Insights from Viewing Ranked Retrieval as Rank Aggregation

Holger Bast Ingmar Weber
Max-Planck-Institut für Informatik
Stuhlsatzenhausweg 85
66123 Saarbrücken, Germany
{bast, iweber}@mpi-sb.mpg.de

Abstract

We view a variety of established methods for ranked retrieval from a common angle, namely as a process of combining query-independent rankings that were precomputed for certain attributes. Apart from a general insight into what effectively distinguishes various schemes from each other, we obtain three specific results concerned with concept-based retrieval. First, we prove that latent semantic indexing (LSI) can be implemented to answer queries in time proportional to the number of words in the query, which improves over the standard implementation by an order of magnitude; a similar result is established for LSI's probabilistic sibling PLSI. Second, we give a simple and precise characterization of the extent, to which latent semantic indexing (LSI) can deal with polysems, and when it fails to do so. Third, we demonstrate that the recombination of the intricate, yet relatively cheap mechanism of PLSI for mapping queries to attributes, with a simplistic, easy-to-compute set of document rankings gives a retrieval performance which is at least as good as that of the most sophisticated concept-based retrieval schemes and which does not require any precomputation.

1. Introduction

Ranked retrieval, where in response to a query search results are presented as a list sorted by relevance scores, has no doubt become the method of choice for retrieving information from very large data repositories, in particular the world wide web, where the entirety of *all* somehow relevant items is often too large to be even skimmed through, let alone checked thoroughly. The main challenge in ranked retrieval is, of course, to compute scores which reflect the intuitive notion of relevance in an adequate manner. Many schemes have been devised to that end, making use of quite different kinds of information for obtaining such scores, for example, term frequencies, assumed underlying concepts, links between documents, or explicit directories made and maintained by humans.

The starting point of this work is to take a common, somewhat unusual view on schemes for ranked retrieval. Namely, we see them as consisting of the following three components:

1. a number of query-independent and hence precomputable *document rankings*, where each ranking is associated with what we call an *attribute*.

   Such an attribute could correspond to any of the information sources mentioned above. It could be a term, or it could refer to a cluster of documents, or an entry in a directory or some ontology. But attributes do not have to correspond to concepts in an intuitive sense. They could also stand for a user's list of favorite documents or a global popularity measure like PageRank [16]. All three of our main results come from an unusual choice for these attributes.
2. The second component is a *folding-in mechanism* that maps query to attributes;
3. and as a third component we need an *aggregation mechanism* that makes rankings pertaining to a given (weighted) set of attributes into one global ranking.

This view of ranked retrieval as rank aggregation is inspired by the works of Fagin [5, 6], but our goal is completely different from Fagin's. His work is aimed at the particular problem of efficient top-k retrieval. We are looking for new insights into which parts of various established schemes work how well and at which price.

We specifically apply our view to retrieval schemes that try to uncover and benefit from the *latent concepts* underlying a collection. We consider schemes that are based on a *linear transformation of the original term space* like the basic vector space model, and latent semantic indexing (LSI) and its many variants (Section 2), as well as schemes that make use of *non-linear such transformations*, like PLSI (a probabilistic variant of LSI) and NMF (Sections 3 and 4). We obtain the following three results:

Exploiting query sparseness: Most queries consist of only few words, in web search typically only one, two, or three [20]. The straightforward implementation of LSI-type schemes, however, cannot exploit this sparsity for faster retrieval. We show, in Section 2.1 that by taking as our attributes not the latent factors, but simply the original terms, at most q rankings have to be combined where q is the number of words in the query. For PLSI, a similar result follows from a careful analysis of its folding-in mechanism; this is described in Section 3.1.

Dealing with polysems: The (in)ability of linear vector-based retrieval methods to cope with polysems (words with different meaning in different contexts, e.g., surfing the web vs. surfing on water), has only been vaguely commented on in previous work [17] [8]. In Section 2.2, we give a precise characterisation of the extent to which the linear schemes, including LSI, can deal with polysemy, and in which situations they are bound to fail.

Cheap but effective concept-based retrieval: In Section 4, we will replace the expensive precomputation of PLSI, which is a major bottleneck for large-scale use, by a simplistic and easy-to-compute set of document rankings, where this time we take *the documents as attributes* (again not the concepts). In a series of experiments, we get results which are consistently superior to those obtained by a number of state-of-the-art methods, including the original PLSI scheme.

The overall character of this paper is that of a collection of insights and small results as we obtained them from taking an unusual, bird's eye view on the vast literature on ranked retrieval. We hope that others find this view useful and fruitful, too.

2. LSI and its many relatives

Latent semantic indexing (LSI) [1] became famous as the first fully automatic technique for ranked retrieval that addresses the problems of both synonyms and polysems. LSI is based on what is usually referred to as the *vector space model* (VSM) [18], where queries and documents are represented as vectors in a high-dimensional space, with each dimension corresponding to one of the words, in that context called *terms*, that make up the given document collection, and with each entry being a measure of how prominent that term is for that query or document, e.g., the number of times it occurs. The whole corpus is thus viewed as a large term-document matrix. The relevance of a query to a document is measured by the similarity between the corresponding vectors, the most common measure being the cosine of the angle, which is equal to the dot product of the normalized vectors, and which is 1 for parallel vectors, 0 for orthogonal ones, -1 when the angle is 180 degrees, and somewhere in-between otherwise.

The key idea of LSI is to map both queries and documents from the m-dimensional space, where m is the number of terms, to a space of significantly lower dimension k, and compute vector similarities in the reduced space instead of in the original one. The hope is that while the dimensions of the original space correspond to the specific terms used, the dimensions of the lower-dimensional space correspond more to the (relatively few) intuitive concepts underlying this term usage. We will therefore refer to the original m-dimensional space as *term space* and to the reduced k-dimensional space as *concept space*.

For LSI the map from term space to concept space is obtained via the *singular value decomposition* (SVD) [7] of the term-document matrix; more precisely, the vectors in term space are projected on the k most significant left singular vectors, i.e., those pertaining to the k largest singular values.

What we have to say in the following, however, applies to *any* scheme that uses a linear map L from term space to concept space, that is, L is an $k \times m$ matrix. In particular, this covers the numerous variants and extensions of LSI, e.g., [9] [4] [11], the straightforward use of non-negative matrix factorization (NMF) [14] [21], semi-discrete matrix factorization [12], and the basic vector space model (take $k = m$ and L as the $m \times m$ identity matrix).

2.1. Exploiting query sparseness

Consider any concept-reduction scheme with a linear map L, as described above. A straightforward way to cast such a scheme in our framework would be to take as attributes the k dimensions of the concept space, and as similarity score between a document and a concept simply the corresponding (normalized) coordinate in the concept space; this would work. Due to the linearity of L, it is, however, also possible — and will in fact give more interesting insights — to take the *terms* to be the attributes, which we will demonstrate next.[1]

We write the query as $q = \sum_t n_{q,t}\, e_t$, where e_t denotes the unit vector for the dimension corresponding to term t and $n_{q,t}$ is the term frequency of this term in the query. The cosine similarities between a query q and a document d can then be written as

$$\begin{aligned}\mathrm{sim}_{\mathrm{LSI}}(q,d) &= \frac{(Lq)^T Ld}{|Lq| \cdot |Ld|} \\ &= \frac{1}{|Lq|} \sum_t n_{q,t} \frac{(Le_t)^T Ld}{|Ld|},\end{aligned}$$

1 Mapping the concept space back to the term space is itself not a new idea; it is the conjunction with our rank aggregation view which gives us a new result here.

where we have used the linearity of L. Now simply define the score of document d for attribute (term) t as

$$s_t(d) := \frac{(Le_t)^T Ld}{|Ld|}.$$

For a given query, combining these scores with weights $n_{q,t}$ then gives the same overall scores for a document d as before, except for the factor of $1/|Lq|$, which, however, has no influence on the ranking of the documents,

We have thus shown that LSI and all its (linear) relatives, despite their being designed as concept-based retrieval techniques, can be viewed as aggregating rankings for individual *terms*, not concepts. This has a number of interesting consequences. As a first immediate result, we get

Theorem 1. *LSI, or any other method that uses a linear map from term to concept base, can be implemented as a rank aggregation scheme such that a query with q keywords can be processed by combining merely q rankings.*

In comparison, the straightforward implementation of these methods requires the combination of k rankings, independent of the query, where k is the number of concepts. For the majority of queries, which consist of only very few words [20], Theorem 1 improves on this by an order of magnitude.

2.2. LSI and polysems

The fact that LSI and its relatives do mere linear aggregation of rankings for the individual *terms* makes it easy to characterize to which extent these methods can deal with polysems and when they fail. Note that the wit of the following theorem does not lie in its mathematics (which is trivial) but in its interpretation: if only for some polysem in the query the score of an undesired document d^- is sufficiently high compared to that of a desired document d^+, then d^- will be ranked before d^+, even if the other words in the query give evidence against d^-.

Theorem 2. *For a query q containing a polysem p, a document d^+ (matching the intended meaning) is ranked before a document d^- (matching an unintended meaning) if and only if*

$$s_p(d^-) - s_p(d^+) \quad < \sum_{t \in q \setminus \{p\}} (s_t(d^+) - s_t(d^-)).$$

As a concrete example consider the *Classic3* corpus, which was considered in several recent works [19] [3], and which is a simple union of the three classical collections Medline (1033 medical abstracts), Cranfield (1400 abstracts on aeronautics), and Cisi (1460 abstracts on information science)[10]. The term "pressure" (or rather its stem) is a polysem in that it occurs frequently in both Medline and Cranfield documents; however, it occurs about 9 times more often in Cranfield than in Medline. The term "blood", on the other hand, only occurs in Medline, but its overall frequency is 4 times less than that of "pressure". Due to the phenomenon formalized in Theorem 2, LSI fails badly on this query, in that there is *not a single Medline document among the top 10 documents*.[2] In contrast, the PLSI scheme of [8], which we discuss in the next section, does exactly the right thing in that it *ranks all related Medline documents higher than those from Cranfield.*

3. PLSI and NMF

Probabilistic latent semantic indexing (PLSI) [8] follows the same principle as LSI in that both queries and documents are mapped from a given high-dimensional term space to a concept space of given, and typically much lower dimension k. There are two main differences between LSI and PLSI. One is that the approximate matrix decomposition computed by PLSI has only non-negative entries, that is, we can view PLSI as computing for each document a probability distribution over concepts, and, by duality, for each concept a probability distribution over terms. (As a consequence, the concepts in PLSI often make sense to a human, i.e., terms with a high probability for a given concept do indeed belong to the same semantic topic — the same is rarely ever true for the latent factors of LSI.)

The other main difference is that PLSI's mechanism for mapping queries to concepts is inherently non-linear, while for LSI and its relatives it is linear. In order to map a query from term space to concept space, a process referred to as 'folding-in', PLSI tries to find the probability distribution over the k concepts which maximizes the likelihood of obtaining the given query, assuming that the query is generated by repeatedly picking from that distribution over the concepts a concept, and then picking from that concept a random term, according to the precomputed distribution of terms for that concept. This is a complex optimization problem (with no closed-form solution), which PLSI solves approximately by a variant of the EM algorithm [2]. The formulas of the iteration are stated in the proof of Theorem 3.

3.1. Exploiting sparse queries in PLSI

We have seen that LSI can be made to exploit sparse queries by implementing it as an aggregation of rankings that were precomputed for the individual *terms*. This approach does not work for PLSI, however, due to its nonlinear folding-in mechanism.

2 We remark that we observed the same phenomenon, though not necessarily for the same query, for various (randomly selected) variants and extensions of LSI.

Nevertheless, PLSI can process sparse queries with equal efficiency, owing to the following theorem, which says that by PLSI's folding-in a query cannot have more non-zero entries in concept space than in term space.[3] Note that the analogous statement for LSI is not true.

Theorem 3. *If we run PLSI's folding-in algorithm until convergence then a query with r distinct terms is "generally" assigned at most r concepts with non-zero probability. Here "generally" refers to the assumption that a random overdetermined linear system does not have a solution.*

Proof. Hofmann [8] uses the following iterative update equations to obtain an approximate non-negative factorization of the given term-document matrix. The notation is explained after the formulas.

$$p(z_k|d_i, w_j) = \frac{p(w_j|z_k)\, p(z_k|d_i)}{\sum_l p(w_j|z_l)\, p(z_l|d_i)} \quad (1)$$

$$p(w_j|z_k) = \frac{\sum_i n(d_i, w_j)\, p(z_k|d_i, w_j)}{\sum_m \sum_i n(d_i, w_m)\, p(z_k|d_i, w_m)}$$

$$p(z_k|d_i) = \frac{\sum_j n(d_i, w_j)\, p(z_k|d_i, w_j)}{n(d_i)} \quad (2)$$

Following Hofmann's terminology, we here use $p(w_j|z_k)$ to denote the probability of the jth term for the kth concept z, and $p(z_k|d_i)$ to denote the probability of the kth concept for the ith document; it should be understood that $\sum_j p(w_j|z_k) = 1$ and $\sum_k p(z_k|d_i) = 1$. Further, $p(z_k|d_i, w_j)$ denotes the probability that the occurrence of the jth term in the ith document was generated by the kth concept. The $n(d_i, w_j)$ are simply the frequencies of the jth term in the ith document, and $n(d_i)$ refers to the total number of terms in the ith document.

Essentially the same equations are used to fold-in a query. That is, for a given query q containing certain words w_j we try to, conceptually, find the mix of concepts present in the query, i.e., a distribution $p(z_k|q)$. For the folding-in computation the $p(w_j|z_k)$ are kept constant. We replace all d_i's by q as there is only one "document" of interest. Equations (1) and (2) are then iterated *until convergence* with the aim of estimating the $p(z_k|q)$.

For clearer notation we define the empirical distribution of terms in the query $\hat{p}(w_j|q) = n(q, w_j)/n(q)$.

On convergence of (1) and (2) we have

$$p(z_k|q) = \sum_j \hat{p}(w_j|q)\, \frac{p(w_j|z_k)\, p(z_k|q)}{\sum_l p(w_j|z_l)\, p(z_l|q)}$$

Under the assumption that $p(z_k|q) \neq 0$ we can cancel this term on both sides so that the equation above becomes

$$1 = \sum_j \hat{p}(w_j|q)\, \frac{p(w_j|z_k)}{\sum_l p(w_j|z_l)\, p(z_l|q)}$$

Let $c_{k,j} = \hat{p}(w_j|q)\, p(w_j|z_k)$ and $x_j = 1/\sum_l p(w_j|z_l)\, p(z_l|q)$. With this notation, we get for each k with $p(z_k|q) > 0$ an equation

$$1 = \sum_{j=1} c_{k,j}\, x_j .$$

Written in matrix form, this is

$$\mathbf{1} = C\, x,$$

where the bold one on the left-hand side denotes an l-dimensional column vector with all ones, where l is the number of concepts z_k for which $p(z_k|q) > 0$. Also observe that $\hat{p}(w_j|q) = 0$ implies $c_{k,j} = 0$ for all k. For a query with only r distinct terms, the matrix C therefore has dimensions $l \times r$ after its all-zero rows have been removed.

Then, if C has rank at least r, r rows of C will already define a unique solution x, which will in general not be consistent with the remaining $l - r$ rows. In that case, we must have $l \leq r$ for a solution to exist, that is, for a query with r distinct terms we use *at most* r distinct concepts to generate the query.

Observe that for any reasonable document collection the term distribution $p(w_j|z_k)$ computed by PLSI for the individual concepts will indeed be linearly independent, since this helps to improve the cross-entropy relative to the original term-document matrix, which is the objective that PLSI tries to minimize.

□

We finally remark that in practice a finite number of iterations will usually give only *approximate convergence*. Similarly, the use of a temperature parameter β (see [8] for details) for the folding-in of queries has the effect of pulling the $p(z_k|q)$ toward an equidistribution where all concepts are equally relevant to the query. For these cases, Theorem 3 at least guarantees that there will be only a small number of *dominant* concepts.

4. The power of non-linear folding-in

As reported in [8], and as indicated by our example from Section 2.2, PLSI can to a large extent overcome the problem with polysems which we have formally characterised for LSI-type schemes in Section 2.2. Unfortunately, however, PLSI's approximate factorization of the term-document matrix is costly to compute: when nz is the number of non-zero entries in the term-document matrix and k is the number of concepts, each iteration requires

3 In particular, in response to a single-term query, always exactly one of k precomputed rankings is returned. In that case, we can even show that the iteration cannot get stuck at a local minimum: a slight increase in the optimal $p(z_k|q)$ will always have a positive impact on the relative entropy between the empirical distribution $p(w_j|q)$, then concentrated at a single term, and $\sum_k p(w_j|z_k)\, p(z_k|q)$.

$O(nz \cdot k)$ operations, and the number of iterations required typically lies in the hundreds. Mapping the query to the concepts, i.e., the folding-in mechanism, on the other hand requires merely $O(q \cdot k)$ operations, where q is the number of terms in the query, and is thus relatively cheap, especially for short queries.

What we show in this section is, that the expensive document rankings of PLSI can be replaced by very simplistic and easy-to-compute rankings at *no loss in retrieval precision*. Our findings give strong evidence that the superior retrieval precision of PLSI-type schemes is above all due to the folding-in mechanism, and *not* due to the heavy precomputation.

4.1. A new simple algorithm

The algorithm we present here was again born from looking at an existing ranked-retrieval scheme — PLSI in this case — as a rank aggregation process as we described it in the introduction and then making an alternative, unusual choice of attributes. Namely, we here take *as attributes the documents*. To obtain a ranking for each such attribute, we simply take the cosine similarities between each document and the document that constitutes the respective attribute and normalize the contribution of each 'attribute document' in the 2-norm, in order to avoid that an 'average' document dominates the final ranking. These attribute rankings can in fact, as we have done in our implementation, be computed on-the-fly when they are required. Thus, our scheme does *not need any precomputation* such as a clustering or a matrix factorization. The folding-in mechanism we leave as in the original PLSI scheme, except that queries are now mapped to distributions over documents, not concepts, and that we use the normalized term-document matrix for this process. The running time of this procedure is proportional to the number of occurrences of the query terms in the corpus, which lies about an order of magnitude (but not more) above the requirements for the original PLSI.

4.2. Experiments

We tested our simple algorithm against two state-of-the-art concept-finding schemes, presented in [15]. Both of these compute an approximate non-negative matrix factorization (NMF) by means of an iterative local search. One method is aimed at minimizing the *Euclidean distance* from the product of the factorization to the original matrix, which is exactly what LSI does without the non-negativity constraint; we therefore name this method NMF-LSI. The other method is aimed at minimizing the *Kullback-Leibler divergence* or *cross-entropy* which is exactly the objective PLSI tries to minimize, too; we therefore name this method NMF-PLSI.[4] For both methods, we chose document rankings in the straightforward way, as we described it at the beginning of Section 2.1.

		Cranfield	Reuters	Ohsumed
NMF-LSI	EM	30.1%	41.9%	2.3%*
NMF-PLSI	EM	22.5%	—	—
SIMPLE	**EM**	**38.2%**	**46.5%**	**20.0%**
COSINE	LIN	32.8%	36.0%	12.8%
LSI	LIN	31.9%	20.5%*	—
NMF-LSI	LIN	25.8%	27.2%	0.8%*
NMF-PLSI	LIN	17.9%	—	—
SIMPLE	LIN	30.1%	—	—

Table 1. Average precisions for Cranfield (2836 terms, 1400 documents, 225 queries), Reuters (5701 terms, 21578 documents, 119 queries) , and Ohsumed (99117 terms, 233445 documents, 63 queries)[10].

The results of our comparison surpassed our own expectations. On three standard test collections — one small, one of medium size, and one large — our algorithm was consistently and significantly better than all its competitors. Our results are summarized in Table 1 below. A '—' indicates that the computation either ran out of memory or did not finish within one day on a dedicated compute server with two 3-GHz processors and 4 GB of main memory. A * indicates that the same memory and time constraints permitted only a relatively small, suboptimal number of concepts. All computations were done with Matlab. For the NMF computations we used the source code provided by Lee and Seung at [13].

To further support our hypothesis of the outstanding significance of folding-in for retrieval, the lower part of Table 1 also gives the figures for a number of schemes, including ours and the two NMF variants, when implemented with a simple linear folding-in (LIN). For this, we considered two variants: multiplication with the transpose of the matrix of concept vectors, and multiplication with its pseudo-inverse (for LSI, the two coincide). For each scheme, we picked the variant which gave the best results. As can be seen from the comparison of the rows for NMF-LSI, NMF-PLSI, and SIMPLE, average precision for the linear schemes was consistently worse among all methods and all collections where the factorizations could still be computed.

4 We also implemented PLSI itself, which, however, we could not, despite our best efforts, get to perform better than NMF-PLSI, which is why we omitted it in our figures.

Here are some more details on our experiments. We removed obvious stopwords and used stemming (Porter). For Cranfield and Ohsumed, average precisions were computed based on the queries and relevance assessments that are available for these collections. Reuters only comes with topic labels, from which we synthesized queries by taking for each topic the most significant terms from a random sample, in a fashion very similar to that of [4]. We ran all our experiments with tf as well as with tf.idf term weighting, and for each experiment took the better of the two (which in almost all cases was tf.idf). We also tried various dimensions for each entry in Table 1 between 48 and 1000 and in each case picked the one which gave the best result. Similarly, the EM-based folding-in has a parameter β (called the *inverse temperature* [8]), and for each entry in Table 1, we picked that β from the sequence $0.1, 0.2, \dots, 1.0$ which gave the best result. We actually found the choice of β a crucial performance factor: for all the methods we considered, including ours, there was a difference of about 10% in average precision between the best choice and setting $\beta = 1.0$. A choice of $\beta = 0.6$ gave an average precision at most 2% away from the optimum for every method. The number of iterations of the EM-based folding-in was set to 10 for the small Cranfield corpus and to 60 for the larger corpora to ensure a high degree of sparseness as guaranteed by Theorem 3.

5. Conclusions

By viewing ranked retrieval as a process of aggregation of precomputed rankings, we obtained three small but apparently new results: a reformulation of LSI that can exploit query sparseness, a precise characterization of the extent to which LSI can deal with polysems, and a replacement of the compute-intensive part of PLSI by one that is much simpler and easier to compute, without loss in retrieval quality.

A more general insight to be obtained from our view is a generic way to combine ranked retrieval schemes of completely different origins. It would be interesting to implement a scheme combination along these lines and test its effectiveness in a web-scale setting.

References

[1] 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.

[2] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39:1–38, 1977.

[3] I. S. Dhillon and Y. Guan. Information theoretic clustering of sparse co-occurrence data. In *Proceedings ICDM'03*, pages 517–520, 2003.

[4] G. Dupret. Latent concepts and the number of orthogonal factors in latent semantic analysis. In *Proceedings of the 26th Annual ACM conference on Research and Development in Informaion Retrieval (SIGIR'03)*, pages 221–226, 2003.

[5] R. Fagin. Combining fuzzy information from multiple systems. In *Proceedings PODS'96*, pages 216–226, 1996.

[6] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *Proceedings PODS'01*, pages 102–113, 2001.

[7] G. H. Golub and C. van Loan. *Matrix Computations (3rd edition)*. John Hopkins University Press, 1996.

[8] T. Hofmann. Unsupervised learning by probabilistic latent semantic analysis. *Machine Learning Journal*, 42(1):177–196, 2001.

[9] P. Husbands, H. Simon, and C. H. Q. Ding. On the use of the singular value decomposition for text retrieval. *Computational Information Retrieval*, pages 145–156, 2001.

[10] Glasgow IDOMENEUS - test collections. http://www.dcs.gla.ac.uk/idom/ir_resources/test_collections/.

[11] S. D. Kamvar, D. Klein, and C. D. Manning. Spectral learning. In *18th International Joint Conference on Artificial Intelligence (IJCAI'03)*, pages 561–566, 2003.

[12] T. Kolda and D. O'Leary. A semidiscrete matrix decomposition for latent semantic indexing in information retrieval. *ACM Transactions on Information Systems*, 16(4):322–346, 1998.

[13] D. D. Lee and H. S. Seung. Non-negative matrix factorization discussion and matlab code. http://journalclub.mit.edu/jclub/publication?com_id=2.

[14] D. D. Lee and H. S. Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401:788–791, 1999.

[15] D. D. Lee and H. S. Seung. Algorithms for non-negative matrix factorization. In *Proceedings NIPS'00*, pages 556–562, 2000.

[16] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.

[17] C. H. Papadimitriou, H. Tamaki, P. Raghavan, and S. Vempala. Latent semantic indexing: a probabilistic analysis. In *Proceedings PODS'98*, pages 159–168. ACM Press, 1998.

[18] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.

[19] K. Shin and S. Han. Fast clustering algorithm for information organization. In *Proceedings CICLing'03*, pages 619–622, 2003.

[20] C. Silverstein, H. Marais, M. Henzinger, and M. Moricz. Analysis of a very large web search engine query log. *Sigir Forum*, 33(1):6–12, 1999.

[21] A. Vinokourov. Why nonnegative matrix factorization works well for text information retrieval. http://citeseer.ist.psu.edu/458322.html.