Applying Latent Semantic Indexing on the TREC 2010 Legal Dataset

Andy Garron and April Kontostathis
Department of Mathematics and Computer Science,
Ursinus College, Collegeville PA 19426, akontostathis@ursinus.edu

Abstract—We applied both Latent Semantic Indexing (LSI) and Essential Dimensions of LSI (EDLSI) to the 2010 TREC Legal Learning task. This year the Enron email collection was used and teams were given a list of relevant and a list of non-relevant documents for each of the eight test queries. In this article we focus on our attempts to incorporate machine learning into the LSI process. We show the EDLSI continues to outperform LSI on large datasets.

For 2011 we plan to enhance our system by adding parallel and distributed approaches to LSI and EDLSI. We believe our retrieval performance would be improved if we could process more dimensions. Our current system resources limited us to 70 dimensions this year. Even with 70 dimensions our system performance was greater than or equal to the median for 6 of the 8 queries on the F1 metric.

1 Introduction

In the 2009 TREC Legal Competition, which used the tobacco lawsuit collection, we tested a new matrix factorization for Latent Semantic Indexing (LSI). The system we used relied on MATLAB for the matrix operations, and this severely restricted our ability to process the entire term-document matrix as a unit. As a result, we relied on distributed techniques for processing the collection, and we had no way of comparing our distributed techniques to a run of the entire corpus.

For 2010 we have redeveloped and restructured our system for performance reasons, and instead of MATLAB we used a combination of CLAPACK and SVDLIBC for our matrix operations. Indexing and term weighting were provided by Lemur. We describe the process we used in section 3. Due to time constraints we were unable to develop a parallel version of this system to compare to the sequential version. That will be our primary goal for 2011.

Teams this year were permitted three runs. Our first run was a standard LSI run with the maximum number of dimensions we could process. Our second two runs used EDLSI instead of LSI, because EDLSI has been shown to work as well as or better than LSI with many fewer dimensions [10].

2 Background

In this section we begin with a description of vector-space retrieval, which forms the foundation for Latent Semantic Indexing (LSI). We also present a brief overview of LSI [5]. We discuss Essential Dimensions of LSI (EDLSI) and its improvements over LSI.

2.1 Vector-Space Retrieval

In vector-space retrieval, a document is represented as a vector in t-dimensional space, where t is the number of terms in the lexicon being used. If there are d documents in the collection, then the vectors representing the documents can be represented by a matrix $A \in \Re^{t \times d}$, called the *term-document matrix*. Entry $a_{i,j}$ of matrix A indicates how important term i is to document j, where $1 \leq i \leq t$ and $1 \leq j \leq d$.

The entries in A can be binary numbers (1 if the term appears in the document and 0 otherwise), raw term frequencies (the number of times the term appears in the document), or weighted term frequencies. Weighting can be done using either local weighting, global weighting, or a combination of both. The purpose of local weighting is to capture the relative importance of a term within a specific document; therefore, local weighting uses the frequency of the term within the document to calculate the weight and assigns a higher weight if the frequency is higher. The purpose of global weighting is to identify terms that discriminate effectively between documents; thus, global weighting uses the frequency of the term within the entire document collection to calculate the weight and assigns a higher weight if the frequency is lower. Because document size often varies widely, the weights are also usually normalized; otherwise, long documents are more likely to be retrieved. See, e.g., [2], [14] for a comprehensive discussion of local and global weighting techniques. In our experiments we used tf-idf weighting and cosine normalization.

Common words, such as *and*, *the*, *if*, etc., are considered to be *stop-words* [2] and are not included in the term-document matrix. Words that appear infrequently are often excluded to reduce the size of the lexicon.

Like the documents, queries are represented as t-dimensional vectors, and the same weighting is applied to them. Documents are retrieved by mapping $\mathbf{q}$ into the row (document) space of the term-document matrix, A:

$$\mathbf{w} = \mathbf{q}^T A.$$

After this calculation, $\mathbf{w}$ is a d-dimensional row vector, entry j of which is a measure of how relevant document j is to query $\mathbf{q}$. In a traditional search-and-retrieval application, documents are sorted based on their relevance score (i.e., vector $\mathbf{w}$) and returned to the user with the highest-scoring document appearing first. The order in which a document is retrieved is referred to as the *rank*[1] of the document with respect to the query. The experiments in this paper run multiple queries against a given dataset, so in general the query vectors $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_n$ are collected into a matrix $Q \in \Re^{t \times n}$ and their relevance scores are computed as

$$W = Q^T A,$$

where entry $w_{j,k}$ in $W \in \Re^{n \times d}$ is a measure of how relevant document j is to query k.

There are two immediate deficiencies of vector-space retrieval. First, W might pick up documents that are not relevant to the queries in Q but contain some of the same words. Second, Q may overlook documents that are relevant but that do not use the exact words being queried. The *partial singular value decomposition* (PSVD) that forms the heart of LSI is used to capture term relationship information in the term-document space. Documents that contain relevant terms but perhaps not exact matches will ideally still end up 'close' to the query in the LSI space [5].

2.2 Latent Semantic Indexing

LSI uses the PSVD to approximate A, alleviating the deficiencies of vector-space retrieval described above.

The PSVD, also known as the truncated SVD, is derived from the SVD. The (reduced) SVD decomposes the term-document matrix into the product of three matrices: $U \in \Re^{t \times r}$, $\Sigma \in \Re^{r \times r}$, and $V \in \Re^{d \times r}$, where r is the rank of the matrix A. The columns of U and V are orthonormal, and Σ is a diagonal matrix, the diagonal entries of which are the r non-zero singular values of A, customarily arranged in non-increasing order. Thus A is factored as

$$A = U \Sigma V^T.$$

The PSVD produces an optimal rank-k ($k < r$) approximation to A by truncating Σ after the first (and largest) k singular values. The corresponding columns from $k+1$ to r of U and V are also truncated, leading to matrices $U_k \in \Re^{t \times k}$, $\Sigma_k \in \Re^{k \times k}$, and $V_k \in \Re^{d \times k}$. A is then approximated by

$$A \approx A_k = U_k \Sigma_k V_k^T.$$

In the context of LSI, there is evidence to show that A_k provides a better model of the semantic structure of the corpus than the original term-document matrix was able to provide for some collections [5], [6], [7], [3]. For example, searchers may choose a term, t_1, that is synonymous with a term, t_2, that appears in a given document, d_1. If k is chosen appropriately and there is ample use of the terms t_1 and t_2 in other documents (in similar contexts), the PSVD will give t_1 a large weight in the d_1 dimension of A_k even though t_1 does not appear in d_1. Similarly, an ancillary term t_3 that appears in d_1, even though d_1 is not 'about' t_3, may well receive a lower or negative weight in A_k matrix entry (t_3, d_1).

Choosing an optimal LSI dimension k for each collection remains elusive. Traditionally, an acceptable k has been chosen by running a set of queries with known relevant document sets for multiple values of k. The k that results in the best retrieval performance is chosen as the optimal k for each collection. Optimal k values are typically in the range of 100–300 dimensions [6], [12].

2.3 Essential Dimensions of LSI

As the dimension of the LSI space k approaches the rank r of the term-document matrix A, LSI approaches vector-space retrieval. In particular, vector-space retrieval is equivalent to LSI when $k = r$.

Figures 1–3 show graphically that the performance of LSI may essentially match (or even exceed) that of vector-space retrieval even when $k \ll r$. For the CACM [15] and NPL [8] collections, we see that LSI retrieval performance continues to increase as additional dimensions are added, whereas retrieval performance of LSI for the MED collection peaks when $k = 75$ and then decays to the level of vector-space retrieval. Thus we see that vector-space retrieval outperforms LSI on some collections, even for relatively large values of k. Other examples of collections that do not benefit from LSI can be found in [11] and [9].

These data suggest that we can use the term relationship information captured in the first few SVD vectors, in combination with vector-space retrieval, a technique referred to as Essential Dimensions of Latent Semantic Indexing (EDLSI). Kontostathis demonstrated good performance on a variety of collections by using only the first 10 dimensions of the SVD [10]. The model obtains final document scores by computing a weighted average of the traditional LSI score using a small value

1. This rank is unrelated to the *rank* of a matrix mentioned below.

Fig. 1. LSI vs. vector-space retrieval for the CACM Corpus ($r = 3\,204$).

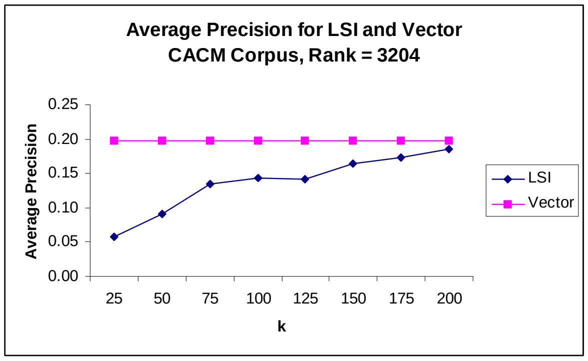


Fig. 2. LSI vs. vector-space retrieval for the NPL Corpus ($r = 6\,988$).

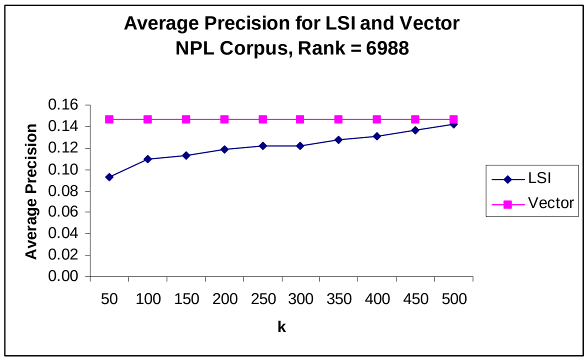
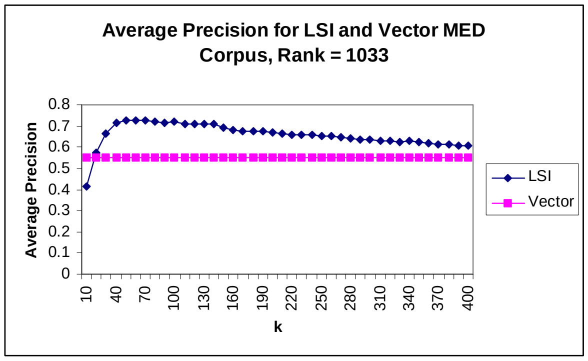


Fig. 3. LSI vs. vector-space retrieval for the MED Corpus ($r = 1\,033$).



for k and the vector-space retrieval score. The result vector computation is

$$W = x(Q^T A_k) + (1 - x)(Q^T A),$$

where x is a weighting factor ($0 \leq x \leq 1$) and k is small.

In [10], parameter settings of $k = 10$ and $x = 0.2$ were shown to provide consistently good results across a variety of large and small collections studied. On average, EDLSI improved retrieval performance an average of 12% over vector-space retrieval. All collections showed significant improvements, ranging from 8% to 19%. Significant improvements over LSI were also noted in most cases. LSI outperformed EDLSI for $k = 10$ and $x = 0.2$ on only two small datasets, MED and CRAN. It is well known that LSI happens to perform particularly well on these datasets. Optimizing k and x for these specific datasets restored the outperformance of EDLSI.

Furthermore, computation of only a few singular values and their associated singular vectors has a significantly reduced cost when compared to the usual 100–300 dimensions required for traditional LSI. EDLSI also requires minimal extra memory during query run time when compared to vector-space retrieval and much less memory than LSI [10].

3 Methodology

The dataset for the 2010 TREC Legal Track Learning Task contains a large volume of information, approximately 700,000 documents in many different formats (.txt, .pst, .ppt, etc.) that take up 3.71 gigabytes of disk space. Each document needed to be added to an index in order to create the term-document matrix for the dataset. The Lemur Toolkit was used to index the collection [13]. The Lemur Toolkit is designed to take a set of documents formatted in a certain way and add them quickly to an index that can then be used to glean information about the dataset as a whole, as well as information about each document.

All terms that appeared in more than 100,000 documents were considered bad discriminators between documents and were removed from the index. Terms that included numeric characters or other characters not in the alphabet were also eliminated. Documents which contained no indexed terms after parsing were considered to be 'blank' and were removed from further processing. After pruning our term-document matrix size was 302,119 (terms) by 456,968 (documents). Pruned documents were added to our run submission file, with a probability of zero, to meet the requirements for submission.

After a final term-document matrix was created for the TREC dataset, the LSI process began. We evaluated several different libraries used to perform operations on dense vectors and matrices to determine which would adequately serve our needs. This was an important decision because LSI requires two large matrix multiplications for every query. We chose CLAPACK (a C library designed to provide the FORTRAN LAPACK library functionality in a C/C++ environment) [1] to handle the dense matrix calculations. Although CLAPACK was also our first choice for the calculation of the SVD, this proved ineffective, as CLAPACK does not work with sparse matrices. SVDLIBC was used for the decomposition and it provided much-increased speed and functionality for working with the sparse term-document matrix [4].

3.1 Optimizing k

We were permitted three runs for submission to the TREC Legal learning task. We decided to use one submission using LSI, and another two using EDLSI with different k choices. The k's that were chosen were determined by a simple scoring algorithm: given that we know a subset of the relevant documents and a subset of the irrelevant documents, we can ignore the rest of the documents in our result set, and count up the known-relevant documents that are listed below the cutoff for relevant documents (and the known-irrelevant documents that are listed above the cutoff for irrelevant documents). The sum is the score of a query result, and the higher the sum, the worse the result. The results that scored the best on average across all topics were the k-selections of 35 and 70. Due to memory restrictions on our research computer, we could not select a k value greater than 70.

4 Results and Discussion

The results from our runs are shown in Figures 4 through 9. There are some interesting trends and there is also a lot of room for improvement.

Figure 4 compares the F1 metric for our runs with the best and the median F1 metrics. Using this measure, we did reasonably well, with our EDLSIK70 run meeting or exceeding the median for 6 of the 8 queries. EDLSIK70 outperforms both of our other runs, showing that EDLSI outperforms LSI on this large collection.

Figure 5 tells the opposite story. The Hypothetical F1 would be the F1 if the best cutoff was chosen. This metric is a measure of our ranking. Our runs performed lower than the median on 6 of the 8 queries. EDLSI, however, is still showing improvement over LSI, and a lower k value ($k = 35$ vs $k = 70$) is better for EDLSI.

Figure 6 and 7 are measures of our ability to predict the probability of relevance for each document. Again, we show results better than the median on most of the queries. Generally, our system had a problem with queries 202 and 205. More analysis will be needed to determine the features of these requests that make them so difficult for us. All of our runs were fully-automated, and terms were simply extracted from the short description of the request to generate our query vector. More carefully chosen terms may improve the results dramatically (but would also mean that our run would be semi-automated instead of fully-automated).

Figures 8 and 9 seem to provide conflicting information. The cutoff accuracy figures were pretty reasonable, doing better than the median on 6 of the 8 requests, but the cutoff values seem to be way out of line. We most often have suggested retrieval of many more documents than were needed (17 of 24 instances), and we were often orders of magnitude off the mark (ex. LSI for request 207). Interestingly we consistently under-estimated the number of documents for 205. This probably has less to do with the cutoff assignment, however, than the actual retrieval quality for that request.

5 Conclusion and Future Directions

We have developed a system that uses LSI and EDLSI on the ENRON collection. Our 2010 run results show that EDLSI outperforms LSI. However, there were two requests that did not perform well at all. We will begin 2011 with an analysis of these queries. We are curious to see if simply modifying (increasing) k can improve our results. We also are planning to do some testing that would optimize k for each request (rather than system-wide, as is typically done). Other directions for future research are noted below.

k-optimization: The selection of k in LSI and EDLSI (the extent to which we use the SVD) is a very important choice to make. If k is too low, the LSI portion of the results will be inaccurate. If k is too high, computation time increases substantially, and the effect of the approximation of the term-document matrix is mitigated; if k = total number of singular values then LSI and EDLSI are equivalent to vector-space retrieval. Finding an optimal k for a dataset is an area of ongoing research. Given a dataset where we know some of the relevant documents, we can run the query with multiple k values and select which one returns the most of the relevant documents as the optimal k. We were limited by memory resources in our current system, but should be able to overcome these limitations when we implement a parallel version of our system.

Selective Query Expansion: Knowing a handful of relevant documents for a query is useful in that if they are all relevant to one thing, they must be similar in at least one way. A method of reflecting that in a query is to find significant terms that co-occur in most or all of the documents and add them to our query. The mentality is that if document set A is relevant to a query and each member is similar to each other member, other sets of documents relevant to that query could also be similar to the members of A.

Automatic Query Expansion: Query vectors in the LSI sense are represented in the same form as document vectors. If we know a set of relevant documents for a general topic, it should be possible to create query vectors from those documents and run queries using each of those. Multiple result sets will be returned, and in

Fig. 4. TREC Results Using F1 Metric

F1 Scores

Topic	**TREC best**	**TREC median**	**URSK35T**	**URSK70T**	**URSLSIT**
200	20	2.1	2.1	**2.8**	**2.8**
201	30.9	5	1.5	**5.1**	3.7
202	80.1	16.9	**3.1**	1.6	1.8
203	32.5	11.3	5.5	**11.3**	**11.3**
204	26	3.9	3.2	**8.5**	2.3
205	51.2	31	3.4	**7.0**	**7.0**
206	6.1	2.8	2.8	**4.7**	**4.7**
207	89.9	11.4	8.1	**11.4**	3.8

Fig. 5. TREC Results Using Hypothetical F1 Metric

Hypothetical F1 Scores

Topic	**TREC best**	**TREC median**	**URSK35T**	**URSK70T**	**URSLSIT**
200	25.8	8.9	**8.7**	7.4	7.4
201	72.9	17.1	**11.7**	9.3	11.1
202	88.5	49.9	**13.6**	10.0	7.5
203	63.6	30.4	**15.3**	14.8	14.8
204	26.6	13.3	5.3	15.6	**20.9**
205	57.4	48.8	17.9	**18.9**	**18.9**
206	35.7	11.7	**21.1**	20.4	20.4
207	90.3	19.1	11.5	14.1	**16.3**

order to know the overall theme shared by each of those, an average of a document's score for each query could be taken and used as the final result for that document's relevancy to the general topic. This idea presents the most interesting possibilities due to its flexibility. A 'best estimate' query for the general topic could be used to try and perform k-optimization for the dataset first. Also, if we know a set of documents that are irrelevant to the general topic, as is the case with the TREC task, we could use those documents as queries and reject documents that have high similarities to them.

Query Analysis: Analysis of which topics EDLSI or LSI failed on and the characteristics of those topics could lead to a better machine learning process; perhaps we can find an attributed that can be used to determine which algorithm to use based on the nature of the query and the document set.

6 Acknowledgements

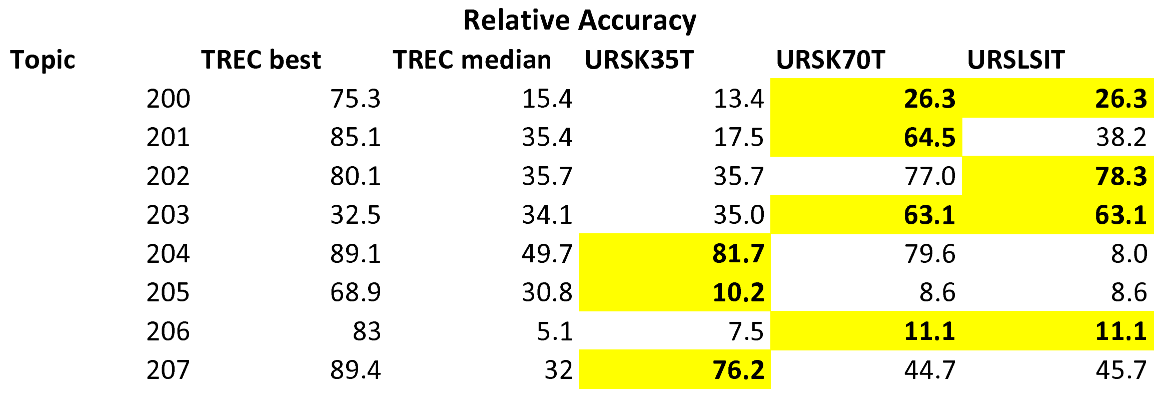
This material is based upon work supported by the National Science Foundation under Grant No. 1003972. Additional funding was provided by the Ursinus Summer Fellows program. We also thank Dr. Raymond J. Spiteri and Erin Moulding from the University of Saskatchewan for their assistance with understanding the Matlab code and providing background on the project from prior years.

References

[1] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999.

[2] R. A. Baeza-Yates, R. Baeza-Yates, and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., 1999.

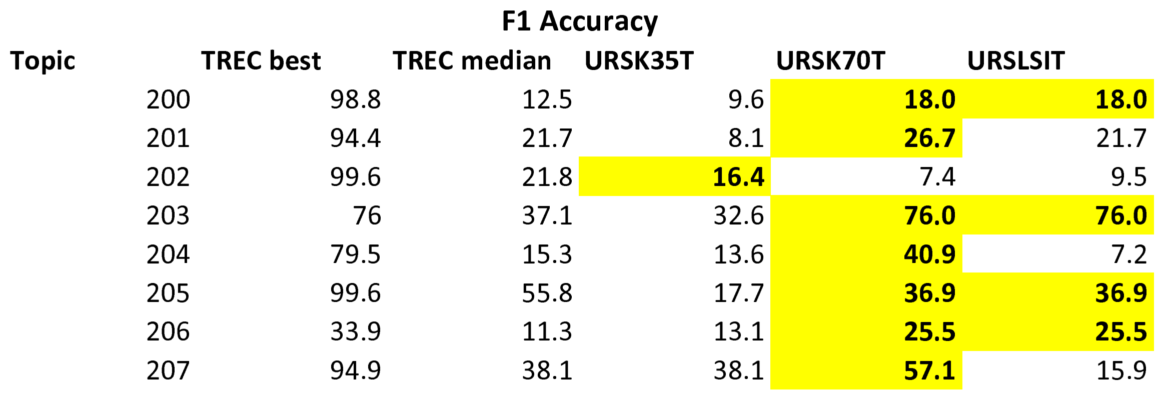
Fig. 6. TREC Results - Relative Accuracy



Relative Accuracy

Topic	**TREC best**	**TREC median**	**URSK35T**	**URSK70T**	**URSLSIT**
200	75.3	15.4	13.4	**26.3**	**26.3**
201	85.1	35.4	17.5	**64.5**	38.2
202	80.1	35.7	35.7	77.0	**78.3**
203	32.5	34.1	35.0	**63.1**	**63.1**
204	89.1	49.7	**81.7**	79.6	8.0
205	68.9	30.8	**10.2**	8.6	8.6
206	83	5.1	7.5	**11.1**	**11.1**
207	89.4	32	**76.2**	44.7	45.7

Fig. 7. TREC Results - F1 Accuracy



F1 Accuracy

Topic	**TREC best**	**TREC median**	**URSK35T**	**URSK70T**	**URSLSIT**
200	98.8	12.5	9.6	**18.0**	**18.0**
201	94.4	21.7	8.1	**26.7**	21.7
202	99.6	21.8	**16.4**	7.4	9.5
203	76	37.1	32.6	**76.0**	**76.0**
204	79.5	15.3	13.6	**40.9**	7.2
205	99.6	55.8	17.7	**36.9**	**36.9**
206	33.9	11.3	13.1	**25.5**	**25.5**
207	94.9	38.1	38.1	**57.1**	15.9

[3] M. W. Berry, S. T. Dumais, and G. W. O'Brien. Using linear algebra for intelligent information retrieval. *SIAM Review*, 37(4):575–595, 1995.

[4] D. Rohde. http://tedlab.mit.edu/ dr/svdlibc/, accessed July 2010.

[5] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman. Indexing by Latent Semantic Analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.

[6] S. T. Dumais. LSI meets TREC: A status report. In D. Harmon, editor, *The First Text REtrieval Conference (TREC1), National Institute of Standards and Technology Special Publication 500-207*, pages 137–152, 1992.

[7] S. T. Dumais. Latent semantic indexing (LSI) and TREC-2. In D. Harmon, editor, *The Second Text REtrieval Conference (TREC2), National Institute of Standards and Technology Special Publication 500-215*, pages 105–116, 1993.

[8] Glasgow Test Collections. http://www.dcs.gla.ac.uk/idom/ir_resources/test_collections/, 2009.

[9] E. R. Jessup and J. H. Martin. Taking a new look at the latent semantic analysis approach to information retrieval. In *Computational Information Retrieval*, pages 121–144, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics.

[10] A. Kontostathis. Essential dimensions of latent semantic indexing (lsi). *Proceedings of the 40th Hawaii International Conference on System Sciences - 2007*, 2007.

[11] A. Kontostathis, W. M. Pottenger, and B. D. Davison. Identification of critical values in latent semantic indexing. In T.Y. Lin, S. Ohsuga, C. Liau, X. Hu, and S. Tsumoto, editors, *Foundations of Data Mining and Knowledge Discovery*, pages 333–346. Spring-Verlag, 2005.

[12] T. A. Letsche and M. W. Berry. Large-scale information retrieval with Latent Semantic Indexing. *Information Sciences*, 100(1-4):105–137, 1997.

[13] P. Ogilvie and J. Callan. Experiments using the Lemur toolkit. In *The Tenth Text REtrieval Conference (TREC2001), National Institute of Standards and Technology Special Publication 500-207*.

[14] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Inf. Process Manage.*, 24(5):513–523, 1988.

[15] SMART. ftp://ftp.cs.cornell.edu/pub/smart/, 2009.

Fig. 8. TREC Results - Cutoff Accuracy

		Cutoff Accuracy			
Topic	**TREC best**	**TREC median**	**URSK35T**	**URSK70T**	**URSLSIT**
200	90.4	11.6	2.5	**11.4**	**11.4**
201	84.8	11.4	2.3	**26.2**	9.4
202	86.8	23.1	14.9	44.8	**54.2**
203	56.3	22.8	22.8	**45.2**	**45.2**
204	89.5	22.6	37.9	**73.2**	2.8
205	63.2	37.7	2.2	**34.4**	**34.4**
206	26.8	4.7	3.2	**11.5**	**11.5**
207	99.3	25.7	17.8	**53.5**	2.2

Fig. 9. TREC Results - Best Cutoff

	EDLSI k=35		**EDLSI k=70**		**LSI k=70**	
Topic	**Cutoff Est**	**Best Cutoff**	**Cutoff Est**	**Best Cutoff**	**Cutoff Est**	**Best Cutoff**
200	66557	1660	27845	3164	27845	3164
201	39719	921	6995	1831	14528	1367
202	33094	4928	5978	13345	11286	20808
203	44180	10073	15125	6840	15125	6840
204	19464	51377	9743	13304	367930	10302
205	15380	684218	12257	35663	12257	35663
206	30901	1000	19574	2260	19574	2260
207	49846	8891	25620	13655	225412	4986