new documents accumulate, every time the buffer reaches the 2,000 documents, it is thinned down to 1,000 documents by random (uniform) removal. Note that if a profile update has taken place, all document scores should be recalculated for a threshold update. This can be computationally heavy for large documents buffers.

EM converges locally, this means that finding a global fit depends largely on the initial settings of the parameters. Initial values for the parameters of the Gaussian and exponential are selected randomly as

$$\mu_{\text{init}} \in [\mu_{\text{rel},n}/2,\, \mu_{\text{rel},n}]\,, \tag{25}$$

$$\sigma^2_{\text{init}} \in [\sigma^2_{\text{rel},n}/4,\, \sigma^2_{\text{rel},n}]\,, \tag{26}$$

$$\lambda_{\text{init}} \in [1/\mu_{\text{half lowest scores}},\, 1/\mu_{\text{all scores}}]\,, \tag{27}$$

where $\mu_{\text{rel},n}$ and $\sigma^2_{\text{rel},n}$ are the biased parameters calculated using the formulae in Section 3.1. The initial mixing parameter is selected as

$$P_{\text{init}}(1) \in [0.5,\, 1 - b_n/N]\,. \tag{28}$$

To find a global fit, EM is run 10 times with initial parameters selected randomly from the ranges above. Then, the fit that has the least squared error with the empirical data is selected.

5 Results

Table 3 summarizes the official results we achieved in TREC-10. The rightmost column shows the final rank of the runs, and the number in parentheses is the total number of runs in the corresponding category submitted by all groups.

Run	T10SU	F05	Av.Prec.	Rank
KUNaU	0.203	0.437	—	12 (30)
KUNaF	0.141	0.356	—	12 (30)
KUNbU	0.307	0.507	—	4 (18)
KUNbF	0.264	0.489	—	8 (18)
KUNr1	—	—	0.136	4 (18)
KUNr2	—	—	0.137	3 (18)

Table 3: TREC-10 results of KUN.

In the routing task, the LCS system has performed very well this year (KUNr2), confirming that its bad performance in TREC-9 was due to the large number of Winnow-iterations that led to over-training. It has achieved a slightly larger average precision than the FilterIt system (KUNr1). According to those results our systems were ranked as the 2nd best.

The batch filtering runs (KUNbU and KUNbF) were performed by FilterIt. We used exactly the same parameter settings as for the routing run KUNr1, except that we thresholded the final document rankings using empirical thresholds estimated on the training set. Ironically, KUNbU optimized for T10SU resulted in larger F05 than KUNbF optimized for F05. The adaptive runs (KUNaU and KUNaF) were performed by FilterIt.

6 Concluding Remarks

Summarizing, we are satisfied with the profile training part of the FilterIt system. It is efficient since it allows incremental training, and it has proved effective as well. LCS and FilterIt are radically different systems, with different learning methods (Rocchio vs. Winnow), and different term selection and weighting schemes. Moreover, LCS did not use the initial queries at all. The fact that two so different systems have achieved similar results implies that we have either reached the "ceiling" of effectiveness for the current pre-processing and representation of the document collection, or the top-1000 documents used in the evaluation were not enough to distinguish between the two systems.

Concerning the threshold optimization for adaptive filtering, we have made a considerable step towards removing the bias introduced by the partial relevance judgements. However, numerous other parameters have been introduced that seem to require extensive tuning in order to achieve good end-results.

We have found EM especially "messy" and difficult to tune. It seems sensitive to the choice of the initial parameter values in converging to a global optimum rather than a local one. The update equations for EM which we have used, do not take into account the relevance judgements available. The available judgements have been used merely for determining usable ranges for initializing the parameters. Note that it may be possible to derive other update equations that will take into account the partial judgements. This may improve the quality of the fit.

Another source of inaccuracies lies onto the document sampling. The current sampling function is certainly not the best that can be used, considering the underlying total score distribution. The number of samples (1,000 to 2,000 max.) used did not seem enough for some topics. However, increasing the size of the document buffer introduces a serious computational overhead in threshold updates since all document scores must be recalculated after profile updates. A reasonable trade-off between threshold accuracy and efficiency has yet to be established.

Despite the "roughness" of these new methods we integrated into thresholding, and the fact that the "bug" in document preprocessing introduced a serious disadvantage into profile updates, our adaptive results ranked FilterIt above the median system.

Acknowledgments

I would like to thank Marc Seutter (KUN) for running KUNr2 with the LCS system, Panos Giannopoulos (University of Utrecht) and Kees Koster (KUN).

References

[1] A. Arampatzis. *Adaptive and Temporally-dependent Document Filtering.* PhD thesis, University of Nijmegen, Nijmegen, The Netherlands, 2001.
Available from `www.cs.kun.nl/~avgerino`.

[2] A. Arampatzis, J. Beney, C. Koster, and T. van der Weide. Incrementality, Half-Life, and Threshold Optimization, for Adaptive Document Filtering. In *The Nineth Text REtrieval Conference (TREC-9)*, Gaithersburg, MD, November 13–16 2000. NIST.

[3] A. Arampatzis and T. van der Weide. Document Filtering as an Adaptive and Temporally-dependent Process. In *Proceedings of the BCS-IRSG European Colloquium on IR Research*, Darmstadt, Germany, April 4–6 2001.

[4] A. Arampatzis and A. van Hameren. The Score-Distributional Threshold Optimization for Adaptive Binary Classification Tasks. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, New Orleans, September 9-13 2001. ACM Press.

[5] D. Lewis. Evaluating and optimizing autonomous text classification systems. In *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, July 1995.

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

[7] R. Manmatha, T. Rath, and F. Feng. Modeling Score Distributions for Combining the Outputs of Search Engines. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, New Orleans, 2001. ACM Press.

[8] 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.

[9] B. D. Ripley. *Pattern Recognition and Neural Networks.* Cambridge University Press, 1996.

[10] S. Robertson and I. Soboroff. The TREC-10 Filtering Track Final Report. In *The Tenth Text REtrieval Conference (TREC-10)*, 2001.

[11] 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.

[12] A. Singhal. AT&T at TREC-6. In *The Sixth Text REtrieval Conference (TREC-6)*, Gaithersburg, MD, November 19–21 1997. NIST.

[13] Y. Zhang and J. Callan. Maximum Likelihood Estimation for Filtering Thresholds. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, New Orleans, 2001. ACM Press.