

# SYMBOLIC PREPROCESSING TECHNIQUES FOR INFORMATION RETRIEVAL USING VECTOR SPACE MODELS \*

MICHAEL W. BERRY <sup>†</sup>, PADMA RAGHAVAN <sup>‡</sup>, AND XIAOYAN ZHANG <sup>§</sup>

**1. Introduction.** Consider the vector space model of information retrieval [9] in which both terms and documents in a text-collection are encoded as vectors in high dimensional space. More specifically, some initial preprocessing [2] is used to extract  $n$  documents and  $m$  indexable terms from the text collection to construct an  $m \times n$  term-by-document matrix  $M$ . The element  $M_{i,j}$  is typically represented as the product  $L(i, j) \times G(i)$  where  $L(i, j)$  is the local weighting for term  $i$  in document  $j$  and  $G(i)$  is global weighting for term  $i$ . There are a variety of schemes for determining global and local weightings [2]. Thus, a document vector corresponds to a column of the matrix  $M$  with elements representing weighted term frequencies [2]. A user's query is then represented by a vector which is constructed using the terms in the query in a manner consistent with weighting schemes used in constructing  $M$ . Retrieving documents corresponding to a query vector amounts to searching the column subspace of  $M$  to find document vectors similar to the query vector.

One measure of similarity that is commonly used is the cosine of the angle between the query and a document vectors. Let  $m_j$  denote the  $j$ -th column of  $M$  (and hence the  $j$ -th document) and  $q$  denote the query vector;  $\cos\theta_j$ , the cosine of the angle between these vectors given by:

$$\frac{m_j^T q}{\|m_j\|_2 \|q\|_2}.$$

The two-norms of the document vectors can be computed once and stored. The dot product in the numerator of the expression is computed for a query vector against all document vectors. Now documents can be ranked in relevance by sorting documents in decreasing order of the cosine values. There are several drawbacks of this basic model of information retrieval and many schemes have been proposed to enhance this basic approach. A popular vector space modeling scheme is Latent Semantic Indexing (LSI) [4] which replaces  $M$  by a low-rank approximation; such a low-rank approximation can be computed using either the singular value decomposition(SVD) [1] or the semi-discrete decomposition(SDD) [6].

The basic subspace model requires computing two norms of the document vectors once and reusing them per query; additionally, a multiplication of the query vector with the matrix  $M$  is needed (per query) to compute cosine values. The LSI model requires extra computation, namely, to compute a low-rank approximation of the original term-document matrix  $M$ . Such decompositions are computationally demanding. For example, the cost of computing an SVD using a Lanczos type

---

\* This work was supported in part by the National Science Foundation through grants NSF-ACI-97-21361 and NSF-CCR-98-18334.

<sup>†</sup> Department of Computer Science, The University of Tennessee, Knoxville, TN 37996, E-mail:berry@cs.utk.edu.

<sup>‡</sup> Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA 16802, E-mail:raghavan@cse.psu.edu.

<sup>§</sup> Department of Computer Science, The University of Tennessee, Knoxville, TN 37996, E-mail:zhang@cs.utk.edu.

procedure is directly proportional to cost the of a large number of matrix-vector multiplications by  $M$  and  $M^T$ .

In this paper, we consider inexpensive preprocessing methods aimed at reducing the cost of query processing using vector space models using a method such as LSI. Our goal is to formulate schemes that extract a small submatrix  $\tilde{M}$  from the original term-document matrix  $M$ . Then LSI is applied to  $\tilde{M}$  to retrieve a set of relevant documents. Such preprocessing can also be used in a dynamic setting, for example, information retrieval based on fast-changing term-document collections on the world wide web. Our preprocessing methods are ‘symbolic,’ i.e., they consider the ‘graph’ counterpart of the matrix  $M$ . We show that some natural adaptations of well known graph-theoretic schemes can effectively extract small subsets of  $M$  which can be used with LSI to retrieve documents without degrading the quality of retrieval.

The rest of this paper is organized as follows. Section 2 provides some background material. Sections 3 and 4 contain our main contributions. Section 5 contains some concluding remarks.

**2. Background.** For background information on information retrieval we recommend the textbook by Korfhage [7]. The book by Berry and Browne provides a good overview of mathematical modeling and text retrieval using search engines [2]. The survey paper by Berry, Drmac, and Jessup [3] is a good starting point for understanding the role of matrix decompositions in vector space methods for information retrieval. The rest of this section provides some basic definitions used in this paper.

**Quantifying quality of retrieval.** Two quantities, *precision* and *recall* are typically used to measure the quality of retrieval. Precision is defined as the ratio of the number of relevant documents retrieved to the total number of documents retrieved. Recall is defined as the ratio of the number of relevant documents retrieved to the total number of relevant documents in the collection.

Vector space methods provide a relevance ranking of all documents in the collection with respect to a given query. The *average precision* is the most common measure used to assess the performance of such methods. If the method returns an ordered list of  $n$  documents judged relevant to a query, a recall-precision pair  $(r_i, p_i)$  can be computed at each point  $i$  in the list as follows. The pseudo-precision at recall level  $x \in [0, 1]$  is defined as:

$$P(x) = \max\{p_i : r_i \geq (x * r_n), i = 1, \dots, n\}.$$

The  $k$ -point interpolated average precision for a single query is defined as:

$$P = (1/k) \sum_{i=0}^{i=k-1} \{P(i/(k-1))\}.$$

Typically, a 11-point interpolated average precision is computed for each query using  $P(x)$  for  $x = 0, 0.1, 0.2, \dots, 1.0$ . Results for multiple queries are specified using the mean or the median value of  $P$  over all queries; we use the mean in this paper.

**Computational cost of LSI.** The computational cost of LSI using the SVD is dominated by the cost for computing the matrix decomposition using a Lanczos type procedure. This cost is directly proportional to cost of a large number of matrix-vector multiplications by  $M$  and  $M^T$ . In terms of the matrix  $M$ , the cost is proportional to the sum of the number of rows and columns and the number of nonzeros. Our preprocessing techniques are aimed at reducing the cost by replacing  $M$  by a smaller

submatrix  $\tilde{M}$ . To measure the effectiveness of our filtering methods we use the number of nonzeros in  $\tilde{M}$  as well as its size, i.e., the number of rows and columns.

**The graph of a sparse matrix.** A matrix  $A$  is symmetric if  $A_{ij} = A_{ji}$  and it is sparse if most of its elements are zero. The *undirected* graph of a sparse symmetric matrix  $A$  has a vertex for each row/column and an edge between vertices  $(i, j)$  iff  $A_{i,j} \neq 0$ . This graph shows the sparsity structure, i.e., linkages between rows and columns of the matrix. The term document matrix  $M$  is typically sparse but it is not symmetric; the same is true of  $B = [M|q]$ , the query augmented term-by-document matrix. Using an undirected graph model makes associated algorithms simpler. We therefore use  $G$ , the graph of the symmetric matrix:

$$\begin{pmatrix} 0 & B \\ B^T & 0 \end{pmatrix}.$$

This matrix is not constructed explicitly, instead, the graph is traversed implicitly using the standard sparse storage scheme for the matrix. For more on graphs of matrices and their use in matrix computations, we refer the reader to the book by George and Liu [5].

**3. Symbolic preprocessing techniques.** Our methods are based on traversing the graph defined above to filter out a smaller relevant subgraph (and hence submatrix). In short, we construct a ‘query-specific’ matrix  $\tilde{M}$  which is much smaller than  $M$  and we apply LSI on  $\tilde{M}$ . The preprocessing cost is typically negligible and is no more than the cost of accessing elements in  $\tilde{M}$  a few times.

We first considered a simple *level search* (or breadth first search) of the graph [8]. We start the search with the vertex in the graph corresponding to the query. The initial level, level 0, consists of the query itself. The next level (level 1) consists of the terms of the query. Level 2 contains all documents that contain a term in level 1. Level 3 contains terms (not in previous levels) that are in documents at level 2 and so on (even numbered levels contain documents and odd numbered levels contain terms). The level search could be arbitrarily terminated at a given level; alternatively it will terminate when the collection of terms and documents is exhausted.

Simple level search does not result in many levels on typical term-document matrices. A significant fraction of the total number of documents are encountered in the first few levels. Generating a submatrix  $\tilde{M}$  that corresponds to the first few levels for a query results in relatively small savings in size. We therefore concluded that filtering using simple level search is not useful.

**Weighted level search.** We next considered level search schemes that can take into account the values of nonzero elements in the matrix  $M$ . Recall that the element  $M_{i,j}$  is represented as the product  $L(i, j) \times G(i)$  where  $L(i, j)$  is the local weighting for term  $i$  in document  $j$  and  $G(i)$  is global weighting for term  $i$ . In our weighted level search, we start with the query in level 0 and its terms in level 1. We assign weight  $L(i, j) \times G(i)$  to an edge leading to document  $j$  from term  $i$ . The weight of document in the level search is set to the two norm of the weights over all edges leading into the document. Only documents with weights greater than a certain threshold are *retained* in the level search; those with lesser weights are *discarded*. This weighting process typically leads to fewer documents at an even numbered level and has the effect of increasing the maximum number of levels for a given query. Figure 3.1 illustrates weighted level search.

Weighted level search with different threshold conditions, such as the mean or median performed significantly better than simple level search. Consider using such

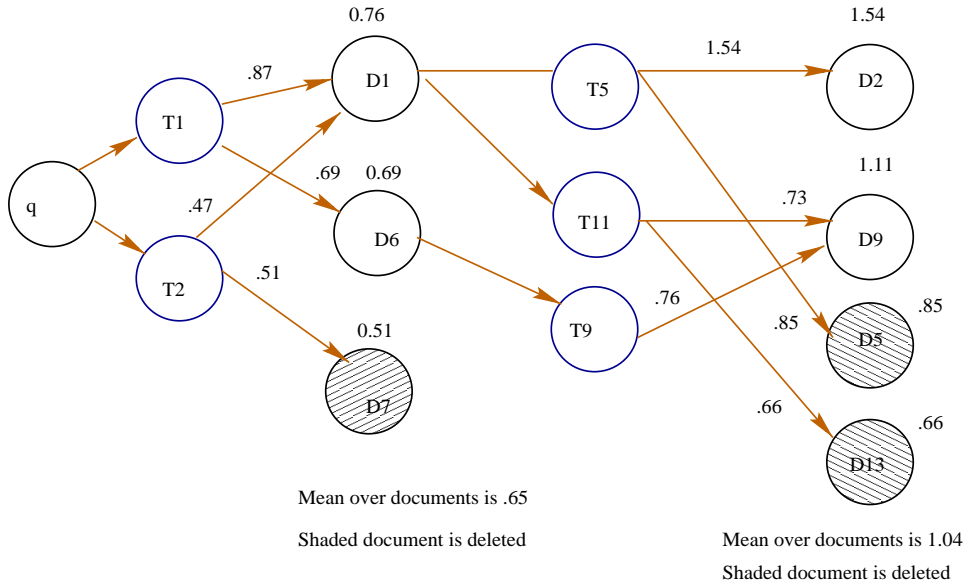


FIG. 3.1. Example illustrating weighted level search.

a level search as a retrieval method:

- the document collection returned was a small fraction of the original size, in the range 7% – 25% of the original,
- the average recall over all queries was high, in the range 56% – 75%, but
- the precision was very poor.

These results indicate that such a scheme is not an adequate retrieval method but that it can be an effective filter; the precision can be improved using LSI on the filtered subset.

**Pruned weighted level search.** We next considered ways to further refine our weighted level search. We observed that the subset returned by weighted level search contained many terms that occurred in a very small number of documents in the subset. We modified the subset by pruning such terms; we call this pruned weighted level search. Once again the pruning can be implemented using a threshold, i.e., a condition of the form prune terms connected to less than 3 documents.

**Merging documents or terms.** A successful scheme in sparse matrix/graph applications is that of merging vertices that are nearly ‘indistinguishable’, i.e., have similar connectivities. We add a document merging step to further reduce the filtered submatrix size. We use a greedy heuristic with the following basic selection step. Pick a document  $d_i$  of maximal degree (containing many terms); next pick document  $d_j$  of maximal degree containing terms in  $d_i$  ( $d_j$  is a neighbor of a term in  $d_i$ ). If the number of terms common to  $d_i$  and  $d_j$  exceeds a threshold, merge them into a new document and remove  $d_i$  and  $d_j$  from the original set. This step is repeated until all documents have been selected. The same merging process can also be applied to terms.

A natural question is whether the merge should be performed before or after level search. We show in the next section that merging after level search is better. For merged documents returned by LSI, we select members that pass the cosine test.

	MEDLINE	CISI	TIME	FBIS	LATIMES
# of docs	1,033	1,460	425	4,625	1,086
# of terms	5,831	5,609	10,804	42,500	17,903
Nonzeroes	52.0 K	68.2 K	83.6 K	1,573 K	250 K
# of queries	30	112	82	43	48
Mean terms/docs	50.35	46.74	196.71	316.31	230.42
Mean terms/query	10.2	21.8	7.8	39	28.8

TABLE 4.1

*Characteristics of term-document matrices in test collection.*

**4. Empirical Results.** We now provide some empirical results demonstrating the effectiveness of our filtering techniques. More detailed empirical results related to our level search methods can be found in our earlier paper [10]. We use a well known collection of term-document matrices shown in Table 4.1. Each matrix has an associated set of queries with a specified set of relevant documents per query. This information is used to compute precision recall curves for reporting the retrieval quality of our methods.

We compare the performance of LSI with and without filtering techniques. We begin with Figure 4.1 which shows the performance of LSI on the original term document matrices and on the submatrices obtained using filtering by means of weighted level search (WLS). The precision recall curves are the mean of 11-point precision recall curves over all queries for a given term-document collection. For the same level of recall, LSI is slightly better than LSI-WLS for MEDLINE, CISI, and LATIMES. However, for the TIME and FBIS collection, filtering with weighted level search actually improves the precision for the same level of recall.

Figure 4.2 shows the performance of LSI on the original term document matrices and on the submatrices obtained after pruned weighted level search (PWLS; results reported correspond to pruning away terms connected to only one document in the subset). The precision-recall curves are practically unchanged from those in Figure 4.1 indicating that pruning away poorly connected terms does not affect the quality of retrieval.

Figure 4.3 shows the mean over all 5 data collections of the plots in Figure 4.1 and 4.2. This figure shows that on average, weighted level search filtering improves precision at less than 50% recall. On the other hand, the precision degrades for larger levels of recall. The performance of pruned weighted level search is somewhat worse for all levels of recall. The lowest plot in Figure 4.3 shows the gain or loss in precision at all levels of recall. It is interesting to observe that despite filtering out on average more than 70% of the nonzeroes, precision degrades by less than 5.94% and can be improved by as much as 3.27%.

Figure 4.4 shows the effect of merging documents. We found that it is important to merge documents after filtering out the subset using weighted level search. If the merging is applied to the original term document matrix before level search, there is an additional problem of computing the weights of the merged documents. A simple method such as using the average weight per term over both documents is not very effective. We show the performance for the FBIS collection in Figure 4.4; note the dramatic degradation in performance when the merge is applied first.

We next consider the effect of weighted level search and pruned weighted level search on the size of the submatrix. Recall that our goal was to filter out a significantly smaller subset. The size data with and without preprocessing is shown (over all queries per data collection) in Figure 4.5. On average, LSI with weighted level search reduces the number of nonzeros to 33.84%, the number of terms to 74.03%, and the number of documents to 22.14%. Pruned weighted level search improved these results further; on average the number of nonzeros is reduced to 28.33%, and the number of terms is reduced to 35.26% (pruning does not affect the number of documents). Merging of documents on average reduces the number of nonzeros and documents by an additional 12%.

We were somewhat surprised to see that filtering could improve the retrieval quality in some instances, namely, TIME and FBIS. We conjecture that this a ‘noise reduction’ effect related to the significantly larger number of terms per document (on average) for these two collections; see Figure 4.6.

**5. Conclusions.** Symbolic filtering using level search followed by document merging appears to be an effective strategy. On average the number of nonzeros is reduced by 72% and precision-recall curves are comparable to those for LSI on the original matrix. For term-document collections with a large ratio of terms to documents, such filtering appears to improve precision by 10% – 60%.

#### REFERENCES

- [1] M. BERRY AND S. DUMAIS, *Using linear algebra for intelligent information retrieval*, SIAM Review, 37 (1995), pp. 573–595.
- [2] M. W. BERRY AND M. BROWNE, *Understanding Search Engines: Mathematical Modeling and Text Retrieval*, SIAM, Philadelphia, 1999.
- [3] M. W. BERRY, Z. DRMAC, AND E. R. JESSUP, *Matrices, vector spaces, and information retrieval*, SIAM Review, 41 (1999), pp. 335–362.
- [4] S. DEERWESTER, S. DUMAIS, G. FURNAS, T. LANDAUER, AND R. HARSHMAN, *Indexing by latent semantic analysis*, Journal of American Society for Information Science, 41 (1990), pp. 391–407.
- [5] A. GEORGE AND J. W. H. LIU, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, 1981.
- [6] T. G. KOLDA AND D. P. O’LEARY, *A semi-discrete matrix ddecomposition for latent semantic indexing in information retrieval*, ACM Transactions on Information Systems, 16 (1998), pp. 322–346.
- [7] R. R. KORFHAGE, *Information storage and retrieval*, John Wiley & Sons Inc., New York, 1997.
- [8] K. MEHLHORN, *Graph Algorithms and NP-Completeness*, Springer-Verlag, Berlin Heidelberg, 1984.
- [9] G. SALTON AND M. MCGILL, *Introduction to Modern Information Retrieval*, McGraw-Hill, New York, 1983.
- [10] X. ZHANG, M. W. BERRY, AND P. RAGHAVAN, *Level search techniques for scalable information filtering and retrieval*, Information Processing and Management, (2000). To appear.

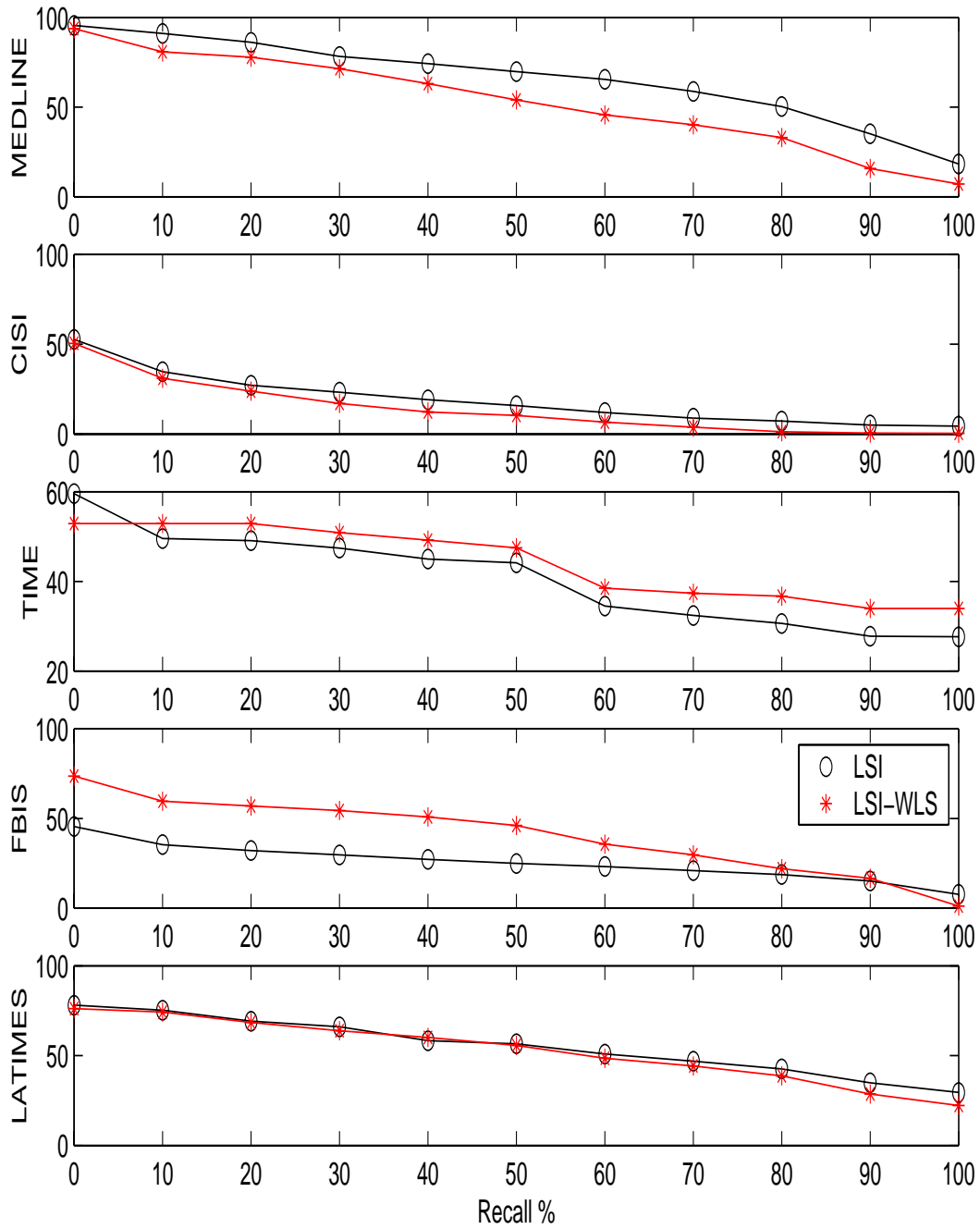


FIG. 4.1. Precision-recall curves for LSI and LSI-WLS; precision is shown along the Y-axis.

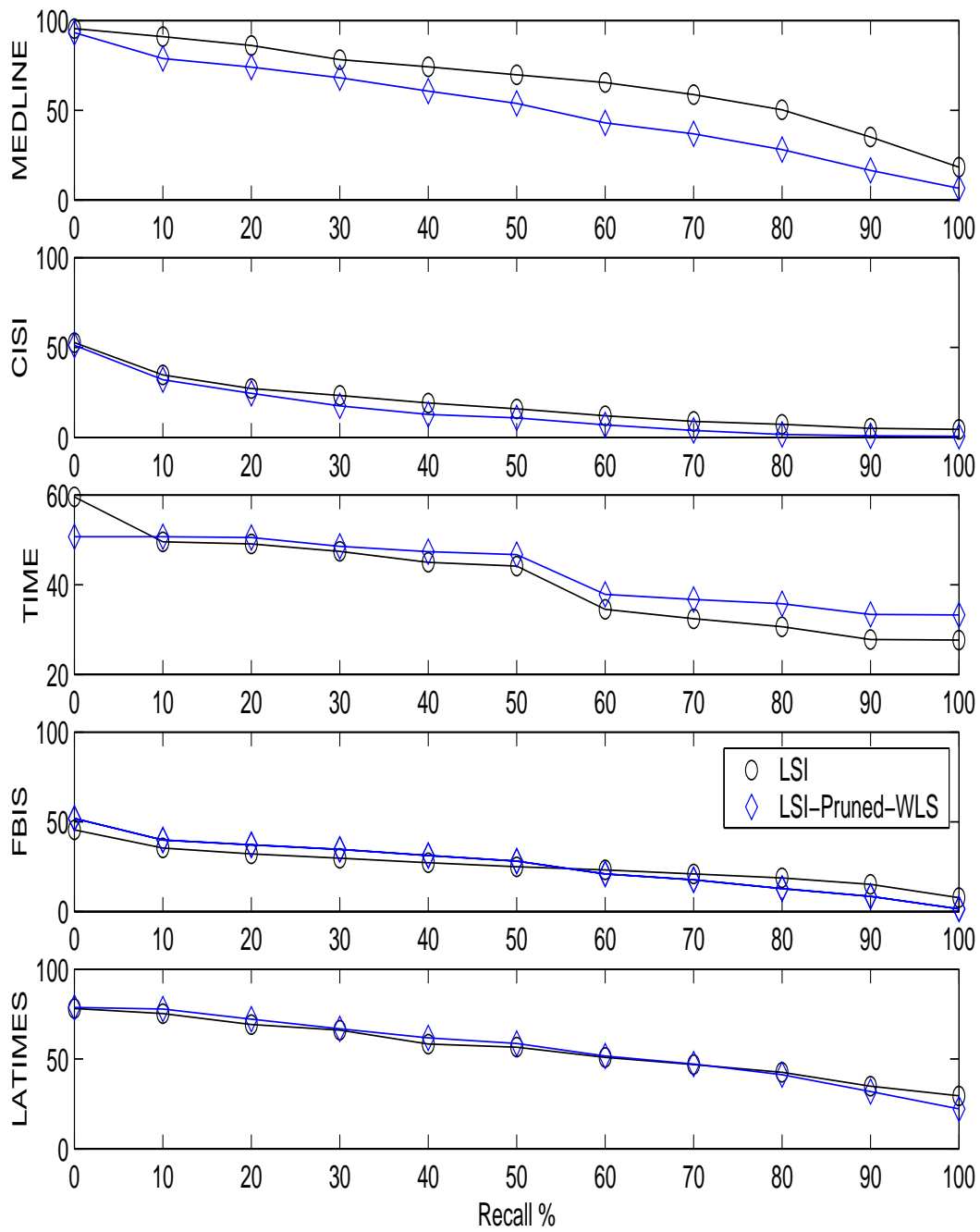


FIG. 4.2. Precision-recall curves for LSI and LSI-Pruned-WLS; precision is shown along the Y-axis.

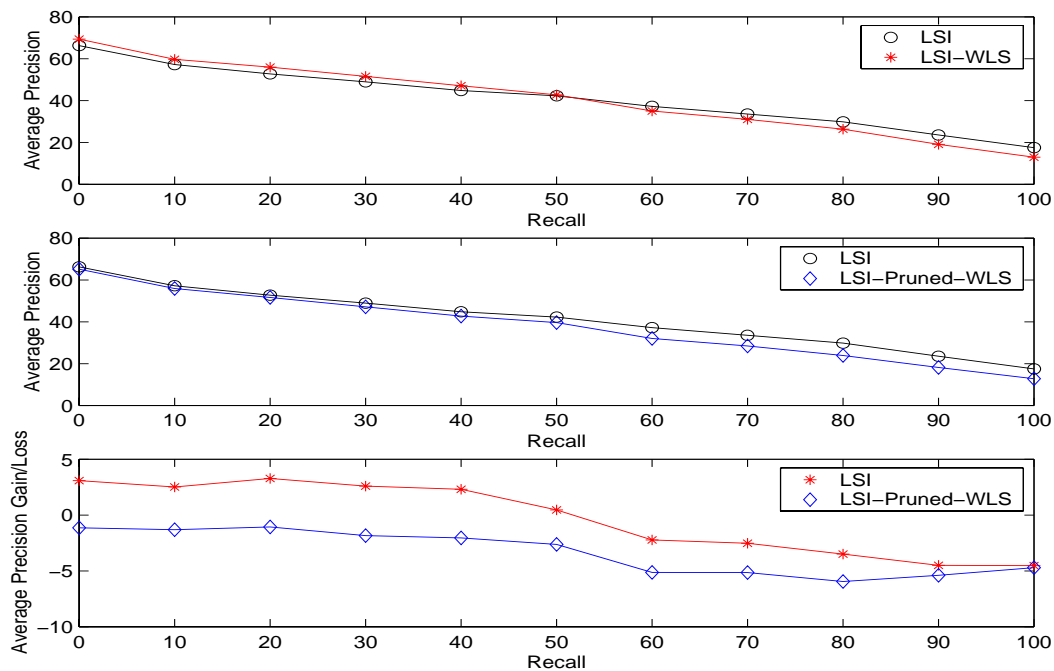


FIG. 4.3. Mean precision-recall curves for LSI, LSI-WLS, and LSI-Pruned-WLS over all 5 data collections.

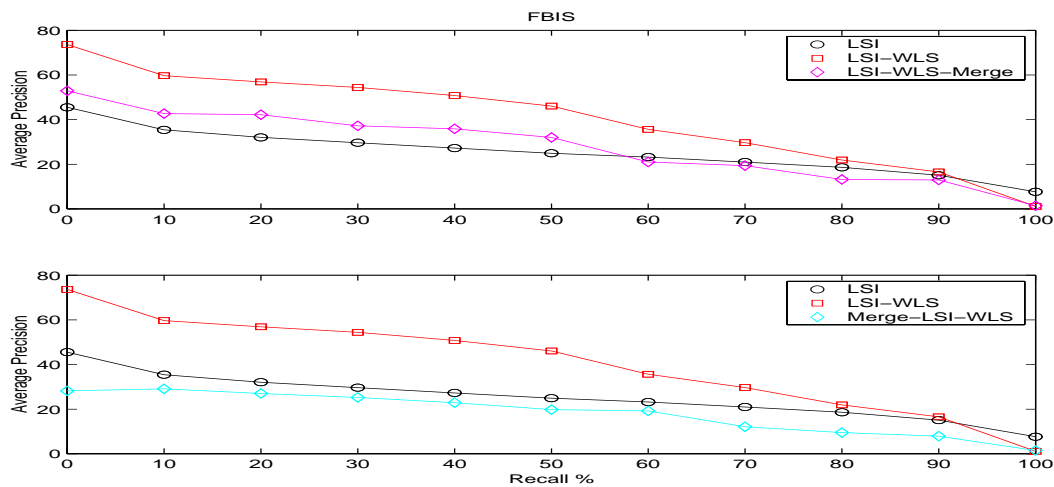


FIG. 4.4. Precision-Recall Graphs with document merging.

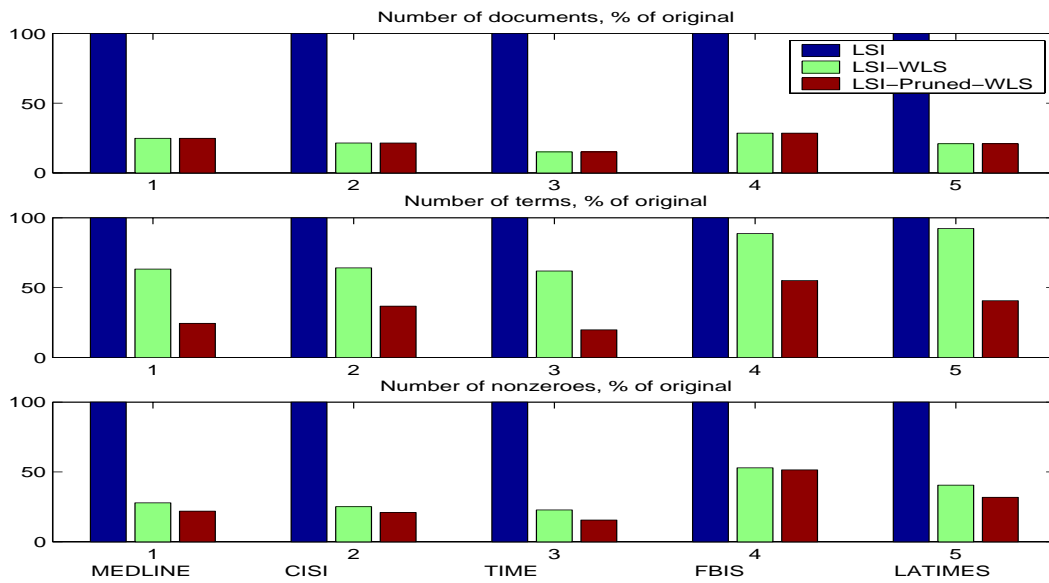


FIG. 4.5. Bar Graphs of Average Submatrix Size.

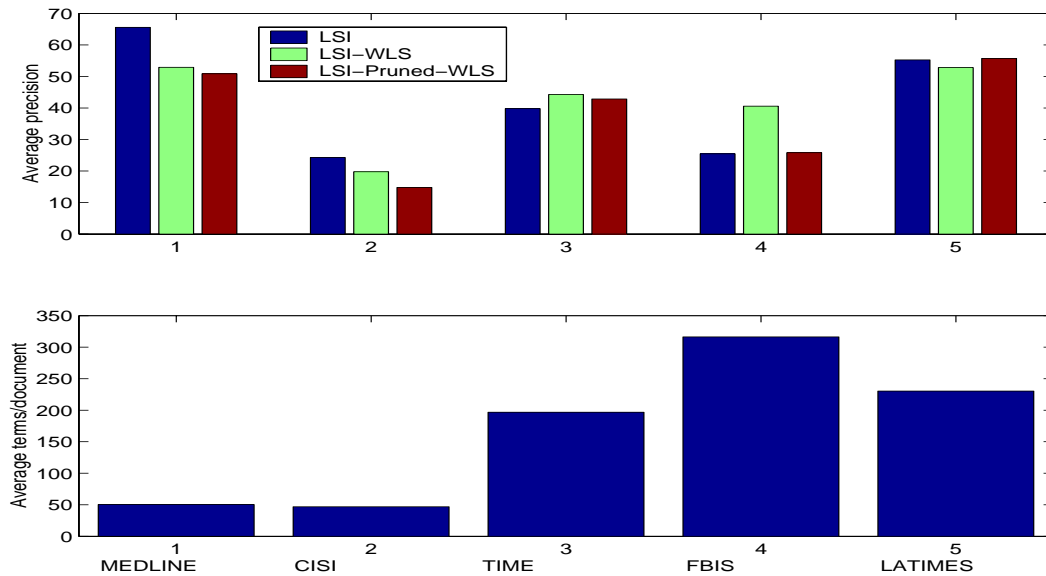


FIG. 4.6. Association of Performance with Matrix Attribute.