Search Engines and Data Clustering

Ralph Abbey
North Carolina State University
rwabbey@ncsu.edu

Jeremy Diepenbrock
Washington University in St. Louis
jcdiepen@wustl.edu

Dexin Zhou
Bard College
dz989@bard.edu

Dr. Carl D. Meyer
North Carolina State University
meyer@math.ncsu.edu

Shaina Race
North Carolina State University
slrace@ncsu.edu

Abstract

We discuss the basic foundation of search engines and various clustering algorithms. Additionally, we propose a new clustering algorithm as an adaptation of an already well-known algorithm. From the results of these algorithms, we form an aggregation matrix and perform clustering on that. Finally, we compare the performance of many of these algorithms on various data sets.

Contents

1 Introduction

This paper details most of the proceedings of our group during the summer 2007 Research Experience for Undergraduates (REU) program. Over the course of the summer, we did some investigation into the workings of search engines, but ended up focusing our work on data clustering. After delving into previously published algorithms, we tried some of our own adaptations in an attempt to improve on existing algorithms. Finally, through the use of Matlab, we applied all these algorithms to a variety of data sets in order to see how their performance compared in terms of both clustering quality and computing time.

2 Search Engines

2.1 Challenges

Due to the unregulated nature of the World Wide Web, search engines face some challenges, including:

- The structure of the World Wide Web has no set pattern.
- Data are not necessarily standardized.
- Data may be falsified, producing undesired results.

2.2 Vector Space Model

There are 3 types of search engines: Boolean, probabilistic model, and vector space model. We chose to focus on the vector space model.

The vector space model is based on a term-by-document matrix A where A_{ij} is some function of the raw frequency, f_{ij}, which is the number of times term i occurs in document j.

$$A_{m\times n} = \begin{array}{c} \\ \text{Term 1} \\ \\ \text{Term i} \\ \\ \text{Term m} \end{array} \begin{array}{c} \begin{array}{cccccc} \text{Doc 1} & & & & \text{Doc j} & \text{Doc n} \end{array} \\ \begin{pmatrix} & & & | & & \\ & & & | & & \\ & & & | & & \\ - & - & - & A_{ij} & & \\ & & & & & \end{pmatrix} \end{array}$$

Typically, A_{ij} is a product of local weighting (l_{ij}), global weighting (g_i), and normalization (d_j) factors. In general, local weighting is used to downplay words used frequently within a certain document while global weighting is used to downplay words used frequently across an entire document collection. Below

are some examples of possible weighting functions.

Local Term Weighting

- $l_{ij} = \delta(f_{ij}) = \begin{cases} 1 \text{ if } f_{ij} > 0 \\ 0 \text{ if } f_{ij} = 0 \end{cases}$ (binary weighting)
- $l_{ij} = \log(1 + f_{ij})$ (logarithmic weighting)
- $l_{ij} = \frac{1}{2}(\delta(f_{ij}) + \frac{f_{ij}}{\max\limits_{k} f_{kj}})$ (augmented normalized term frequency)

Global Term Weighting

- $g_i = 1$ (each term equal weight)
- $g_i = 1 + \frac{1}{\log(n)} \sum_j P_{ij} \log(P_{ij})$ where $P_{ij} = \frac{f_{ij}}{\sum_j f_{ij}}$ (entropy weighting)
- $g_i = \log\left(\frac{n}{\sum_j \delta(f_{ij})}\right) = \log\left(\frac{\#\text{ of docs}}{\#\text{ of docs w/ term i}}\right)$ (inverse document frequency)
- $g_i = \left(\sum_j f_{ij}^2\right)^{-\frac{1}{2}}$ (normal)
- $g_i = \frac{\sum_j f_{ij}}{\sum_j \delta(f_{ij})}$ (GFIdf)

Normalization Factor (d_j): $d_j = \left(\sum (l_{ij} g_i)^2\right)^{-\frac{1}{2}}$

We use these factors to produce:

$$A_{ij} = l_{ij} g_i d_j$$

resulting in $A_{m \times n}$, where m is the number of terms and n is the number of documents.

Once the term-by-document matrix has been prepared, a query can be processed. A query is a column vector q where

$$q_i = \begin{cases} 1 \text{ if term i is in the query} \\ 0 \text{ if term i is not in the query} \end{cases}$$

To calculate relevance between the query and each individual document, angular distance is used. The angular distance between the query vector q and document j is

$$\cos(\theta_j) = \frac{q^T A e_j}{\|q\|_2 \|A e_j\|_2}$$

Documents are then sorted based on their θ_j values and returned in order of relevance to the query (with the smallest θ_j as the most relevant).

But the calculation $q^T A$ ignores connections between words that have similar semantics (synonyms, e.g.). To deal with this issue, we reduce the rank of A using the Singular Value Decomposition (SVD).

Initially, $A_{m\times n}$ is factored such that $A = USV^T$, where $U_{m\times m}$ and $V_{n\times n}$ are orthogonal matrices (i.e. $UU^T = I$ and $VV^T = I$) and $S_{m\times n}$ is a diagonal matrix with the singular values (σ_i) along the diagonal in descending order, with the largest at the upper left. If we view this decomposition as a linear combination of outer products, it becomes clear that the SVD produces a fourier expansion of A.

So

$$A = \sigma_1 U_1 V_1^T + \sigma_2 U_2 V_2^T + \cdots = \sigma_1 Z_1 + \sigma_2 Z_2 + \cdots$$

where $Z_i = U_i V_i^T$. This sum can be truncated after the k^{th} term, resulting in

$$A_k = \sigma_1 U_1 V_1^T + \cdots + \sigma_k U_k V_k^T$$

which is called the rank k approximation of A.

Reducing the rank of A often causes documents with similar semantic meaning to have closer vector representations, a result known as Latent Semantic Indexing.

2.2.1 Choosing k

One challenge of rank reduction is choosing the optimal k in order to maximize the true signal from the data and minimize the noise. In the formula $A = \sigma_1 Z_1 + \sigma_2 Z_2 + \cdots$, each σ value can be thought of as a coefficient in the Z basis where the projection of A onto Z_i is represented by σ_i. Since $\sigma_1 \geq \sigma_2 \geq \sigma_3 \geq \cdots$, the projection of A onto Z_i decreases as i increases. Thus, σ_i is the amount of information in A that is directed along Z_i. If we assume the noise is distributed equally across each Z_i, then it is clear that truncating this sum will reduce noise at a greater rate than it will information, or "signal".

To choose an optimal k, we define two terms:

- % signal remaining $= \dfrac{\sum_{i=1}^{k} \sigma_i}{\sum_{i=1}^{r} \sigma_i}$ where r is the rank of A.
- % noise remaining $= \frac{k}{r}$

These definitions imply that σ_i corresponds to the amount of signal in Z_i and that the amount of noise is evenly distributed across all Z_i.

We sought to find the k that maximizes the value (% signal - % noise).

3 Data Clustering

Due to a close relation between the vector space model and data clustering, we were able to transition away from search engines, a shift in focus motivated by the large potential for exploration in data clustering. We implemented several different clustering algorithms including our own naive method which we called "tree sort", the standard and well-known k-means algorithm, Nonnegative Matrix Factorization, Principal Direction Divisive Partitioning (PDDP), and our own adaptation of PDDP which we called Principle Direction Gap Partitioning (PDGP). In order to process our data-sets and visualize our results, we used two tools which eased the task of text parsing and cluster visualization called Text to Matrix Generator, and Vismatrix.

3.1 Tools

3.1.1 Text to Matrix Generator (TMG)

The Text to Matrix Generator (TMG) is a MATLAB graphical user interface (GUI) created by Dimitrios Zeimpekis and Efstratios Gallopoulos from the Deparment of Computer Engineering and Informatics at the University of Patras, Greece. The tool has been paramount to our investigation of textual search and document clustering. It allows the user to input a set of documents which the program then parses. The tool outputs a dictionary of terms found in the document collection, a list of the document titles, and the term-document matrix, employing either raw frequency or local and global term weighting as dictated by the user.

TMG also includes other useful features for the purposes of clustering, classification, and information retrieval. TMG has a separate GUI for clustering, which allows the user to input a dataset and cluster their document set with k-means, spherical k-means, and PDDP. This feature made it easy to make comparison among algorithms. Implicit in the clustering GUI is the capability for dimension reduction of the data matrix via SVD [5].

3.1.2 Vismatrix

Vismatrix is a tool that was created by David Gleich at the Stanford University Institute for Computational and Mathematical Engineering. This program allows the user to visualize the entries of a sparse matrix by representing each entry in the matrix as a colored marker, where smaller values are represented by

white and blue pixels and more significant values produce orange to red markers. Once a data matrix is clustered, the matrix with permuted columns and rows can be entered into Vismatrix and the hope is that the clustering will be apparent in the formation of large blocks of significant values. The program also allows the user to use a cursor to move around the matrix to different values and see the titles and terms associated with the coordinates of the cursor. Thus, one can move around inside a clustered block to investigate the accuracy of the clustering. As is apparent in the picture below, Vismatrix is an invaluable tool for visualizing the clustering of large matrices. The following matrix has 17,770 rows and columns.

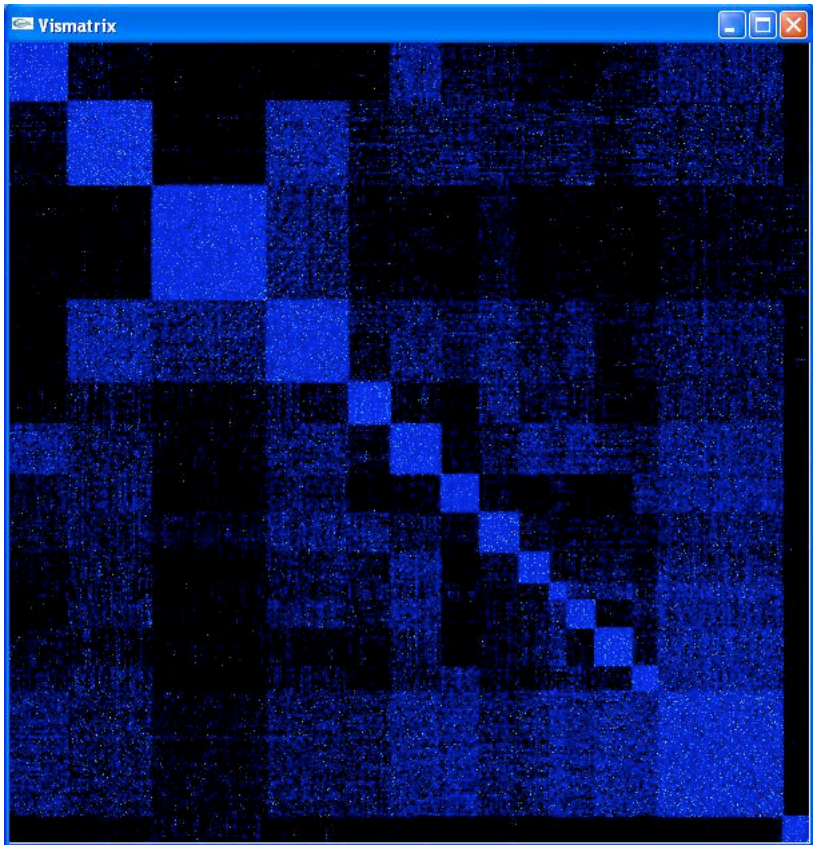

Figure 1: An example of a Vismatrix Visualization

3.2 Tree Sort

We wrote the tree sort algorithm with no prior clustering knowledge. It was our uninformed attempt at writing our own clustering algorithm. Although it performed respectably when the dimension of the data was greater than the

number of items to be clustered, it broke down when the data had more items than dimensions. Nevertheless, it proved to be useful in choosing initializations for other algorithms.

The algorithm works as follows:

1. Start with entire document collection.
2. Find most "popular" word in collection based on maximum row sum.
3. Split collection into documents that include the "popular" word and those that don't.
4. For the documents that include the "popular" word, let that word be a category for that collection.
5. For each collection from step 3, repeat back to step 1.

3.3 K-Means Clustering

K-means seeks to minimize the square distance between each data point and the centroid of the cluster with which it is associated. That is, it minimizes $\sum_{i=1}^{k} \sum_{x_j \in S_i} |x_j - \mu_i|^2$, where S_i represents each cluster, μ_i is the centroid of the cluster S_i, and k represents the number of clusters.

We used Lloyd's algorithm to implement k-means, which performs k-means for k clusters as follows:

1. Randomly initialize k centroids.
2. Associate each data point with the closest centroid based on Euclidean distance.
3. For each centroid, find the mean of all data points associated with it. This becomes the new centroid.
4. Repeat steps 2 and 3 until the centroids converge.

In step 2, Euclidean distance is used to measure closeness. However, we also used angular distance to measure closeness and found that it made little or no difference in the results for our data set.

The outcome of k-means clustering is highly dependent on the location of the initial centroids. Therefore, clustering results are frequently improved by making more educated guesses for the initial centroids. We adapted our tree sort algorithm so that its clustering results served as an initialization for k-means. This made the results of the algorithm better and more consistent.

3.4 Non-Negative Matrix Factorization

Non-Negative Matrix Factorization (NMF) is an algorithm first proposed by Daniel Lee and Sebastian Seung in [4] which seeks to find a parts-based representation of multivariate data by factoring a matrix **A** into 2 non-negative matrices **W** and **H** such that

$$A_{m\times n} \approx W_{m\times k} H_{k\times n}$$

where the columns of W can be thought of as parts or features and the entries in **H** represent the extent to which each feature is present in each data measurement. The factorization provided by NMF is not unique or exact. NMF algorithms attempt to minimize $\|\mathbf{A}-\mathbf{WH}\|$, but convergence to a local or global minimum is not guaranteed. The value of k is input by the user, and determines the number of clusters or "parts" desired from the data.

In the application of textual clustering, we can view such a factorization as decomposing each column of **A** into a linear combination of k "topic-vectors"

$$A_i = \sum_{j=1}^{k} W_j H_{ji}$$

The idea is that the columns of the matrix **W** will yield information about how the terms in the dictionary contribute to each "topics" and the rows of the matrix **H** will provide information about the extent to which each topic is represented in each document. Thus, we are able to obtain clustering information about both terms and documents from the Nonnegative Matrix Factorization of **A**. Although there are many ways in which to use this information, the simplest way is to pick the largest value in each row of **W** and assign that term to that cluster number. For example if the first term in our dictionary were Apple and the largest entry in the first row of **W** appeared in column 5, then the term Apple would be assigned to cluster 5. Similarly, we use the values in the columns of **H** to designate each document to a cluster.

In [3], Patrik O. Hoyer developed a way to enforce sparsity on both matrices **W** and **H** explicitly. In many applications, a sparse, nonnegative matrix factorization is preferred because information can be pulled more readily from the resulting factors, and storage is reduced. Although it is apparent that enforcing sparsity could ease the task of data clustering, we found that the sparsity of the original term-document matrix usually permeated through the algorithm and resulted in sparse factors **W** and **H**. Enforcing further sparsity seemed to diminish the quality of the clustering.

3.5 Principal Direction Divisive Partitioning

Principal Direction Divisive Partitioning (PDDP), developed by Daniel Boley of the University of Minnesota, is one of many clustering algorithms based upon

the Singular Value Decomposition.

The first step is to center the data. That is, the mean of the document vectors is set to zero. The mean μ of all the columns of $\mathbf{A}_{m\times n}$ is $\mu = \mathbf{Ae}/n$, where $\mathbf{e}$ is a $n \times 1$ vector of all ones. In order to center the matrix, μ is subtracted from each column, resulting in

$$\mathbf{C} = \mathbf{A} - \mu\mathbf{e}^T = \mathbf{A}\left(\mathbf{I} - \frac{\mathbf{ee}^T}{n}\right)$$

The centered term-by-document matrix $\mathbf{C}$ is factored into $\mathbf{C} = \mathbf{USV}^T$. The first column in $\mathbf{U}$, or $\mathbf{u}_1$, is known as the principal direction because it is the direction in which the data have the greatest variance. The data in $\mathbf{C}$ are then projected onto $\mathbf{u}_1$. Conveniently, this projection ends up being proportional to the values of $\mathbf{v}_1$, as shown below:

$$\mathbf{u}_j^T\mathbf{C} = \mathbf{u}_j^T\mathbf{USV}^T = \sigma_j\mathbf{v}_j^T$$

where σ_j (the j^{th} singular value) is the proportionality constant [2].

Thus the projection of the data onto the principal direction is easily obtained from the values of $\mathbf{v}_1$. Next, the data are split into 2 groups: those that have $\mathbf{v}_{1i} > 0$ (positive projections onto the principal direction) and those that have $\mathbf{v}_{1i} \leq 0$. Boley arbitrarily chose to put those data with $\mathbf{v}_{1i} = 0$ into the group with negative projections. This split effectively divides the term-by-document matrix into 2 separate matrices, each corresponding to the sign of their respective projections.

After this initial split, a measure of scatter is used to determine which cluster is to be split next. Since the term-by-document matrix was split in the previous step, each newly-created matrix must be recentered in the same manner as at the beginning. Then, the scatter for each cluster is computed as the Frobenius norm of the centered term-by-document matrix corresponding to that cluster. The cluster with the greatest scatter value is split and the process is repeated until the desired number of clusters is attained [2].

In some cases, using uncentered data may be desirable due to computation time or data storage needs, as centering the matrix destroys the sparse nature of the data. In this case, the data is not centered before each iteration of PDDP. Due to the fact that all entries of the $\mathbf{v}_1$ will have the same sign, the $\mathbf{v}_2$ vector is now the vector where the split at 0 must occur. The difference between clusterings when using centered data or uncentered data, however, is not currently known.

3.6 Principal Direction Gap Partitioning

Principal Direction Gap Partitioning is our modification of Boley's PDDP algorithm.

3.6.1 Definitions

Recall that our centered term-by-document matrix $\mathbf{C}$ is factored into $\mathbf{C} = \mathbf{USV}^T$. For convenience, we simplify the notation $\mathbf{v}_1$, the first column in $\mathbf{V}$, as $\mathbf{v}$. We define a gap and a gap interval of a vector $\mathbf{v}_{n\times 1}$ as follows:

- Sort the values in $\mathbf{v}$ in ascending order. Call this $\mathbf{v}'$.
- Define $d\mathbf{v}'$ such that $d\mathbf{v}'_i = \mathbf{v}_{i+1} - \mathbf{v}_i$ for $i = 1, 2, \ldots, n-1$.
- Say the maximum value of $d\mathbf{v}'$ occurs at $d\mathbf{v}'_k$. Then the gap in $\mathbf{v}$ occurs between $\mathbf{v}_k$ and $\mathbf{v}_{k+1}$.
- The gap interval of $\mathbf{v}$ is the interval $[\mathbf{v}_k\ \mathbf{v}_{k+1}]$.

3.6.2 Description

We felt that the PDDP split in $\mathbf{v}$ at 0 was fairly arbitrary so we looked into different methods for determining the split. Therefore, we sorted $\mathbf{v}$ in order to see where the proposed split was taking place. After plotting the values of the sorted $\mathbf{v}$ vector, we noticed a significant gap in the values. When doing the first division on our data set, 0 was included in the gap interval (Figure 2). So splitting based on signs performed the same action as splitting at the gap. However, when we did the second iteration and went to split the cluster with greatest scatter, the gap interval did not include 0 (Figure 3). In fact, there is not a noticeable break in the values of $\mathbf{v}$ at 0. The horizontal line representing the split of PDDP crosses through the sorted $\mathbf{v}$ values at a point that is clearly not a clean break in the values. On the other hand, the gap in the sorted $\mathbf{v}$ values occurs around 0.1, and the gap interval is entirely above 0. We suggest splitting at this gap instead of at 0.

However, these gaps sometimes occur very close to the ends of $\mathbf{v}'$, which would result in unbalanced clusters. Therefore, we chose to ignore a certain percentage of the values in $d\mathbf{v}'$ in order to avoid choosing gaps close to either end, which might isolate too few data points.

3.6.3 Using Multiple Singular Vectors with PDGP

We have also tried to incorporate the information in singular vectors other than $\mathbf{v}_1$. To do this, we looked at the gaps in both $\mathbf{v}_1$ and $\mathbf{v}_2$ and used the vector that had the longer gap interval. For labeling purposes, we called this algorithm PDGP2. Theoretically, this method should use whichever vector produces a cleaner split in the data. However, we have yet to get this method to produce better clusters than those formed by the normal PDGP algorithm.

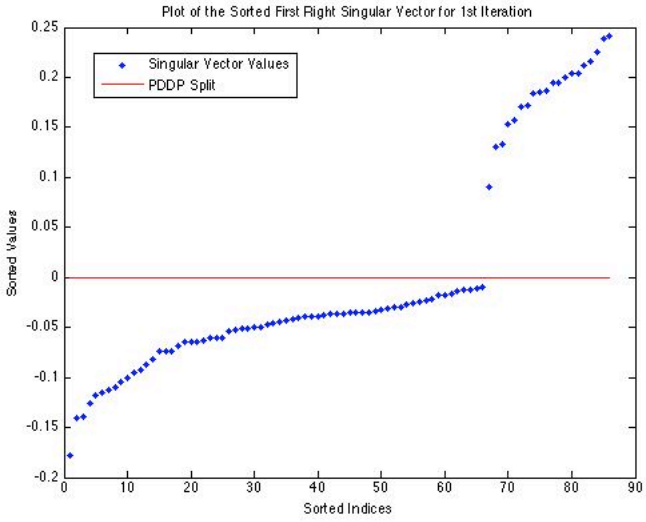

Figure 2: Sorted **v** values for first iteration

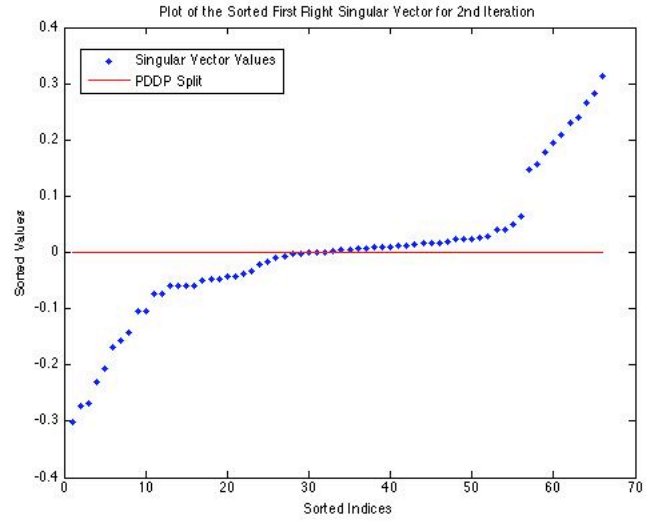

Figure 3: Sorted **v** values for second iteration

3.7 Cluster Aggregation

Cluster aggregation is the idea of using the results of multiple clustering algorithms to form one overall clustering result. The motivation for cluster aggre-

gation comes from the fact that different algorithms yield different clustering results. When attempting to cluster data sets about which there is no prior knowledge, it cannot be known which clustering algorithm will show the best clustering results. The hope is that by aggregating the clusters from different algorithms into one overall cluster a better quality clustering will occur.

3.7.1 How Cluster Aggregation Works

Cluster aggregation is a several step process that relies on other algorithms to create adjacency matrices, which are square matrices that express the similarity between documents. These adjacency matrices are added together to form an overall aggregated adjacency matrix which can then be clustered one final time to yield clusters. The process is as follows:

1. A set of algorithms is run on the term-by-document matrix representing the data.
2. For each algorithm an adjacency matrix is formed by assigning 1 to M_{ij} if document i and document j are in the same cluster. It can be easily seen that these matrices will be symmetric.
3. The adjacency matrix for each algorithm is added to all of the other adjacency matrices.
4. A clustering algorithm is run on the aggregated matrix.

3.7.2 Our Claims For Cluster Aggregation

We claim the following about cluster aggregation, and will test this method to investigate the extent to which our claims are valid,

- Cluster aggregation can yield better clustering than using a single algorithm.
- While slow, cluster aggregation can theoretically be made faster by use of parallel computing.
- Cluster aggregation can act as a filter so that poor clusters are filtered out.

3.7.3 Cluster Aggregation Example

For this example we shall imagine that we have four clusters of documents: 1-5, 6-10, 11-15, and 16-20. In this example we will assume that there have been three clustering algorithms run: A, B, and C. However, as the clustering algorithms are not perfect algorithm A has clustered document 1 with documents 2, 3, 6, and 11; algorithm B has clustered document 1 with documents 3, 4, 5,

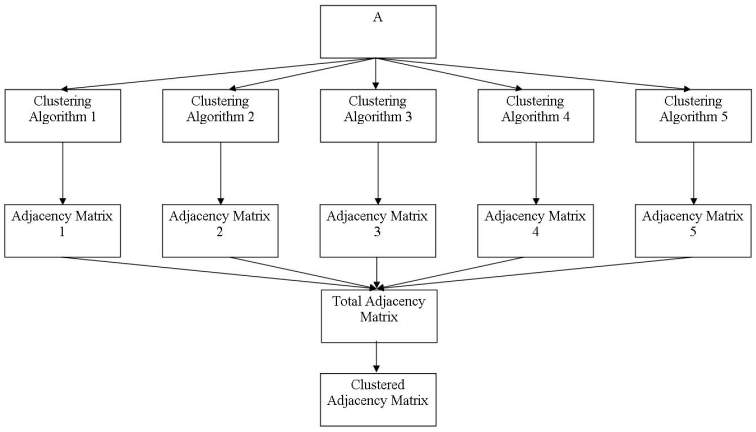

Figure 4: Flowchart of the Cluster Aggregation Process

and 19; and algorithm C has clustered document 1 with documents 2, 5, 8, 12, and 20. Each of these clusters can be seen to be less than optimal, however with cluster aggregation the fact that document 1 was clustered with 2, 3, and 5 twice, while with 4, 6, 11, 19, 8, 12, and 20 once, shows that 1 is more relevant to documents 2, 3, and 5 than the others. In this case 4 was not included, but when more algorithms are run there is a larger chance the the better clusters will fall out.

The reason behind the good clustering is that in the previous example there are four good connections that document 1 can make (2, 3, 4, and 5), while there are fifteen poor connections that document 1 can make. If we assume that poor connections are made with almost random distribution, and that each clustering algorithm will get some good connection and some poor connections, we can see that the error will spread over a large area and make less of an impact on the final clustering.

4 Data Sets

To test these various algorithms, we made our own textual data set and downloaded other benchmark sets from the Internet.

4.1 Our Own Data Set

Our data set consists of 86 documents from the World Wide Web, containing 2413 terms. We used the Term Matrix Generator tool to parse our data and produce the term-by-document matrix. This tool has options for local weighting, global weighting, normalization, and stemming, but we eventually used augmented normalized term frequency for local weighting, no global weighting, no normalization, and no stemming. The general topics of the documents in our data set are chess (grandmasters, tactics, openings), global warming, computer learning (chess, poker, backgammon), Blackstone (IPO, buyout), backgammon, UNC sports, non-negative matrix factorization, and poker.

4.2 Benchmarks

We also worked with the following data sets commonly used as benchmarks in data mining:

- Reuters-10 [1]: A subset of Reuters-21578, a collection of documents published by Reuters in 1987.
- Cranfield: A collection of articles on aerodynamics
- Medline: Medical documents
- CISI

4.3 Other Data Sets

- Boley's Data Set: For comparison purposes, we obtained the J1 document collection used by Daniel Boley in his PDDP paper.
- Grocery Store Data: SAS provided us with some grocery store sales data with 48 stores and 10982 items
- Netflix: Derived from user ratings for various movies, we obtained a movie-by-movie matrix (17770×17770) that gives associations between movies

5 Results

5.1 Metrics

To compare the clusterings of different data sets, we used two different metrics. The density metric requires no prior knowledge of the data but does not

provide a single value for comparison. On the other hand, the entropy metric requires labels on each document yet it returns a single value, allowing for easy comparison among the algorithms.

5.1.1 Density Metric

The density metric inputs the term-by-document matrix, row clusters, and column clusters. Then, it creates a grid of blocks by crossing the row clusters with the column clusters. For each blocks, it calculates the block density, where

$$\textbf{Block Density} = \frac{\text{\# of non-zero points in the block}}{\text{\# of lattice points in the block}}$$

If the block density is greater than an inputted tolerance, then that block is deemed an "acceptable" cluster and all the nonzero entries are considered relevant. After this procedure is done on all the blocks, the overall density is calculated as

$$\textbf{Overall Density} = \frac{\text{\# of relevant points}}{\text{total \# of nonzero entries}}$$

Finally, so that lower numbers imply higher quality clusters, the metric returns $1 -$ (Overall Density).

Unfortunately, this metric is dependent on the density tolerance chosen so the results of various algorithms must be compared over a range of tolerances. This requirement produces a graph that shows the relative performance of algorithms yet does not produce an easy comparison between them.

5.1.2 Entropy

A common metric to measure the quality of clusterings is the entropy method. Its formula is as follows:

$$e_{total} = -\frac{1}{m}\sum_j n_j \cdot \sum_i \left(\frac{c(i,j)}{\sum_i c(i,j)}\right) \log\left(\frac{c(i,j)}{\sum_i c(i,j)}\right)$$

where n_j is the number of documents in cluster j and c(i,j) is the number of occurrences of label i in cluster j.

This formula comes from a popular measure in information theory called "self information" or "surprisal". The surprisal of a variable x is defined as $\log_2\left(\frac{1}{p(x)}\right)$ where $p(x)$ denotes the probability that x occurs. The value of the surprisal indicates the extent to which one is surprised by the occurence of event x. If the probabilty that x occurs is 0, then one is infinitely surprised that the event occured. On the other end, if the probability that x occurs is 1, then it would be no surprise to the observer that the event occured. Entropy is essentially the

expected value of the self-information or surprisal of an event.

The difficult part about applying this metric is that it requires prior knowledge of clusters in order to give labels. Additionally, for very large data sets, it may not be practical to label every document. Essentially, if labels are provided or easily obtained, entropy is a good standard for measuring cluster quality. However, these labels frequently require significant time and effort to obtain.

5.2 86 Mini-Document Set

The following figures display the effect of row and column permutations on the term-by-document matrix. Though there seems to be very little white space in the unclustered matrix, the clustered matrix reveals significant block structure and the ability to concentrate the white space. Each dot represents a point where the term corresponding to that row occurs at least once in the document corresponding to the column.

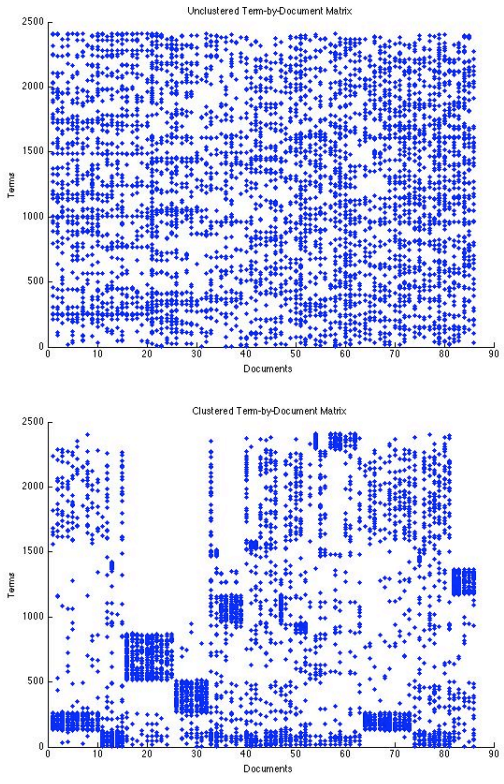

Figure 5: Unclustered (top) and clustered (bottom) term-by-document matrix

The results of various algorithms run on the 86 mini-document set are shown in the charts and table below.

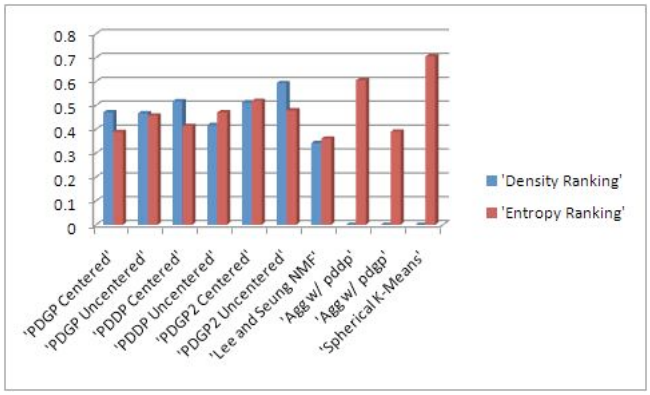

Figure 6: Entropy and Density Rankings for 86-mini document set

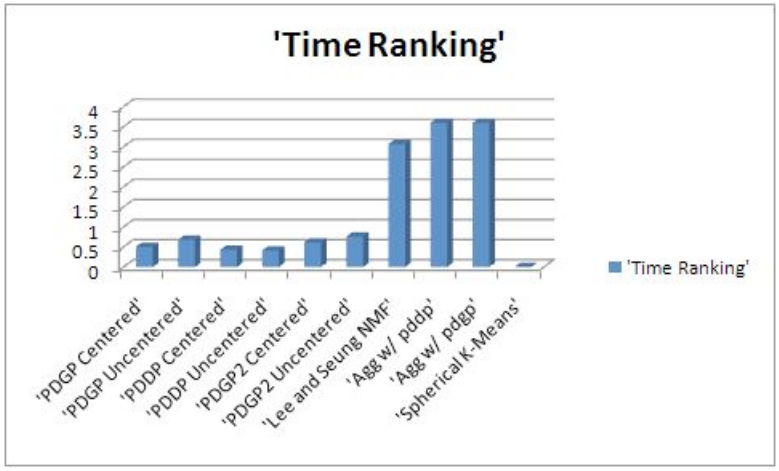

Figure 7: Time Rankings (in seconds) for 86-mini document set

Table 1:

Methods	Density Ranking	Entropy Ranking	Time Ranking (in seconds)
PDGP Centered	0.4707	0.389	0.4846
PDGP Uncentered	0.4675	0.4578	0.6865
PDDP Centered	0.5177	0.4146	0.4342
PDDP Uncentered	0.4183	0.4713	0.4006
PDGP2 Centered	0.5132	0.5194	0.4814
PDGP2 Uncentered	0.5938	0.4798	0.7313
Lee and Seung	0.3424	0.3611	3.1115
Agg w/ pddp	0.5194	0.6303	3.5607
Agg w/ pdgp	0.4491	0.3905	3.5649
Spherical K-Means		0.7054	

5.3 Daniel Boley's Document Set (J1) Results

We normalized the columns in the term-by-document matrix so that the euclidean norm for each column was one, so that we could corroborate our data with the results that Daniel Boley was able to obtain in his paper. The results from various algorithms run on the PDDP paper document set are shown in the charts and table below.

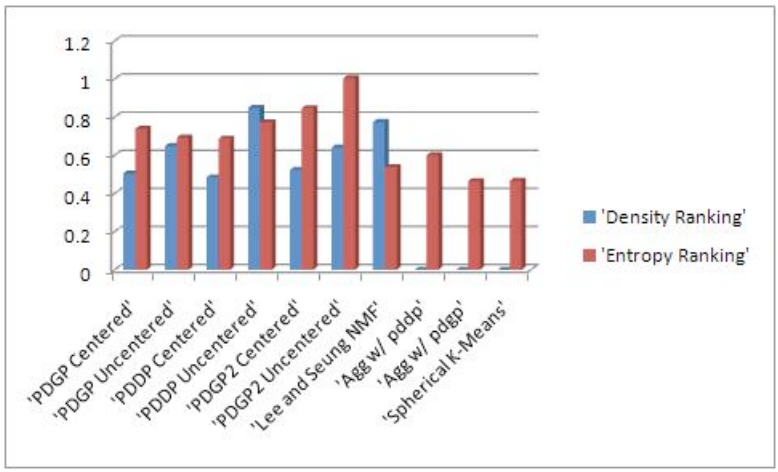

Figure 8: Entropy and Density Rankings for PDDP paper document set

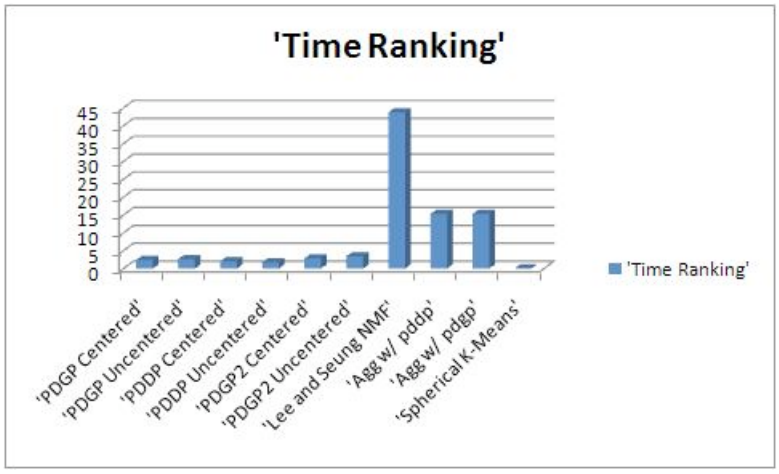

Figure 9: Time Ranking (in seconds) for PDDP paper document set

Table 2:

Methods	Density Ranking	Entropy Ranking	Time Ranking (in seconds)
PDGP Centered	0.505	0.7405	2.3876
PDGP Uncentered	0.6496	0.6946	2.5744
PDDP Centered	0.4854	0.6886	2.0479
PDDP Uncentered	0.8488	0.7742	1.762
PDGP2 Centered	0.5249	0.8485	2.8313
PDGP2 Uncentered	0.6411	1.0049	3.4367
Lee and Seung	0.7924	0.4661	44.1468
Agg w/ pddp		0.6012	15.5568
Agg w/ pdgp		0.4655	15.5855
Spherical K-Means		0.4675	

5.4 Discussion of Document Set Results

It is important to note, before any detailed observations are made, that for PDGP, 10% of $d\mathbf{v}'$ is being disregarded, and that for PDGP 2, 20% of $d\mathbf{v}'$ is being disregarded. Changing the percent of $d\mathbf{v}'$ that is disregarded can drastically change the values for entropy and density rankings for both of these algorithms and further tests with varying percents should be run for a full understanding of its effect on the results and capabilities of each algorithm.

In both the 86 mini-document set and the PDDP paper document set, Lee and Seung's NMF algorithm clusters well according to the entropy metric. However

the downside to this algorithm is that it is very time expensive, nor does it converge to the same value each time the algorithm is run, as other test runs of the Lee and Seung NMF have shown higher values in the entropy metric. A detailed examination of the performance of Lee and Seung, then, would only prove useful in the case of multiple data sets each clustered many times. As of now we have not looked into this as there are more efficient non-negative matrix factorization algorithms available. However, the inclusion of Lee and Seung is important to give at least a general idea of how well different algorithms are clustering. It should be noted that this NMF algorithm is one of the slowest available. As shown in [1], there are more efficient and consistent ways of using NMF for data-clustering.

5.4.1 Centering

We can see that from Table 1 that the centered algorithms of PDGP and PDDP yield results closer to the known clusters than do the uncentered PDGP and PDDP algorithms. However, the uncentered PDGP 2 outperforms the centered PDGP 2. A similar trend follows in Table 2, where the centered PDDP and PDGP 2 algorithms perform better than the uncentered ones, while the uncentered PDGP does a better job of clustering than the centered PDGP. In each case two of the three algorithms examined, in terms of centering vs. noncentering, yields more desireable results when centered.

Due to the code used for each of the algorithms using centered data takes less time than using uncentered data in the case of PDGP and PDGP 2. This contrasts the case of PDDP where uncentered data allows for slightly faster computation time. However, as previously mentioned, PDDP used with centering performs better in either document set than used with noncentering.

5.4.2 PDDP, PDGP, and PDGP 2

With the two data sets tested PDGP 2 does not cluster as well as either PDDP or PDGP. However, between PDDP and PDGP there is not consistency to which provides better clustering. In the 86 mini-document set PDGP with centering performs better than PDDP with centering and noncentering. In the PDDP paper document set PDDP with centering performs better than PDGP with centering and noncentering.

From Tables 1 and 2 PDDP is seen to be faster than PDGP, which itself is seen to be faster than PDGP 2 in the case where centering is only compared with centering speeds and noncentering is only compared to noncentering speeds.

5.4.3 Cluster Aggregation

Cluster aggregation as used for these document sets used only SVD based algorithms in the aggregation. That is, only PDGP, PDDP, and PDGP 2, with and without centering, were incorporated into the aggregation matrix. In both cases of cluster aggregation, cluster aggregation using pdgp to cluster the aggregated adjacency matrix provides better results than using pddp. The times between the two indicate that clustering the aggregated matrix with pdgp is slightly slower, but not to any significant amount.

In the test cases cluster aggregation performs just as well, if not better, than the other algorithms examined, excepting the cases of Lee and Seung, and the slight difference between pdgp and the cluster aggregation, both seen in Table 1. For the PDDP paper document set, cluster aggregation with pdgp clustering yields an entropy that is in the neighborhood of 66% of the entropies of the other SVD based clustering algorithms. For the 86 mini-document set the entropy yielded is about 95% of the other entropies, excluded the entropy in the case of PDGP with centering.

5.5 Grocery Store Data

In clustering the SAS Grocery Store Data, we performed cluster aggregation on both the 48 stores and the 10982 items.

5.5.1 Store by Store

The store-by-store aggregation matrix depicts the strength of the association between each pair of stores. In the figures, reds and yellows signify that those 2 stores have been clustered together by most of the algorithms, blues and greens signify that those 2 stores have been clustered together by a couple algorithms, and black signifies that the 2 stores have never been clustered together. The second figure is produced by a permutation of the rows and columns of the first figure in order to display the desired block diagonal structure.

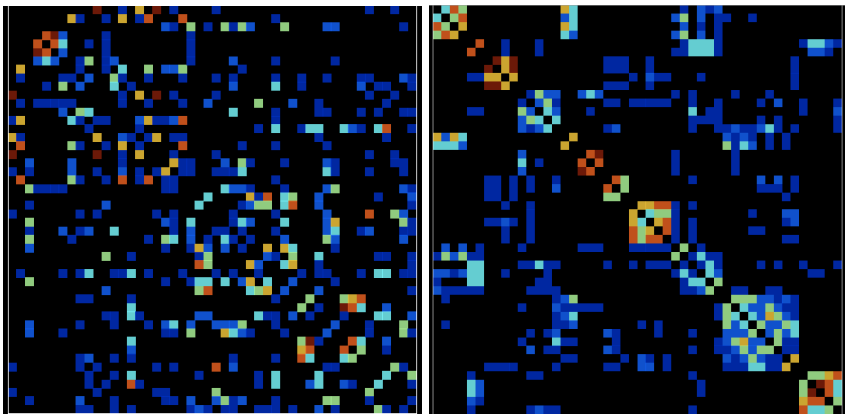

Figure 10: Unpermuted (left) and permuted (right) store-by-store aggregation matrix

5.5.2 Item by Item

Due to the size of the item-by-item aggregation matrix (10982×10982), the matrix shows only a binary representation, where red represents a strong correlation between items and black represents weak or no correlation. Again, the row and column permutations on the original matrix results in a block diagonal structure that was not initially apparent.

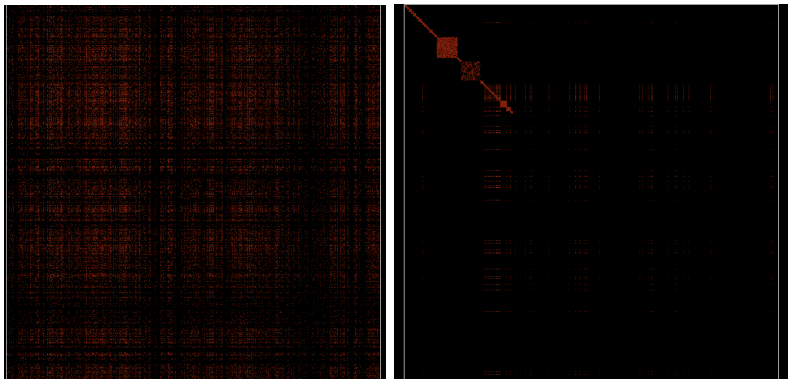

Figure 11: Unpermuted (left) and permuted (right) item-by-item aggregation matrix

5.6 Netflix

We ran both PDDP and PDGP on the movie-by-movie matrix of the Netflix data in order to establish categories among the movies. The larger number of

blocks on the diagonal in the PDGP picture signifies that PDGP found more categories in the data than PDDP found.

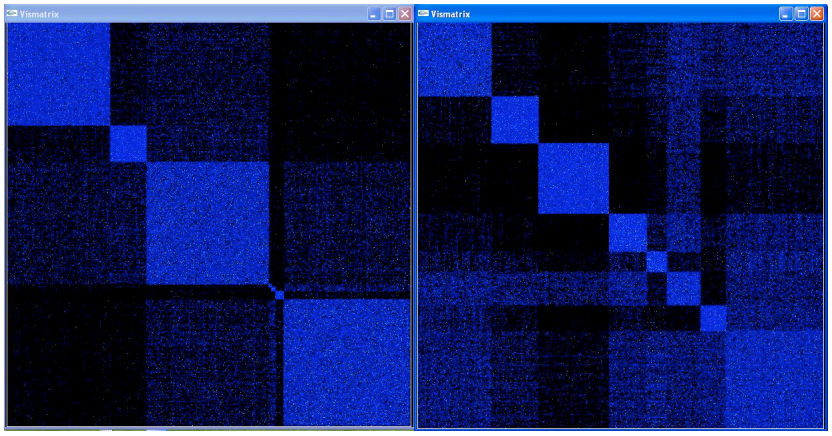

Figure 12: Netflix Matrix Clustered Using PDDP (left) and PDGP (right)

6 Conclusions

From our results we can make some general conclusions in response to various questions that we have investigated throughout our research. Specifically we can make statements regarding our adaptation of PDDP, PDGP, cluster aggregation, and the issue of centering and noncentering data.

6.1 PDGP

As discussed in our results section and seen in Tables 1 and 2 therein we know that PDGP provides promise as a clustering algorithm in terms of clustering effectiveness, as well as computation time. While we have only experimented with the most basic form of PDGP we have thought of future questions that can be investigated using PDGP which we address in our future work section.

6.2 Cluster Aggregation

Our results show empirical evidence to validate our claim that cluster aggregation acts as a filter so that poor clustering that occurs in the component algorithms gets filtered.

We know from our results that cluster aggregation doesn't always outperform all of it's component algorithms. However, as only one of the component algorithms outperformed cluster aggregation by a very minute difference, we still feel that cluster aggregation may outperform all of its component algorithms a very high percentage of the time.

We have been unable to show either way if parallel computing will work to increase the speed of cluster aggregation.

6.3 Centering and Noncentering

We have found techniques that allow for fast computing of the singular value decomposition while centering the data. This means that using centered data does not take significantly longer for computation time than using uncentered data.

We have been unable to determine the effect that centering the data has on the quality of the clusters. However, we feel that there is no loss in quality of data when the data is centered.

6.4 SAS Grocery Store Data

After using cluster aggregation on the SAS grocery store data we can conclude that there are some very definitive store clusters as well as item clusters. This would imply that certain stores sell similar items in similar quantities, and also

that certain items are often sold at the same stores. However, these implications cannot be checked leaving us only with the knowledge that there are clusters among the stores and items sold.

7 Further Work

We have many ideas left untested and/or unproven. The following is a brief description of them and possible directions of research.

7.1 More Work With PDGP

- A major issue we encountered when working with PDGP is that the gaps frequently occur very close to the beginning or end of $\mathbf{v}'$.
- When working with large data sets, the extreme location of the gaps is an issue because a very small portion of the data set is split off each time.
- Initially, we tried ignoring a certain percentage of the differences from each end. Although it was an improvement over looking at all of $d\mathbf{v}'$, the results were not entirely satisfactory.
- We plan on working on a weighting scheme that gives more weight to gaps closer to the middle of $\mathbf{v}'$ and diminishes the weight of gaps near the end of $\mathbf{v}'$.

7.2 Centering vs. Uncentering

- For computation time, uncentered data sets are preferred because it preserves the sparse structure of the data set. Centering the data destroys all sparsity in the term-by-document matrix.
- However, with procedures such as the Lanczos method that can quickly compute the first few columns of $\mathbf{V}$, centering the data does not increase the time by nearly as much as first anticipated.
- We have yet to determine whether or not the decreased computing time involved with uncentered data is enough to make up for the decreased quality of the clustering.

7.3 Multiple Singular Vectors

- We would still like to improve our use of columns of $\mathbf{V}$ in addition to $\mathbf{v}_1$.
- We believe other singular vectors contain pertinent information but have been unable to get algorithms using more than one singular vector to perform better than those that use just $\mathbf{v}_1$.

7.4 Cluster Aggregation

- We have tried to combine the results from many of the algorithms mentioned above into an adjacency matrix that shows how many times a pair of documents has been clustered together.
- We are still working on clustering this adjacency matrix, though it has produced clusters for some data sets that are better than any of the individual clustering methods.

8 Acknowledgments

We would like to thank Daniel Boley for sending us the data from his paper, Dimitrios Zeimpekis for sharing his code, David Gleich for creating Vismatrix, Amy Langville for providing us with advice, and Russell Albright, Jim Cox, and David Duling for providing us with the grocery store data and inviting us to their campus to discuss the algorithms.

This project was supported in part by NSF grant DMS-0552571 and NSA grant H98230-06-1-0098.

References

[1] Carl D. Meyer Amy N. Langville and Russell Albright. Initializations for the nonnegative matrix factorization. 2006.

[2] D.L. Boley. Principal direction divisive partitioning. *Data Mining and Knowledge Discovery*, 2(4), 1998.

[3] Patrik O. Hoyer. Nonnegative matrix factorization with sparseness constraints. *Journal of Machine Learning Research*, 5(1457-1469), 2004.

[4] Daniel D. Lee and H. Sebastian Seung. Learning the parts of objects by nonnegative matrix factorization. *Nature*, 401(6755), 1999.

[5] Dimitrios Zeimpekis and Efstratios Gallopoulos. *Text to Matrix Generator User's Guide*. 2007.