Dimensionality Reduction by Random Projection and Latent Semantic Indexing

Jessica Lin Dimitrios Gunopulos

Department of Computer Science & Engineering
University of California, Riverside
{jessica, dg}@cs.ucr.edu

Abstract

Document categorization and classification is of seminal importance for information retrieval. During the past decade the growth of Internet has fundamentally changed the ways that information is shared, and it has made clear that efficient methods for searching and exploring vast amounts of data should be put forward. The largest challenges associated with information retrieval are synonymy and high dimensionality. An elegant and accurate technique to solve both problems has been presented in the form of Latent Semantic Analysis (LSI). However, its high computational cost makes it infeasible for large databases. Therefore other models such as random projection have been proposed. A suggestion for combining these two approaches has also been proposed. However, best to our knowledge, no empirical results have been presented on this "hybrid" method. In this paper we evaluate and compare these three approaches and discover that the seemingly promising combination of LSI and random projection does not always result in faster running time as expected.

1. Introduction

The demand for efficient storage and retrieval of multidimensional data such as images or text has increased drastically over the past decade [12]. These types of data share the characteristics of having high dimensionality, usually in thousands or more (for example, the dimensionality of text data is the size of the vocabulary for the entire dataset). As the dimensionality of data increases, query performance diminishes. This problem, known as the "curse of dimensionality," has been given a great deal of attention by researchers, in an effort to find techniques that process queries in large databases accurately and efficiently. While this remains a challenging task, researchers have found that dimensionality reduction offers a middle ground, which generally results in faster computational time, while yielding reasonable accuracy.

An ideal dimensionality technique has the capability of efficiently reducing the data into a lower-dimensional model, while preserving the properties of the original data. In practice, however, information is lost as the dimensionality is reduced. It is therefore desirable to formulate a method that reduces the dimensionality efficiently, while preserving as much information from the original data as possible. One common way to reduce the dimensionality of data is to project the data onto a lower-dimensional subspace [13]. The two methods discussed in this paper, latent semantic indexing (LSI) and random projection, are both examples of dimensionality reduction techniques that project data onto its subspace.

1.1 Our Contribution

In this paper we provide an experimental comparison of LSI and random projection. In addition, we experiment on the hybrid algorithm, originally proposed by [16], that combines random projection and LSI. In [16], the authors show mathematically that this hybrid algorithm will be able to mitigate the computational cost associated with LSI. To the best of our knowledge, there is no experimental implementation and evaluation of this technique. This paper provides empirical results and shows that this seemingly promising idea might not always be feasible in reality. More specifically, the use of random projection prior to LSI does *not* always result in a faster algorithm than LSI.

The purpose of this paper is not to invalidate the suggestion made by previous researchers – in fact, the claim about the combination is perfectly valid theoretically. Rather, we try to interpret the discrepancy and possibly determine the factors that cause the combination method to fail.

The rest of this paper is organized as follows: Section 2 provides a detailed description of the dimensionality techniques mentioned above and a review of related work. In Section 3 we discuss the motivation of combining these techniques. We show the empirical results in Section 4 and discuss some observations made from these results. In section 5 we apply clustering on the reduced data and demonstrate the results in terms of time and accuracy. Finally we conclude and discuss future work in Section 6.

2. Background and Related Work

In this section we provide necessary background on data representation and the dimensionality reduction techniques used in this paper.

2.1 Vector Space Model

Before discussing the dimensionality reduction techniques, it is essential to understand how the data is represented. All dimensionality reduction techniques described in this paper use the same data representation model: the vector-space model [18].

In the vector-space model, each document is represented as a vector in the vector space [13]. Each dimension of the vector corresponds to one word, and the value of each component is the relative frequency of occurrence for the corresponding word in the document. As a result, an *m*-by-*n* term-to-document matrix *X* is constructed, where m is the number of unique terms in the text collection, n is the number of documents, and each element *X(i,j)* is the frequency of the i^{th} word occurring in the j^{th} document.

2.2 LSI by SVD

Singular Value Decomposition (SVD) is a well-known dimensionality reduction technique. In the process of SVD, a given rectangular *m*-by-*n* matrix *X* is decomposed into three matrices of special forms [9]:

$$X = U_{[m\times r]} \cdot S_{[r\times r]} \cdot V'_{[r\times n]} \qquad (1)$$

where *U* and *V* are orthogonal matrices that contain the left and right singular vectors of *X*, respectively, S is the diagonal matrix that contains the singular values of *X*, and the subscript *r* denotes the number of singular values (i.e. the rank of *X*). If the singular values are sorted in descending order, SVD can project the data onto a lower, *k*-dimensional space spanned by their singular vectors corresponding to the *k* largest singular values [9]. The new decomposition becomes:

$$\tilde{X} = \tilde{U}_{[m\times k]} \cdot \tilde{S}_{[k\times k]} \cdot \tilde{V}'_{[k\times n]} \qquad (2)$$

The resulting matrix $\tilde{X}$ is an approximation of *X*, and is of rank k [9]. It has been shown that $\tilde{X}$ is the best rank-*k* approximation of the original data in the least-squares sense [9].

More specifically, SVD rotates the axes to maximize variance along the first few dimensions, which usually contain the most important information about the data [9]. The chosen dimensions (i.e. the eigenvectors corresponding to the *k* largest singular values) are therefore the best axes that result in the minimum sum of squares of projection errors. See Figure 1 for a simple, 2-dimensional example.

Latent Semantic Indexing (LSI) utilizes the same idea to reduce the high dimensionality of text data [5]. It does so by keeping the first *k* largest singular values and omitting the rest. While some researchers believe that the choice of *k* is critical to the performance of LSI [5], in our experiments we show that LSI can typically preserve similarities well with relatively small *k* compared to the original dimensionality.

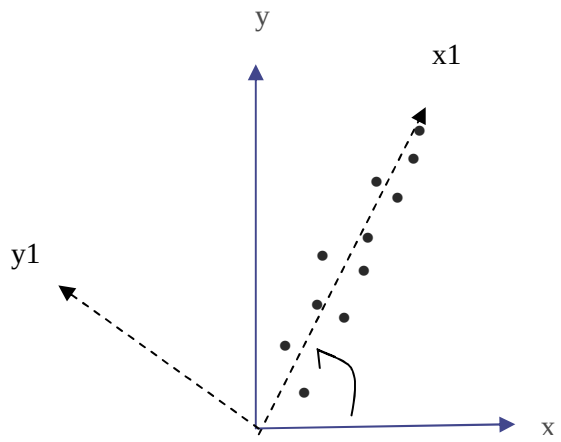

Figure 1. The axes are rotated to maximize variance along the first dimension (x1). The first dimension is chosen because it results in the minimum sum of squares of projections errors.

Since LSI is simply a dimensionality reduction technique by SVD, for the rest of the paper, we will use the terms "LSI" and "SVD" interchangeably.

2.2.1 Document Similarity Models for LSI

In [5], different document similarity models are described. From the component matrices $\tilde{U}$, $\tilde{S}$, and $\tilde{V}$, one can compute the similarities between two documents, two terms, or between a document and a term. Since we are only interested in finding similarities between documents, we explain document-to-document similarity model below and direct interested readers to [5] for further readings.

Doc-to-Doc similarities:

Recall that each column represents one document in the vector space model. In Eq. (2), the dot product between two columns of $\tilde{X}$ indicates the similarity between those two documents. Simple algebra shows that to compute the similarity between document *i* and document *j*, we can simply calculate the dot product between the ith and jth rows of the matrix $\tilde{V} \cdot \tilde{S}$.

2.2.2 Discussion

An important aspect of LSI is that it not only reduces the dimensionality of data, but it also "captures" the hidden (latent) semantic structure of the data. It constructs a semantic space and places terms and documents that are highly correlated together [5]. It finds the correlations in the data by discovering the semantic structure, rather than relying solely on the usage of terms in a given document. Consequently, the document vectors with overlapping terms are grouped together, and the terms from these documents are also pulled together. For example, "car" and "automobile" will be placed close to each other because they both co-occur in documents related to vehicles [5]. With this ability, LSI resolves the issues raised by synonymy (i.e. in many cases, there exists more than one way to describe a certain concept or knowledge; different words can have the same meaning) that other information retrieval algorithms fail to resolve. As a result, when given a query, a document of related concept may be retrieved even if it does not contain the exact terms that appear in the query.

Another advantage of LSI is that it yields high accuracy after dimensionality reduction and therefore the similarities between documents are well preserved.

One common argument against LSI, however, is that it is computationally expensive and therefore impractical for large datasets. For a given *m*-by-*n* matrix, the time complexity to run SVD is $O(m^2 n)$ [9]. It has been noted in some literature [3, 16] that, if the input matrix is large and sparse, then the running time for SVD can be reduced to $O(cmn)$, where c is the average number of non-zero entries in a vector (i.e. average number of terms in a document). There exist some SVD packages for such large, sparse matrices such as SVDPACKC [2], svds in Matlab, etc.

2.3 Random Projection

Random projection is another powerful technique for dimensionality reduction. The idea is deceptively simple: given a matrix *X*, the dimensionality of the data can be reduced by projecting it thru the origin onto a lower-

dimensional subspace, formed by a set of random vectors [13]:

$$A_{[k\times n]} = R_{[k\times m]} \cdot X_{[m\times n]} \qquad (3)$$

The *k* in the subscripts is the desired, reduced dimensionality.

The idea of random projection is motivated by the Johnson-Lindenstrauss lemma [11]:

Theorem 1 (Johnson-Lindenstrauss lemma)

For any $0 < \varepsilon < 1$ and any integer *n*, let k be a positive integer such that

$$k \geq 4(\varepsilon^2/2 - \varepsilon^3/3)^{-k} \ln n \qquad (4)$$

Then for any set *W* of *n* points in R^d, there is a map $f : R^d \rightarrow R^k$ such that for all *u, v* ∈ *W*,

$$(1-\varepsilon)\|u-v\|^2 \leq \|f(u)-f(v)\|^2 \leq (1+\varepsilon)\|u-v\|^2 \qquad (5)$$

In other words, the lemma states that a set of *n* points in high-dimensional Euclidean space can be mapped down onto an $O(\log n/\varepsilon^2)$ dimensional subspace such that the distances between the points are approximately preserved (i.e. not distorted more than a factor of $(1 \pm \varepsilon)$), for any $0 < \varepsilon < 1$ [11]. See [4] for the proof of the lemma. Further, this map can be found in randomized polynomial time [13].

2.3.1 Choice of Random Matrix

There are many proposals for the choice of the random matrix. Typically, the elements in R are Gaussian distributed. Achlioptas [1] has proposed two simpler distributions:

$$r_{i,j} = \begin{cases} +1 & with\ prob.\ \tfrac{1}{2} \\ -1 & with\ prob.\ \tfrac{1}{2} \end{cases} \qquad (6)$$

or

$$r_{i,j} = \sqrt{3} \cdot \begin{cases} +1 & with\ prob.\ \tfrac{1}{6} \\ 0 & with\ prob.\ \tfrac{2}{3} \\ -1 & with\ prob.\ \tfrac{1}{6} \end{cases} \qquad (7)$$

These simple distributions reduce computational time for the calculation of $R \cdot X$. For the second distribution, the speedup is threefold because only one-third of the operations are needed [13].

2.3.2 Orthogonalization

Theoretically, if the random vectors are orthogonal, then the similarities between the original vectors will be preserved exactly [13]. Ideally we would want the random matrix to be orthogonal but, unfortunately, orthogonalization is very costly. However, Hecht-Nielsen et. al have noted that in a high-dimensional space, there exist a much larger number of almost orthogonal vectors than orthogonal vectors [3, 10, 13]. Therefore, the random vectors might be sufficiently close enough to orthogonal to offer a reasonable approximation of the original vectors.

2.3.3 Discussion

Random projection can be applied on various types of data such as text, image, audio, etc. It is based on a simple idea and is efficient to compute, yet it yields results that are close approximation (to true similarities) that are sufficient enough for our purposes. The process of establishing the random *k*-by-*m* matrix *R* and projecting the original *m*-by-*n* matrix *X* to the lower-dimensional subspace takes $O(kmn)$ time. Furthermore, if the original data matrix is sparse, as usually the case when we work with term-document data, the time complexity is reduced to $O(kcn)$, where c is the average number of non-zero entries in a column (i.e. the average number of terms in a document) [3].

2.4 Related Work

Much work has been done on LSI, as well as random projection. Kaski [13] presented experimental results using random projection in the WEBSOM system. Bingham and Mannila [3] presented experimental results using random projection on text and image data. Papadimitriou et al. [16] provides theoretical analysis on LSI and suggests that data can be pre-processed with random projection prior to LSI to speed up the algorithm. However, to the best of our knowledge, no empirical results have been presented on combining random projection and LSI. Other projection methods have also been proposed. For example, Sasaki and Kita [19]

proposed using spherical k-means to find the concept vectors and apply projection using these concept vectors. However, they used only a very small dataset (1033 documents), and it seems that in order for their algorithm to outperform other vector space models, the number of clusters chosen has to be at least 500. This implies that every 2 documents form one concept! It's unclear how well the concept projection will perform on large datasets.

3. Combining Random Projection and LSI: Is it Feasible?

We have seen the advantages and disadvantages for both random projection and LSI: random projection is efficient in terms of computational time but does not preserve as much information as LSI; LSI, on the other hand, is computationally expensive, but produces highly-accurate results, in addition to capturing the underlying semantics of the documents. As mentioned earlier, a hybrid algorithm was proposed that combines the two approaches to benefit from the advantages of both algorithms. The algorithm works as follows: first the data is pre-processed with random projection to a lower dimension k_1, then LSI is applied on the reduced, lower-dimensional data, to further reduce the data to the desired dimension, k_2. This algorithm supposedly will improve running time for LSI, and accuracy for random projection. In this paper we empirically demonstrate that this unfortunately is not the case, and we believe the key factor lies in the sparseness of the data.

As mentioned earlier, the time complexity of SVD is $O(cmn)$ for large, sparse datasets. It is reasonable, then, to assume that a lower dimensionality will result in faster computation time, since it's dependent of the dimensionality m. However, one important property has lost from the process of random projection, and that's the sparseness of the data. The sparseness of the original data matrix makes it possible to attempt the otherwise costly SVD algorithm. After the reduction, the matrix becomes dense[1], and the cost for running sparse SVD routine becomes even higher than running on the original data! Another factor that contributes to the deterioration of efficiency is the memory consumption. More memory is needed to store the reduced matrix in the sparse format and to process the decomposition. Even if we have enough memory, the high memory consumption is likely to affect the efficiency.

4. Experimental Study

We performed a series of experiments using the dimensionality reduction techniques mentioned in the previous sections. We used two subsets of Reuters categorization text collection [15], omitting empty documents and those without topic labels. Common and rare words are removed and the vocabulary is stemmed with the Porter Stemmer [17]. The words are converted into a term-document matrix, in which each entry denotes the frequency of the occurrences of the corresponding word. Each document vector is normalized to unit length before further processing.

The larger subset has 10377 documents. After stemming, the vocabulary size is 12113. The term-document matrix is extremely sparse, with a density of only 0.004. The dimensionality of the original data (i.e. 12113) is then reduced to lower k-dimensions. The smaller dataset is pre-processed in a similar fashion, and has 1831 documents, 5414 terms, and density of 0.008. We conducted a set of experiments with different choices of $k \in \{50, 100, 200, 300, 400, 500, \text{and } 600\}$, and compared their results. The same experiments are repeated for each algorithm: LSI, random projection (RP) and the combination of random projection and LSI (we will refer to this as RP_LSI). For RP_LSI, we experimented on the permutation of different parameters. First, we reduced the dimensionality using random projection to $k_1 \in \{400, 600\}$. We also tried $k_1 = 1000$, but couldn't complete the experiment for the larger dataset due to memory constraint[2]. From the reduced matrix, we further reduced it to

[1] This result is independent of the choice of random matrix. We have also tried the simple random distribution (see Eq. 7), but the resulting matrix is still too dense for the sparse SVD routine to perform well.

[2] This is done in Matlab, on a machine with 1-GB of memory. Recall the burden on memory for processing large dense matrix using the sparse SVD routine. If $k1 = 1000$, then the input matrix for the sparse SVD routine is a dense 1000-by-10000 matrix!

$k_2 \in \{50, 100, 200, 300\}$ using LSI (and 400 when $k1 = 600$).

4.1 Metrics for Measuring the Similarity

To measure the accuracy of the results, we have to determine how well the similarities are preserved during the process of dimensionality reduction. Two commonly used similarity measures are the Euclidean distance and the cosine of the angle between vectors. As mentioned in section 2.2.1, the similarity between two documents can be measured as the dot product between two document vectors. However, we also included Euclidean distance as a distance measure because it's the criterion based on which SVD seeks to optimize [16].

The purpose for dimensionality reduction is apparent – it makes information retrieval faster than operating on the original data. Since *k* does not vary with different reduction techniques, the goal for this part of experiments is therefore to show how well the reduction techniques preserve and reconstruct the original relationship. To achieve this, we have to compare the document-to-document correlation *before* and *after* the reduction, by calculating the distance between them using the measures mentioned above. We randomly selected 100 pairs of documents and measured their similarities on the original and reduced data matrices. We repeated the process 10 times for each experiment, each time with random document pairs, and took the average.

4.1.1 Euclidean Distances

According to the Johnson-Lindenstrauss lemma, the expected form of a projection of a unit vector onto a random subspace through the original is $\sqrt{m/k}$ [3, 11]. Therefore, to take into account the decreased dimensionality of the data, the Euclidean distance is scaled by $\sqrt{m/k}$.

4.1.2 Cosine of angle between documents (Dot Product)

The similarity between two document vectors v_j and v_k can be determined by computing the cosine of angle ϑ between them:

$$sim(v_j, v_k) = \cos(\theta) = \sum_{i=0}^{m} \frac{v_{ji} \cdot v_{ki}}{\|v_j\| \cdot \|v_k\|} = \frac{v_j \cdot v_k}{\|v_j\| \cdot \|v_k\|} \quad (8)$$

Since we normalize the vectors to unit length prior to reduction (i.e. $\|v_i\| = \|v_j\| = 1$), the cosine of the angle between them is simply their dot product.

4.1.3 Determining Error

As mentioned earlier, we randomly selected 100 document pairs, and calculated their distances before and after dimensionality reduction. To determine the accuracy, or the amount of similarity preserved, we computed the projection error ε. The error is determined by computing the correlations of the distance vectors obtained from the original and reduced data, respectively.

The correlation coefficient *r* can be expressed as follows [18]:

$$r = \frac{1}{n} \sum \left(\frac{x - \bar{x}}{s_x}\right)\left(\frac{y - \bar{y}}{s_y}\right) \quad (9)$$

where *n* is the number of document pairs (i.e. 100); x and y are the distance vectors computed from the original and reduced data, respectively; $\bar{x}$ and $\bar{y}$ are the means of *x* and *y*, and S_x and S_y are the standard deviations for the distance vectors. After finding the correlation between the original and the reduced distances, we can determine the error by subtracting the correlation from 1:

$$\varepsilon = 1 - r. \quad (10)$$

4.2 Experimental Results

In this section we show the results from different dimensionality reduction algorithms in terms of running time and accuracy (measured by error ε).

4.2.1 Evaluating the Distances before and after Dimensionality Reduction

Table 1 shows an example of the errors resulted from different dimensionality reduction methods, from the larger dataset. The smaller dataset produces similar results. The reduced dimensionality is 200. For RP_LSI, the

dimensionality is first reduced to 600 via random projection.

	LSI	RP	RP_LSI
Dot Products	0.0439	0.1088	0.0645
Euclidean Distances	0.0415	0.0847	0.0500

Table 1. Errors from different methods. The parameter k for the instance shown here is 200, and k_1 = 600 (for RP_LSI).

From Table 1 we observe that RP_LSI yields fairly good results, with very small error rate.

4.2.2 Comparing Running time

Table 2 shows the running time for all methods from our experiments for the large dataset. For RP_LSI, dimensionality is first reduced to 600 by random projection.

k	LSI	RP	RP_LSI
50	145.3	0.91	820.86
100	384.4	1.93	1476.94
200	1254.9	4.14	2715.90
300	2731.5	6.46	3783.8
400	4334.95	8.75	4843.0
500	6631.08	11.06	N/A
600	9368.15	13.34	N/A

Table 2. Running time for all methods, measured in seconds, for the larger dataset. Note that we did not further reduce k to 500 or 600, as it's not quite as meaningful since the data has already been reduced to 600 with random projection.

The following figure displays the result presented in Table 2.

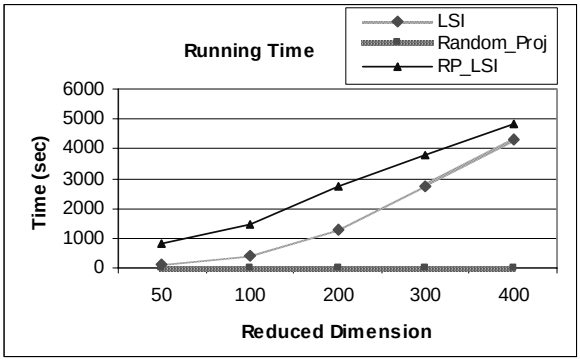

Figure 2. Running time for all algorithms. Note that for LSI and RP_LSI the running times tend to converge as k increases.

Table 2 and Figure 2 show that while random projection takes very little time to compute, applying it before LSI makes the total execution time even longer than running LSI alone. These results are unexpected and contradict the common anticipation for the combination approach. However, it seems that the running time tends to converge for LSI and RP_LSI as k increases. On the other hand, when we run the algorithms on the smaller dataset, we get conflicting results. More specifically, we discover that RP_LSI is indeed faster than LSI for the smaller dataset. Table 3 demonstrates this. This set of experiment (i.e. for the smaller dataset) was run at an earlier date on a different machine. Therefore, the numbers from Table 3 should not be compared side-by-side with the number in Table 2. However, examining the data *within* Table 3, we can see that RP_LSI is about twice as fast as LSI. This inconsistency suggests that the efficiency of the hybrid algorithm is data-dependent. While random projection seems of little utility for our large, sparse dataset, it might be useful as a pre-processing tool for other types of data (i.e. small or non-sparse).

k	LSI	RP	RP_LSI
50	130.1	1.1	107.4
100	379.9	2.1	236.5
200	1406.8	4.2	677.6
300	3141.4	6.3	1469.5
400	5589.2	8.4	2312.3
500	8918.7	10.5	4205.4
600	14007.9	12.5	6680.4

Table 3. Running time for all methods for the smaller dataset.

4.2.3 Comparing the Error

Figure 3 and 4 summarize the errors from all the algorithms with different similarity measures. The results show that the combination method produces more accurate results than random projection. LSI typically produces the best results, as expected.

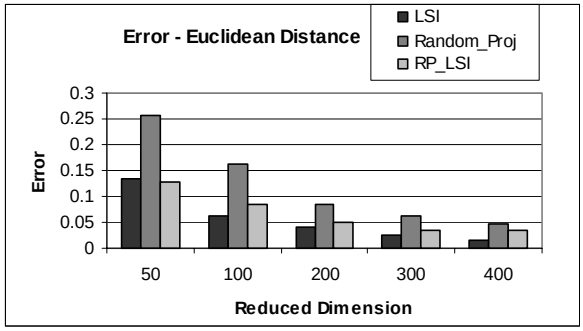

Figure 3. Errors for the Euclidean distance metric. We can see that RP_LSI improves the accuracy of random projection.

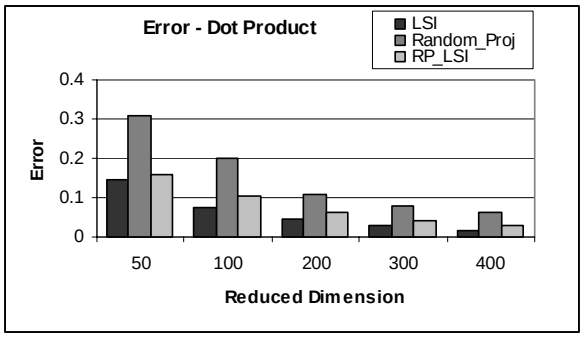

Figure 4. Errors for the dot product. Again, RP_LSI improves the accuracy of random projection.

Figure 5 shows the error for combining random projection and LSI. The bars in the front row are the errors for Euclidean distance, and the back row for the dot product. Each "column" of bars along the x-axis denotes the errors from different levels of dimensionality reduction: the first number in each label is the "intermediate" dimensionality resulted from random projection, and the second number from LSI (also the final dimension). From the plot we observe that the amount of second reduction (i.e. by LSI) is more critical than the amount of the first reduction. This observation suggests that LSI plays a more important role in preserving similarity.

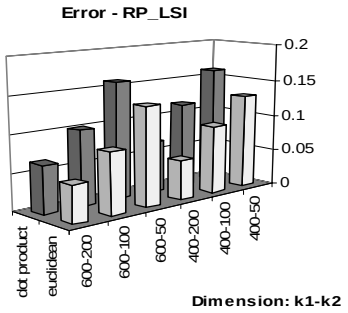

Figure 5. Error after combining random projection and LSI. Note the amount of second reduction (k2) makes more significant differences than the first reduction.

5. Application Study: Clustering Documents

As the last part of the experiments, we apply clustering on the data before and after the dimensionality reduction to verify the results from the previous section. The results shown here are from the smaller dataset. Our choice of clustering algorithm is k-Means, for its effectiveness and considerably lower computational cost as compared to other clustering algorithms such as hierarchical clustering. There are many variations of k-Means, such as the one in [20] that finds the global optimum. For simplicity, we use the classic k-Means in this work.

5.1 Background on K-Means Clustering

The basic intuition behind k-Means [8, 14] (and a more general class of clustering algorithms known as iterative refinement algorithms) is to partition the dataset by assigning each data point to its closest cluster center. The cluster centers can be randomly selected initially. With each subsequent iteration, the centers are re-computed and the points are re-distributed. The process is repeated until no data point changes cluster membership.

5.2 Clustering Results

As in previous section, we use Euclidean distances and the cosine similarity as the distance metrics. Similar to the spherical k-Means algorithm described in [7], the document vectors are normalized to unit length before clustering, as well as the centroids computed thereafter. Clustering is applied *before* and *after*

dimensionality reduction. Since the k-Means algorithm seeks to optimize the objective function by minimizing the sum of intra-cluster errors, we evaluate the quality of clustering by comparing the objective functions. However, since the dimensionality of data is reduced when we apply the three algorithms, we have to compute the objective functions on the original data to make the comparison possible. This can be easily achieved by mapping the class labels for the document vectors, returned by the k-Means algorithm, to the original data space, and computing the objective functions using the information on clustering assignment. The number of clusters *k* we choose is 5 (since there are roughly 5 main topics from the smaller dataset). Since we use random seeds as initial centers, we repeat k-means 20 times on each experiment and take the average.

Figure 6 summarizes the accuracy of clustering for all techniques, using cosine similarity. Clustering using Euclidean distance produces similar results, so the plot is omitted here. The bottom thick line is the objective function obtained when clustering the original data, and the rest three "curves" show the mapped objective function obtained when clustering the reduced data. Though it's difficult to see, LSI produces better results (i.e. smaller objective functions) than RP_LSI. The fact that all data vectors, as well as the computed centroids, are normalized to unit length might be the reason that the mapped objective functions show little variation.

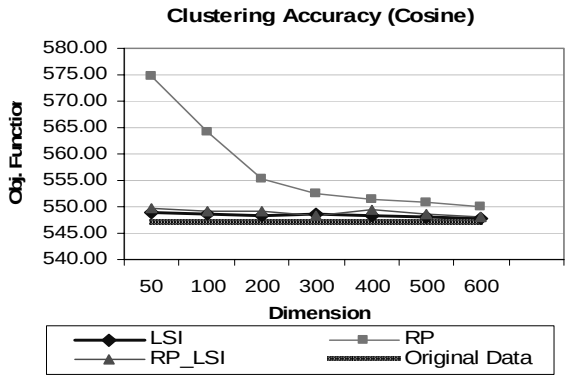

Figure 6. Clustering accuracy for cosine similarity. The bottom thick line is the objective function for the original data. RP shows significant improvement as dimensionality increases, while LSI and RP_LSI show very little improvement.

Figure 7 shows the running time for clustering after the dimensionality reduction. We only show the clustering time from one set of experiments here because the running time is mostly dependent on the dimensionality of the (reduced) dataset (i.e. it is invariant of how the data is reduced). The number shown on the y-axis is the fraction of time needed to cluster the original data.

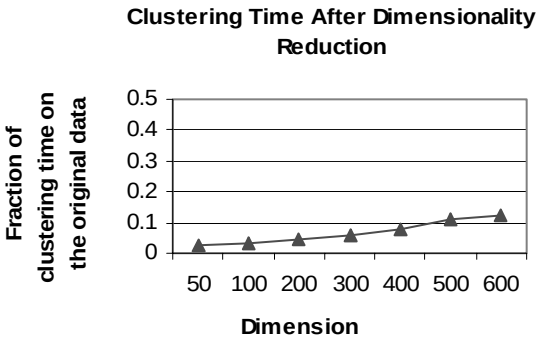

Figure 7. Running time for clustering after dimensionality reduction, represented as the fraction of clustering time on the original data.

6. Conclusions and Future Work

In this paper we compare two well-known dimensionality reduction techniques, random projection and latent semantic indexing. We verify the effectiveness of these methods. We also combine LSI with random projection to test the "effectiveness" of the combination, though our results are inconclusive. In attempt to interpret the controversy on the running time, we believe that the sparseness of the data plays an important role on how SVD performs. On the other hand, we can safely conclude that LSI helps improve the accuracy of random projection.

As mentioned earlier, the purpose of our experiments is not to invalidate the suggestion made by previous researchers – in fact, the claim about the combination is perfectly valid theoretically. Instead, we present this unexpected observation in attempt to find the factors that cause the combination approach to fail and hopefully, as future work, a resolution.

Furthermore, a more comprehensive set of experiments is needed to verify whether the sparseness of data is indeed the cause of the

discrepancy on running time. One way to do that, and to eliminate the possibility of implementation bias, is to use a different SVD package, such as the widely-used SVDPACKC [2]. In addition, we can use other dimensionality reduction algorithms that preserve the sparseness of the data, and see if that helps improve the running time for LSI.

7. References

[1] D. Achlioptas. Database-Friendly Random Projections. In *Proc. ACM Symp. On the Principles of Database Systems*, pages 274-281, 2001.

[2] M. W. Berry, T. Do, G. W. O'Brien, V. Krishna, and S. Varadhan. SVDPACKC (Version 1.0). *User's Guide*. University of Tennessee, April 1993.

[3] E. Bingham and H. Mannila. Random Projection in Dimensionality Reduction: Application to Image and Text Data. *Knowledge Discovery and Data Mining*, 1998.

[4] S. Dasgupta and A. Gupta. An Elementary proof of the Johnson-Lindenstrauss lemma. *Technical Report TR-99-006*, International Computer Science Institute, Berkeley, California, USA, 1999.

[5] S. Deerwester, S. T. Dumais, G. W. Furnas, and T. K.Landauer. Indexing by Latent Semantic Analysis. *Journal of the Am. Soc. for Information Science*, 41(6):391-407, 1990.

[6] M. H. Degroot and M. Schervish. Probability and Statistics. *Addison-Wesley Publishing*.

[7] I. S. Dhillon and D. S. Modha. Concept Decompositions for Large Sparse Text Data Using Clustering. *Technical Report*, IBM Almaden Research Center, 1999.

[8] R. C. Dubes and A. K. Jain, Algorithms for Clustering Data. *Prentice Hall*, 1988.

[9] G. H. Golub, C. F. Van Loan. Matrix Computations. *Johns Hopkins Press*, Baltimore, MD, 1989.

[10] R. Hecht-Nielsen. Context Vectors: General Purpose Approximate Meaning Representations Self-Organized from Raw Data. J. M. Zurada, R. J. Marks II, and C. J. Robinson, editors, *Computational Intelligence: Imitating Life*, pages 43-56. IEEE Press, 1994.

[11] W. B. Johnson and J. Lindenstrauss. Extensions of Lipshitz Mapping into Hilbert Space. In Conference in modern analysis and probability, *volumn 26 of Contemporary Mathematics*, pages 189-206. Amer. Math. Soc., 1984.

[12] K. V. Kanth, D. Agrawal, and A. Singh. Dimensionality Reduction for Similarity Searching in Dynamic Databases. *SIGMOD*, 1998.

[13] S. Kaski. Dimensionality Reduction by Random Mapping. In *Proc. Int. Joint Conf.* On Neural Networks, volume 1, pages 413-418, 1998.

[14] L. Kaufman and P. J. Rousseeuw, Finding Groups in Data: An Introduction to Cluster Analysis. *John Wiley and Sons*, 1990.

[15] D. Lewis, Reuters-21578 Text Categorization Test Collection Distribution 1.0. http://www.research.att.com/~lewis

[16] C. H. Papadimitriou, P. Raghavan, H. Tamaki, and S. Vempala. Latent Semantic Indexing: A Probabilistic Analysis. In *Proc. 17th ACM Symp. On the Principles of Database Systems*, pages 159-168, 1998.

[17] M. Porter. An algorithm for suffix stripping, Program, 14(3): 130-137, 1980. http://www.tartarus.org/~martin/PorterStemmer

[18] G. Salton, A. Wong, and C. S. Yang. A Vector Space Model for Automatic Indexing. *Communications of the ACM*. 18:613-620, 1975.

[19] M. Sasaki and K. Kita. Vector Space Information Retrieval Using Concept Projection. In *Proc. Of the 19th International Conference on Computer Processing of Oriental Languages*, pp. 73-76, 2001.

[20] H. Zha, C. Ding, M. Gu, X. He, and H. Simon. *Advances in Neural Information Processing Systems 14*. T. G. Dietterich, S. Becker and Z. Ghahramani, editors. MIT Press. Cambridge, MA, 2002.