A Similarity-based Approach to Relevance Learning

Rickard Cöster[1] and **Lars Asker**[2]

Abstract. In several information retrieval (IR) systems there is a possibility for user feedback. Many machine learning methods have been proposed that learn from the feedback information in a long-term fashion. In this paper, we present an approach that builds on user feedback across multiple queries in order to improve the retrieval quality of novel queries. This allows users of an IR system to retrieve relevant documents at a reduced effort.

Two algorithms for long-term learning across multiple queries in the scope of the retrieval system Latent Semantic Indexing have been implemented in a system, REGRESSOR, in order to test these ideas. The algorithms are based on k-nearest-neighbor searching and back propagation neural networks. Training examples are query vectors, and by using Latent Semantic Indexing, the examples are reduced to a fixed and manageable size.

In order to evaluate the methods, we performed a set of experiments where we compared the performance of Latent Semantic Indexing and REGRESSOR. The results demonstrate that REGRESSOR automatically improves on the performance of Latent Semantic Indexing by utilizing the feedback information from past queries.

1 INTRODUCTION

For most people today, the problem is not lack of information, but rather an overload of it. To handle this problem, a number of techniques for efficient information retrieval [5] [15] [16] and text classification [9] [10] have emerged.

One successful technique, for retrieval of relevant documents from large text collections is Latent Semantic Indexing (LSI) [5]. The performance of LSI and other similar retrieval systems can usually be further improved by applying a technique known as relevance feedback [6]. This semi-automatic technique requires a user to explicitly evaluate the relevance of retrieved documents to supply as feedback to the system. Relevance feedback may be seen as a short-term learning mechanism, where the valuable feedback information is lost when the user starts a new query.

We have developed a technique which automatically extends LSI with the functionality of a long-term relevance feedback learner. The technique (hereafter called REGRESSOR) exploits the implicit information acquired from past sessions with the information retrieval system. This information, which represents knowledge about the relationship between past queries and their retrieved documents, is used to improve the system's response to similar future queries.

Several methods have been proposed to make IR systems learn from relevance feedback [3] [4] [17]. One problem is this domain is imposed by the high dimensionality of the feature space. There are several ways to solve this problem. One way is to use e.g Support Vector Machines which handle data sets with large number of features and which has been successfully applied to text classification [9]. Another approach is to reduce the dimensionality of the feature space and then use other learning methods such as Artifical Neural Networks.

We investigate solutions to the problem of learning from relevance feedback in scope of the LSI retrieval system. LSI combines Singular Value Decomposition (SVD) [1] and the Vector Space Model [15] into an efficient retrieval system. The SVD is applied to objects such as documents and terms in the IR system, and LSI has demonstrated equal or better performance than traditional IR techniques [5]. From the machine learning point of view, SVD is a well known technique for reducing the dimensionality of high-dimensional data when working with e.g neural networks [2].

The algorithms implemented in REGRESSOR are based on the hypothesis that queries which are similar in the LSI representation have similar result sets. From this hypothesis, we develop two learning algorithms, based on techniques from nearest neighbor searching and back propagation neural networks. Our training examples consists of query vectors in the LSI representation.

1.1 Latent Semantic Indexing

Latent Semantic Indexing is a fairly new method for automatic indexing and retrieval. It is developed to overcome some of the problems of traditional IR systems, which are based on lexical matching of terms. In such systems a document is retrieved if it contains one or more terms occurring in a query. The problems are that a concept or meaning may be expressed by a number of different terms (*synonymy*) and that words may have many different meanings (*polysemy*).

LSI approaches these problems by estimating the word usage across documents, revealing its underlying semantic structure. The method tends to capture term associations in such a way that terms which occur frequently together are associated [5].

The underlying mathematical operation in LSI is the Singular Value Decomposition (SVD) [1]. The SVD is a way of approximating a rectangular matrix in the least-square sense. The result is a much lower-dimensional space in which all relationships in the original matrix can be approximated using the dot product or cosine measure. The matrix subject for SVD in LSI is the term-document matrix, and each dimension in the new space can be thought of representing common meaning components of many different words and documents [5]. Each term, document and query is represented as a vector in the low-dimensional space.

In terms of effectiveness, the LSI method has demonstrated equal or better performance than the traditional Vector Space Model [5]

[1] Department of Computer and System Sciences, Stockholm University, Electrum 230, SE-164 40 Kista, SWEDEN, email: rick@dsv.su.se

[2] Department of Computer and System Sciences, Stockholm University, Electrum 230, SE-164 40 Kista, SWEDEN, email: asker@dsv.su.se

1.2 Relevance feedback

The relevance feedback process may be viewed as an iterative dialogue between the user and the system, initiated by the user when posing a query. The query is evaluated and the system responds with a ranked list of documents which, according to the system's ordering, best match the query. If the user is not satisfied, s/he may mark one or several documents as relevant and ask the system to refine the search. Again, the system produces a ranked list of documents and this iteration continues until the user stops interacting with the system, either because s/he was content or gave up.

A common way to implement relevance feedback is to construct a query vector which is closer to the relevant, and further from the non-relevant documents as specified by the user. This can be seen as trying to construct an *optimal query*, i.e one which separates the relevant documents from the non-relevant ones. There are several ways of implementing relevance feedback, e.g [12] [13] [14], we will focus on the technique most suited for our work.

One effective method to implement relevance feedback in LSI is to heuristically approximate the optimal query vector. This is carried out by replacing the query vector with the centroid of the vectors representing the relevant documents [6]. This method is defined by equation 1. In the equation, the approximation of the optimal query (hereafter called *improved*) is denoted $q^{'}$ whereas the set $R^{'}$ contains the vectors d_i for the documents determined relevant by the user.

$$q^{'} = \sum_{d_i \in R^{'}} \frac{d_i}{\mid R^{'} \mid} \tag{1}$$

2 DESCRIPTION OF REGRESSOR

The REGRESSOR package implements two algorithms for long-term learning from relevance feedback in LSI. REGRESSOR takes as input a LSI query vector and produces a new, optimized query vector for retrieval. We base our algorithms on regression, as opposed to classification. That is, we do not assign each input to one of a number of discrete classes. The input is instead mapped to a new query vector, with dimension equal to the number of factors used in the SVD.

LSI improves the initial query by replacing it with the centroid of the vectors representing the relevant documents. At this point, the system has gathered much information: the initial query in text format, the initial query vector, the list of relevant documents, the improved query vector, the text in each of the relevant documents etc. REGRESSOR makes use of only the initial query vector q and the improved query vector $q^{'}$. Each training example is thus on the form $\{q, q^{'}\}$.

The two algorithms implemented in REGRESSOR are Nearest Neighbor Regressor (NNR) and Back Propagation Regressor (BPR). The first is based on the k-nearest neighbor learning algorithm [11], the second is based on a feed-forward back-propagation neural network [2] [8]. We model the problem in a similar way in both algorithms, so that it is feasible to compare the experimental results. We use the term *optimize* for the process of generalizing from previously posed queries to improve the performance of a new, unseen query.

2.1 Nearest Neighbor Regressor (NNR)

NNR uses a k-nearest neighbor method which is slightly similar to locally weighted regression [11], a k-nearest neighbor method for regression problems. There are however some differences, the first is that the 'nearness' is measured by the cosine of the angle. The second is the way we weight the contribution of the nearest vectors.

During the training phase, the examples are simply stored. Each example is a tuple $(q, q^{'})$ where q is the initial query vector and $q^{'}$ the improved query vector.

When the user poses a new query, NNR performs a nearest neighbor search with the new query vector against all stored initial query vectors. If there is one or more initial query vectors which is sufficiently 'near' the new query, we have the hit case. Otherwise, we have the miss case and the new query is used as is. In this second case, no optimization is performed.

When the hit case occurs, NNR calculates the optimized query in three steps. The vectors of the k pairs of initial and improved query vectors are first normalized to unit length, as is the new query vector. The centroid of the initial and the centroid of the improved query vectors are then calculated. To obtain the optimized query vector the first centroid is subtracted from the new query vector and the second centroid is added to it.

More formally, let k be the number of neighbors to the new query. Denote the new query vector v, the initial query vectors q_i and the improved query vectors $q_i^{'}$. The optimized query vector $v^{'}$ is calculated by:

$$v^{'} = \frac{v}{|v|} - \frac{1}{k}\sum_{i=1}^{k}\frac{q_i}{|q_i|} + \frac{1}{k}\sum_{i=1}^{k}\frac{q_i^{'}}{|q_i^{'}|} \tag{2}$$

Note that equation 2 has a desirable property when $k = 1$: if the new query is identical to a previous initial query, it will be replaced by its relevance feedback query. When the new query is similar to one or more previous initial query vectors, a combination of the previous initial and improved query vectors is added to it. This combination favors the direction of the improved vectors if the new query vector is close to the nearest neighbors. The resulting vector points more in the direction of the improved queries, and the amount of the change in direction is determined by the difference between the new query and its nearest neighbors.

The vectors are normalized to unit length because it is the angular position which is of interest. If they are not normalized, some vectors may swamp the contribution of others. Furthermore, NNR will not try to optimize every query, the hit case will only occur if there are any vectors in the training set which are sufficiently similar to the new query. A threshold value is used to determine if two vectors are sufficiently similar, a way to determine if there is enough information in the training set to generalize from.

2.2 Back Propagation Regressor (BPR)

The BPR algorithm is based on a combination of the back propagation algorithm [2], [8] and NNR. When training the network, each example is composed by the initial query vector as the input, and the difference between the improved query vector and the initial query vector as the target. Using the terminology from the previous section, the network input vector is q and the network target vector is $q^{'} - q$. Thus, we train the network to learn the difference between an improved query and a new query, given the new query.

If there is no information in the network regarding a new query, it is not reasonable to demand generalization from the network. To alleviate this problem, we set up a criterion which must be obeyed by a new query if it is to be run through the network. The criterion is that there should be at least one example in the training data which is sufficiently similar to the new instance, using cosine as similarity measure.

When given a new query, an exhaustive nearest neighbor search is performed in the training set. If there is at least one query vector which is similar to the new query, it is run through the network, otherwise the query vector is used as is.

After the new query is run through the network, it is optimized in two steps. First, both the new query and the network output are normalized to unit length. Secondly, the new query is added to the network output to form the optimized query.

Using vector notation, let the new query vector be v and the network output o. The optimized query $v^{'}$ is then constructed by

$$v^{'} = \frac{v}{|v|} + \frac{o}{|o|} \tag{3}$$

The input vectors are normalized in the range $(-1.0, 1.0)$ whereas the target vectors are normalized in the range $(0.1, 0.9)$. The input vector normalization is necessary to improved training time, since the vector values are typically rather small. The target vector normalization is necessary since we use a bounded activation function (the sigmoid function).

3 EXPERIMENTAL EVALUATION

In order to investigate the effectiveness of the two methods, we performed a set of experiments. We implemented REGRESSOR in C++ and used an implementation of LSI provided by Telcordia (formerly Bellcore).

3.1 Setup

Three standard IR test collections; CACM, CISI and CRAN, were retrieved from the Glasgow IR group home page (`http://ir.dcs.gla.ac.uk/home/`). Each collection consists of a set of documents, a set of queries and a list describing which documents are considered relevant for each query. Table 1 provides a summary of the collection statistics. The third column (Queries) contains the number of queries with relevance judgments. The last column (Relevant) lists the average number of relevant documents per query.

Table 1. Test collection statistics

Collection	Documents	Queries	Relevant
CISI	1460	76	41.0
CACM	3204	52	15.3
CRAN	1398	225	8.2

After some initial testing, it became clear that the CACM and CISI collections were not very much affected by the algorithms. In fact, in only very few cases were the algorithms able to optimize queries from those collections. This was due to the very small amount of queries with a high cosine similarity measure. Table 2 displays the results of this analysis. The second column shows the number of possible targets for optimization, the third shows the top cosine similarity measure found. Consequently, we decided not to use the CACM and CISI collections, since the training sets turned out to be too small. The CRAN collection however, turned out to be more useful for an experimental evaluation of our algorithms.

For the CRAN collection, the query set was divided into two randomly selected subsets of equal size. The first subset was used for training and the second for testing. This process was repeated 7 times.

Table 2. Collection analysis

Collection	Targets	Top cosine
CISI	5	0.77
CACM	16	0.98
CRAN	55	0.97

Since the purpose of the algorithms is to optimize new queries on the basis of a set of stored queries, only the test data is subject for evaluation. Since we use a threshold value to determine which queries should be subject for optimization, only these particular queries are included in the evaluation. In Table 3 we list the number of optimized queries for each threshold value. The third column in the table lists the average number of queries subject for optimization. The last column lists the percentage of the optimized queries in the test data set.

Table 3. Optimization statistics

Collection	Threshold	Optimized	Percentage
CRAN	0.7	15.4	13.8
CRAN	0.8	8.8	7.9

At first we used all relevant documents to construct the approximation of the optimal queries, as described in [6]. However, we noticed that this did not yield the best performance. The results improved further when we used the top (1-5) relevant documents from the result set of the approximation of the optimal query. The improved queries were constructed using this method.

The NNR algorithm was evaluated using two different threshold values: 0.7 and 0.8. After some experiments, we found that values below 0.7 caused the algorithm to make incorrect judgments with respect to the similarity between queries while values higher than 0.8 caused very few queries to be optimized.

For each threshold value, we set the the maximum number of neighbors to first to 1 and then to 3, making a total of 4 experiments. It should however be noted that not all queries had 3 neighbors above the threshold.

The BPR algorithm was evaluated using the same threshold values (0.7 and 0.8) as for NNR. We performed some initial tests to find a reasonable set of parameter values for the network. First of all, we saw that even though we did not have many examples, we needed more nodes in the hidden layers than in the input or output layers. This should both be due to the fact that the function we wish to approximate is fairly complex and that the network output should be an approximation of real values.

When we used more than one hidden layer the convergence of the network was very slow since the network needed many epochs to recover from local minima. We found that one hidden layer of 200 nodes proved fairly good, both in terms of running speed and results. For the CRAN collection, LSI compressed the term, document and query vectors to 113 dimensions. Therefore we used 113 nodes in the input and output layers of the network.

We experimented with different values on learning rate and momentum constants, and noticed that values closer to 0.0 than to 1.0 were to prefer. We set both the learning rate and the momentum to 0.2, which made the network slowly converge to a small error on the

test set.

Another problem we faced during this experiment was that the stop condition for the neural network was not easy to determine. Therefore, we decided to study the behavior of two different networks to see if they exposed similar behavior. The networks were trained for 10000 epochs each, and we listed the network error at each epoch. Figure 1 plots the network error versus the number of epochs (on a logarithmic scale). Both networks follow the same pattern; a high initial error which quickly gets smaller, and approximately the same error after 1000 epochs. In the figure, the error after 1000 epochs is approximately 4.0, whereas the error at epoch 10000 is close to 0.0001. We saw that the difference in average precision between using the network after 1000 and after 10000 epochs was approximately ± 0.02. In light of this, we decided to evaluate the networks after 1000 epochs.

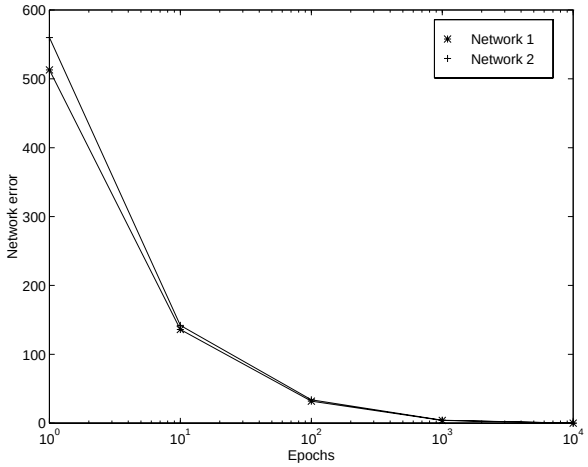

Figure 1. Network error versus number of epochs

3.2 Results

We present the main results as precision-recall graphs in Figures 2 and 3. The graphs were constructed by first calculating precision and recall for all documents in the retrieved list. Precision was then interpolated on recall levels $0.0 \ldots 1.0$ by taking the precision at some recall level equal to the maximum precision at that and all higher levels.

In Figure 2 we used a threshold value of 0.7, in Figure 3 we used 0.8. NNR(1) and NNR(3) refer to the Nearest Neighbor Regressor algorithm using 1 and max 3 neighbors, whereas BPR refer to the Back Propagation Regressor method. The results obtained by using no optimization at all (LSI) is also included. The four curves (LSI, BPR, NNR(1) and NNR(3)) are the results of using completely *automatic* methods, and they show the performance of the initial user query.

For comparison, we have also included the maximal precision that can be obtained by the relevance feedback mechanism (RF) described in section 1.2. Recall that this performance is based on perfect information about all 1398 documents returned by a query. In reality however, an ordinary human user will be able to give an accurate relevance score of only 5 or 10 of the top ranked documents. Hence, performance using relevance feedback for an ordinary user, would be more comparable with that of NNR or BPR (depending on the number of documents that the user has the patience to rate). The NNR and BPR methods on the other hand, improve on the original query automatically, based on their knowledge about the relevance of (similar) past queries.

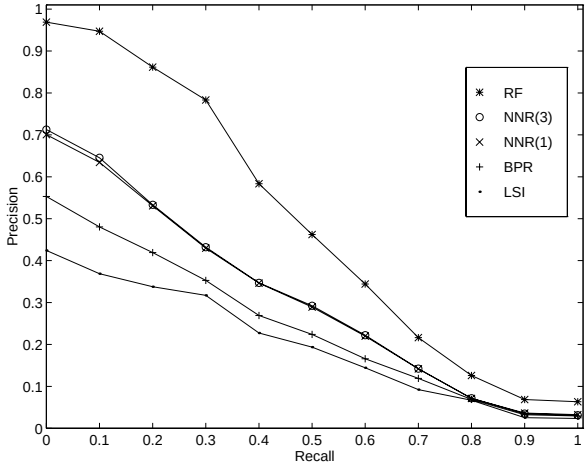

Figure 2. Precision/Recall using threshold 0.7

From the figures, it is clear that the methods outperforms the LSI search technique. Furthermore, the nearest neighbor method also performs better than the back propagation method.

The difference between using 1 neighbor and max 3 neighbors is very small. We noticed that 3 neighbors were only used twice (1.8% of all optimized queries) in the test runs with threshold 0.8. 2 neighbors were only used 11 times (10.1%). The test data was therefore not very much clustered, in terms of the cosine measure.

The figures show only a small difference between using 0.7 and 0.8 as threshold value, in terms of effectiveness. But using 0.7 caused twice as many queries to be optimized, without considerably degrading the results.

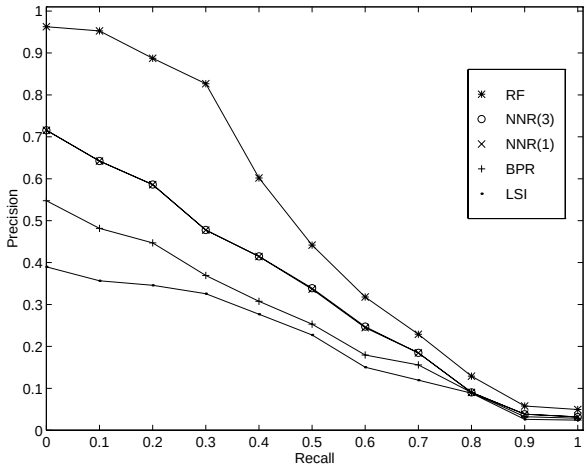

Figure 3. Precision/Recall using threshold 0.8

It is interesting to see that the nearest neighbor method performs much better than the back propagation method. This can be due to a number of reasons, perhaps the most likely that we were incapable

of finding the best network topology. There is also an error source involved in the network algorithm; the test data vectors are normalized according to the minimum and maximum vector values in the training set.

In terms of effectiveness, the methods performed better than the original LSI query. However, the number of optimized queries were rather small. The knowledge used from previous queries is also very specific (or vertical) since one neighbor was mostly used for generalization. The results also indicate that the less complex solution (nearest neighbor) is capable of better results than the complex one (neural network).

4 SUMMARY AND CONCLUSIONS

We have presented results regarding two learning algorithms for improving previously unseen user queries in Latent Semantic Indexing.

The proposed methods are based on two different learning schemes from the area of machine learning. The first method, Nearest Neighbor Regressor, uses a nearest neighbor search of the training set to find queries similar to the initial query. If found, the initial query is then optimized by changing its direction to be more similar to that of the improved queries.

The second method, Back Propagation Regressor, implements a back propagation neural network and is trained to learn the difference between an improved query and its initial query. When a new query is found similar to at least one of the queries in the training set, it is optimized by adding the network output to itself.

In order to evaluate the methods, we retrieved three standard IR test collections. We found that there were only a few queries in the first two collections with high similarity values, according to the similarity measure we chose. We evaluated the methods on the third test collection, using a standard measure of effectiveness.

The experiments revealed an array of results. First, we found that the methods improved new queries considerably. About 15% of the queries in the test set were improved. Secondly, we found that the nearest neighbor method outperformed the back propagation neural network. This, we believe, was mainly because it was difficult to find the optimal parameters for the network. We also found that using the 1-5 top-ranked documents from the result list of the improved query during training generally increased the effectiveness of both methods. It thus seems like clustered documents are preferred as training examples by our algorithms.

We have shown that relevance feedback in Latent Semantic Indexing can be improved by using information implicitly represented in feedback from previous queries. Most techniques for relevance feedback today use information that is collected from the user at the time of posing new queries to an information retrieval system. This is often time-consuming and requires an unneccessarily big effort from the user. It also means that collected information is lost at the end of a query session. As an alternative, we have developed a method that uses knowledge about the user's preferences as they have appeared in earlier sessions. By being able to learn from past sessions, our system minimizes the effort by the user while at the same time improves on the relevance of retrieved documents.

5 FUTURE WORK

We are currently exploring the usefulness of the methods in the scope of much larger text collections. The critical factor would be whether the SVD can be computed for the collection or not, and if the vector dimensions then are small enough.

The dimension of a query vector in LSI is equal to the number of singular values one chooses in the SVD operation. Tests with LSI at the TREC conference [7] have demonstrated that even with collections of hundreds of thousands of documents, only a few hundred singular values are required for good performance. Thus, it is reasonable to believe that for collections with more documents there will not be a tremendous increase in the number of dimensions.

Algorithms for computing the truncated SVD are getting faster all the time. However, the complexity of the current algorithms is fairly high. On the other hand there is no need to compute the truncated SVD of the entire term-document matrix. A carefully selected sample of the document set can be used to construct the LSI space. The rest of the document set can then be *folded-in* to the space [1], in a manner similar to how the query vector is represented.

Another interesting direction for future work is to investigate how the proposed methods can be modified to work for a group of users with similar frames of reference and interests. In such a scenario the common feedback from all users in the group will be available to optimize new queries. This is a scenario where individual efforts (in terms of giving feedback to the system) will be minimized while the benefits (in terms of being presented less irrelevant documents) will be maximized. We believe that this is the environment in which the suggested methods will prove to be most useful and make a difference for the next generation of information retrieval systems.

REFERENCES

[1] M.W. Berry, S.T. Dumais, and G.W. OBrien, 'Using linear algebra for intelligent information retrieval', *SIAM Review*, (1995).

[2] C.M. Bishop, *Neural Networks for Pattern Recognition*, Oxford University Press, 1995.

[3] H. Chen, 'Machine learning for information retrieval: Neural networks, symbolic learning, and genetic algorithms', *Journal of the American Society for Information Science*, (1995).

[4] F. Crestani, 'Learning strategies for an adaptive information retrieval system using neural networks', in *IEEE International Conference on Neural Networks*, (1993).

[5] S. Deerwester, S.T. Dumais, T.K. Landauer, and R. Harshman, 'Indexing by latent semantic analysis', *Journal of the American Society for Information Science*, (1990).

[6] S.T. Dumais, 'Improving the retrieval of information from external sources', *Behavior Research Methods, Instruments and Computers*, (1991).

[7] S.T Dumais, 'Latent semantic indexing (lsi): Trec-3 report', in *NIST Special Publication 500-226: Overview of the Third Text REtrieval Conference (TREC-3)*, (1995).

[8] J. Hertz, A. Krogh, and R. Palmer, *Introduction to the Theory of Neural Computation*, Addison-Wesley, 1991.

[9] T. Joachims, 'Text categorization with support vector machines: Learning with many relevant features', in *Proceedings of the European Conference on Machine Learning*, (1998).

[10] D. Lewis, R. Schapire, J. Callan, and R. Papka, 'Training algorithms for linear text classifiers', in *International ACM SIGIR Conference on Research and Development in Information Retrieval*, (1996).

[11] T.M. Mitchell, *Machine Learning*, McGraw Hill, 1997.

[12] S. Robertson and K. Sparck-Jones, 'Relevance weighting of search terms', *Journal of the American Society for Information Science*, (1976).

[13] J. Rocchio, *The SMART Retrieval System: Experiments in Automatic Document Processing*, chapter 313–323, Prentice-Hall, 1971.

[14] G. Salton and C. Buckley, 'Improving retrieval performance by relevance feedback', *Journal of the American Society for Information Science*, (1990).

[15] G. Salton and M.J. McGill, *Introduction to Modern Information Retrieval*, McGraw Hill, 1983.

[16] C.J van Rijsbergen, *Information Retrieval*, Butterworth, 1979.

[17] C.C. Vogt, G.W. Cottrell, R.K. Belew, and B.T. Bartell, 'User lenses - achieving 100 % precision on frequently asked questions', Technical report, UCSD CSE, (1997).