Information Retrieval using very short Krylov sequences

Katarina Blom and Axel Ruhe*

1 Introduction

The task is to search among the *documents* in a large data base for those that contain information of interest stated in a *query*. We summarize the contents of the data base as an $m \times n$ *term document* matrix X, where each column represents one document and each row one *term*, in the simplest case just a specific word. An element x_{ik} is nonzero whenever term i is present in document k.

The query is now q, an m vector of terms, and we may form the scalar product

$$p^T = q^T X \tag{1}$$

to get p a *choice vector* whose nonzero elements indicate which of the documents that contain any of the terms in the query.

This is the way one looks at it in the *vector space methods* of information retrieval [9], each document is regarded as a column vector x_k in m space and the query q is another. In the simplest case, we retrieve those documents k whose angle to the query is smallest, say that we take those for which the cosine

$$c_k = \cos \angle(q, x_k) = \frac{q^T x_k}{\|q\|_2 \|x_k\|_2}$$

is largest.

*Department of Mathematics, Chalmers Institute of Technology and the University of Göteborg, S-41296 Göteborg, Sweden, email: blom,ruhe@math.chalmers.se. Support to the first author from Chalmers University Graduate students travel Fund and to the second from the Royal Society for Arts and Sciences in Göteborg is gratefully acknowledged.

2 Latent Semantic Indexing

The Latent Semantic Indexing (LSI) method has become rather well established, see the works of Berry, Dumais et. al. [8, 1, 4]! Its aim is to find the global structure in the data by computing the leading part of the singular value decomposition of the term document matrix,

$$X = U\Sigma V^T$$

containing the r leading components and look at

$$X^{(r)} = U_r \Sigma_{rr} V_r^T \,. \tag{2}$$

Here we let the superscript in $X^{(r)}$ signify that we take a rank r matrix of full size, $m \times n$, while subscripts like U_r mean that we take the r first columns of U and double susbcripts in Σ_{rr} that we take the leading rows and columns of Σ.

The reason for doing the SVD is that the leading singular vectors contain the global information of the data base, and we filter out local peculiarities as e. g. spelling errors and nonstandard use of terms.

The rank r is chosen as a much smaller number than m or n, but is still quite large, typically values like $r = 100$ to $r = 300$ are reported. One uses the singular vectors computed to find a choice vector,

$$p^{(r)T} = q^T X^{(r)} = q^T U_r \Sigma_{rr} V_r^T = \left[U_r^T q\right] \left[\Sigma_{rr} V_r^T\right] = \hat{q_r}^T \hat{X}_{rn} \,, \tag{3}$$

now with a r dimensional transformed query vector $\hat{q_r}$ and a $r \times n$ transformed term document matrix $\hat{X}_{rn}$, the latter can be regarded as some kind of catalogue of the document collection, where relevant information is gathered.

One can avoid computing this transformed term document matrix $\hat{X}_{rn}$ by instead computing a projected query vector, noting that

$$p^{(r)T} = q^T U_r \Sigma_{rr} V_r^T = q^T U_r U_r^T U_r \Sigma_{rr} V_r^T = \left[U_r U_r^T q\right]^T U_m \Sigma_{mn} V_n^T = \tilde{q}^T X \,, \tag{4}$$

a product between the *projected* query vector $\tilde{q}$ and the *original* term document matrix X.

In LSI the choices use to be determined by angles between the transformed query $\hat{q}_r$ and the transformed document vectors $\hat{x}_k$, which is the same as between the projected query $\tilde{q}$ and the projected document vectors $\tilde{x}_k$ in the leading singular subspace spanned by U_r,

$$\tilde{c}_k^1 = \cos \angle(\tilde{q}, \tilde{x}_k) = \frac{p_k^{(r)}}{||\tilde{q}||_2 ||\tilde{x}_k||_2} \,, \tag{5}$$

we will still need to get all the norms of the projected documents $\tilde{x}_k$.

We may avoid this extra computation, by instead letting the angles between the *projected* query $\hat{q}_r$ and the *original* documents x_k determine the scoring,

$$\tilde{c}_k^2 = \cos \angle(\tilde{q}, x_k) = \frac{p_k^{(r)}}{||\tilde{q}||_2 ||x_k||_2} \,. \tag{6}$$

This is simpler, since now we only need to compute the norms of the original sparse x_k and can use the simpler multiplication (4) to compute p_k. This simplified choice gives a preference to documents that are closer to the leading singular subspace of the documentspace X. We will see that this is of advantage when we deal with Krylov subspaces later in this contribution.

One drawback of LSI is that the computation of several hundred singular values of a huge matrix is a rather time consuming task. We also need to store a large amount of singular vectors in the form of real numbers, while the original matrix X is a very sparse matrix of small integers. It is also nontrivial to determine the most appropriate rank r, and changes in the data base, additions and deletions of terms and documents, will need some kind of update of the SVD.

3 Our approach

We have tried a simpler way to find the global information in the data base. Run Lanczos, or more properly the Golub Kahan bidiagonalization algorithm [2, 7], starting with the query vector q, and look at what you can get from the Krylov subspaces thus computed.

Take the normalized query vector q as a start, $q_1 = q/\|q\|$, and compute a bidiagonal representation,

$$X^T Q_j = P_j B_{j,j}^T, \quad XP_j = Q_{j+1} B_{j+1,j} \tag{7}$$

with P and Q orthonormal bases of dimensions indicated by the subscripts and B a lower bidiagonal matrix, see [3] section 9.3.3!

The bases Q and P are computed, adding one column in each step j using the following:

ALGORITHM BIDIAG
Start with $q_1 = q/\|q\|_2, \beta_1 = 0$
For $j = 1, 2, \ldots$ *do*

1. $\alpha_j p_j = X^T q_j - \beta_j p_{j-1}$
2. $\beta_{j+1} q_{j+1} = X p_j - \alpha_j q_j$

End

The quantities α_j and β_{j+1} are computed to give the vectors p_j and q_{j+1} unit Euclidean norm.

After step j we have got the bidiagonal matrix,

$$B_{j+1,j} = \begin{bmatrix} \alpha_1 & 0 & & & 0 \\ \beta_2 & \alpha_2 & 0 & & \\ 0 & \beta_3 & \alpha_3 & & \\ & & \ddots & \ddots & \\ & & & \beta_j & \alpha_j \\ & & & 0 & \beta_{j+1} \end{bmatrix}$$

We can show that P_j is an orthonormal basis of the Krylov subspace $K^j(X^TX, X^Tq)$ and Q_j of $K^j(XX^T, q)$. A Krylov space $K^j(A, x)$ is a j dimensional subspace of a starting vector x and successive applications of the matrix operator A to x,

$$K^j(A, x) = \text{span}\left(x, Ax, A^2x, \ldots, A^{j-1}x\right) .$$

Interpretation The columns of Q have the dimension of a kind of query, while those of P can be interpreted as choices among the documents in the collection X. The first column p_1 contains those documents that contain terms in the query, say its brothers and sisters, and the next p_2 can similarly be interpreted as cousins and so on. The number of documents reached will grow in a chain letter fashion, so we can hope that rather few steps j will be sufficient to reach all documents that have any connection to the original query.

Each step j can be interpreted as first applying the current query q_j, in matrix language doing X^Tq_j, giving a new choice p_j that is strongly different from previous choices, in matrix language we say that it is orthogonal. Then all terms from the chosen documents are combined in the multiplication Xp_j, to give a new query q_{j+1} strongly different (orthogonal) to all previous queries $q_1, \ldots, q_j$. After j steps we have made j queries to the data base, all of them strongly different from each other. Some readers may remember the children's game "master mind".

Measuring progress We are interested in the *reached subspace* spanned by all the documents that are combinations of all choices up to step j,

$$XP_j = Q_{j+1}B_{j+1,j} = Q_{j+1}H_{j+1,j}R_{j,j} , \tag{8}$$

where H and R are the results of a QR-factorization of B. Note that H is both orthogonal and Hessenberg, i. e. it is upper triangular with just one subdiagonal added and can be computed as a product of j elementary Givens rotation matrices. The matrices

$$W_j = Q_{j+1}H_{j+1,j}$$

will be orthogonal bases of these interesting subspaces for a sequence of steps j. Let us project the query onto this subspace.

$$\tilde{q}^{(j)} = W_jW_j^Tq = W_jH_{j+1,j}^Te_1 = W_j \begin{pmatrix} h_{1,1} \\ h_{1,2} \\ \vdots \\ h_{1,j} \end{pmatrix} ,$$

and we see that the first row of H gives the coordinates of the query in the basis W. When we run several steps j of our algorithm, new columns are added to H, but it is only the last column that is modified.

In step j the distance between the query q and the projected query $\tilde{q}^{(j)}$ is

$$\begin{aligned} r^{(j)} &= q - \tilde{q}^{(j)} \\ &= Q_{j+1}e_1 - Q_{j+1}H_{j+1,j}H_{j+1,j}^T e_1 \\ &= Q_{j+1}(I - H_{j+1,j}H_{j+1,j}^T)e_1 \\ &= Q_{j+1}h_{j+1}^{(j)}h_{j+1}^{(j)T}e_1 \\ &= Q_{j+1}h_{j+1}^{(j)}h_{1,j+1}^{(j)} \end{aligned} \tag{9}$$

and its norm is just

$$\|r^{(j)}\| = |h_{1,j+1}^{(j)}|\,. \tag{10}$$

It decreases as we let j grow, but will not tend to zero unless the query is a linear combination of the documents in X.

We will get a quantity that tends to zero, if we follow the normal equation residual,

$$\begin{aligned} X^T r^{(j)} &= X^T Q_{j+1}h_{j+1}^{(j)}h_{1,j+1}^{(j)} \\ &= P_{j+1}B_{j+1,j+1}^T h_{j+1}^{(j)}h_{1,j+1}^{(j)} \\ &= P_{j+1}\begin{pmatrix} B_{j+1,j}^T \\ 0 \quad \alpha_{j+1} \end{pmatrix} h_{j+1}^{(j)}h_{1,j+1}^{(j)} \\ &= P_{j+1}\begin{pmatrix} 0 \\ \alpha_{j+1}h_{j+1,j+1}^{(j)} \end{pmatrix} h_{1,j+1}^{(j)} \end{aligned} \tag{11}$$

which has the norm

$$\|X^T r^{(j)}\| = |\alpha_{j+1}h_{j+1,j+1}^{(j)}h_{1,j+1}^{(j)}| \tag{12}$$

Scoring documents There are several possible ways to use the quantities obtained from our algorithm to score documents for relevance with respect to the query q. It is natural to mimic LSI and choose the angles either between the projected query and documents in the reached subspace (5),

$$c_k^1 = \cos\angle(\tilde{q}, \tilde{x}_k) = \frac{p_k}{\|\tilde{q}\|_2\|\tilde{x}_k\|_2}\,, \tag{13}$$

or between the projected query and the original documents (6),

$$c_k^2 = \cos\angle(\tilde{q}, x_k) = \frac{p_k}{\|\tilde{q}\|_2\|x_k\|_2}\,. \tag{14}$$

It is easy to find these from quantities computed in ALGORITHM BIDIAG. Apply an elementary orthogonal transformation $S_{j,j}$ from the right to the Hessenberg matrix $H_{j+1,j}$ so that the resulting $H_{j+1,j}S_{j,j}$ has only its leading element nonzero in the

first row. Then $Q_{j+1}H_{j+1,j}S_{j,j}$ is a new basis of the reached subspace. Premultiply the document vector x_k with this basis and,

$$y_k = S_{j,j}^T H_{j+1,j}^T Q_{j+1}^T x_k$$

will give the component of x_k along $\tilde{q}^{(j)}$ as its first coordinate $y_{1,k}$, and the rest of the projected $\tilde{x}_k$ as the norm of the remaining $(y_{2,k}, \ldots, y_{j,k})^T$.

Our experiments have shown that the second choice (14) gave better precision so we use it as our standard.

4 Illustrations

We have tested our algorithm on several widely circulated test sets coming from a. o. the TREC conferences [6]. Here we choose to report results on the well known MEDLINE collection [5]. It is admittedly a toy problem, but since it has been tested by a wide range of people, it is good for comparison and illustration purposes.

Our original matrix X is of size $m \times n = 7014 \times 1033$, with $x_{ik} = 1$ if term i is present in document k, no counting of appearances is done. Instead we *scale* the matrix in a way that is natural in the analysis of general data. First all rows are scaled to have Euclidean norm (sum of squares) equal to unity. This is sensible, since it deemphasises common terms, as is done in most weighting schemes. Then we scale the columns to have unit length. This disturbs the row scaling somehow, but is good for illustration purposes, we get the same length on all vectors and can let the length of a projection tell how close the vector is to a subspace it is projected into. Moreover, we will most often let angles between vectors and spaces determine choices, and those are independent of the length of the vectors. We plot the row (term) and column (document) norms in figure 4. The lines are the row and column norms in the original matrix X. After the row scaling all rows have unit norm, so we do not plot them, but we plot the column norms as points in the right plot. After the column scaling all columns have unit norm, but the rows have the norms indicated by the dots on the left plot. We see that the row norms are no longer strictly equal but reasonably equilibrated.

There are 30 queries supplied with the test matrix, together with indices k of relevant documents for each each query. This gives between 9 and 39 relevant documents for each query, altogether 696 documents are relevant to some query and no document is relevant to more than one query. We compare our results to these correct answers, and are interested to know whether any quantities obtained during the computation can be used to determine if a query is a difficult or a simple one to handle with our method.

Let us list results for some typical and interesting queries. First take query 1, it has rather many, 37 relevant documents, and is on the easy side for our algorithm. In figure 4 we follow the progress in linear algebra terms, as we execute the algorithm for steps $j = 1, \ldots, 12$. Circles are the residual norms $\|r^{(j)}\|$, (10), they decrease unnoticeably slowly from 1 to 0.892. This means that the query q is at a rather large angle to the reached subspace, it has a projection of length 0.451. We plot the normal equation residuals $\|X^T r^{(j)}\|$, (12), as pluses, and note that they decrease

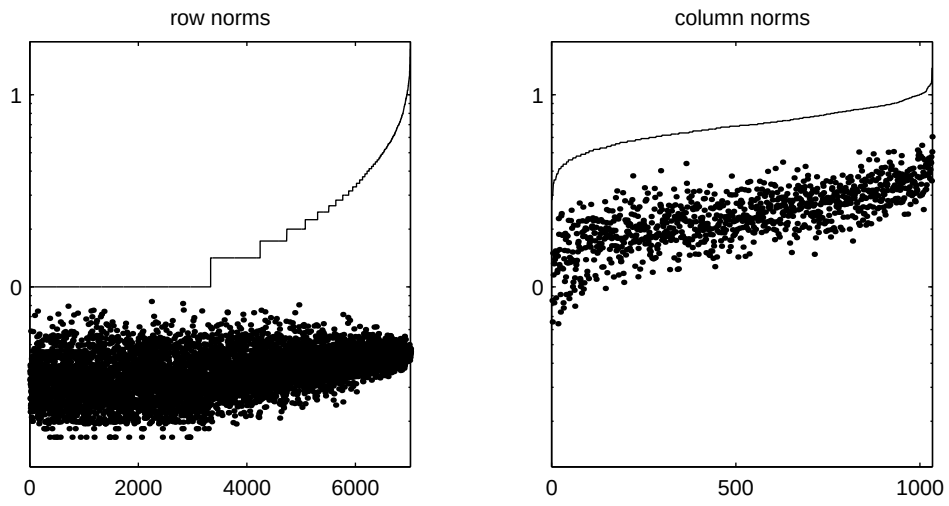

Figure 1. *Medline matrix row and column norms, before and after scaling. Sorted before scaling*

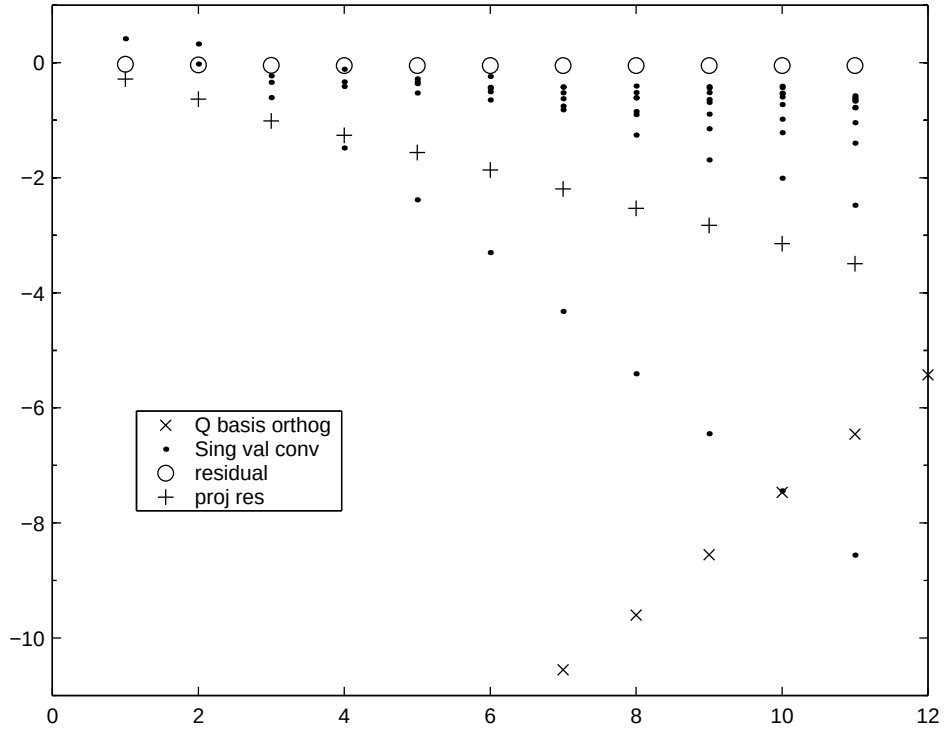

Figure 2. *Medline matrix, follow convergence of bidiagonalization procedure.*

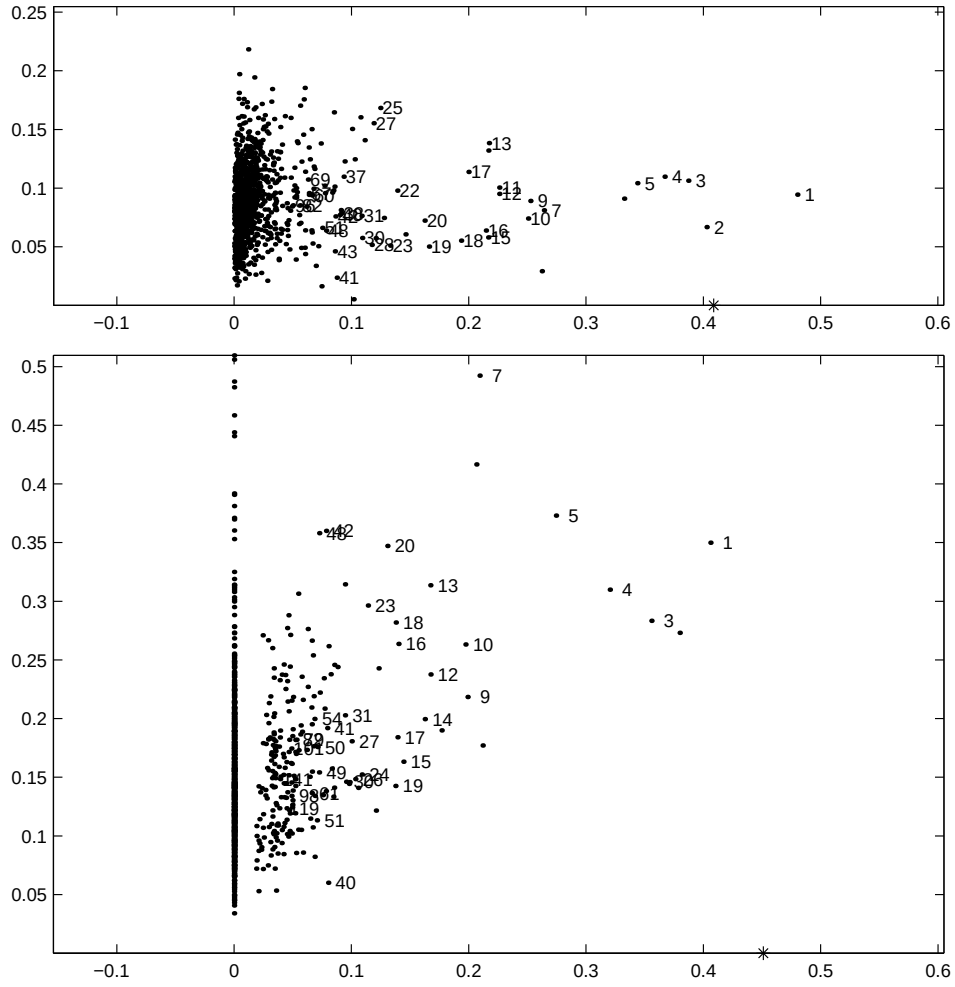

Figure 3. *Medline matrix, Query 1, upper half step $j = 2$, lower half step $j = 12$, numbers scores of relevant documents.*

fast enough at a linear rate. After 12 steps we have found the projection of the query into the document space spanned by X to at least 3 decimals. We were curious to see how the singular values converged and plotted estimates of their accurracies as points. Note that the leading singular value converged very fast, after 12 steps its vector is accurate to 9 decimals and the singular value to full machine precision. It is well known that the basis vectors Q_j keep orthogonal until one of the singular values converges, to verify that we plotted the orthogonality of each basis vector q_j to its predecessors Q_{j-1} as crosses and, true to theory, the crosses and points intersect at half the machine accuracy level during step 10.

Let us now turn to a view of all the documents, and see how well we find

the relevant documents for query 1. We plot them in a two dimensional coordinate system in figure 4. The x axis is along the projected query $\tilde{q}^{(j)}$. The y axis is used to plot the component of each x_k in the reached subspace (8) orthogonal to $\tilde{q}^{(j)}$. This makes up two of the three components of each x_k vector. We can infer the length of the third component, which is orthogonal to the reached subspace, by remembering that all vectors x_k were normalized to unit length, so the distances of the points plotted to the origin indicate how close the vectors are to the reached subspace. Those close to the origin are far from the reached subspace. If we continue the bidiagonalization to full length $j = n$, most of the vectors will get unit length, because then the reached subspace is the whole span of X, except in the rare case when the query is totally unrelated to a part of the document collection.

If we use our standard scoring method (14), taking angles between the original documents and the projected query, we would choose documents from right to left as plotted in figure 4, and we can check how well we find the relevant documents. We show this by giving the score number beside each relevant document. Look at the lower part of figure 4! First comes document 1, it is a relevant one, then comes an unnumbered point down to the left, it is not numbered since it is not relevant. Then come 3, 4, 5 all relevant, while 6 is missing. Again 7 is relevant but 8 is not, then 9 and 10 are good. Now our algorithm has given us 10 suggestions, of which we find that 7 are relevant. We say that the *precision* is 0.7 now when the *recall* is 7 out of 37, that is 0.2. APR, the averaged precision over all relevant documents [6], is slightly lower at 0.634 since the last relevant documents are found much later, we see that the last one scores as number 141. This is still not too bad, we had 1033 documents to score.

One final look at the lower half of figure 4! There are many points along the y axis, they denote documents that are orthogonal to the projected query, they will be the last ones scored. Actually all but 234 of the 1033 documents are orthogonal to both the original and the projected first query.

Our experiments have shown that it is overkill to run the algorithm as far as $j = 12$, the APR reaches its maximum already after $j = 2$ steps. Look at the upper part of figure 4! The projected query is slightly shorter, 0.408 compared to 0.451 at step 12, and all the x_k points are quite a bit closer to the origin, indicating that the components of the document vectors in the reached plane are smaller. One the other hand, there is one more relevant document among the first 10 scored, and the last relevant document is now scored as number 96, which brings up the APR to 0.762.

When scoring documents by angles in the reached plane (13), these can be seen as angles to the x axis in figure 4. It did not differ much from the standard scoring (14), actually it gave slightly worse precision, in this case APR 0.621 for $j = 2$. The third scoring choice, angles to Krylov subspace, cannot be directly seen in the plots in figure 4, it amounts somehow to choosing those documents plotted far from the origin and gives about the same choices but with still somewhat lower precision, in this case APR 0.495.

We have performed this kind of analysis for all the 30 queries given, some give better and some give worse results than this first query, see figure 4 where the APR is plotted against the step j for some of the queries and figure 4 that gives a

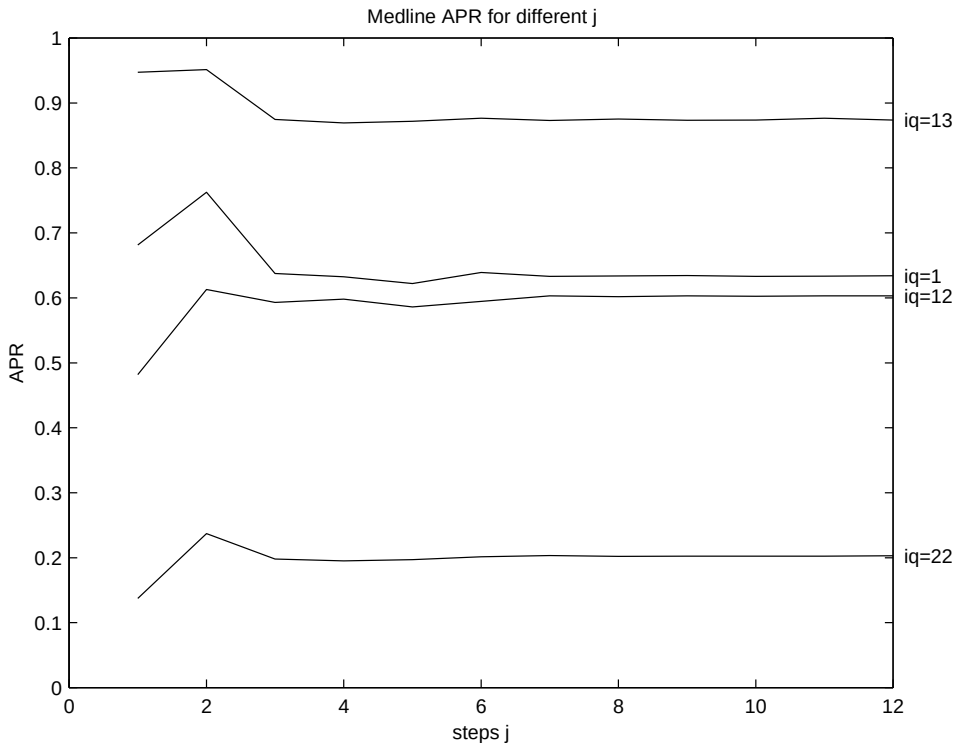

Figure 4. *Medline matrix, averaged precision recall for different steps j and some queries iq.*

(noninterpolated) precision recall diagram for these same queries at step $j = 2$.

Let us study a good question, query 13 in figure 4. It gives full score for the first 16 documents scored, and the 21 st and last relevant document is scored as number 32. We do not show the figure corresponding to figure 4 for this query, it is very similar. This query actually deteriorates if we run up to 12 steps, see the lower half of figure 4. Now we see that no less than 941 of the documents are orthogonal to the query, including one relevant document that is scored as 582. This needs to be studied further, one advantage claimed for LSI is that it can handle synonymy, see [1] and return documents that are orthogonal to the original query. Here it looks like our method has this property at step $j = 2$ but looses it when $j = 12$ is reached.

The final query we will bother the readers with is 22, the worst performer. See figure 4! Here one relevant document is the first to be scored, but then we have to look until number 6 and 13 to find the next relevant documents, and the 25 th and last does not appear until as number 292, and at 0.237 the APR is not that impressive. There does not seem to be anything wrong with the query, its projection is 0.46 not smaller than the others.

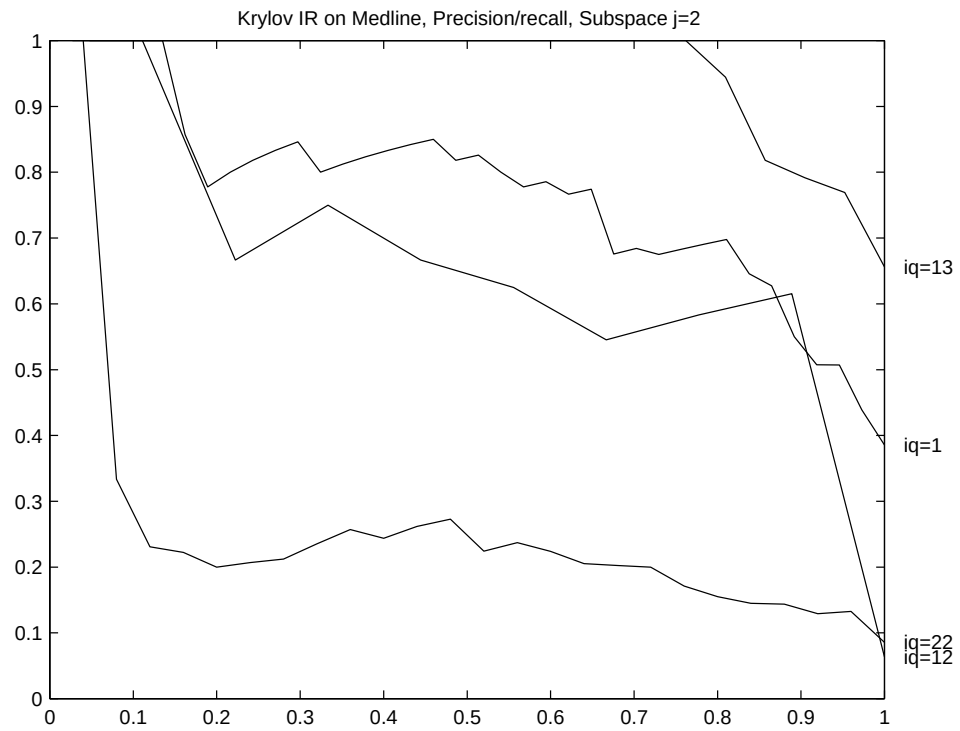

Figure 5. *Medline matrix, precision recall at step $j = 2$ and some queries iq.*

One might believe that it should be difficult to distinguish relevant document vectors from irrelevant ones but that appears far from true. Plot the cosines of the angles between all the document vectors x_k and those relevant to query 22 in figure 4 left half. The relevant documents form a 25 dimensional subspace, that is remarkably well separated from all the other vectors, the closest irrelevant vector has a cosine of 0.277 to the subspace of relevant vectors. The query vector is at a cosine of 0.3, not very close either, but this is true also for the lucky query 13, where it is 0.43. We see in the right half of figure 4, that now the relevant document vectors get good scoring cosines, the first points are higher and the later lower than those for query 22. So, after the fact, we can see that the separation is better for query 13 than for query 22.

Let us once more point out that the information needed for figure 4 is only available when we know which documents are relevant. It cannot be used in the information retrieval process.

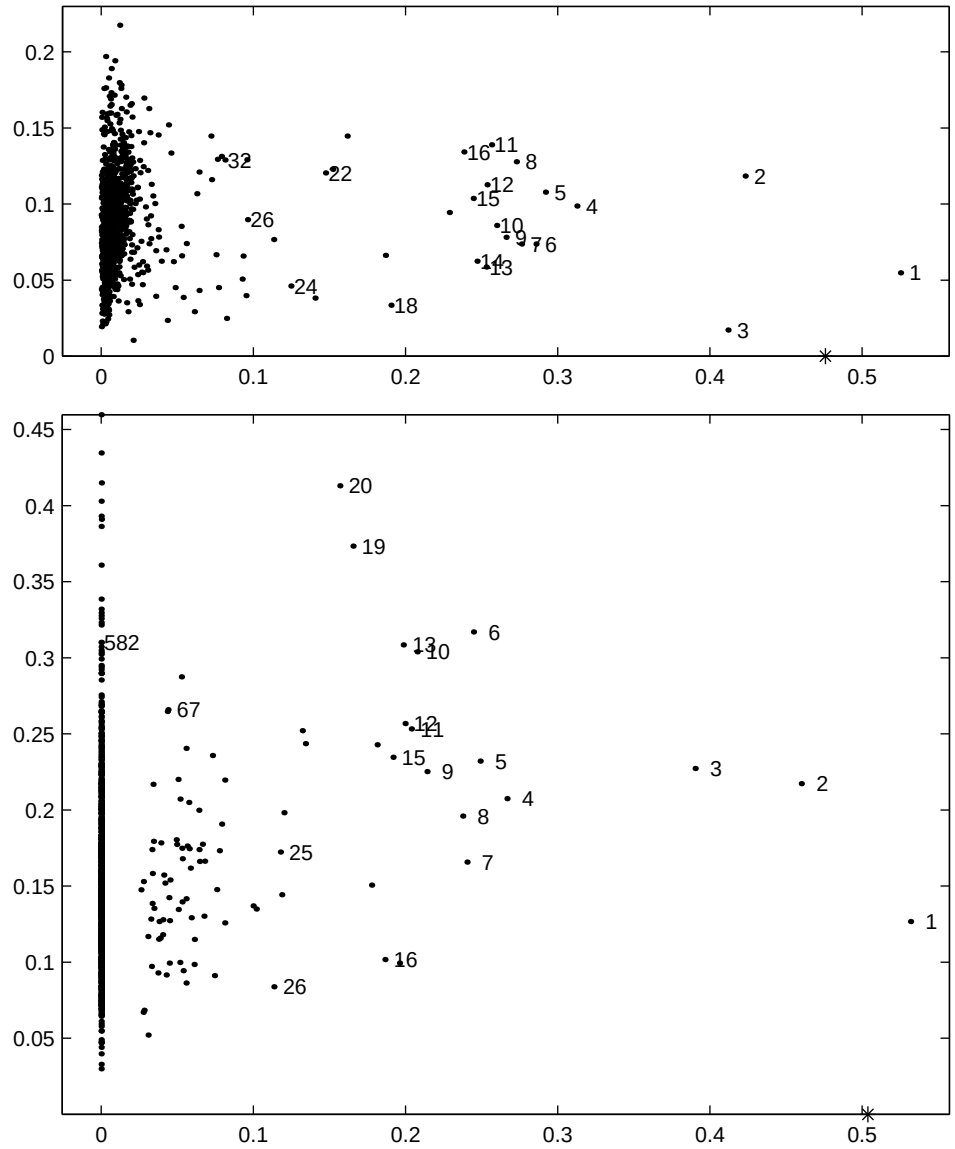

Figure 6. *Medline matrix, Query 13, upper half step* $j = 2$*, lower half step* $j = 12$*, numbers scores of relevant documents.*

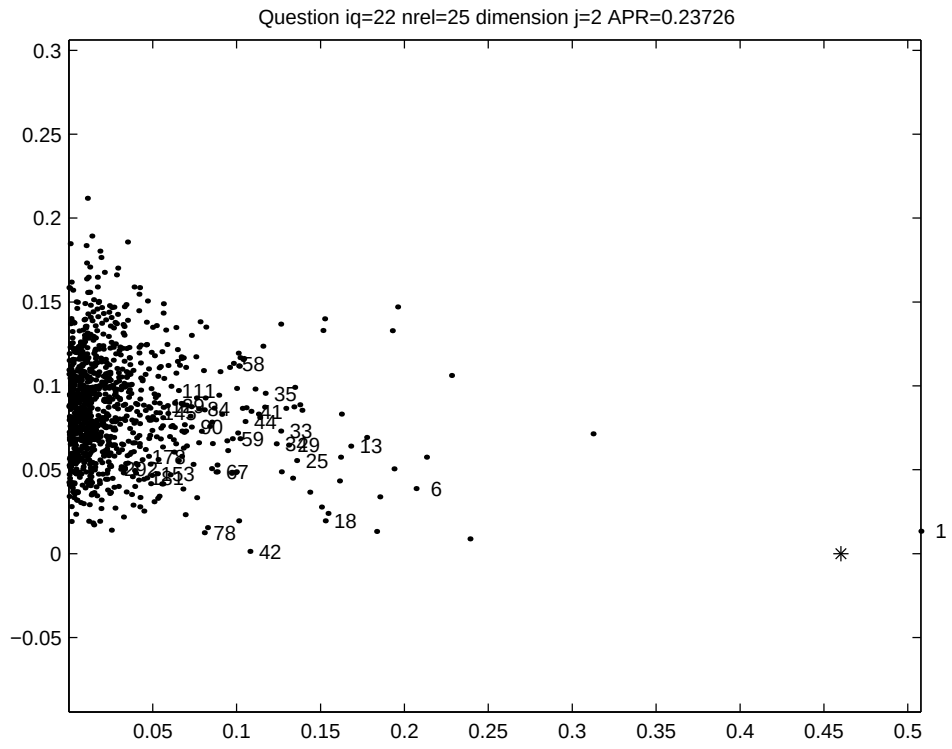

Figure 7. *Medline matrix, Query 22, $j = 2$, numbers scores of relevant documents.*

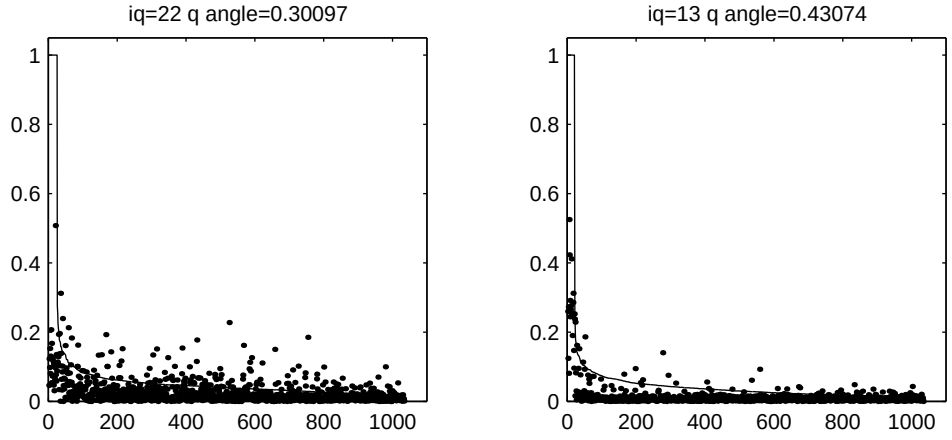

Figure 8. *Medline matrix, lines cosines of angles to subspace of relevant documents, points cosines of angles to projected query. Sorted after angle to subspace.*

Bibliography

[1] M. W. Berry, S. T. Dumais, and G. W. O'Brien, *Using linear algebra for intelligent information retrieval*, SIAM Review, 37 (1995), pp. 573–595.

[2] G. H. Golub and W. Kahan, *Calculating the singular values and pseudo-inverse of a matrix*, SIAM Journal on Numerical Analysis, 2 (1965), pp. 205–224.

[3] G. H. Golub and C. F. Van Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore, Maryland, 3 ed., 1996.

[4] K. Lochbaum and L. Steeter, *Comparing and combining the effectiveness of latent semantic indexing and the ordinary vector space model for information retrieval*, Information Processing and Management, 25(6) (1989), pp. 665–76.

[5] *Medline collection.* (Available at URL: `ftp://ftp.cs.cornell.edu/pub/smart/med`).

[6] NIST Special Publication 500-246, *The Eighth Text REtrieval Conference (TREC 8), November 16-19, 1999*, 2000. (Available at URL: `http://trec.nist.gov/pubs/trec8/t8_proceedings.html`).

[7] C. C. Paige and M. A. Saunders, *LSQR. an algorithm for sparse linear equations and sparse least squares*, ACM Transactions on Mathematical Software, 8 (1982), pp. 43–71.

[8] S. T. Dumais, G. W. Furnas, T. K. Landauer, S. Deerwester, R. Harsman, *Indexing by latent semantic analysis*, Journal of the American Society for Information Science, 41 (1990), pp. 391–407.

[9] G. Salton, *Automatic text processing the transformation, analysis, and retrieval of information by computer*, Addison-Wesley, Reading, Mass., 1989.