San José State University

Department of Mathematics and Statistics

Large-Scale Spectral Clustering Methods for Image and Text Data

Team Leaders:
Jeffrey Lee, Scott Li
Team Members:
Jiye Ding, Maham Niaz, Khiem Pham, Xin Xu, Zhengxia Yi, Xin Zhang

Faculty Supervisor:
Dr. Guangliang Chen

August 27, 2018

Abstract

Verizon is a telecommunications company that receives a large amount of data from cell phone users. Clustering is one way to draw insights from this data in order to make better business decisions. While spectral clustering has many advantages over other clustering algorithms, it involves computationally expensive steps which limit its application to large datasets. In this report, we explore three methods - Scalable Spectral Clustering using Cosine Similarity, Landmark-Based Spectral Clustering, and Landmark-Based Bipartite Graph Spectral Clustering to improve the scalability of current spectral clustering algorithms while maintaining clustering performance.

Acknowledgements

We would like to thank Prof. Guangliang Chen for his guidance and supervision with this project and Prof. Slobodan Simic for helping to organize this project We also express deep gratitude to Verizon for their generous sponsorship.

Contents

1 Introduction

This project is sponsored by Verizon. Verizon receives a large stream of data from their cell-phone users, such as browsing history and demographic data. Drawing insights from this data to provide better services or making better business decisions is critical for company growth or survival. Clustering is one way to analyze this data.

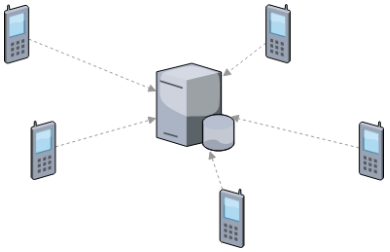


Figure 1: Cell phone user data

Clustering has many business applications including customer segmentation, market segmentation, and grouping web pages.

1.1 CAMCOS Project

This Spring 2018 CAMCOS project is a continuation of the Spring 2017 project, which was also sponsored by Verizon. The previous group performed a proof of concept study to cluster a famous document dataset. They worked on a general framework and considered data visualization, different data preprocessing methods, different similarity measures, and tested different clustering algorithms. They concluded that spectral clustering techniques could be applied effectively to the documents dataset.

This group is focused on techniques to improve the scalability of spectral clustering algorithms, and understanding the clustering results. There are three teams which focused on three different techniques:

1. Use cosine similarity and clever matrix manipulations to bypass some computationally expensive steps.
2. Use landmarks to find a sparse representation of the data.
3. Use landmarks and the given data to build bipartite graph models.

1.2 Report Organization

The report will start by giving background material on general clustering as well as spectral clustering in particular. We will also introduce any datasets we used

Figure 2: Group clustering illustration

for testing. Because the CAMCOS group was divided into three teams, the report will present the methodology, results, and conclusions of each team in order. Then, cluster interpretation will be examined. Finally, overall conclusions and future work can be presented.

2 Background

2.1 Notation

- Vectors are denoted with bold, lower case letters: **b** denotes vector b.
- Matrices are denoted in bold, upper case letters: **B** denotes matrix B.
- Entries in a matrix are denoted as subscripts under matrix notation: $\mathbf{B_{i,j}}$ denotes the matrix entry corresponding to row i and column j.
- The identity matrix will be denoted in bold 'I'. The size of the identity matrix will vary according to context, but we assume that these matrices are of sizes that conform with the given situation: **I** denotes an identity matrix.
- A constant vector of one will be denoted in bold '1'. The length of this vector will vary according to context but we assume that these vectors are of lengths that conform with the given situation: **1** denotes a constant vector of 1.

2.2 Clustering and Clustering Components

Clustering is an unsupervised machine learning task. The goal of clustering is to group data in a way that data within a group are more similar to each other than they are to data in different groups. Some applications for clustering include customer and market segmentation and identifying groups of web pages.
The components in a clustering process is as follows:

- **Data**. We will consider data in the form of a matrix, where each row represents an observation x_i.
- The **number of clusters to be formed** that is specified by the user. It is assumed that the user already has an optimal number of clusters in mind. Methods for selecting a number of clusters given that it is unknown is outside the scope of this project and will require additional research.
- A **similarity measure** to measure how similar different observations are to each other. Similarity measures between two observations x and y are denoted $S(x, y)$
- A **specified clustering algorithm** to perform the clustering. One example of an algorithm could be spectral clustering with cosine similarity or landmark clustering.
- A **criterion to evaluate the clusters**. For our study, we will evaluate our clustering algorithms by accuracy and runtime. For text data, since we have the vocabulary for our data, we will also consider the cluster interpretations of our output by looking at the top significant words in each cluster.

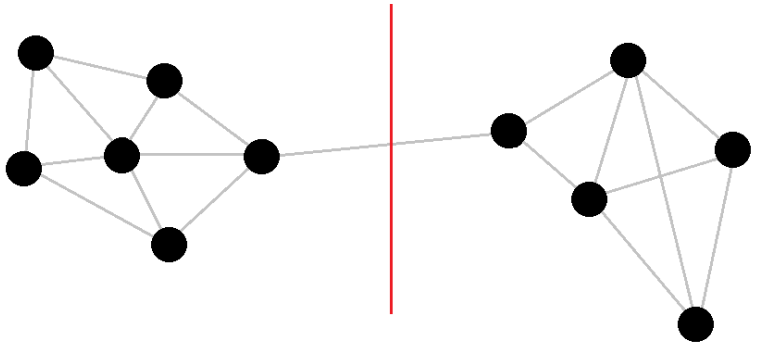
2.3 Spectral Clustering

Given data, spectral clustering uses spectral decomposition with the similarity matrix. Some spectral clustering methods use a degree matrix, **D**, which is a diagonal matrix where each diagonal entry is the sum of all weights for a corresponding observation in the data. Then, k-means clustering is used on the resulting top k eigenvectors from the spectral decomposition. For our project, we consider three different methods of spectral clustering. Simple manipulations of the weight matrix and the degree matrix will allow us to perform spectral clustering with one of the following three algorithms: The Normalized Cut algorithm (NCut) [14], the Diffusion Maps algorithm (DM) [4], and the Ng, Jordan, and Weiss algorithm (NJW) [10].

2.3.1 Normalized Cut (NCut)

We discuss the Normalized Cut method first, because it is the most intuitive. Under Normalized Cut, spectral clustering can be considered to be a graph cut problem where each observation is a vertex and each pair of vertices are connected by an edge with the corresponding similarity weights.

Figure 3: Graph Representation of Data



For Normalized Cut, the clustering criterion will "cut" the edges of the graph in a way that minimizes the separation of the edges while maximizing the remaining weights within each group after the cut.

This criterion can be summarized as follows:

$$min_{A,B}\, Ncut(A,B) = \frac{Cut(A,B)}{Vol(A)} + \frac{Cut(A,B)}{Vol(B)}$$

Where $Cut(A,B)$ is the sum of all of weights of all of the edges that have been removed after partitioning the vertices into two disjoint sets A and B, and $Vol(A)$ is the sum of all of the existing pairwise weights within set A.

It can be shown that this criterion is approximated by solving an eigenvalue problem on the transition matrix $\mathbf{P} = \mathbf{D^{-1}W}$. To cluster, we would use the second to kth largest eigenvectors for k-means clustering.

2.3.2 Diffusion Maps (DM)

The Diffusion Maps algorithm is very similar to the NCut algorithm. The difference is that here, the eigenvalue decomposition is done on $\mathbf{P^t}$ for some positive integer t instead of just $\mathbf{P}$. In this case, we have a transition matrix that represents t time steps.

2.3.3 Ng, Jordan, and Weiss (NJW)

Much of our project is focused on the use of the Ng, Jordan, and Weiss spectral clustering. Here, instead of the transition matrix $\mathbf{P}$, the NJW method uses a symmetric normalization of the weight matrix, $\tilde{\mathbf{W}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{W}\mathbf{D}^{-\frac{1}{2}}$.

2.4 Implementing The Algorithms

We specify the NJW algorithm first and then the adjustments required to perform the other two algorithms.

- **NJW algorithm:**
  Input: Data $x_1, \ldots, x_n$, specified number k, α fraction cutoff for outliers
  1. $\mathbf{W} = (\mathbf{W_{i,j}}) \in R^{n \times n}$, where $\mathbf{W_{i,j}} = S(x_i, x_j)$
  2. $\mathbf{D} = diag(\mathbf{W} \cdot \mathbf{1})$
  3. Symmetric normalization: $\tilde{\mathbf{W}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{W}\mathbf{D}^{-\frac{1}{2}}$
  4. Compute the top k eigenvectors of $\tilde{\mathbf{W}}$. Let these row normalized eigenvectors be $\tilde{\mathbf{U}}$.
  5. Run k-means on $\tilde{\mathbf{U}}$ to cluster.
  Output: Cluster labels
- **NCut algorithm:** Adjustment to step 5: Instead of running k-means on $\tilde{\mathbf{U}}$, run k-means on $\mathbf{U}$, where $\mathbf{U} = \mathbf{D}^{-\frac{1}{2}}\tilde{\mathbf{U}}$, to cluster.
- **DM algorithm:** Adjustment to step 5: Instead of running k-means on $\tilde{\mathbf{U}}$, run k-means on $\mathbf{U^{(t)}} = \mathbf{U}\mathbf{\Lambda^t}$, where $\mathbf{\Lambda}$ is a diagonal matrix of the top k eigenvalues of $\tilde{\mathbf{W}}$, to cluster.

2.5 Strengths and Challenges Of Spectral Clustering

Spectral clustering is an improvement on regular k-means clustering, because it can handle arbitrarily shaped clusters. It is also equivalent to some graph cut problems,

which is desirable because of the availability of the use of graph theory and other applications. From earlier, we see that the Spectral Clustering algorithms can be implemented with 5 or 6 steps.

However, Spectral Clustering is expensive for large datasets in terms of storage, which is $O(n^2)$, and computation, which is $O(n^3)$. This is an important motivation for our project's focus on scalability of the clustering algorithms.

2.6 Datasets

The 6 datasets that we will test our methods on are classified into document data and image data. They are as follows:

Table 1: Datasets

Type	Dataset	Instances	Features	Classes
Text	20 Newsgroups	18,768	55,570	20
	Reuters	8,067	18,933	30
	TDT2	9,394	36,771	30
Image	USPS	9,298	256	10
	Pendigits	10,992	16	10
	MNIST	70,000	784	10

Note that the text datasets are sparse, and the image datasets are low dimension. The 6 datasets that we used are sourced from the following websites:

- 20 Newsgroups: http://qwone.com/∼jason/20Newsgroups/
- Reuters: http://www.cad.zju.edu.cn/home/dengcai/Data/TextData.html
- TDT2: http://www.cad.zju.edu.cn/home/dengcai/Data/TextData.html
- USPS: https://www.csie.ntu.edu.tw/∼cjlin/libsvmtools/datasets/
- Pendigits: https://www.csie.ntu.edu.tw/∼cjlin/libsvmtools/datasets/
- MNIST: http://yann.lecun.com/exdb/mnist/

3 Scalable Spectral Clustering using Cosine Similarity

Although spectral clustering is expensive, we will show that we can omit the similarity matrix calculation, which is one of the more costly parts of the algorithm, and still be able to cluster with the NJW algorithm (as well as NCut and DM) under the use of cosine similarity.

3.1 Setting for Scalable Spectral Clustering

Many clustering problems involve document data or image data. For these types of data, cosine similarity is appropriate to use.

3.1.1 Cosine Similarity

The cosine similarity between objects x and y takes the form:

$$S(x, y) = cos\theta = \frac{x \cdot y}{||x|| \cdot ||y||}$$

Let $\mathbf{A}$ be a data matrix where each row has been normalized into unit vectors. Then, under cosine similarity, the form of the similarity matrix becomes:

$$\mathbf{W} = \mathbf{A}\mathbf{A}^\mathbf{T} - \mathbf{I}$$

3.1.2 Assumptions for Scalable Spectral Clustering

For the scalable method, we will assume that:

- Any given data will either be sparse or low dimensional.
- Cosine similarity is used as the similarity measure.

3.2 Derivation of Scalable Spectral Clustering

Recall the NJW algorithm from Section 2.4. If we plug in $\mathbf{W} = \mathbf{A}\mathbf{A}^\mathbf{T} - \mathbf{I}$, then two important things happen:

(1) $\mathbf{D} = \mathbf{diag}(\mathbf{W} \cdot \mathbf{1})$
$= \mathbf{diag}((\mathbf{A}\mathbf{A}^\mathbf{T} - \mathbf{I}) \cdot \mathbf{1})$
$= \mathbf{diag}(\mathbf{A}(\mathbf{A}^\mathbf{T}\mathbf{1}) - \mathbf{1})$
which can be calculated without $\mathbf{W}$

(2) $\tilde{\mathbf{W}} = \mathbf{D}^{-\frac{1}{2}}(\mathbf{A}\mathbf{A}^\mathbf{T} - \mathbf{I})\mathbf{D}^{-\frac{1}{2}}$
$= \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{A}^\mathbf{T}\mathbf{D}^{-\frac{1}{2}} - \mathbf{D}^{-\mathbf{1}}$
$= \tilde{\mathbf{A}}\tilde{\mathbf{A}}^\mathbf{T} - \mathbf{D}^{-\mathbf{1}}$
where $\tilde{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}$

From (1), we see that because the similarity matrix has a specific form, we can manipulate the matrix multiplications in an efficient way. A result of (1) is that we can omit calculating $\mathbf{W}$ and directly calculate $\mathbf{D}$, the degree matrix, with just $\mathbf{A}$.

From (2), since $\tilde{\mathbf{W}} = \tilde{\mathbf{A}}\tilde{\mathbf{A}}^\mathbf{T} - \mathbf{D}^{-\mathbf{1}}$, we will consider some observations:

- Consider $\tilde{\mathbf{A}}\tilde{\mathbf{A}}^\mathbf{T}$ without the subtraction term. Also consider the singular value decomposition of $\tilde{\mathbf{A}}$, where $\tilde{\mathbf{A}} = \tilde{\mathbf{U}}\mathbf{\Sigma}\mathbf{V}^\mathbf{T}$. Then $\tilde{\mathbf{A}}\tilde{\mathbf{A}}^\mathbf{T} = (\tilde{\mathbf{U}}\mathbf{\Sigma}\mathbf{V}^\mathbf{T})(\mathbf{V}\mathbf{\Sigma}^\mathbf{T}\tilde{\mathbf{U}}^\mathbf{T})$, and it can be shown that under these considerations, $\tilde{\mathbf{U}}$ is analogous to the top k eigenvectors of $\tilde{\mathbf{W}}$ from step 4 of the NJW algorithm in Section 2.4.
- Consider $\tilde{\mathbf{W}} = \tilde{\mathbf{A}}\tilde{\mathbf{A}}^\mathbf{T} - \mathbf{D}^{-\mathbf{1}}$. If $\mathbf{D}^{-\mathbf{1}}$ has constant diagonals, then the eigenvectors of $\tilde{\mathbf{W}}$ will be the same as those of $\tilde{\mathbf{A}}\tilde{\mathbf{A}}^\mathbf{T}$ (but their eigenvalues are

different). This implies that one can use the SVD of $\tilde{\mathbf{A}}$ to compute the eigenvectors of $\tilde{\mathbf{W}}$ when $\mathbf{D^{-1}}$ has a constant diagonal.

Although we mention constant diagonal entries for $\mathbf{D^{-1}}$, this condition is rarely the case in practice. However, this condition can be made nearly true if we remove an appropriate fraction of the observations that have the lowest degrees from the data when we perform the clustering, which would result in a $\mathbf{D^{-1}}$ with small and nearly constant diagonals. In this sense, the data that are removed can be considered as outliers, due to their low degrees.

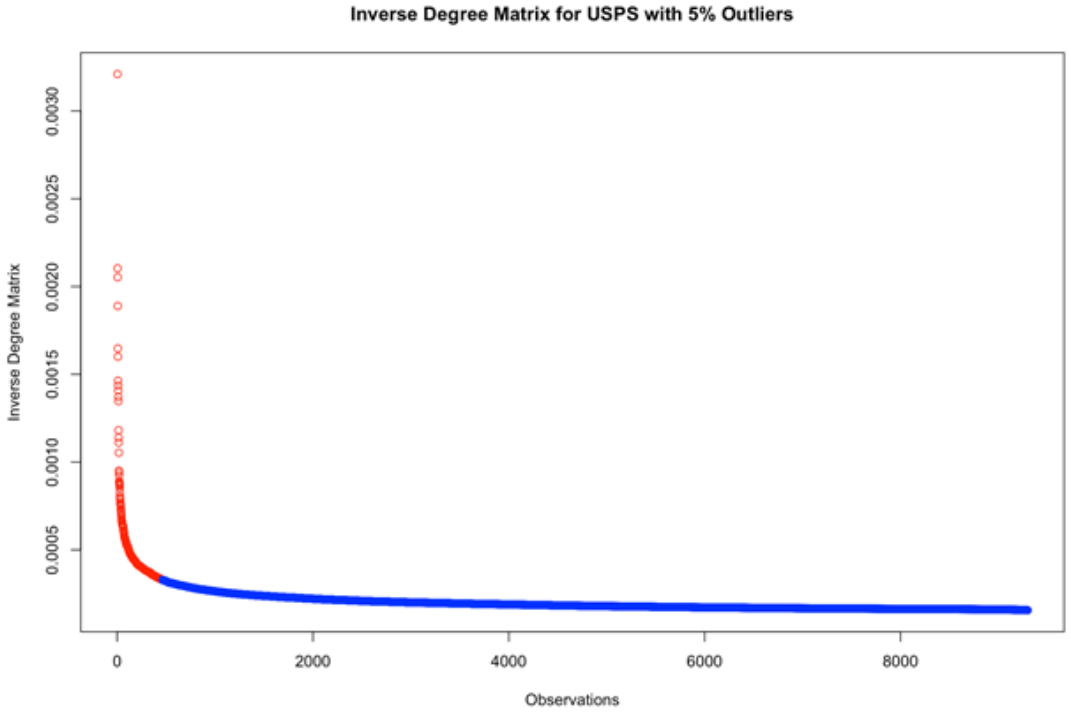


Figure 4: Inverse degrees of the observations ordered from large to small. Red degrees indicate that the corresponding observations are considered to be outliers.

We note the following while keeping in mind that low degree means high inverse degree:

- There tends to be a small fraction of observations with low degree denoted in red. If there exist data where a large portion of the observations have low degree, then clustering would be bad for analyzing the data in the first place.
- After removing the red observations, the remaining blue observations are numerically similar. For practical purposes, we will consider the blue observations to have approximately equal degrees.

With the above graph, equations (1) and (2), and their results in mind, it is possible to implement a scalable spectral NJW clustering algorithm that omits the **W** calculation and only requires the data matrix **A** to approximate clustering results of the NJW algorithm from Section 2.4 (from this point on, that algorithm will be referred to as the Plain NJW algorithm).

3.3 Scalable Spectral Clustering Algorithm Overview

Input: Data **A**, Specified number k, clustering method (NJW, Ncut or DM) and α fraction cutoff for outliers

1. L2 normalize **A**. Compute degree matrix **D** (refer to equation (1)), remove outliers from **D** and **A**
2. Compute $\tilde{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}$
3. Compute the $\tilde{\mathbf{U}}$, the top k left singular vectors of $\tilde{\mathbf{A}}$
4. Convert $\tilde{\mathbf{U}}$ according to clustering method and run k-means

Output: Cluster labels, including a label for outliers

3.3.1 Outlier Classification

In order to check accuracy of a clustering algorithm, we can compare the true labels and cluster labels while excluding the algorithm-marked outliers. If we want to include all of the observations, then we classify the outliers by doing the following:

- Take the clustered observations and average each cluster. These averages will be a cluster center.
- Classify each of the outliers with the Nearest Centroid Classifier, which is done by computing each of the cluster centers and assigning each of the outliers to its nearest cluster center.

3.4 Experiments

To test the scalable algorithm, we use it and compare it with the plain algorithm for all 6 datasets mentioned in Section 2.6. The criteria used for evaluation of cluster algorithms are accuracy and runtime. Each specified clustering run is conducted for 5 seeds each, and the results are averaged. All codes in these experiments are implemented in R and conducted on a server called Golub at San Jose State University, which is equipped with the following:

- Two Xeon E5620 Intel Processors with four cores and support for eight threads per physical processor running at 2.4GHz
- The server has 48GB of memory and two 2TB mirror providing 2TB of storage

3.4.1 Speed and Accuracy

The following results in this section are from experiments done with $\alpha = 0.01$. Note that under the plain algorithm, MNIST clustering runs were unsuccessful because of memory issues.

Table 2: Here are comparisons between the Scalable NJW algorithm and the Plain NJW algorithm for each of the datasets. From the runtime table, we can see that the Scalable NJW is much faster than the Plain NJW method.

Runtime (Seconds)		
Dataset	**Scalable**	**Plain**
20 Newsgroup	57.7	154.9
Reuters	5.9	51.1
TDT2	25.3	53.9
USPS	1.1	52.9
Pendigits	3.4	102.0
MNIST	36.2	Out of Memory

Table 3: Accuracies in this section are computed after classifying the outliers. From the accuracy table, we can see that both methods are similar in accuracy. The Plain NJW method is slightly more accurate.

Accuracy (%)		
Dataset	**Scalable**	**Plain**
20 Newsgroup	64.40	64.95
Reuters	24.60	25.23
TDT2	51.20	51.80
USPS	67.53	67.47
Pendigits	73.56	73.56
MNIST	52.60	Out of Memory

3.4.2 Robustness to Outliers

For each dataset, we also experimented on the α from 0.01 to 0.10. This is to get an idea of how sensitive the Scalable NJW algorithm is to the fraction of data that are removed, since there is no rule of thumb to decide how much of the data should be considered outliers. Note that a larger α means that more data will be considered to be outliers (see Section 3.3).

Figure 5: Accuracy as a function of α for each dataset. For this figure, the accuracies of the Scalable NJW algorithm are calculated after classifying each of the outliers (see Section 3.3.1).

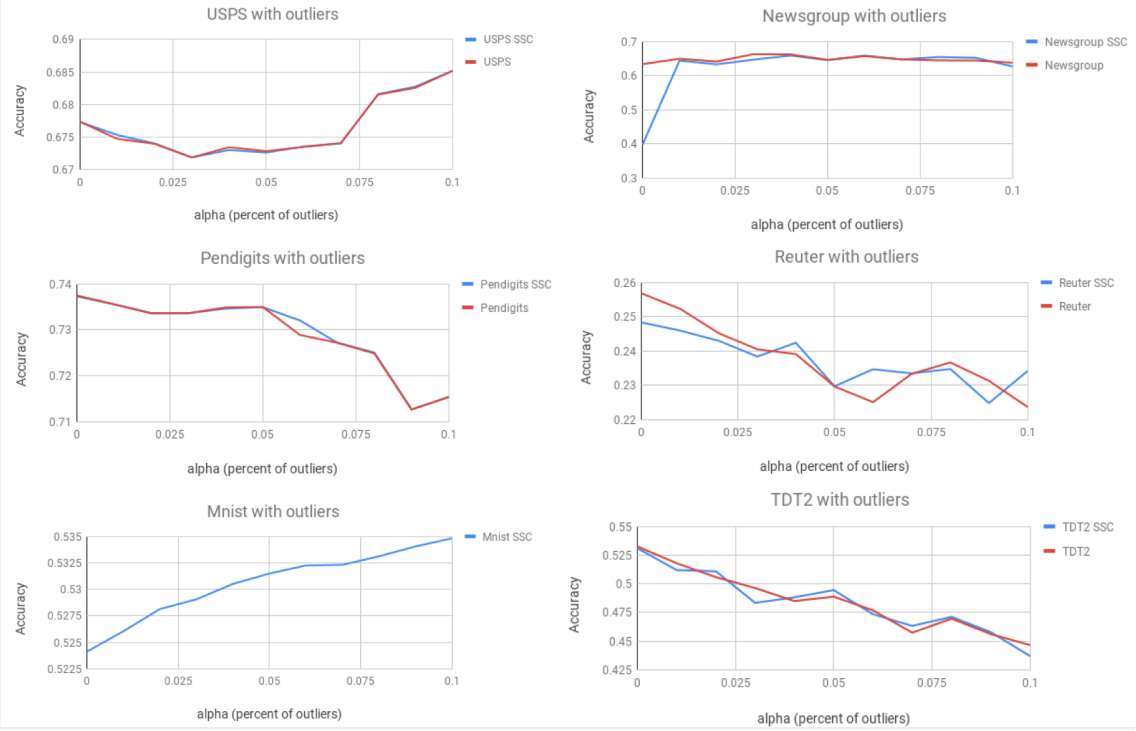
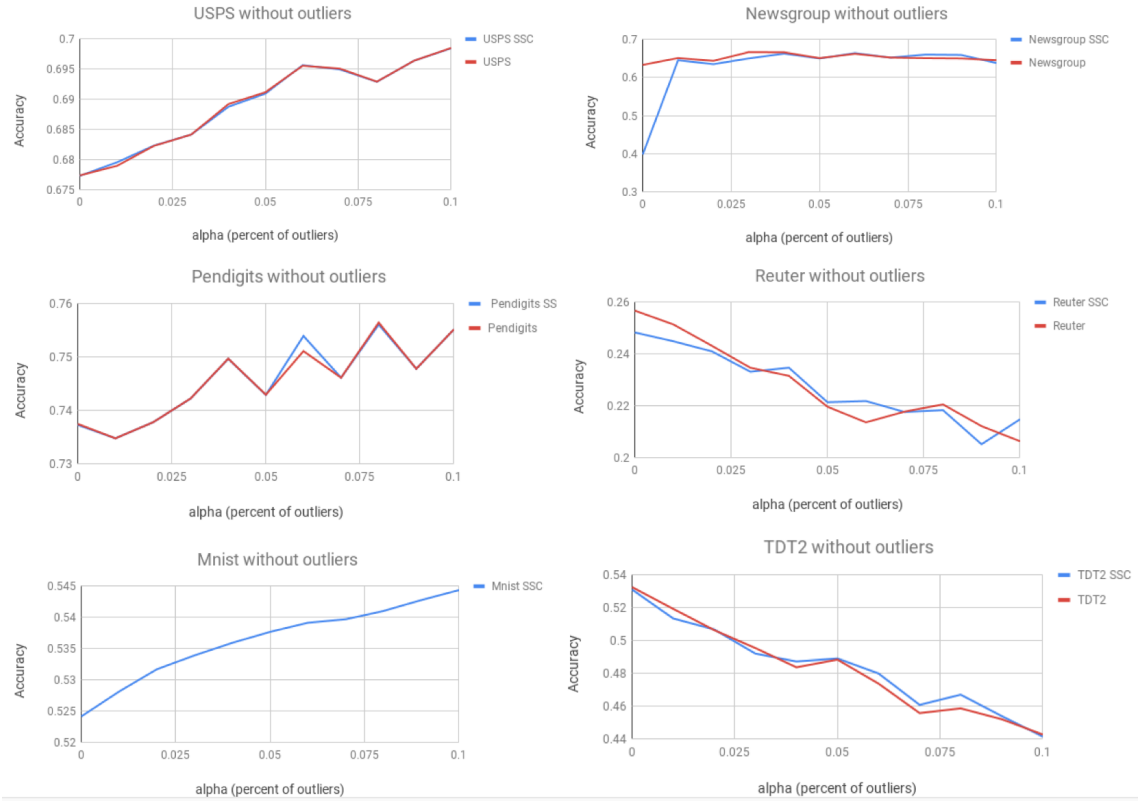


Figure 6: Accuracy as a function of α for each dataset. For this figure, the accuracies of the Scalable NJW algorithm are calculated without the outliers.



In the both figures above, both the Scalable NJW and the Plain NJW algorithms perform similarly in accuracy for different α. For the Newsgroup dataset, we see that accuracy of the Scalable NJW is low at $\alpha = 0$, but it improves and becomes comparable with the Plain NJW after increasing α to 0.125.

Figure 7: Run time as a function of α for each dataset. For this figure, the run times of the Scalable NJW algorithm are calculated after classifying each of the outliers (see Section 3.3.1).

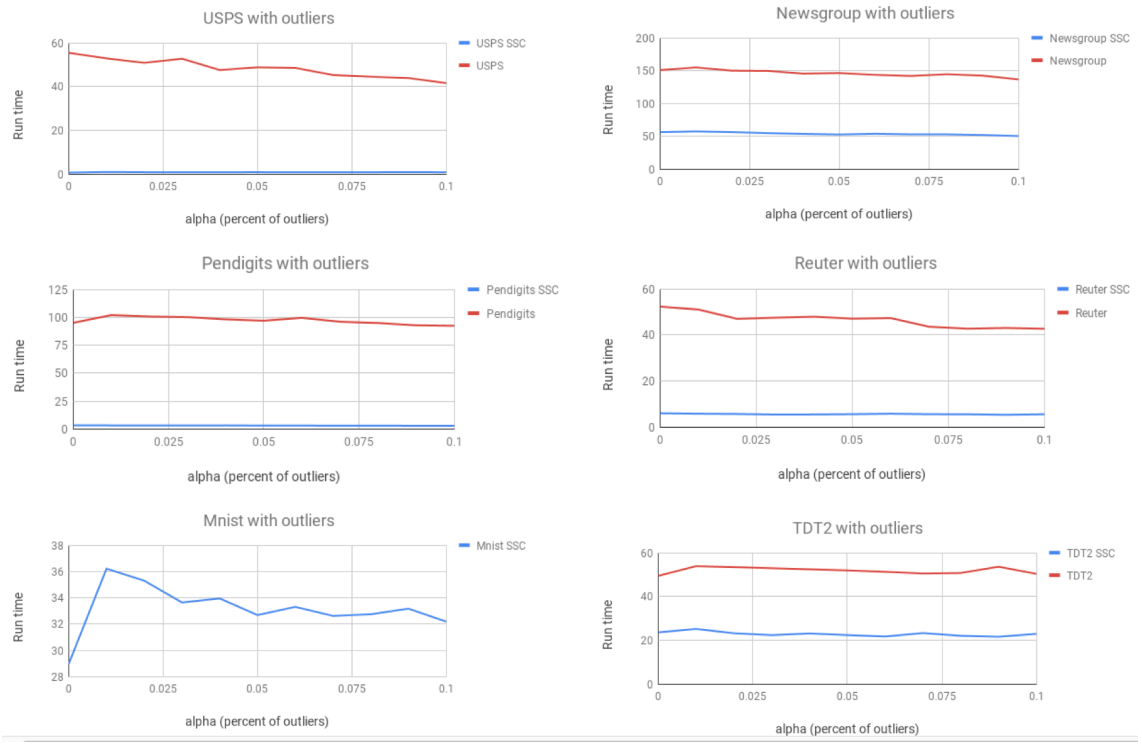
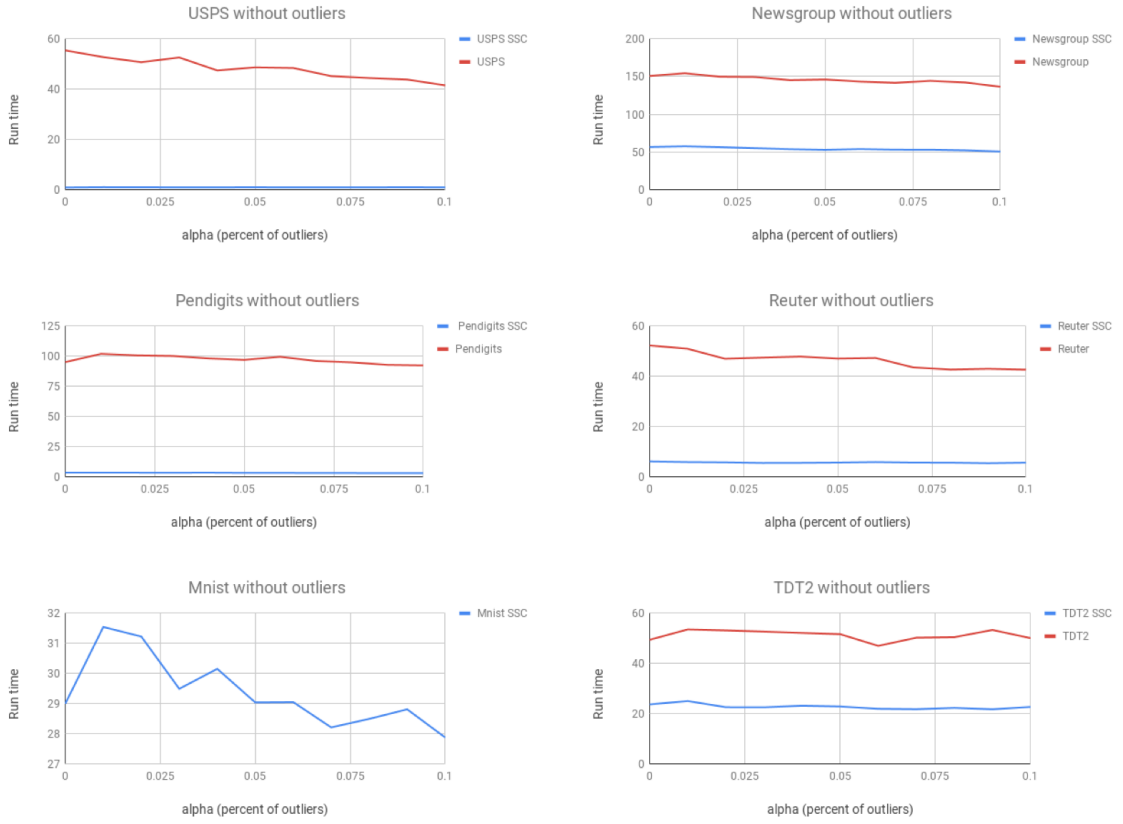


Figure 8: Run time as a function of α for each dataset. For this figure, the run times of the Scalable NJW algorithm are calculated without the outliers.



From the run time plots, we see that with the exception of MNIST experiments, in most cases, run time does not increase much, and the Scalable NJW remains faster in general.

By applying the cosine similarity and adequately removing the low degree observations, the scalable spectral clustering method saves computation time while having accuracies that are comparable to the plain spectral clustering method.

Because we see that accuracy can have a positive (for USPS, Pendigits, and MNIST) or negative (for 20 Newsgroup, Reuters, TDT2) association with α, we only recommend an α that is large enough only to capture all of the outliers for the purpose of implementing the Scalable NJW algorithm. For our study, we found 1% to be sufficient, but further studies on α are recommended.

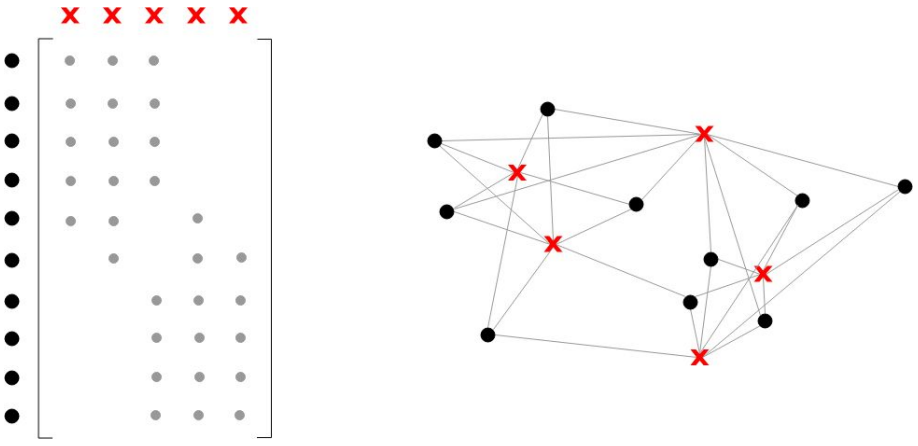
3.5 Other Considerations

In addition to implementing and testing the Scalable NJW clustering algorithm with cosine similarity, we explored using the Scalable NJW algorithm with different similarity measures, and we also extended the NJW algorithm to be able to perform Normalized Cut and Diffusion Maps.

3.5.1 Different Similarity Measures

Scalable NJW Clustering assumed the use of sparse data matrices and cosine similarity. One idea that we explored involves using data with n observations to create a similarity matrix of dimension n by n with any similarity measure, like Gaussian similarity. Then, a random landmark selection method is used to reduce the dimension of the matrix. Landmark selection is a way to select a subset of the observations to represent the entire data when computing the similarities so that each of the n observations will only be compared with each of the p landmarks. Under this setting, the resulting similarity matrix becomes size n by p, which has smaller dimensions. At this point, each row will keep only the top r most similar landmarks, and the rest of the similarities in that row are set 0. The result is a sparse matrix of weights with low dimension. This matrix will become our "data" matrix $\mathbf{A}$ that we use for the Scalable spectral NJW algorithm.

Figure 9: Illustration of Landmark Selection. Left: the form of the affinity matrix after taking the nearest landmarks. Landmarks are x's and observations are dots. Right: a graph representation of this matrix.



Implementation of Scalable NJW with Gaussian similarity.

- Randomly select k observations to be landmarks

- Calculate the Gaussian similarity weight matrix to form an n by k matrix
- Keep the top r weights for each row and turn all other weights to 0
- Use the resulting sparse matrix as an input for the Scalable NJW, which is specified in Section 3.3.

Test Results (accuracy and speed). For this extension of the scalable NJW, we tested only on the image datasets with $\alpha = 0.01$.

Table 4: Runtime and Accuracy of Scalable NJW with Gaussian Similarity. Note that results for MNIST data is an average of only 3 runs under different seeds due to the length of time per run.

Dataset	**Runtime (Seconds)**	**Accuracy (%)**
USPS	88.7	53.69
Pendigits	117.2	72.33
MNIST*	16138.1	50.0

For future work, we do note that the similarity matrix computation can be sped up in terms of the Euclidean distance:

$$||\mathbf{x} - \mathbf{y}||^2 = ||\mathbf{x}||^2 + ||\mathbf{y}||^2 - 2\mathbf{x} \cdot \mathbf{y}$$

This would speed up computation under Gaussian similarity, which takes the form:

$$S(x, y) = e^{-\frac{||x-y||^2}{2\beta\sigma^2}}$$

with variance parameters β and σ.

3.5.2 Other Clustering Methods

Implementation - Turning NJW into NCut and DM. As mentioned in Section 2.4, the Scalable NJW algorithm can be easily modified into a Normalized Cut algorithm by turning $\tilde{\mathbf{U}}$ into $\mathbf{U}$ or a Diffusion Maps algorithm of t steps by turning $\tilde{\mathbf{U}}$ into $\mathbf{U}^{(\mathbf{t})}$. In this setting, we note that $t = -1$ represents NJW, $t = 0$ represents Normalized Cut, and Diffusion Maps of step t are represented by a positive t.

Results (accuracy and speed).

Table 5: Accuracy of Scalable Normalized Cut and Diffusion Maps.

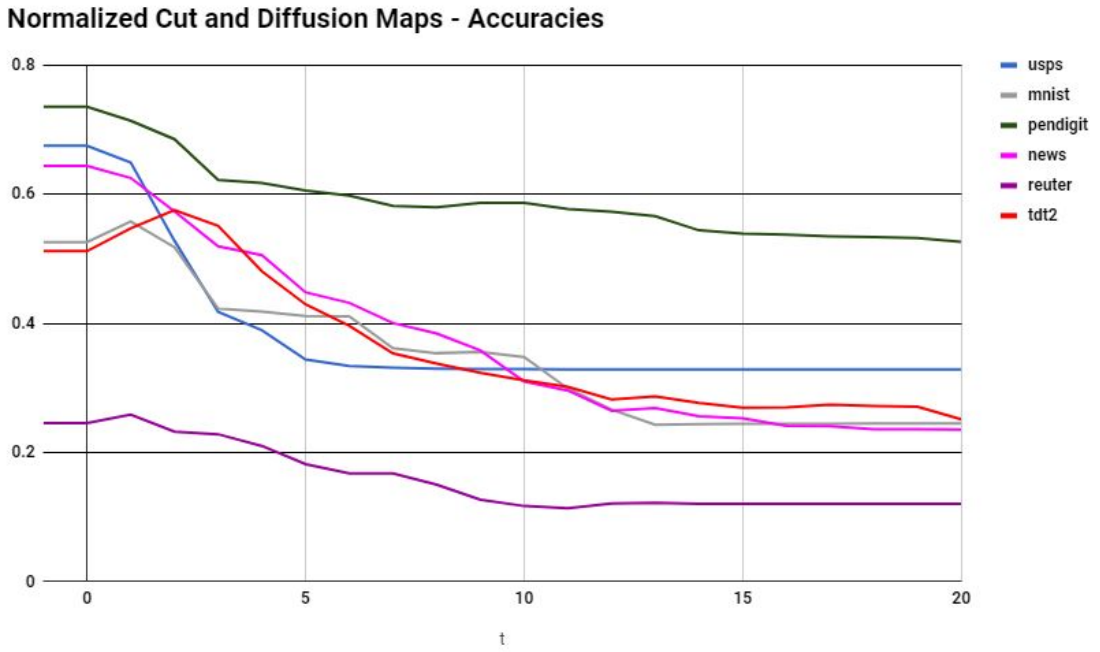
t steps **Dataset**	**0**	**1**	**2**	**3**	**4**
20 Newsgroup	64.41	62.54	57.36	51.95	50.59
Reuters	24.6	25.9	23.25	22.84	21.03
TDT2	51.2	54.76	57.57	55.12	48.08
USPS	67.53	64.93	52.83	41.8	38.95
Pendigits	73.56	71.41	68.55	62.25	61.76
MNIST	52.61	55.79	51.8	42.31	41.85

Table 6: Runtime of Scalable Normalized Cut and Diffusion Maps.

t steps **Dataset**	**0**	**1**	**2**	**3**	**4**
20 Newsgroup	49.5	49.8	49.8	50.0	49.7
Reuters	6.4	6.2	6.2	6.3	6.2
TDT2	22.4	22.1	22.1	21.9	22.0
USPS	1.2	1.3	1.3	1.4	1.4
Pendigits	3.4	3.1	3.2	3.2	3.1
MNIST	38.0	37.9	38.0	36.4	36.6

The runtimes are very similar, but we note that as t increases, the accuracy tends to decrease.

Figure 10: Plot of accuracies as t increases. Here, $t = -1$ is a representation of Scalable NJW to compare with $t = 0$, which is NCut, and positive t, which is DM.



3.5.3 Additional Remarks

Although the extension of Scalable NJW is slow, there exist faster ways to implement different similarities, especially with Gaussian Similarity. Also, for future studies, it would be of interest to take a closer look at Diffusion Maps and how our matrix manipulations affect the accuracies of the clustering algorithm.

4 Landmark-Based Spectral Clustering

Another approach to improve the efficiency of spectral clustering is Landmark-based methods [2]. These methods involves selecting p ($p << n$) landmarks, or representative points from n observations to find a sparse representation of the data. This representation is smaller, takes up less memory, and is faster to perform computations on.

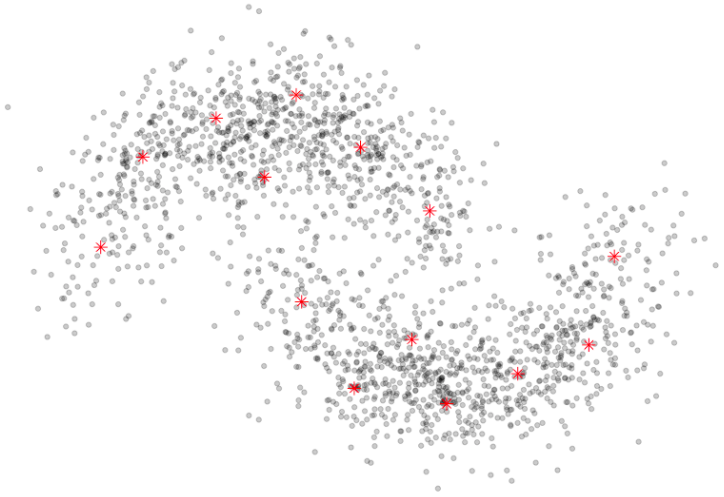


Figure 11: Landmark or representative points highlighted in red.

The sparse affinity matrix between p landmarks y_j and n data points x_i is denoted as $\mathbf{A}$. $\mathbf{A}$ =$(\mathbf{A_{i,j}}) \in R^{n \times p}$, where $\mathbf{A_{i,j}} = S(x_i, y_j)$. The original LSC algorithm clusters the data based on the left singular vectors of the matrix $\mathbf{A}$, after two normalization steps. First, Let $\mathbf{A_1}$ be the L_1 row normalization of $\mathbf{A}$ ($\mathbf{A_1} = \mathbf{D_1}^{-\mathbf{1}}\mathbf{A}$, where $\mathbf{D_1} = diag(\mathbf{A} \cdot \mathbf{1})$). Next, Let $\mathbf{A_2}$ be L_2 column normalization of $\mathbf{A_1}$ ($\mathbf{A_2} = \mathbf{A_1}\mathbf{D_2}^{-\frac{1}{2}}$, where $\mathbf{D_2} = diag(\mathbf{1}^T \cdot \mathbf{A_1})$). The weight matrix of the LSC algorithm is $\mathbf{W} = \mathbf{A_2}\mathbf{A_2^T}$. We show that the degree matrix is the identity matrix below:

Proof. First, by definition,
$\mathbf{W} = \mathbf{A_2}\mathbf{A_2^T}$,
$\mathbf{A_1} = \mathbf{D_1}^{-\mathbf{1}}\mathbf{A}$,
$\mathbf{A_2} = \mathbf{A_1}\mathbf{D_2}^{-\frac{1}{2}}$.
It follows that,
$\mathbf{A_2}\mathbf{A_2^T} = \mathbf{A_1}\mathbf{D_2^{-\frac{1}{2}}}\mathbf{D_2^{-\frac{1}{2}}}\mathbf{A_1^T} = \mathbf{A_1}\mathbf{D_2^{-1}}\mathbf{A_1^T}$,
$\mathbf{D} = diag(\mathbf{W} \cdot \mathbf{1}) = diag(\mathbf{A_1}\mathbf{D_2^{-1}}\mathbf{A_1^T}\mathbf{1}) = diag(\mathbf{A_1}\mathbf{1}) = \mathbf{I}$.
($\mathbf{A_1^T}\mathbf{1}$ is a vector of column sums of $\mathbf{A}$, which means it's a vector of $diag(\mathbf{D_2})$.)

The eigenvectors of $\mathbf{W}$ can be computed by SVD of $\mathbf{A_2}$, which are the left singular vectors of $\mathbf{A_2}$. K-means is performed on the eigenvectors of $\mathbf{W}$ to cluster the data.

4.1 Algorithm Overview

Input: n data points $x_1, x_2, ..x_n$, cluster number k, number of landmarks p, and number of nearest landmarks r

1. Produce p landmark points using k-means or random selection;
2. Construct an affinity matrix $\mathbf{A} \in R^{n \times p}$ between data points and landmark points, with the affinity calculated using Gaussian or cosine similarity;
3. Make $\mathbf{A}$ sparse by preserving only r largest entries per row
4. Normalize $\mathbf{A}$ using L_1 row normalization first to get A_1 and square root L_1 column normalization, to obtain $\mathbf{A_2}$
5. Apply SVD on $\mathbf{A_2}$ to compute the top k left singular vectors and form a matrix $\mathbf{U} = [u_1 \dots u_k]$ and perform k-means on the row vectors of $\mathbf{U}$.

Output: k clusters

4.2 Landmark Selection

Two main methods of selecting landmarks were considered: Random Selection and k-means selection.

Random selection randomly selects p landmarks from the data using Uniform probability distribution. Therefore, each data point has a 1/n probability of becoming a landmark. This method is really fast compared to k-means.

k-means selection uses the k-means algorithm to select p centroids from the data. As seen in the figure, these centroids are much more evenly spread throughout the data and are more representative, thus, boosting accuracy. However, k-means selection can be really slow for large datasets. Therefore, there is a runtime-accuracy trade-off between the two methods.

4.3 Similarity Computation

Gaussian kernel with Euclidean distance is one of the most commonly used similarity measures. This method requires estimation of the variance. When using random selection method, estimation of the variance requires calculating Euclidean distance between n data points and p landmarks. Equation 4.3a is used to calculate Gaussian

Figure 12: Left: random uniform landmark selection. Right: k-means landmark selection.

similarity between x and y.

$$S(x,y) = e^{-\frac{||x-y||^2}{2\beta\sigma^2}} \tag{4.3a}$$

With k-means selection, the average variance between p landmark centroids provided by the k-means algorithm can be used as an estimate for variance.

Cosine Similarity was also considered to compute the nxp affinity matrix. As explained in Section 2, this method works really well for text data. It is computationally efficient and fast. Equation 4.3b is used to calculate cosine similarity between x and y.

$$S(x,y) = cos\theta = \frac{x \cdot y}{||x|| \cdot ||y||} \tag{4.3b}$$

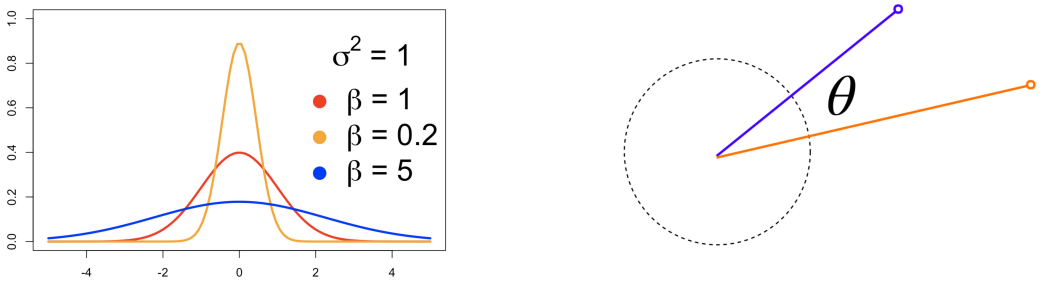


Figure 13: Left: an example of the Gaussian kernel with varying the variance tuning parameter. Right: an example of cosine similarity.

4.4 Nearest Landmarks

After computing the similarity matrix, the matrix needs to be made sparse. In order to do that, r largest similarities are kept for each row, and the rest are set to zero. The sparsity speeds up computations and makes clustering more robust to noise since only nearest landmarks are considered. At this point, the affinity matrix is n rows (corresponding to the observations) by p columns (corresponding to the landmarks), with r nonzero entries in each row. Figure 14 illustrates the affinity matrix for a small dataset with $r = 3$.

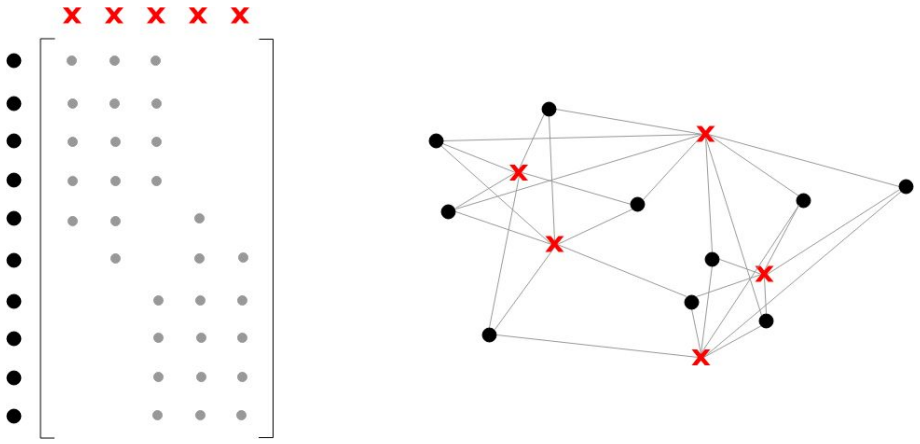


Figure 14: Left: the form of the affinity matrix after taking the nearest landmarks. Landmarks are x's and observations are dots. Right: a graph representation of this matrix.

4.5 Outlier Removal

From the affinity matrix, we can define two kinds of outliers: data outliers and landmark outliers. Data outliers have low row sums, meaning they are dissimilar from most landmark points. Landmark outliers have low column sums meaning they are dissimilar from most data points.

Landmark outliers can simply be removed. Data outliers can either be given the zero label, or removed from the clustering steps and reclassified later with the k nearest neighbor (kNN) classifier.

4.6 Clustering Steps

After computing the sparse affinity matrix, the final step is to do clustering. The original LSC algorithm clusters data points, while we will propose new methods

that cluster landmarks and then use the kNN algorithm to classify observations into clusters.

4.6.1 Data Clustering

The first step in data clustering is normalization. First the rows of A, the sparse affinity matrix between the data and landmarks, are normalized using $L1$ row normalization. This step ensures that each row sums to 1 and the non-zero entries are weights for the landmarks. After normalizing the rows, the columns are also normalized by square root $L1$ normalization. After the two normalization steps, top k left singular vectors are extracted, resulting in matrix $\mathbf{U} = [u_1 \dots u_k] \in \mathbb{R}^{n \times k}$. Then, k-means is applied to the row vectors of $\mathbf{U}$ to get k clusters

4.6.2 Landmark Clustering

Landmark Clustering essentially has the same steps as data clustering and a few additional steps. The rows are normalized using $L1$ normalization and columns are normalized using square root $L1$ normalization. Instead of selecting top k left singular vectors, top k right singular vectors are used, resulting in matrix $\mathbf{V} = [v_1 \dots v_p]$. The row vectors of this matrix represent a k-dimensional embedding of the p landmarks. The rest of the steps are similar to clustering the given data points into k clusters. After clustering landmarks, kNN classification is used to label the original data. We tested a commonly used majority vote and a simple weighted classification scheme. For the majority vote, the most common label, or class, among the nearest landmarks is chosen, with ties randomly broken. Equation 4.6.2a, for majority vote, states that the predicted class for point y is the majority class for the knn number of nearest points, where f returns the class indicator. For the weighted classification, the votes are weighted by the similarities of each label. Equation 4.6.2b is the weighting scheme in which a class's total weight is the sum of the cosine similarity values of the points in that class.

$$\hat{f}(y) = argmax \sum_{i=1}^{knn} f(x_i) \tag{4.6.2a}$$

$$\hat{f}(y) = argmax \sum_{i=1}^{knn} s(y, x_i) f(x_i) \tag{4.6.2b}$$

4.6.3 Diffusion Maps

The diffusion map idea can also be applied to data clustering in this setting. This it done by treating $\mathbf{W} = \mathbf{A_2 A_2}^{\mathbf{t}}$ as a probability transition matrix. Then the t-step

transition probability matrix is given by:

$$\mathbf{U}^{(t)} = \mathbf{U}\Sigma^t$$

where Σ is the diagonal matrix of singular values and $\mathbf{U} = [u_1 \dots u_k]$. By default the time-step parameter is 0.

4.7 Experimental Setup

The following section presents different experiments to evaluate the different LSC methods and parameter settings. The evaluation metrics considered are overall accuracy and run-time. In order to account for the variability in landmark selection and k-means, all tests are conducted for 20 seeds and the results are averaged. All codes in these experiments are implemented in R. With these LSC methods we tested on the six datasets mentioned before. However, we used the 100-dimensional projection for the 20Newsgroups dataset.

4.7.1 Comparisons

The main comparisons focus on cosine similarity. The NJW algorithm with cosine similarity serves as a baseline. There are four LSC methods to compare:

- **R-D:** Random landmark selection with data clustering
- **R-LM:** Random landmark selection with landmark clustering
- **KM-D:** k-means landmark selection with data clustering
- **KM-LM:** k-means landmark selection with landmark clustering

To compare these methods, the parameters are fixed at $p = 500, r = 6, knn = 1$ and no outlier removal. These parameters are also the default in these experiments unless varied for parameter sensitivity.

4.7.2 Parameter Sensitivity

There are many parameters to choose with these LSC methods. To test their sensitivity, each parameter is varied with the others fixed for one particular method for some of the datasets. The parameters tested are as follows:

- p: the number of landmarks
- r: the number of nearest landmarks
- t: time-step parameter
- β: variance tuning parameter
- α_1: percentage of data outlier removal

- α_2: percentage of landmark outlier removal
- knn: number of nearest neighbors for landmark clustering

4.8 Results

The accuracy and run-times for the compared algorithms are presented in the tables below.

Table 7: Clustering accuracy (%)

Dataset	Random LM Selection		k-means LM Selection		NJW
	Data Clustering	Landmark Clustering	Data Clustering	Landmark Clustering	
20Newsgroups	65.51	58.37	**69.42**	60.69	63.36
Reuters	25.37	27.50	27.38	**31.21**	25.68
TDT2	59.85	64.34	59.45	**65.69**	44.38
USPS	62.12	66.70	67.83	**74.70**	67.74
Pendigits	78.81	78.76	77.94	**81.59**	73.75
MNIST	63.32	59.41	**69.43**	65.10	–

Table 8: Clustering run-time (s)

Dataset	Random LM Selection		k-means LM Selection		NJW
	Data Clustering	Landmark Clustering	Data Clustering	Landmark Clustering	
20Newsgroups	5.95	**3.78**	12.75	11.16	150.96
Reuters	7.38	**6.61**	451.88	444.28	52.31
TDT2	12.12	**11.67**	1912.68	1862.29	49.46
USPS	3.93	**3.56**	11.65	11.76	55.46
Pendigits	2.70	**2.25**	3.76	3.63	95.13
MNIST	31.05	**27.62**	584.06	619.06	–

We see from Table 7 and Table 8 that k-means landmark selection gives a better accuracy than random landmark selection for most datasets but the run-time does not scale well. For the larger datasets, the landmark selection steps take much longer than the entire NJW algorithm, even with 10 maximum iterations of the k-means algorithm. Landmark clustering gives a better accuracy than data clustering for all datasets except for 20Newsgroups and MNIST. Considering random landmark selection, there is a clear speed improvement over NJW with all the datasets.

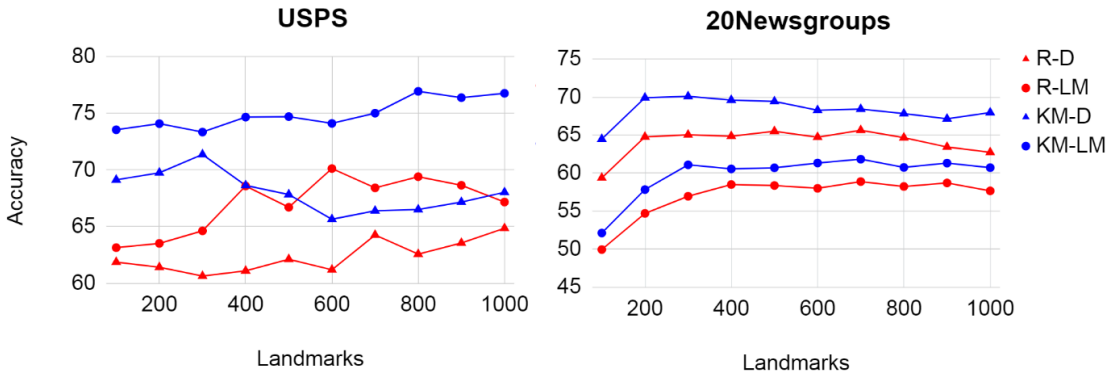


Figure 15: Varying the number of landmarks (%)

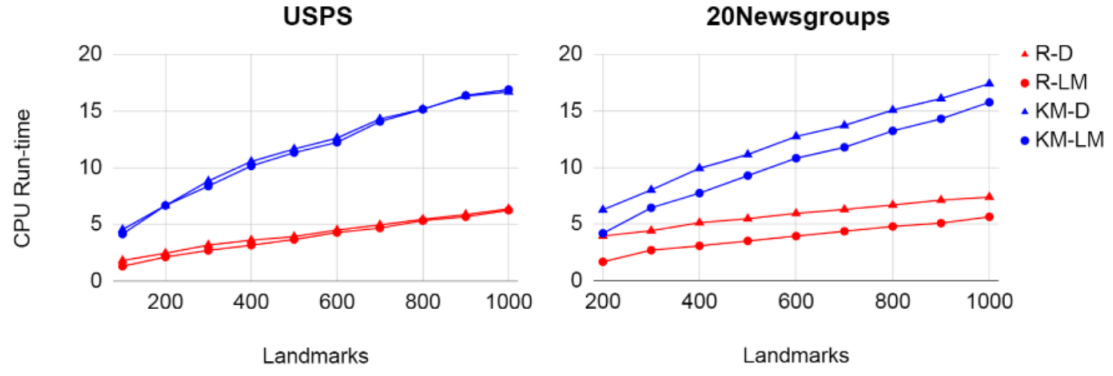


Figure 16: Varying the number of landmarks (s)

Varying the number of landmarks affects the accuracy for USPS but it is fairly robust for 20Newsgroups. Still, the number of landmarks is a parameter that should be tuned for higher accuracy. The increase in run-time is linear with increasing number of landmarks, as expected.

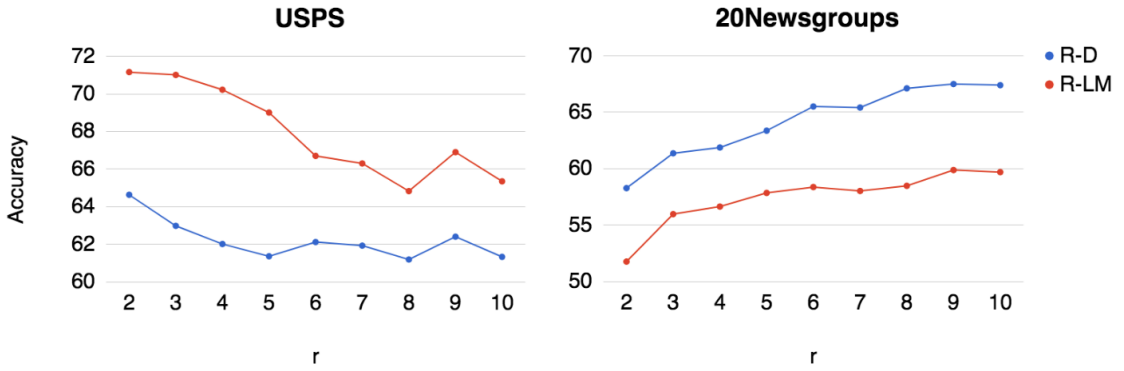


Figure 17: Varying the number of nearest landmarks

Figure 17 shows that varying the number of nearest landmarks can moderately influence the accuracy. The behavior is different between these two datasets. USPS prefers less nearest landmarks whereas 20Newsgroups prefers more.

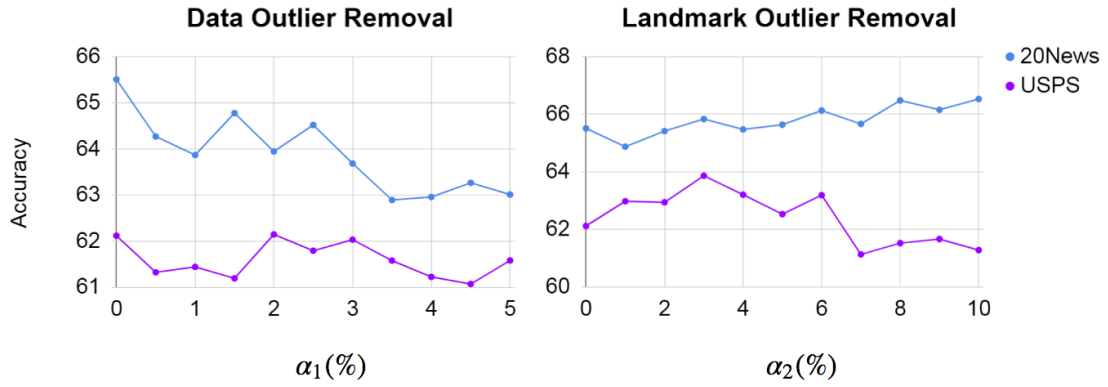


Figure 18: Data and landmark outlier removal

Figure 18 shows that landmark removal was not very effective in improving the overall accuracy. Further work is needed to determine a better way of defining outliers.

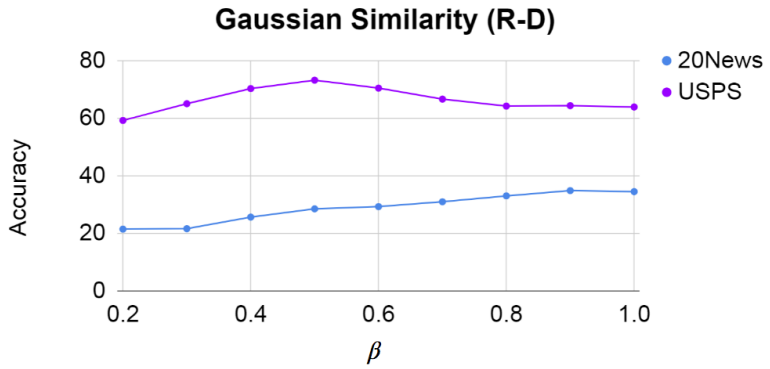


Figure 19: Varying the variance tuning parameter for Gaussian similarity

As seen in Figure 19, for Gaussian similarity, the accuracy for 20Newsgroups is poor but works well for USPS. However, tuning the β parameter results in a moderate increase in accuracy.

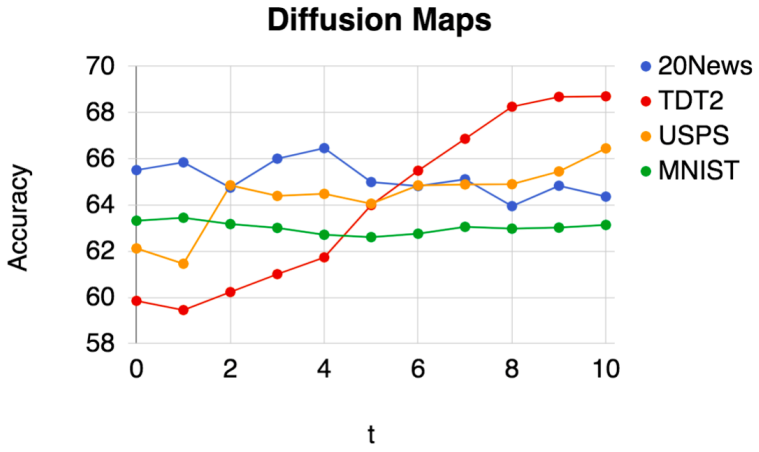


Figure 20: Varying the time-step parameter

Figure 20 shows accuracy for random landmark selection with the diffusion maps method and varying time-step numbers. Increasing the parameter does not affect MNIST or 20News a significant amount but improves accuracy noticeably for TDT2 and USPS.

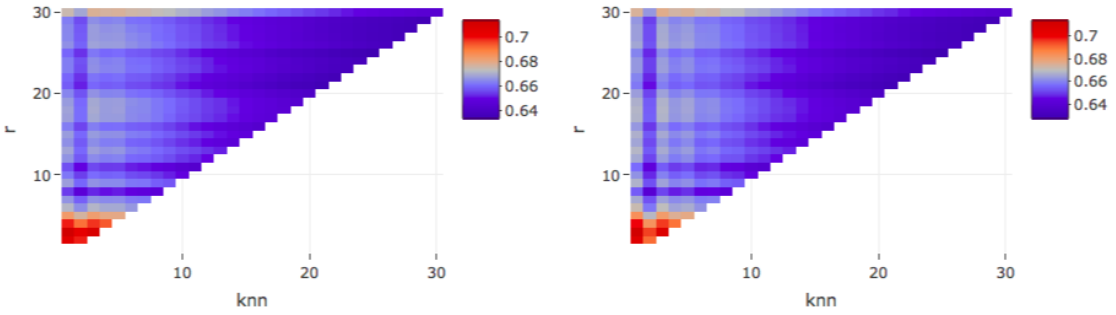


Figure 21: Varying r and knn for random landmark selection with landmark clustering for USPS. Left: majority vote. Right: weighted vote.

Landmark clustering is often faster than data clustering because kNN is much faster than k-means, especially because the required similarities are already computed in a previous step. In this test for the USPS dataset, we vary the number of nearest neighbors for kNN from 1 to r and vary r from 2 to 30. The majority vote and weighted vote methods achieve very similar results. Low r and knn parameters achieve the best results for USPS. With a fixed r, a lower knn parameter achieves slightly better results. For simplicity, we recommend using a majority vote with a knn parameter of 1. It is the fastest to compute and gives good results with this dataset.

4.9 Image Segmentation

Image segmentation is partitioning an image into different regions for different objects (see Figure 22 for an example). One application for image segmentation is with medical imaging, where the size and location of a tumor can be determined. In our case, only gray scale images are considered.

The input data is an image with $m \times n$ pixels and k objects inside. The output is the k clusters of mn pixels. NJW and LSC is applied to the pixels after producing similarity matrix.

Figure 22: Image segmentation example for four clusters.

While spectral clustering algorithms can perform image segmentation tasks, this is very computationally demanding because pixel to pixel similarity is considered, resulting in a $mn \times mn$ similarity matrix. Thus, we test and compare the scalable LSC algorithm and NJW algorithm for this task.

4.9.1 Similarity Matrix

Similarities are computed by Gaussian similarity considering both location and intensity values. When the pixels are far from each other, they are considered to have zero similarity. Therefore, for a pixel, the similarities are only computed for the pixels inside the $B \times B$ patch, whose center is this pixel, of the image.

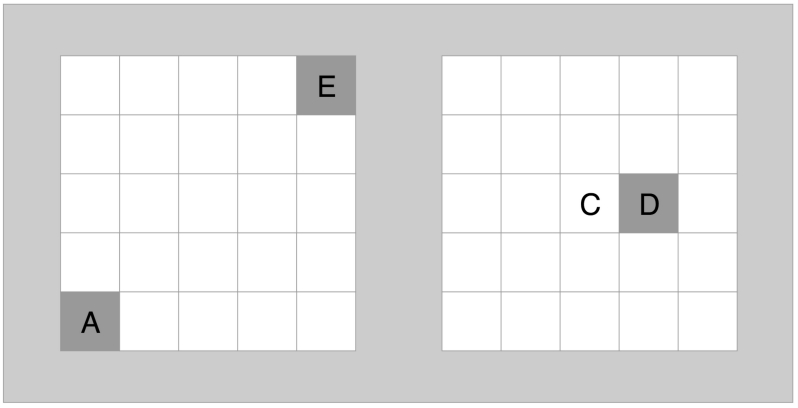


Figure 23: If $B = 3$, Pixels A and E have zero similarity because pixel A is not within a 3×3 patch of the image with center pixel E. Pixels C and D have zero similarity because of intensity.

For the original NJW spectral clustering, the elements in the similarity matrix

are computed by:

$$\mathbf{W}_{i,j} = e^{-\frac{|L_i - L_j|^2}{2\sigma_L^2} - \frac{|P_i - P_j|^2}{2\sigma_P^2}}$$

where L_i and L_j are the location of pixels i and j; P_i and P_j are the pixel values of pixels i and j; $\sigma_L = (B-1)/2$; σ_P is a parameter.

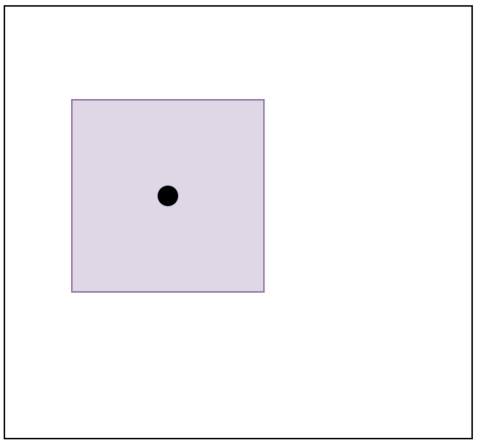


Figure 24: For the black point, only consider the pixels in the small box in the image.

For the landmark based clustering, the elements in the similarity matrix are computed by:

$$\mathbf{A}_{i,j} = e^{-\frac{|L_i - L_j|^2}{2\sigma_L^2} - \frac{|P_i - P'_j|^2}{2\sigma_P^2}}$$

where L_i and L_j are the location of pixel i and j; P_i is the pixel values of pixel i; P'_j is the mean intensity of the pixels in the grid box of the landmark j; $\sigma_L = (B-1)/2$; σ_P is a parameter.

The points in Figure 25 are the landmarks of the image. The size of the big box is $B \times B$ and only the pixels inside it can have similarities to the landmark in the small box. When computing similarities for this landmark, the mean intensity of the pixels in the small box is used.

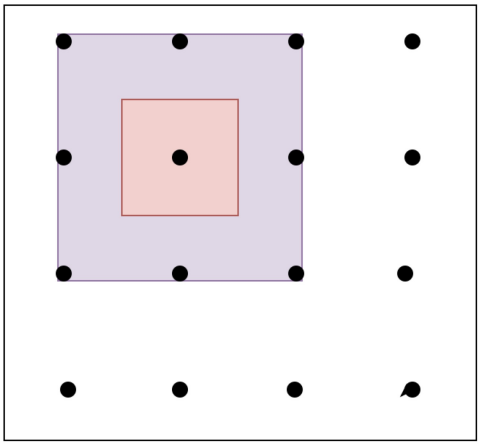


Figure 25: In the image, the black points are landmarks. Only the pixels in the big purple box have similarities with the landmark in the center of the small red box. When calculating the similarities for this landmark, the mean intensity of the pixels in the small red box is used for robustness.

4.9.2 Results

Three images are tested by both NJW and LSC methods. The segmentation results are shown in Figures 26 to 31.

Figure 26: The size of original image is 2883×3875. Reduced to 93×125 to speed the processing.

Figure 27: NJW(left): time = 74.17s, $B = 11$, $\sigma_P = 0.05$. LSC(right): time = 6.85s, $B = 25$, $b = 3$, $\sigma_P = 0.03$.

Figure 28: The size of original image is 710×1150. Reduced to 71×115 to speed the processing

Figure 29: NJW(left): time = 28.02s, $B = 25$, $\sigma_P = 0.02$. LSC(right): time = 3.55s, $B = 25$, $b = 3$, $\sigma_P = 0.03$.

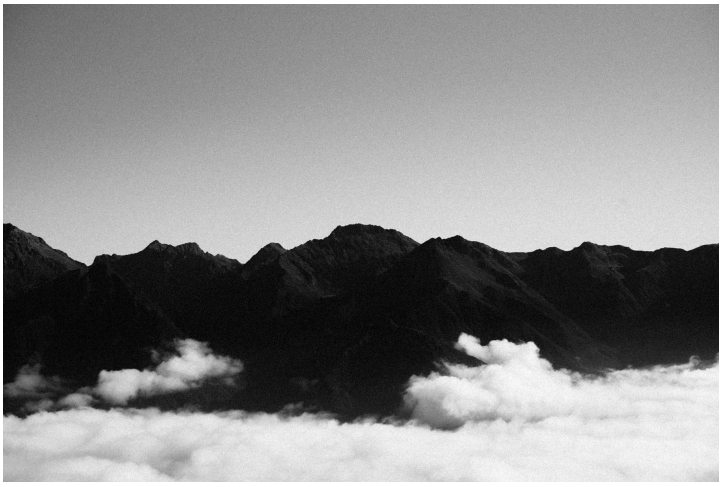


Figure 30: The size of original image is 3360×3510. Reduced to 112×117 to speed the processing

Figure 31: NJW(left): time = 27.22s, $B = 11$, $\sigma_P = 0.1$. LSC(right): time = 3.45s, $B = 17$, $b = 3$, $\sigma_P = 0.1$.

The LSC method is much faster than the NJW method on image segmentation because of the size of the similarity matrix of them. The number of columns of $\mathbf{W}$ is nine times the number of columns of $\mathbf{A}$ in all three images (the grid size b is 3), and they have same number of rows. For achieving similar results, the LSC algorithm is about ten times faster than the NJW algorithm.

5 Landmark-Based Bipartite Graph Spectral Clustering

In this section, we study the bipartite graph and use it as the underlying model for a fast approximate Spectral Clustering algorithm. Bipartite graph is naturally used when there is relation between unequivalent groups, or when the relation only holds one way but not the other (i.e. directed graph). This section will be ordered as follows: Section 5.1 introduces the bipartite graph. Section 5.2 shows how to apply spectral clustering on bipartite graph. Section 5.3 introduces the main algorithm: landmark-based bipartite spectral clustering and diffusion map. Section 5.4 uses experiments to compare the effectiveness of our algorithm with several state-of-the-art approximate clustering algorithms.

5.1 Introduction

5.1.1 The bipartite graph

Various datasets have a common structure: There is a set of main entities called observations (i.e images, documents...). The observations share a set of feature. The feature is represented by a value that varies among the observations, and thus it is also called variable. For example, in a natural image dataset, the features are pixels. The pixel at position 1 in an image may be different from the pixel in position 1 of a different image. The features of an image, however, can be other characteristics as well, such as colors, edges, or more elaborated features such as Gabor filters, histogram of oriented gradients and GIST descriptors. Sometimes, the features can be subject to learning as well, i.e we want to perform classification or clustering on these features. The most common way to do this is to treat feature the same as observation. For example, a popular type of document dataset is the bag-of-word model, where each feature is the number of occurrence of a word in a document. Word can be observation as well, of which documents where it appears become its feature. We can see a symmetry of relationship: whereas word feature tells how different the documents are, document features conversely tells how different the words are. This relationship can be represented as an edge in a bipartite graph. We will define it next.

Let $\mathbb{G}$ be a graph consisting of a set of vertices $\mathbb{V}$ and a set of edges $\mathbb{E}$. Commonly, existence of an edge represents connection between 2 vertices, but a vertex can also have an edge to itself (self-edge). Let $\mathbb{V} = \mathbb{V}_1 \cup^* \mathbb{V}_2$ be the disjoint union of 2 sets of vertices. As a recurring example, consider $\mathbb{V}_1$ the set of documents, $\mathbb{V}_2$ the set of words and $\mathbb{E}$ the set of edges between documents and words. Associate with each edge the number of occurrence of the word in a document. The larger

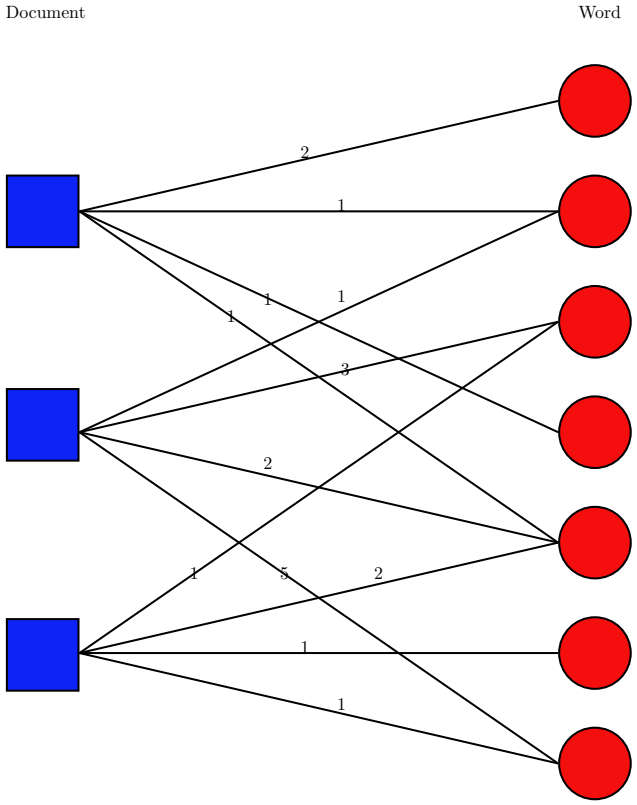


Figure 32: Biparite graph of bag-of-word model. The number on each edge represents the number of word occurrence.

the value, the more often that word appears in that document. There is no edge between any 2 documents as well as between any 2 words: we have only information of relationship between document and word. Thus it is a bipartite graph. We can add edges between 2 vertices of the same group, but the graph will no longer be bipartite. Figure 32 demonstrates the equivalence of bipartite graph and bag-of-word model.

5.2 Spectral Clustering on Bipartite Graph

5.2.1 Formulation and a shortcut

The Spectral Clustering algorithm derived from section 2.3 can be immediately applied to a bipartite graph. Let the affinity matrix of the bipartite graph be:

$$M = \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix} \tag{1}$$

We have ordered the vertices such that document vertices go first. Thus in the first row, 0 represents the affinity matrix between document and document (of which there is no information) and A represents the affinity matrix between document and word, which is simply the bag-of-word model (A^T is the "bag-of-document"). The normalized Laplacian matrix is then:

$$\begin{aligned}\hat{M} &= \begin{bmatrix} D_1^{-1} & 0 \\ 0 & D_2^{-1} \end{bmatrix} \times M \\ &= \begin{bmatrix} 0 & D_1^{-1}A \\ D_2^{-1}A^T & 0 \end{bmatrix}\end{aligned} \tag{2}$$

where D_1 is diagonal matrix of row-sums of A and D_2 is diagonal matrix of row-sums of A^T (equivalently D_2 is of column-sums of A)

Let the number of documents be m and number of words be n. Then A is of size $n \times m$ and M is of size $(n+m) \times (n+m)$. The eigenvalue decomposition takes $O(n+m)^3$ times to compute. This is however rather redundant, because the matrix M only contains as much information as A in the non-zero blocks and the rest are 0. In fact, as shown in [5], we can retrieve all eigenvectors and eigenvalues of $\hat{M}$ from a singular-value decomposition of a smaller matrix:

$$\widetilde{A} = D_1^{-1/2} A D_2^{-1/2} \tag{3}$$

Theorem 1. *Let $\widetilde{V}_1$ and $\widetilde{V}_2$ be the left and right singular vectors of $\hat{A}$. Also let S be the diagonal matrix of singular values of $\widetilde{A}$. Let $V = \begin{bmatrix} V_1 \\ V_2 \end{bmatrix}$ be the eigenvector of $\hat{M}$ in which V_1 is the upper half of V corresponding to document vertices and V_2 is the lower half corresponding to word vertices (remember that we placed documents first). Then*

$$\widetilde{A} = \hat{V}_1 \times S \times \hat{V}_2^T$$

and

$$\hat{M} = \begin{bmatrix} D_1^{-1/2}\hat{V}_1 \\ D_2^{-1/2}\hat{V}_2 \end{bmatrix} \times S \times \begin{bmatrix} D_1^{-1/2}\hat{V}_1 \\ D_2^{-1/2}\hat{V}_2 \end{bmatrix}^T \tag{4}$$

Thus the eigenvectors of $\hat{M}$ are $V_1 = D_1^{-1/2}\hat{V}_1$ and $V_2 = D_2^{-1/2}\hat{V}_2$ and the diagonal eigenvalue matrix is exactly S

Proof. Since V is eigenvectors of M, we have

$$\hat{M}V = \Lambda V$$

From (3) and by definition, we have the system:

$$D_1^{-1}AD_2^{-1/2}V_2 = \Lambda D_1^{-1/2}V_1$$
$$D_2^{-1}A^T D_1^{-1/2}V_1 = \Lambda D_2^{-1/2}V_2$$

Multiply the first equation by $D_1^{1/2}$ and the second by $D_2^{1/2}$

$$D_1^{-1/2}AD_2^{-1/2}V_2 = \Lambda V_1$$
$$D_2^{-1/2}A^T D_1^{-1/2}V_1 = \Lambda V_2$$

From (4)

$$\widetilde{A}V_2 = \Lambda V_1$$
$$\widetilde{A}^T V_1 = \Lambda V_2$$

From definition of singular value decomposition, V_1 and V_2 are left and right singular vectors of $\hat{A}$, and $\Lambda = S$ the singular values of $\widetilde{A}$. □

The singular value decomposition of $\hat{A}$ takes $O(nm^2 + m^3)$. If we had used a much smaller m than n, we would get a significant speed-up.

5.2.2 Computational complexity for sparse matrix

The above analysis applies for general matrix, however in many cases the input matrix is sparse: it has only a very small percentage of non-zero entries. This is especially true for graph that comes from real world such as social network. An edge in a social network represents the connection of 2 people. In real life, a person has on average hundreds of friends, whereas most inactive people will have only dozens of friends. Compare to the size of the number of participating people, which could be in the millions, a person only has about 0.01% of possible connections. On the other hand, when constructing a relational graph by the similarity distance of datapoints, we can also use sparse affinity matrix from using the k-nearest-neighbor or ϵ-nearest neighbor criterion. This has the advantage of fast computation and less storage. Given $x_1, x_2, ..., x_n \in R^n$, the distance between 2 points is calculated as:

$$\text{For k-nearest-neighbor: } dist(x_i, x_j) = \begin{cases} e^{\frac{||x_i - x_j||^2}{2\sigma^2}} \text{ if} x_i \in \mathbb{N}_k(x_j) \text{ or } x_j \in \mathbb{N}_k(x_i), \\ 0 \text{ otherwise} \end{cases}$$

$$\text{For } \epsilon\text{-nearest neighbor: } dist(x_i, x_j) = \begin{cases} e^{\frac{||x_i - x_j||^2}{2\sigma^2}} if e^{\frac{||x_i - x_j||^2}{2\sigma^2}} \leq \epsilon, \\ 0 \text{ otherwise} \end{cases}$$

As suggested by [8], sparse nearest-neighbor graph is often preferred and should be tried first. There are also works in the manifold literature which says adequately

sparse affinity matrix can well approximate topological property of the full one. Intuitively, nearest-neighbor tries to preserve the local structure surrounding each datapoint only, letting global structure to be implied from it.

More importantly, regarding computational complexity of algorithm, operations on sparse matrix can benefit a lot from a supporting linear algebra package. Suppose the matrix is of size $n \times m$ and k neighbors are used to. It still takes $O(mn)$ time to construct the full matrix, $O(mnk)$ time to find k nearest neighbors for each row. However, rank-s SVD of the matrix can be calculated in only $O(nks)$, in contrast to $O(nm^2 + m^3)$. This can be done, for example, using the Lanczos method, where only sparse matrix-vector operations are used.

5.3 Landmark-based Bipartite Spectral Clustering

In this section, we will use the bipartite graph model to approximate the original spectral clustering (specifically the Ncut algorithm). First we provide the intuition behind it along with detail of implementation. We also extend the original algorithm with diffusion map. Finally, we show experiments comparing it with several other approximate algorithms and also study parameter sensitivity.

5.3.1 Intuition and related work

The main idea of our work is to reduce the size of the dataset while still preserving much of the original structure. The first work along this line was [16], where the author uses k-means sampling and perform original spectral clustering algorithm on the reduced set. However, this algorithm completely skipped the information of original datapoints and instead worked with reduced set. This often leads to crude separation.

The second work is LSC [2]. The authors used dictionary learning and sparse coding to create new feature vectors based on landmark points. This algorithm improves from KASP in the sense that datapoints appear on the final matrix, thus often allowing points to be clustered free of centroids. Moreover, the algorithm takes advantage of sparsity which both leads to run-time boost and superior clustering quality. However, they used a normalization formula that lacks foundation.

Our work can be seen as a direct generalization of KASP as well as an improvement to LSC. First we also sample using k-means like KASP, then build a sparse affinity matrix like LSC. However, we then use the bipartite formulation to justify a normalization step that not only makes intuitive senses but also enables 1) landmark clustering and 2) diffusion map.

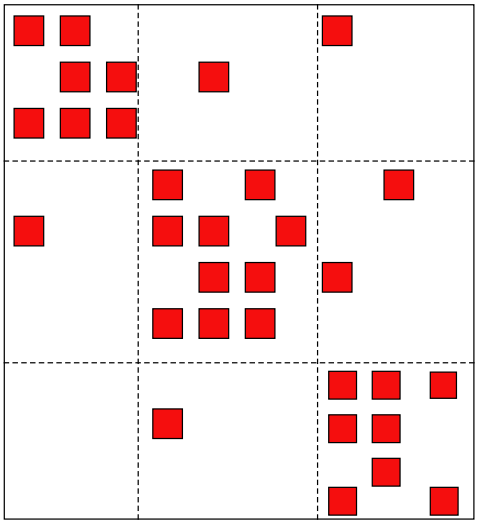


Figure 33: The affinity matrix between data points and sampled centroids, where number of nearest neighbors is 3. Datapoints (rows) have been ordered together if they are in the same cluster. The result is a majority of non-zero entries stay in well-separated blocks in the diagonal.

5.3.2 Diffusion map

Diffusion map [4] is a close variant of spectral clustering. It integrates local similarities by generating random walks in data, connecting points that are far but lie in a continous manifold. It does this by taking the power of the transition probability matrix. Recall the normalized Laplacian matrix in equation 2. $\hat{M}$ itself is a random walk matrix that contains 2 block random walk matrices: from datapoints to landmarks ($D_1^{-1}A$) and from landmarks to datapoints ($D_2^{-1}A^T$). To apply diffusion map, we generate random walks from each data points to get probability of "jumping" from one node to the other in a certain step size. Consider figures 34 and 35. Because initially we only consider landmarks neighbors, there is no connection between the 2 blue datapoints (figure 34). After running a random walk of length 2, however, the datapoints connect with probability equaling the sum of probabilities of 3 possible paths through the landmarks (figure 35).

With odd random walk length, datapoints go to landmarks, retaining the bipartite structure. We can thus follow co-clustering algorithm of Dhillon [5]. With even random walk length, we have 2 disconnected components: one consists of datapoints to datapoints and the other landmarks to landmarks (see figure 34 and 35). Clustering with datapoints embedding results directly in cluster assignments, while clustering with landmarks requires one more step of extending the landmark cluster assignments to datapoints, which can be easily done through s-nearest-landmark-

neighbors. The algorithm, named Landmark-based Diffusion Map (LBDM) is shown in algorithm 1
Note this duality is the result of using the bipartite graph formulation, whereas LSC used the same affinity matrix in figure 33 but came to a different normalization. The landmarks in LSC do not have an equal interpretation and thus can not be used for landmark clusterings.

More formally, we have the following theorem:

Theorem 2. *The p-dimensional diffusion coordinates at time step α are*

$$\begin{bmatrix} D_1^{-1/2}\hat{V}_1 S^\alpha \\ D_2^{-1/2}\hat{V}_2 S^\alpha \end{bmatrix} \in \mathbb{R}^{(m+n)\times p}$$

where S is the diagonal matrix of eigenvalues of $\hat{M}$.

Finally, we note that landmark clustering turns out to be the best algorithm on average, as demonstrated in later experiments. It is also faster because of running last k-means clustering only on the smaller set of landmarks. We think that it could be due to k-means having an easier time clustering the landmark sets which are regularized and less noisy.

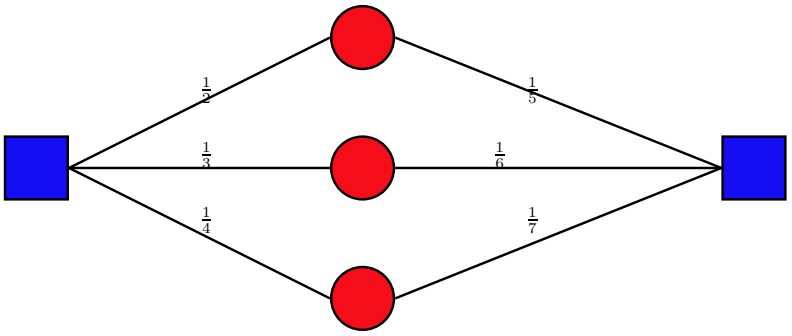


Figure 34: datapoints-to-landmarks. There is no edge between any 2 datapoints or any 2 landmarks. We can use co-clustering to cluster both.

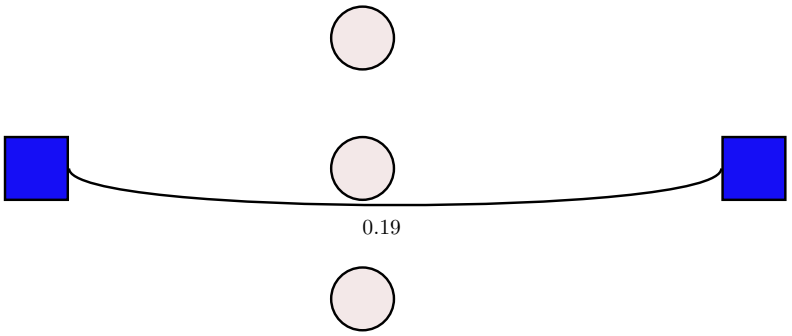
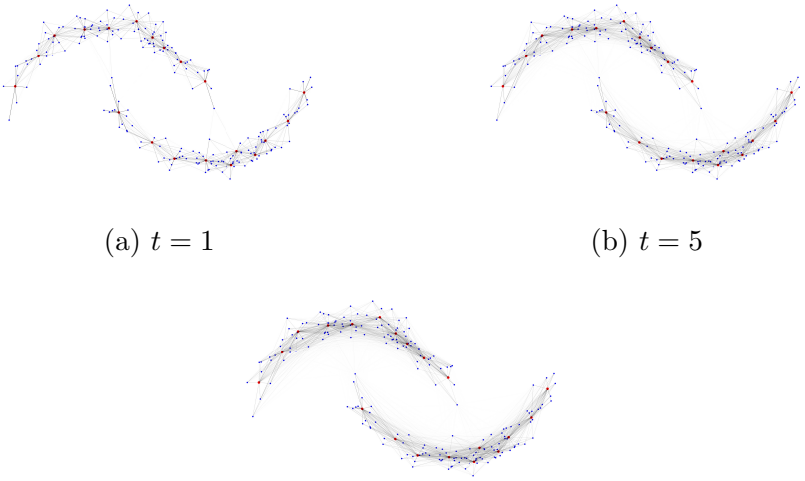


Figure 35: datapoints-to-datapoints connection resulting from running random walk of length 2. There is only edge between datapoints (conversely between landmarks), thus we can cluster datapoints or landmarks separately.

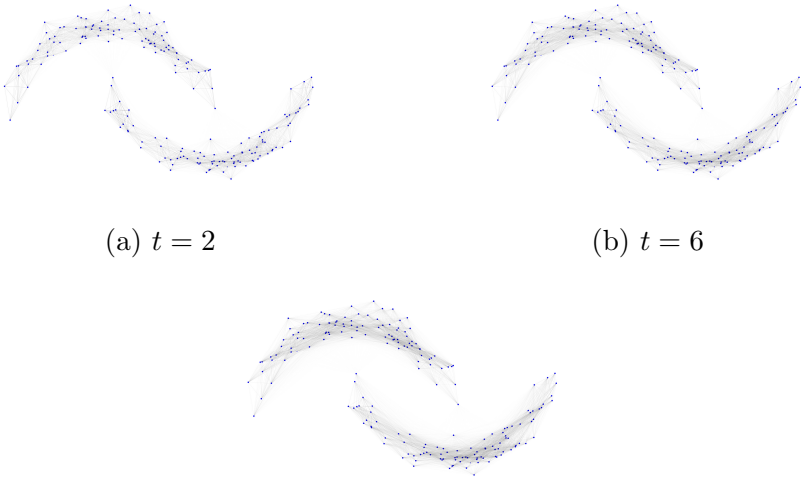
The following figures demonstrate the ability of diffusion map to strengthen the connections of points in the same manifolds. As t increases, each half-moon manifold darkens by the increasing number of edges generated by diffusion map.



(a) $t = 1$ (b) $t = 5$

(c) $t = 9$

Figure 36: diffusion map of odd length



(a) $t = 2$ (b) $t = 6$

(c) $t = 10$

Figure 37: diffusion map of even length, only datapoints are shown

5.4 Experiments

In this section, we conduct extensive experiments to evaluate the practical performance of Alg. 1 with $\alpha = 1, 2, 3$. For the two odd values of α (i.e., 1 and 3), for which the co-clustering method has to be used, we denote the corresponding implementations by LBDM$^{(1)}$ and LBDM$^{(3)}$. For the even value $\alpha = 2$, we use both the direct clustering and landmark clustering methods and denote them as LBDM$^{(2,X)}$ and LBDM$^{(2,Y)}$, respectively.

5.4.1 Experimental setup

We compare our methods with the following algorithms, all implemented/executed in MATLAB 2016b on a computing server with 48GB of RAM and 2 CPUs with 12 total cores:

- **KASP** [16]: We implement a multiway version of the authors' R code at `https://people.eecs.berkeley.edu/~jordan/fasp.html`.
- **LSC** [2]: We use the fully optimized MATLAB code (implemented by the authors), available at `http://www.cad.zju.edu.cn/home/dengcai/Data/Clustering.html`.
- **cSPEC**$^{(n)}$: We adapt the column-sampling spectral clustering (cSPEC) algorithm [15], which only works for unnormalized spectral clustering, for the

Algorithm 1 Landmark-based Bipartite Diffusion Maps (LBDM)

Require:

Data set $X = \{x_1, x_2, ..., x_n\} \subset \mathbb{R}^d$,
clusters k,
similarity function δ (Gaussian RBF, cosine, etc.),
landmark selection method ($k$means, uniform sampling),
landmark points m,
nearest landmark points s,
diffusion steps α,
clustering method: direct or landmark (for even α), or co-clustering (for odd α).

Ensure:

A partition of X into k clusters.

1: Use the indicated landmark selection method to find m landmarks $Y = \{y_1, \ldots, y_m\} \subset \mathbb{R}^d$.
2: Form the affinity matrix A between the input data X and their respective s nearest landmark points in Y by using the given similarity function δ.
3: Find the row and column sums of A and use them to normalize A to obtain $\widetilde{A}$ (as in (3)).
4: Find the largest $k-1$ singular values (excluding 1) and corresponding left and right singular vectors of $\widetilde{A}$.
5: Compute the diffusion coordinates matrix $\mathbb{V}^{(\alpha)}$ according to theorem 2 (with $p = k-1$).
6: Use the indicated clustering method to divide the input data set X into k clusters.

Table 9: Data sets used in the experiments of this section.

Datasets	**# instances**	**# features**	**# classes**
usps	9,298	256	10
pendigits	10,992	16	10
letter	20,000	16	26
protein	24,387	357	3
shuttle	58,000	9	7
mnist	70,000	784	10

normalized spectral clustering [10]

$$\widetilde{W} = D^{-1/2} W D^{-1/2}, \quad D = \mathrm{diag}(W1) \tag{5}$$

by regarding the first D matrix as having the row sums of W and the second D the column sums of W. Since our goal is now to sample columns of $\widetilde{W}$ in order to use the matrix SVD to estimate its eigenvectors, we propose to normalize A in a similar way:

$$\widetilde{A} = D_1^{-1/2} A D_2^{-1/2}, \tag{6}$$

where D_1, D_2 are diagonal matrices consisting of the row sums and column sums of A, respectively. Interestingly, this leads to the same matrix $\widetilde{A}$ that is used by our algorithm (but we obtained it from a bipartite graph model). However, such a normalized version of cSPEC, which we denote by cSPEC$^{(n)}$, will just use the left singular vectors of $\widetilde{A}$ as an embedding of the input data while our methods will further multiply them by the degree matrix (from left) and the singular values of $\widetilde{A}$ (from right) to generate multiscale diffusion coordinates.

- **Dhillon's co-clustering algorithm** [5]: Originally designed for simultaneously clustering documents and terms, we use it here in the same way as LBDM$^{(\alpha)}$ (for odd values of α) to cluster only the input data, that is, first co-clustering the input data and landmarks and then removing the landmarks from the clusters.

We choose for our study six benchmark data sets - *usps*, *pendigits*, *letter*, *protein*, *shuttle*, *mnist* - from `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`; see Table 9 for their summary information.

In order to have fair comparisons between the different algorithms, we use the same parameter values (whenever the parameter is shared). In particular, we fix $m = 500$ (for all methods) and $s = 5$ (for LSC, Dhillon and LBDM with $\alpha = 1, 2, 3$). Also,

we feed all the algorithms with the same landmark set found by $k$means (with 10 iterations, 1 restart), which is initialized with the centroids obtained by preliminary $k$means clustering on 10% of the data set (with 100 iterations, 10 restarts). In all methods we use the Gaussian RBF kernel as the similarity function. For the choice of σ in the RBF kernel, we use (for our method):

$$\sigma = \frac{1}{m} \sum_{i=1}^{m} \sqrt{\frac{\sigma_i}{|S_i|}} \tag{7}$$

where σ_i and $|S_i|$ are the total variance and size of the i^{th} k-means cluster, respectively, and m is the number of k-means clusters. This means we calculate the variance of each cluster, then approximate our global variance by averaging over all clusters.

In the last step of each algorithm, where $k$means is applied to cluster data in the associated embedding space, we use 100 iterations and 10 restarts. The reason is that the quality of the sampled centroids is not too dependent on k-means' optimality, whereas the quality of clustering of the last k-means is obviously critical. Moreover, the first k-means is the main bottleneck of the algorithm, which will become clear.

Although not directly compared in clustering task, k-means is a critical component in the algorithms that use it. We used the same MATLAB implementation of [2], which can be found in their web page [1]. K-means has 2 main variables: the maximum number of iterations and the number of restarts. For k-means in step 1, we use a similar approach by [16]: First run k-means (max 100 iterations, 5 restarts) on 10% of the data set to select a set of initial centroids, then run k-means (max 50 iterations, 1 restart) on the full data set initialized with those centroids. For k-means in step 7, we run k-means with max 100 iterations and 10 restarts. The reason is that the quality of the sampled centroids is not too dependent on k-means' optimality, whereas the quality of clustering of the last k-means is obviously critical. Moreover, the first k-means is the main bottleneck of the algorithm, which will become clear.

We evaluate the different algorithms in terms of clustering accuracy and CPU time, with the former being calculated by first finding the best match between the output clusters and the ground truth and then computing the fraction of correctly assigned points. Each algorithm is repeated 50 times in order to average out the randomness caused by landmark sampling.

5.4.2 Results for K-means sampling LBDM

We refer to [12] for the results of K-means sampling. It is found that LBDM and especially LDBM$^{(2,Y)}$ performed consistently well, beating other baseline algorithms

[1] http://www.cad.zju.edu.cn/home/dengcai/Data/Clustering.html

Table 10: Average clustering accuracy (%) using random landmark sampling

Dataset	**LSC**	**cSPEC**$^{(n)}$	**Dhillon**	**LBDM**$^{(1)}$	**LBDM**$^{(3)}$	**LBDM**$^{(2,X)}$	**LBDM**$^{(2,Y)}$
usps	62.00	**67.05**	62.39	62.26	62.95	63.72	62.52
pendigits	**77.44**	70.12	77.35	77.35	77.35	77.43	77.30
letter	30.14	**36.33**	25.55	25.42	25.66	24.86	25.25
protein	45.17	40.79	45.16	45.16	45.16	**45.19**	45.16
shuttle	40.31	34.64	47.14	49.69	48.90	29.61	**51.05**
mnist	59.01	54.31	59.49	59.13	59.71	**60.08**	59.55

on a large number of datasets.

5.4.3 Results for Random sampling LBDM

Lastly, instead of k-means sampling of landmarks which can be time-prohibitive ($O(nmk)$), we can uniformly random a subset of original data points. Experimental setup is kept the same as last section with 1 exception: since we can not estimate the Gaussian RBG kernel parameter σ with k-means, we will use the mean of squared distance from each point to its 7-nearest neighbor:

$$\sigma = \frac{1}{n}\sum_{i}^{n} ||x_i - 7nn(x_i)||$$

Note that this can be a better estimation for σ but it is also more costly (finding the 7-nearest-neighbor of n points requires computing all n^2 distances and $7n$ more operations). A better strategy may be to estimate from a subset of data e.g 50 points. In the experiments, we actually used 5000 points to get as accurate as possible the variance.

We now present each experiment result with random sampling.

5.4.4 Clustering accuracy and Run time

In general, clustering accuracy decreases but the amount varies for each dataset (table 10). The maximum accuracy in k-means sampling and random sampling are:

- usps 69.45 $(LBDM^{(2,X)}) \rightarrow$ 67.05 $(cSPEC)$
- pendigits 77.93 $(LSC) \rightarrow$ 77.44 (LSC)
- letter 32.21 $(LBDM^{(2,X)}) \rightarrow$ 36.33 $(cSPEC)$
- protein 45.88 $(LBDM^{(2,Y)}) \rightarrow$ 45.19 $(LBDM^{(2,Y)})$
- shuttle 82.78 $(cSPEC) \rightarrow$ 51.05 $(LBDM^{(2,Y)})$

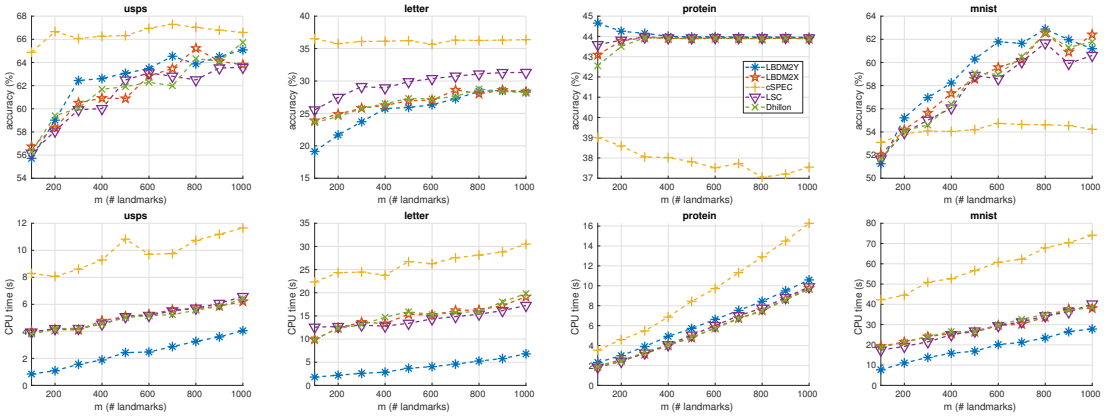


Figure 38: Sensitivity study of the parameter m with random landmark sampling (20 replications). Top row: clustering accuracy; bottom row: CPU time.

- mnist 73.29 $(LBDM^{(2,Y)}) \rightarrow 60.08$ $(LBDM^{(2,X)})$

Surprisingly, cSPEC performs very well on the usps and letter datasets. It is a good place to look into since cSPEC uses a dense matrix, and random sampling may requires dense information from the dataset.

Another surprise is that most algorithms perform better than k-means sampling in the pendigits and protein, and cSPEC in the letter dataset. Do these changes only depend on the datasets? Or is it because we did not use the same parameter σ for these experiments? We suspect the former reason because all algorithm performs poorly on shuttle and mnist with random sampling by a large margin.

5.4.5 Parameter sensitivity study

We observe the general trend in landmark sensitivity (figure 38) as in k-means sampling: In 3 of the datasets the accuracy increases monotonically as the number of landmarks increases. cSPEC has the best accuracies on usps and letter throughout but the worst on protein and mnist again. This further shows that the datasets are quite different in nature.

Disregarding cSPEC, $LBDM^{(2,Y)}$ performs quite well, which is again similar to k-means sampling in the same experiment.

Again, the accuracies decrease as the number of nearest landmarks increases, in accordance with earlier experiment.

Overall, k-means sampling are much more robust. Only when k-means becomes prohibitive should we resort to random sampling, since we use k-means to save time in the first place. For example, when each feature vector is too large, making distance calculation significantly slower, or when memory is limited since k-means requires accessing a matrix of size $n \times m$ for each iteration.

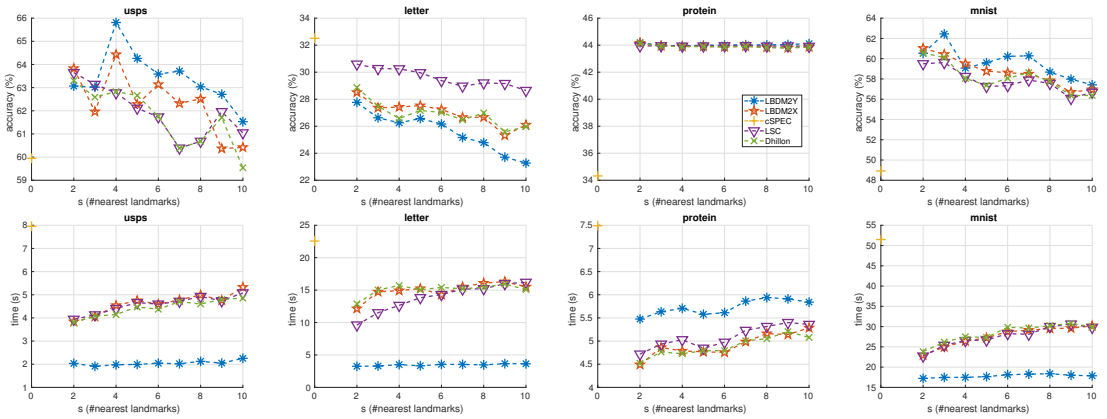


Figure 39: Sensitivity study of the parameter s with random landmark sampling (20 replications). Top row: clustering accuracy; bottom row: CPU time.

6 Simultaneous Document and Word embeddings

In this section, we introduce a "side application" of spectral clustering: Embedding both words and documents in the same space so that nodes (either word or document) that are close in the embedding space are closely related. It is known that the eigenvectors of Spectral Clustering can be used as good non-linear dimensionality reduction. The Laplacian eigenmap [1] is one such algorithm that uses almost the same algorithm procedure. The authors noted that the quadratic form of the Laplacian, fLf', where f is a scalar embedding (a number) of nodes, is equivalent to:

$$fLf' = \frac{1}{2}\sum_{i,j}(f_i - f_j)^2 W_{ij} \tag{8}$$

Proof. Note $L = D - W$. We have:

$$\begin{aligned}
&\sum_{i,j}(f_i - f_j)^2 W_{ij} \\
&\qquad = \sum_{ij} f_i^2 W_{ij} + \sum_{ij} f_j^2 W_{ij} - 2 f_i f_j W_{ij} \\
&\qquad = \sum_{ij} f_i^2 D_{ii} + \sum_{ij} f_j^2 D_{jj} - 2 f_i f_j W_{ij} \\
&\qquad = 2 f L f'
\end{aligned}$$

□

We take a closer look at the above equation when the task is to minimize fLf'. There are 3 observations:

- When W_{ij} is large, f_i and f_j tend to be close together to reduce the loss.
- W_{ij} and $(f_i - f_j)^2$ are non-negative
- The additional condition $fDf' = 1$ translates the optimization to an eigen-problem. It also removes an arbitrary scaling factor in the embedding [1].

For embedding $F \in R^n$, we can show that the problem is minimization of $tr(FLF)$. Going back to bipartite graph, since $W_{ij} = 0$ every time when i and i are both documents or both words, we have:

$$tr(DocLWord') = \sum_{i,j} ||Doc_i - Word_j||^2 W_{ij} \tag{9}$$

It is obvious that when a word j appears many times in document i, then Doc_i and $Word_j$ must be close. This in turn results in proximity among words and among documents themselves. Moreover, even when a word is not in a document, there can still be similarity if through intermediate words and documents they are actually related. As in previous section, we can always run a diffusion map to strengthen this connection.

6.1 The problem of low-degree nodes

Though easily said, a straight eigen-decomposition of the Laplacian matrix will fail for some datasets, especially document datasets such as 20news (figure 40a). The reason is that many words appear in only 2 or less documents, and some documents are relatively shorter and have less distinct number of words.

Stochastic block model [9] is one of the staple models in the statistical network community and is often used to model real-world network exhibiting cluster behavior. However, it was found that the simple model lacks variation in node degrees inside each clusters. To make up for this weakness, recent research has proposed the degree-corrected stochastic block model [7]. Later research [13, 3] proposed several ways to improve spectral clustering via its usage as an approximation to the degree-corrected block model. The simplest variant, the regularized spectral clustering [13], which simply "inflates" the degrees of all nodes such that low-degree nodes become higher asymptotically, and has been well-received.

Technically, given affinity matrix W and diagonal degree matrix D, regularized spectral clustering takes the average degree $\delta = \frac{1}{n}\sum diag(D)$ where $diag(D)$ takes only the diagonal of D. The inflated matrix is:

$$\widehat{W} = W + \delta I$$

Similarly, $\widehat{D} = D + \delta I$. We have the normalized Laplacian:

$$\widehat{L} = \widehat{D}^{-1/2}\widehat{W}\widehat{D}^{-1/2}$$

In the case of bipartite graph, we need to inflate degrees of both words and documents. The take average document degree $\delta_1 = \frac{1}{n}\sum diag(D_1)$ and $\delta_2 = \frac{1}{m}\sum diag(D_2)$. Finally, the inflated version of $\widetilde{A}$ is:

$$\widetilde{A} = \widehat{D}_1^{-1/2}\widehat{A}\widehat{D}_2^{-1/2}$$

6.2 Is co-clustering good?

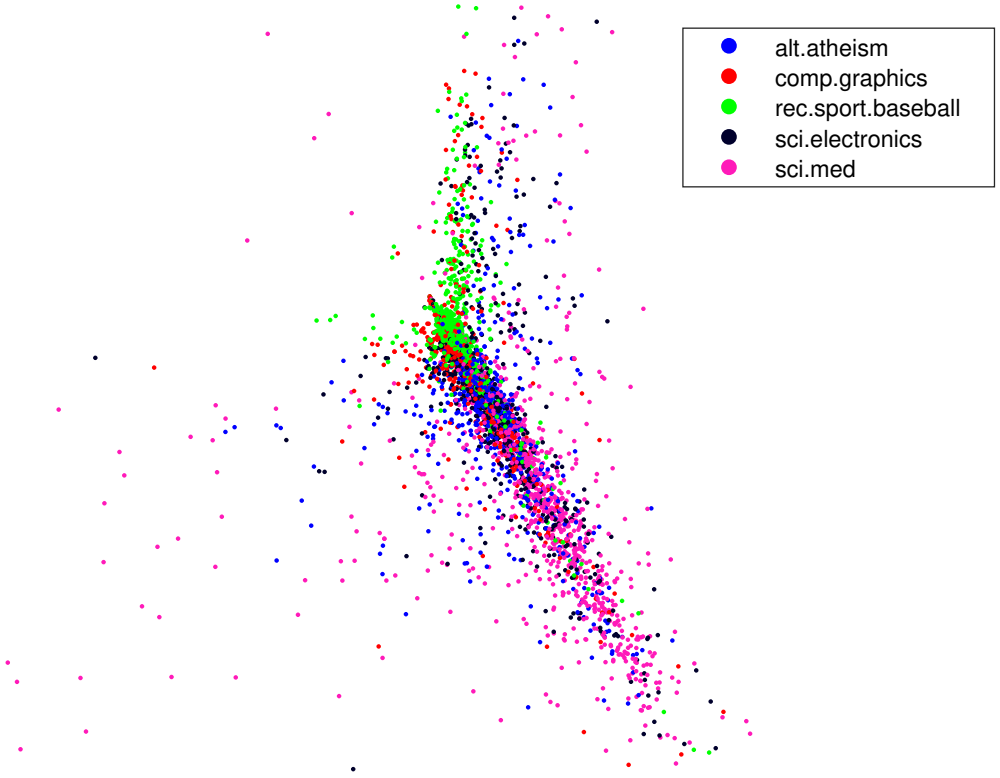
During experiments, we found an intriguing issue: K-means to cluster embeddings might fail to return balanced clusters. When number of words is much bigger than number of documents, it happens often that k-means returns a cluster of only words. Here 10 random groups from the 20news dataset were co-clustered 41:

cluster	#docs	#words
group1	866	28026
group2	2415	7555
group3	129	10484
group4	1103	15123
group5	989	0
group6	1015	0
group7	0	0
group8	947	0
group9	1126	0
group10	968	0

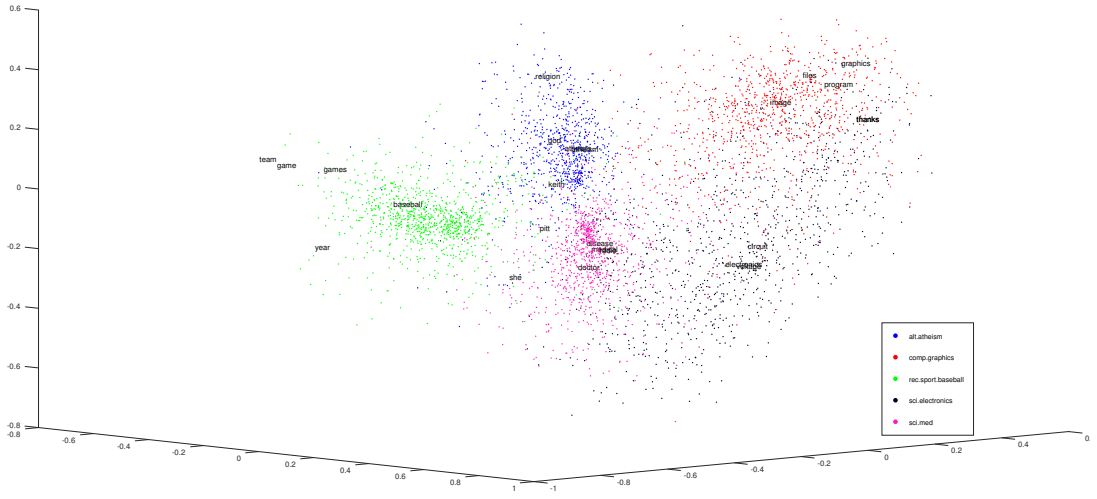
This however does not happen to landmark-based clustering. This issue shows that there is a subtle difference between a "geometric" bipartite graph and a document-word bipartite graph. "Geometric" means that the landmarks encode information of their surrounding datapoints. Landmarks always lie in a mixed cluster with at least its surrounding datapoints inside. On the contrary, document-word bipartite graph is not similarly regular, and can show unbalanced clusters of arbitrary nodes, sometimes only word or only documents. Thus we recommend using direct-clustering for out-of-the-box bipartite graph, even though that means not being able to get topic words through co-clustering. Figure 42 shows confusion matrix of document embeddings only clustering.

6.3 Finding topic words through spectral embedding

In this section, we attempt to find in each cluster a set of words representative of that cluster and through which we can identify what the documents from the same



(a) 5 news embeddings without regularization



(b) 5 news embedding with regularization

Figure 40: spectral embedding of 5 groups from the 20 news dataset. The regularized version has much interpretable words inside clusters

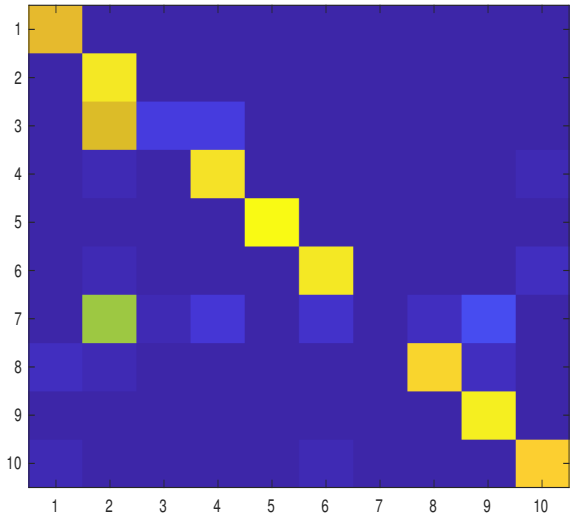


Figure 41: Confusion matrix of 10 random groups (20news) accuracy using co-clustering. Note the zero diagonal entry. The accuracy is 25.5%

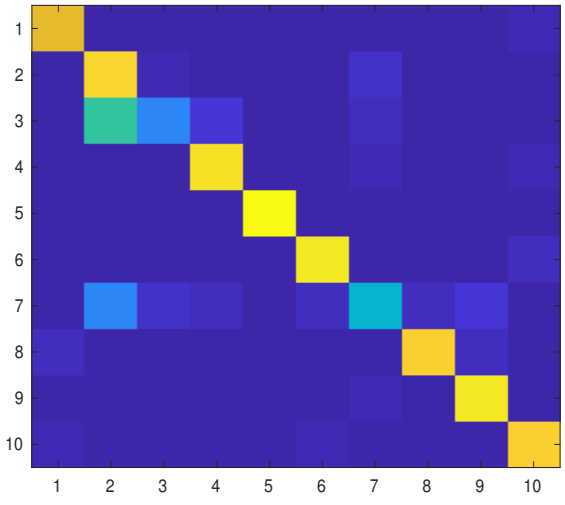


Figure 42: Confusion matrix of 10 random groups (20news) accuracy using direct clustering (clustering document embeddings only). The accuracy is 81.10%

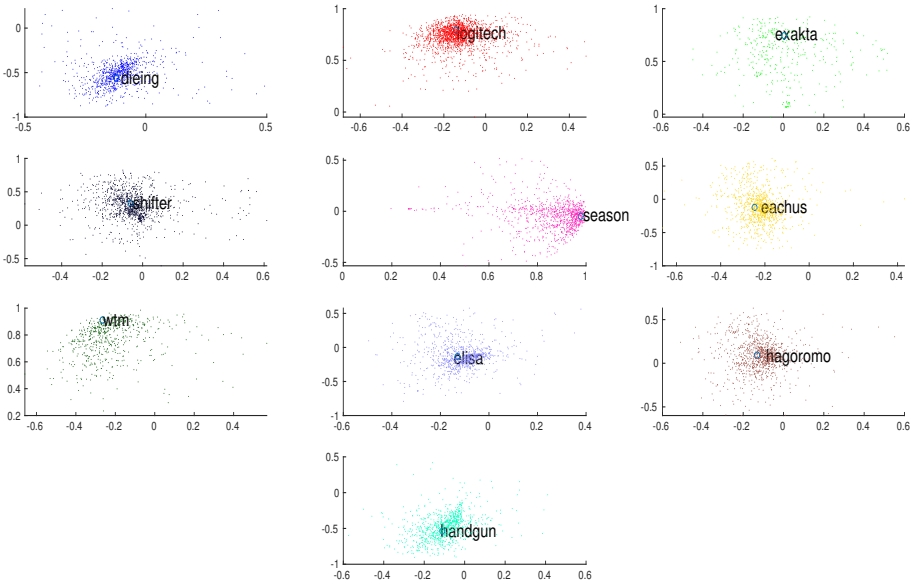


Figure 43: Document embeddings of each cluster along with the centroid and its nearest word

cluster is about. To make the problem simpler, we only focus on a subset of 10 distinguishable news groups: alt.atheism, comp.sys.mac.hardware, misc.forsale, rec.autos, rec.sport.hockey, sci.crypt, sci.electronics, sci.med, sci.space, talk.politics.guns. We used 3 methods: **centroid distance**, **word-document distance** and **kernel density estimation**. Note in 6.2, we have seen that co-clustering might lead to bad clusters. From here on, we only cluster document embeddings and use this grouping to aid us in finding topic words.

6.3.1 Centroid distance

Given a cluster of documents, we can find its centroid, then find which words are closest to it. Words near cluster centroid must be close to other documents in the cluster as well. Centroid is a good representative of the cluster, being the empirical mean, but also a very simple way requiring little computation.

We list the top 20 words nearest to each cluster's centroid along with their cosine distance from their respective centroid:

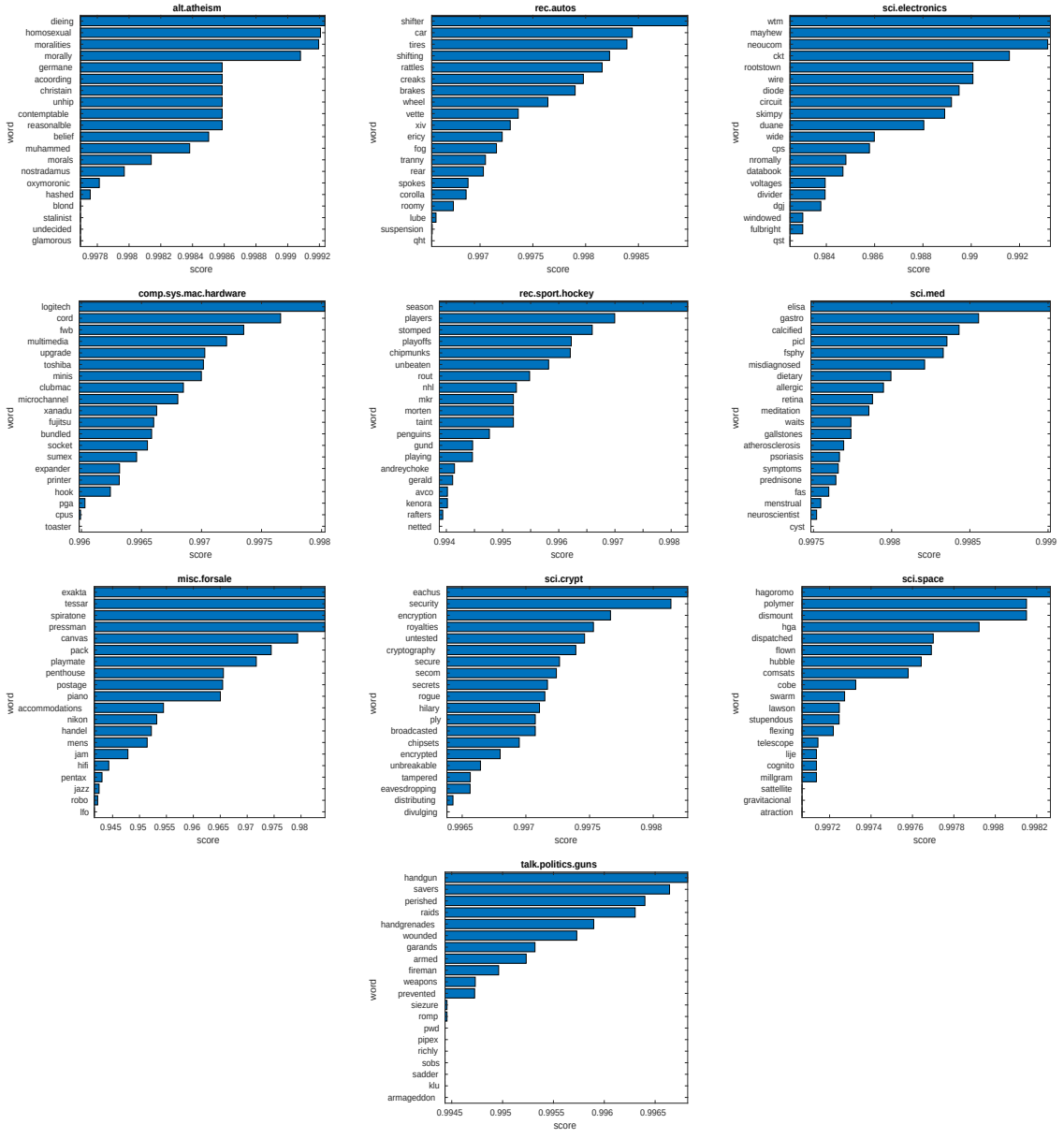


Figure 44: Top 20 words of each of 10 groups in with 81.10% accuracy (only 10 groups are clustered here) ranked by distance to centroid

However, a downside of centroid method is it is too simple: it is very susceptible to noise, especially when the learned cluster contains documents from other groups. Words closer to centroids do not necessarily mean they are more representative of the cluster, especially when the clusters exhibit diversity in topics. For example, in

cluster misc.forsale, the word "sale" does not turn up very often, but the cluster has a diverse vocabulary ranging from car models to superheros (supposedly from comic magazine sales). Thus it is very hard to capture all needed words in a specific region of the cluster when many of them scatter across space.

6.3.2 Word-document embedding distance

To better cover the whole cluster region, we now pick words by how close they are to the documents. For each word $word_i$ and each cluster C containing the set of documents doc_j, we use the score:

$$score_C(word_i) = \sum_{doc_j \in C} \langle word_i, doc_j \rangle$$

It gives how "close" $word_i$ is to C. Note we use inner product (or cosine affinity) because embeddings have been normalized to length 1. We now give the top 20 words for each cluster using this score:

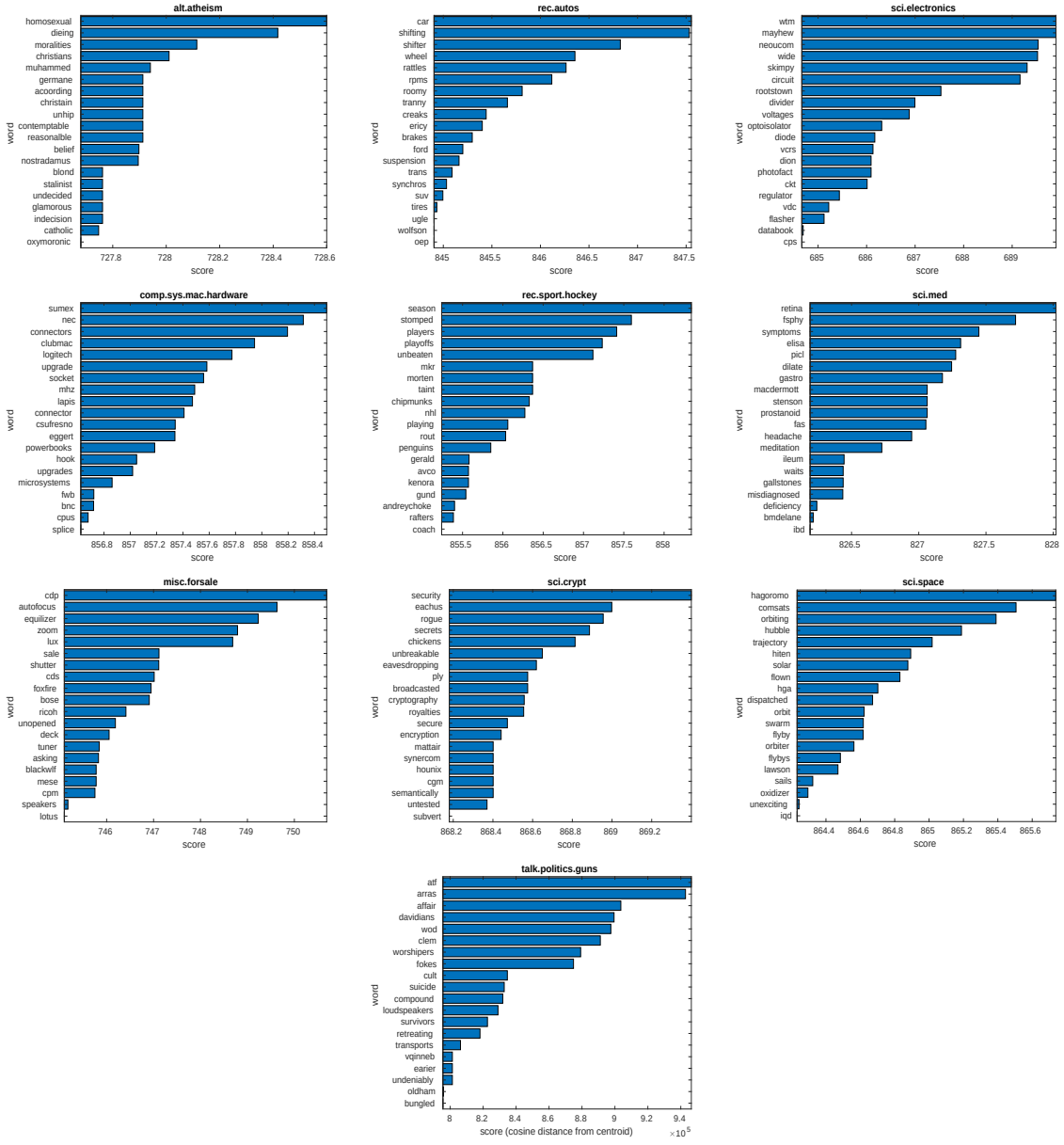
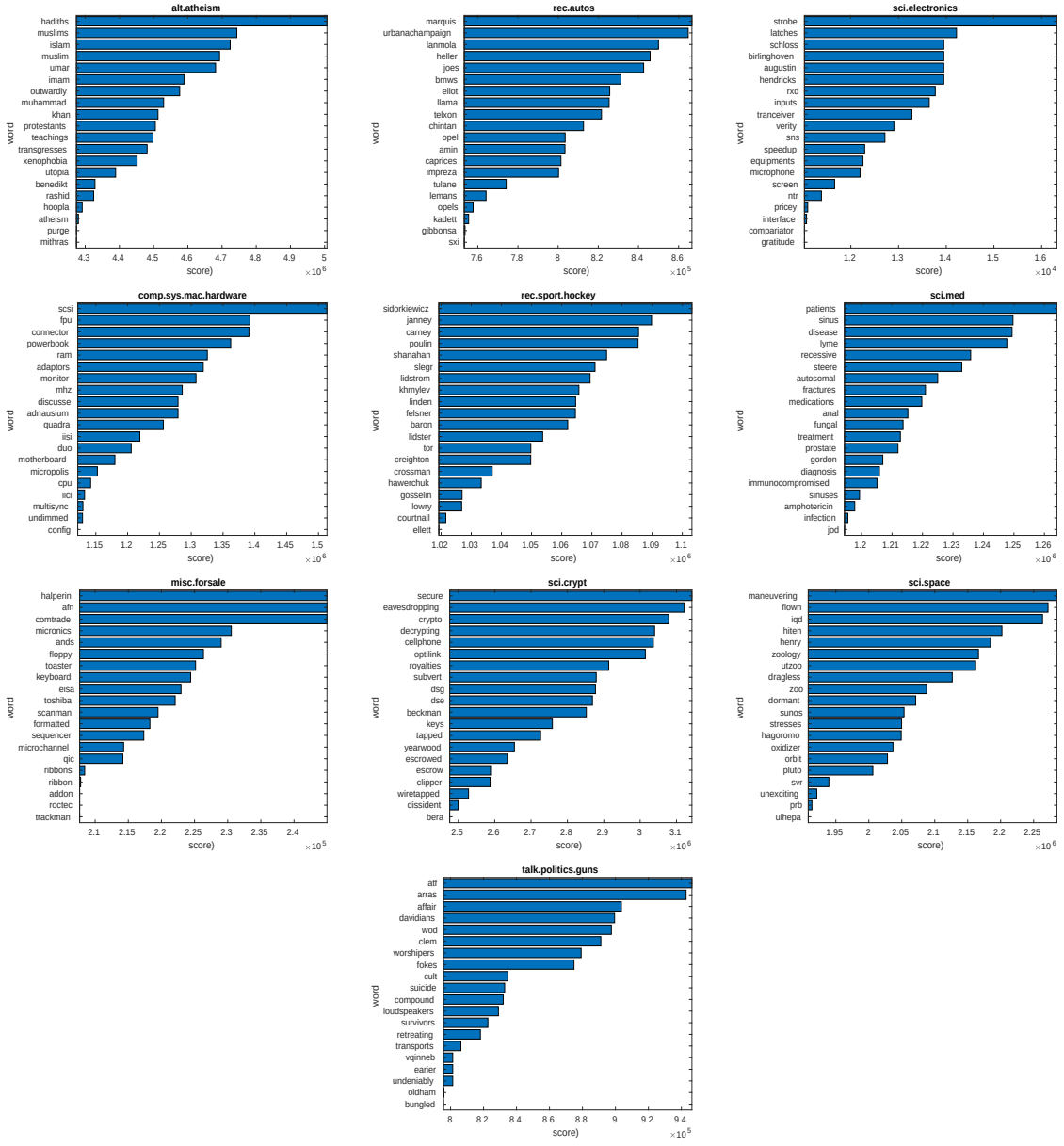


Figure 45: Top 20 words ranked by sum of word distances to cluster.

Note that the distance to centroid is still high, but not as close as the first methods. In fact, many words rank in the 1000th via their distance to centroids.

6.3.3 Kernel density estimation

Recall the problem with centroid distance: if there is noise and if there are diversity in the cluster, then centroid can not capture all the representative words. If we consider each cluster as a probability distribution, then using centroid only makes sense when the distribution is unimodal, having high concentration in only 1 region. When the space is multimodal, having high concentration in separate regions, we need to capture the density of all these regions to better understand the diversity of the topic. Thus we propose to estimate the density documents, then give word the same score as the density of that region. We will use kernel density estimation algorithm [11] with the Gaussian kernel for this purpose.


alt.atheism
hadiths
muslims
islam
muslim
umar
imam
outwardly
muhammad
khan
protestants
teachings
transgresses
xenophobia
utopia
benedikt
rashid
hoopla
atheism
purge
mithras
word
4.3 4.4 4.5 4.6 4.7 4.8 4.9 5
score)
×10^6
rec.autos
marquis
urbanachampaign
lanmola
heller
joes
bmws
eliot
llama
telxon
chintan
opel
amin
caprices
impreza
tulane
lemans
opels
kadett
gibbonsa
sxi
word
7.6 7.8 8 8.2 8.4 8.6
score)
×10^5
sci.electronics
strobe
latches
schloss
birlinghoven
augustin
hendricks
rxd
inputs
tranceiver
verity
srns
speedup
equipments
microphone
screen
ntr
pricey
interface
compariator
gratitude
word
1.2 1.3 1.4 1.5 1.6
score)
×10^4
comp.sys.mac.hardware
scsi
fpu
connector
powerbook
ram
adaptors
monitor
mhz
discusse
adnausium
quadra
iisi
duo
motherboard
micropolis
cpu
iici
multisync
undimmed
config
word
1.15 1.2 1.25 1.3 1.35 1.4 1.45 1.5
score)
×10^6
rec.sport.hockey
sidorkiewicz
janney
carney
poulin
shanahan
slegr
lidstrom
khmylev
linden
felsner
baron
lidster
tor
creighton
crossman
hawerchuk
gosselin
lowry
courtnall
ellett
word
1.02 1.03 1.04 1.05 1.06 1.07 1.08 1.09 1.1
score)
×10^6
sci.med
patients
sinus
disease
lyme
recessive
steere
autosomal
fractures
medications
anal
fungal
treatment
prostate
gordon
diagnosis
immunocompromised
sinuses
amphotericin
infection
jod
word
1.2 1.21 1.22 1.23 1.24 1.25 1.26
score)
×10^6
misc.forsale
halperin
afn
comtrade
micronics
ands
floppy
toaster
keyboard
eisa
toshiba
scanman
formatted
sequencer
microchannel
qic
ribbons
ribbon
addon
roctec
trackman
word
2.1 2.15 2.2 2.25 2.3 2.35 2.4
score)
×10^5
sci.crypt
secure
eavesdropping
crypto
decrypting
cellphone
optilink
royalties
subvert
dsg
dse
beckman
keys
tapped
yearwood
escrowed
escrow
clipper
wiretapped
dissident
bera
word
2.5 2.6 2.7 2.8 2.9 3 3.1
score)
×10^6
sci.space
maneuvering
flown
iqd
hiten
henry
zoology
utzoo
dragless
zoo
dormant
sunos
stresses
hagoromo
oxidizer
orbit
pluto
svr
unexciting
prb
uihepa
word
1.95 2 2.05 2.1 2.15 2.2 2.25
score)
×10^6
talk.politics.guns
atf
arras
affair
davidians
wod
clem
worshipers
fokes
cult
suicide
compound
loudspeakers
survivors
retreating
transports
vqinneb
earier
undeniably
oldham
bungled
word
8 8.2 8.4 8.6 8.8 9 9.2 9.4
score)
×10^5

7 Cluster Interpretation

After getting the clusters of text data, the most important words of each cluster can interpret the meaning of it. Cluster interpretation gives another way to evaluate the clustering results and helps us understand the final results. TF-IDF (term frequency-inverse document frequency) is a statistic that reflects the importance of each word in a document. There are two components to this statistic: TF(t) = (the number of times term t appears in a document)/(the total number of terms in the document) and IDF(t) = log(the total number of documents/the number of documents that contain term t). The final TF-IDF score is the product of these quantities.

Common words such as "the, a, and" are called stop words. Stop words have high frequencies within each document and in the whole data set, so they have low TF-IDF values. A representative word has a higher frequency within a document and lower frequency in the whole data set. The two ways of cluster interpretation are based on the data after TF-IDF transformation.

7.1 Sum of TF-IDF values

This method involves ranking the sum of TF-IDF values in each cluster. The most significant words of each cluster have high total TF-IDF values, and the stop words and less representative words have low total TF-IDF values. The figure below shows the top 15 TF-IDF scores for 12 clusters of the 20Newsgroups dataset after applying the LSC algorithm.

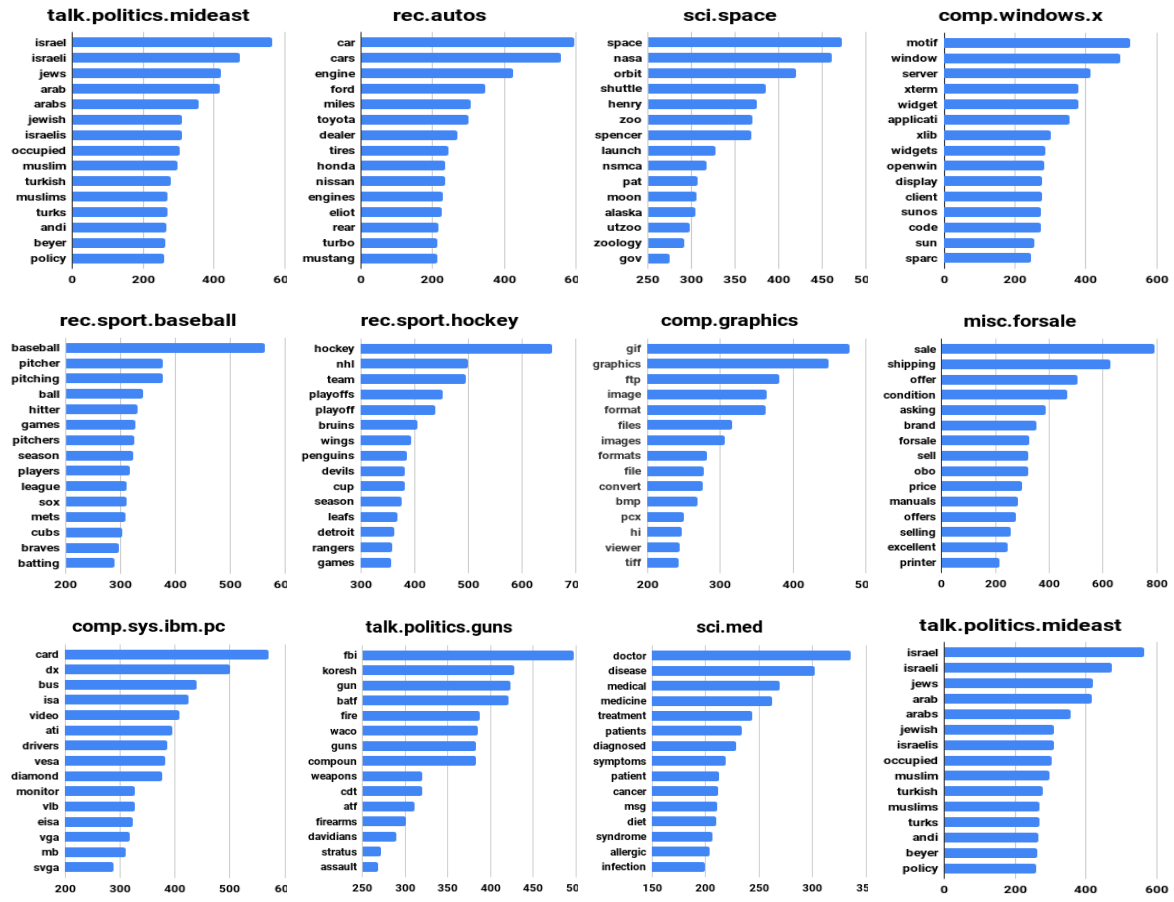


Figure 46: Top 15 words for 12 groups in 20Newsgroups by ranking frequencies based on results of the LSC method with 72.34% accuracy.

7.2 Singular Value Decomposition

Another method is to perform SVD on a subset (the collection of documents belonging to one cluster) of the dataset after TF-IDF weighting. The first right singular vector of SVD for this cluster is the first principal component of this cluster as well. The entries of this vector indicate how close the corresponding words (which are the dimensions) are to the principal direction of the cluster. Thus, one can rank the words based on the magnitude of the entries (larger means closer) and look at the top-ranked words to determine the topic of the cluster. This method produces similar results with ranking sum of TF-IDF values, but it is slower because of the SVD processing step. The results, shown in the figure below, are similar to the TF-IDF sum, but the sets of words differ by a word or two.

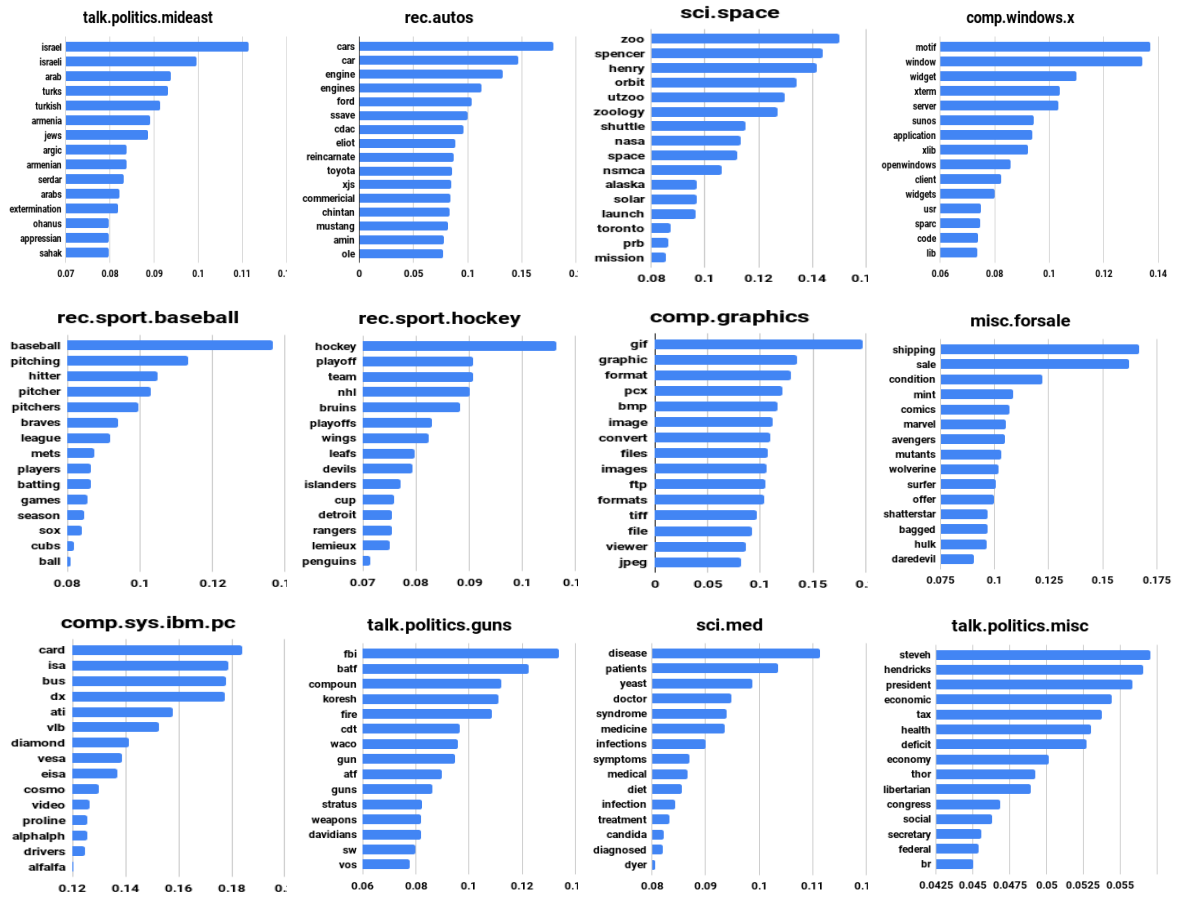


Figure 47: Top 15 words of 12 groups in 20Newsgroups by raking coefficients in the basis vector based on results of the LSC method with 72.34% accuracy.

8 Conclusion and Future Work

We worked on three ideas for scalable spectral clustering methods. They are often faster and more accurate than older spectral clustering algorithms. A selection of results for each method are compared in table below. In terms of each method, we see that the landmark and bipartite methods yield more accurate results, and the Scalable NJW using cosine similarities is more computationally efficient.

Table 11: Comparison between the three methods

	1. Cosine		**2. Landmark**		**3. Bipartite**	
Dataset	**Accuracy**	**Time**	**Accuracy**	**Time**	**Accuracy**	**Time**
USPS	67.5	(1.1)	**74.7**	(11.8)	69.5	(9.4)
Pendigits	73.6	(3.4)	**81.6**	(3.6)	74.7	(6.2)
MNIST	52.6	(36.2)	69.4	(584.1)	**73.3**	(316.4)
TDT2	51.2	(25.3)	64.3	(11.7)	**70.8**	(38.1)
Reuters	24.6	(5.9)	27.5	(6.6)	**38.3**	(36.6)

8.1 More Evaluation Metrics

Overall accuracy is simple to compute and easy to interpret, but it may be a misleading metric with imbalanced data. This is because the accuracy computation will be biased towards majority class instances [6]. This type of data is very common in real world applications and many of our test datasets are imbalanced. There are other evaluation metrics for multiclass problems that are worth considering such as the F-score or adjusted Rand index.

8.2 Recursive Partitioning

Communities are naturally composed of groups and subgroups. In the 20Newsgroup dataset, some groups can be combined into six main clusters, since some groups are more closely related than others. Recursive partitioning can find this natural structure. This idea involves finding k clusters. Then each cluster can be examined to see if it is reasonable to further subdivide the cluster.

This idea can also help determine how many clusters to form since groups can be divided until no further division makes sense. This criteria could be difficult to determine.

8.3 Demographic Data

Most clustering algorithms only work on numeric and continuous tabular data. However, data in the real world are mixed with categorical and continuous variables or have variables of different scales and units. For example, any browsing data can also contain information about the user such as gender, location, and income bracket. This information could be valuable to improve clustering insights but requires complex preprocessing steps or multiple models to cluster.

References

[1] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation*, 15(6):1373–1396, 2003.

[2] D. Cai and X. Chen. Large scale spectral clustering via landmark-based sparse representation. *IEEE Transactions on Cybernetics*, 45(8):1669–1680, Aug 2015.

[3] Kamalika Chaudhuri, Fan Chung, and Alexander Tsiatas. Spectral clustering of graphs with general degrees in the extended planted partition model. In *Conference on Learning Theory*, pages 35–1, 2012.

[4] Ronald R. Coifman and Stéphane Lafon. Diffusion maps. *Applied and Computational Harmonic Analysis*, 21:5 – 30, 2006.

[5] Inderjit S. Dhillon. Co-clustering documents and words using bipartite spectral graph partitioning. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, pages 269–274, New York, NY, USA, 2001. ACM.

[6] Sulaiman Hossin. A review on evaluation metrics for data classification evaluations. *International journal of data mining and knowledge management process*, 5(2), 2015.

[7] Brian Karrer and Mark EJ Newman. Stochastic blockmodels and community structure in networks. *Physical review E*, 83(1):016107, 2011.

[8] Ulrike Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, December 2007.

[9] Elchanan Mossel, Joe Neeman, and Allan Sly. Stochastic block models and reconstruction. *arXiv preprint arXiv:1202.1499*, 2012.

[10] Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*, NIPS'01, pages 849–856, Cambridge, MA, USA, 2001. MIT Press.

[11] Emanuel Parzen. On estimation of a probability density function and mode. *The annals of mathematical statistics*, 33(3):1065–1076, 1962.

[12] Khiem Pham and Guangliang Chen. Large-scale spectral clustering using diffusion coordinates on landmark-based bipartite graphs. In *Proceedings of the Twelfth Workshop on Graph-Based Methods for Natural Language Processing (TextGraphs-12)*, pages 28–37, 2018.

[13] Tai Qin and Karl Rohe. Regularized spectral clustering under the degree-corrected stochastic blockmodel. In *Advances in Neural Information Processing Systems*, pages 3120–3128, 2013.

[14] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(8):888–905, 2000.

[15] Liang Wang, Christopher Leckie, Kotagiri Ramamohanarao, and James Bezdek. *Approximate Spectral Clustering*, pages 134–146. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[16] Donghui Yan, Ling Huang, and Michael I. Jordan. Fast approximate spectral clustering. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pages 907–916, New York, NY, USA, 2009. ACM.

A Packages and Codes

A.1 R Packages

- Matrix
- RSpectra
- pdist
- clue
- R.matlab
- wordspace
- Matrix.utils

A.2 R Accuracy Computation

```
accuracy <- function(truelabels, clusters) {
  # Hungarian algorithm

  # Remove zeros from labels
  if (any(clusters == 0)) {
    print("Zero labels will be removed from accuracy computation")
    zeroindex <- clusters == 0
    truelabels <- truelabels[!zeroindex]
    clusters <- clusters[!zeroindex]
  }

  # Labels from cluster A will be matched on the labels from cluster
   ↪ B
  minWeightBipartiteMatching <- function(clusteringA, clusteringB) {
    require(clue)
    idsA <- unique(clusteringA)  # distinct cluster ids in a
    idsB <- unique(clusteringB)  # distinct cluster ids in b
    nA <- length(clusteringA)  # number of instances in a
    nB <- length(clusteringB)  # number of instances in b
    if (length(idsA) != length(idsB) || nA != nB) {
      stop("The number of clusters or lengths do not match")
    }
    nC <- length(idsA)
    tupel <- c(1:nA)
    # Computing the assignment matrix
    assignmentMatrix <- matrix(rep(-1, nC * nC), nrow = nC)
```

```
26      for (i in 1:nC) {
27        tupelClusterI <- tupel[clusteringA == i]
28        solRowI <- sapply(1:nC, function(i, clusterIDsB, tupelA_I) {
29          nA_I <- length(tupelA_I)  # number of elements in cluster I
30          tupelB_I <- tupel[clusterIDsB == i]
31          nB_I <- length(tupelB_I)
32          nTupelIntersect <- length(intersect(tupelA_I, tupelB_I))
33          return((nA_I - nTupelIntersect) + (nB_I - nTupelIntersect))
34        }, clusteringB, tupelClusterI)
35        assignmentMatrix[i,] <- solRowI
36      }
37      # Optimization
38      result <- solve_LSAP(assignmentMatrix, maximum = FALSE)
39      attr(result, "assignmentMatrix") <- assignmentMatrix
40      return(result)
41    }
42    test <- minWeightBipartiteMatching(clusters, truelabels)
43    predicted = NULL
44    predicted <- rep(NA, length(clusters))
45    for (i in 1:length(test)) {
46      predicted[which(clusters == i)] <- test[i]
47    }
48    table <- table(predicted, truelabels)
49    accuracy <- (sum(diag(table)) / length(truelabels))
50    classaccuracy <- vector()
51    colsums <- colSums(table)
52    for (i in 1:length(test)) {
53      classaccuracy[i] <- table[i, i] / colsums[i]
54    }
55    return(
56      list(
57        "accuracy" = accuracy,
58        "classaccuracy" = classaccuracy,
59        "table" = table,
60        "mapping" = test,
61        "mappedlabels" = predicted
62      )
63    )
64  }
```

A.3 Team 1 Code

A.3.1 Scalable Spectral Clustering

```
#Scalable spectral clustering with cosine similarity########
#Cluster data using cosine similarity and matrix computations.
#Scalable Ng Jordan Weiss algorithm.
#
#############################################################################

##function name
#scalable.cosine.spectral.clustering

##arguments
#the.data - the data in sparse matrix form. Each row is an observation.
#the.k - the number of clusters to be formed.
#the.algorithm - the spectral clustering algorithm to be performed.
  #-1 is NJW, 0 is Ncut, and positive integers are Diffusion Maps with
  #the corresponding time steps. Default is -1.
#cut.D.percent - the percent cutoff for data to be classified as outliers.
  #For example, a value of 0.05 results in 5% of the data to be classified as
  #outliers. Default is 0.01.
#tempSeed - the seed set for the k-means clustering. Default is 1000.
#kmeans.iter.max - See "iter.max" in the kmeans function. Default is 100.
#kmeans.nstart - See "nstart" in the kmeans function. Default is 10.

##output
#The cluster labels


##packages
#RSpectra. For the SVD step - take advantage of sparsity during large matrix SVD.
#Matrix. For handling sparse matrices.
#wordspace. For normalizing matrices efficiently with "normalize.rows"
```

```
31
32 ##function
33 scalable.cosine.spectral.clustering <- function(
34 the.data,
35 the.k,
36 the.algorithm = -1,
37 cut.D.percent = 0.01,
38 tempSeed = 1000,
39 kmeans.iter.max = 100,
40 kmeans.nstart = 10
41 ){
42 ########################################
43 ###########PACKAGES#####
44 #
45 library(RSpectra)
46 library(Matrix)
47 library(wordspace)
48 ########################################
49 #
50
51 ########################################
52 #BEGIN NJW FUNCTION
53 ########################################
54 #
55
56 #Step 0. L2 Normalize each row of data.
57 tempData <- normalize.rows(M=the.data, method = "euclidean")
58 originalData.indexlabel <- cbind( id = (1:nrow(the.data)), cluster =
   ↪  rep.int(0,nrow(the.data)) , the.data)
59 normalized.Data <- tempData
60
61 #Step 0.2 the.k is the number of clusters we want.
62 tempK <- the.k
63
64
65 #Step 1. Calculate D.
66 temp.row <- dim(tempData)[1]
67 temp.col <- dim(tempData)[2]
68 tempD.vector <- as.numeric(
```

```
 tempData %*% ( t(tempData) %*% rep.int(1, temp.row) ) -
  ↪ rep.int(1,temp.row)
 )

#Step 1.2. In D, remove the rows that contain the SMALLEST specified
 ↪ percent of D values.
#          Also remove the corresponding rows (data observations)
 ↪ from the data matrix.
#          These become the new D and new A.
#          STORE THE REMOVED RESULTS.

kept.index <- (1:length(tempD.vector))
removed.index <- NULL
if(cut.D.percent > 0){
temp.n.smallest <- floor(cut.D.percent * length(tempD.vector) )

smallest.Ds <- order(tempD.vector, decreasing =
 ↪ FALSE)[1:temp.n.smallest]

removed.index <- smallest.Ds
kept.index <- (1:length(tempD.vector))[-smallest.Ds]

tempD.vector <- tempD.vector[-smallest.Ds]
tempData <- tempData[-smallest.Ds,   ]
}


#Step 2. Use new D and new A to calculate A tilda.
tempD.vector <- 1/sqrt(tempD.vector)
tempA.tilda <- Diagonal(x = tempD.vector) %*% tempData


#Step 3. SVD on A tilda. Then normalize the U to get V.
svd.W <- svds(A = tempA.tilda,k = tempK, nu = tempK, nv = 0)
tempU <- svd.W$u

#Specifying NJW/NCut/Diffusion Map
current.algorithm <- NULL
if(the.algorithm == -1){
 current.algorithm <- "NJW"
```

```
}else if(the.algorithm == 0){
 current.algorithm <- "Normalized Cut"
 tempU <- Diagonal(x = tempD.vector) %*% tempU
}else if(the.algorithm > 0){
 current.algorithm <- paste("Diffusion Maps", "t =", the.algorithm,
  ↪ sep = " ")
 tempU <- Diagonal(x = tempD.vector) %*% tempU %*% Diagonal(x=
  ↪ (svd.W$d)^the.algorithm )
}

tempV <- normalize.rows(M=tempU, method = "euclidean")


#Step 4. Run the K means on the V.
set.seed(tempSeed)
tempKmeans <- kmeans(x = tempV, centers = tempK, nstart =
 ↪ kmeans.nstart, iter.max = kmeans.iter.max)
tempCluster <- tempKmeans$cluster

originalData.indexlabel[kept.index,2] <- tempCluster

return(originalData.indexlabel[ ,2])

}
```

A.3.2 Plain Spectral Clustering

```
#Plain Spectral Clustering Function###################
#Cluster data.
#Ng Jordan Weiss algorithm.
#
#########################################################################

##function name
#njw.spectral.clustering

##arguments
#the.data - the data in sparse matrix form. Each row is an
 ↪ observation.
#the.k - the number of clusters to be formed.
```

```
#cut.D.percent - the percent cutoff for data to be classified as outliers.
  #For example, a value of 0.05 results in 5% of the data to be classified as
  #outliers. Default is 0.01.
#seed - the seed set for the k-means clustering. Default is 1000.
#kmeans.iter.max - See "iter.max" in the kmeans function. Default is 100.
#kmeans.nstart - See "nstart" in the kmeans function. Default is 10.


##output
#The cluster labels


##packages
#RSpectra - For the Eigendecomposition step - take advantage of sparsity.
#Matrix - For handling sparse matrices.
#wordspace - For normalizing matrices efficiently with "normalize.rows"

##function
njw.spectral.clustering <- function(
the.data,
the.k,
cut.D.percent=0.01,
seed=1000,
kmeans.iter.max = 100,
kmeans.nstart = 10
){
#load packages
library(RSpectra)
library(Matrix)
library(wordspace)


tempData <- the.data
#keep track of outliers
```

```
originalData.indexlabel <- cbind( id = (1:nrow(the.data)), cluster = rep.int(0,nrow(the.data)) , the.data)

#Step 0. Normalize Data.
tempData <- normalize.rows(tempData, method = "euclidean")

#Step 1. Compute W = A %*% t(A). Use sparsity (Matrix package).
tempW <- tcrossprod(tempData)
#Set diagonal to 0
diag(tempW) <- 0

#Step 2.1. Compute D. Row sums of W with Matrix package.
tempD.vector <- rowSums( tempW )
#Step 2.2. Outlier removal.
kept.index <- 1:length(tempD.vector)
removed.index <- NULL

if(cut.D.percent > 0){
temp.n.smallest <- floor(cut.D.percent * length(tempD.vector) )
smallest.Ds <- order(tempD.vector, decreasing = FALSE)[1:temp.n.smallest]
removed.index <- smallest.Ds
kept.index <- (1:length(tempD.vector))[-smallest.Ds]
tempD.vector <- tempD.vector[-smallest.Ds]
tempData <- tempData[-smallest.Ds,   ]
###SUBSET THE W#####
#one other option is to subset the original data, and then recalculate W#
tempW <- tempW[-removed.index, -removed.index]
}

#Step 3. Compute W tilda. Sparse Matrix multiplication.
tempD.vector <- 1/sqrt(tempD.vector)
tempW.tilda <- Diagonal(x = tempD.vector)%*% tempW %*% Diagonal(x = tempD.vector)

#Step 4. Obtain U with eigendecomposition of W tilda. Use eigs() in RSpectra package.
tempU <- eigs(A = tempW.tilda, k = the.k)$vectors

```

```
#Step 5. Normalize U with wordspace package.
tempV <- normalize.rows(tempU, method = "euclidean")

#Step 6. Kmeans on V.
set.seed(seed)
tempKmeans <- kmeans(x = tempV, centers = the.k, nstart =
 ↪  kmeans.nstart, iter.max = kmeans.iter.max)
## nstart option that attempts multiple initial configurations and
 ↪  reports on the best one.
tempCluster <- tempKmeans$cluster


#cluster including label 0 for outliers#
originalData.indexlabel[kept.index,2] <- tempCluster


return(cluster = originalData.indexlabel[ ,2])

}
```

A.3.3 Cluster Interpretation with Rank 1 SVD

```
#Rank 1 SVD on data.###################
#Look at clusters to interpret important variables.
#
#
#############################################################################

##function name
#interpret.cluster

##arguments
#the.data - the observation data. Each row is an observation.
#the.label - the label of the group that an observation is in.
#the.variables - labels for the variables/columns. Default is NULL.
#top.var - the specified number of the most highly correlated
 ↪  variables in the rank 1 SVD. Default number is 10.

##outputs
#A data frame where each column is a group with its most highly
 ↪  correlated variables.
```

```
18
19 ##packages
20 #RSpectra - for the SVD step - take advantage of sparsity during
   ↪ large matrix SVD
21 #Matrix - for sparse matrices
22
23 ##function
24 interpret.cluster <- function(
25 the.data,
26 the.label,
27 the.variables = NULL,
28 top.var = 10
29 ){
30 #load packages
31 library(RSpectra)
32 library(Matrix)
33
34
35 the.label <- as.vector(the.label)
36 if(is.null(the.variables)){the.variables <- 1:ncol(the.data) }
37
38
39 top.var.list <- list()
40
41 for(i in min(the.label):max(the.label)){
42 subset.data <- the.data[ the.label == i,]
43 subset.pca <- abs(svds(A=subset.data,k=1, nu =0, nv = 1)$v)
44 variable.importance <- data.frame(the.variables, subset.pca)
45 the.order <- order(subset.pca, decreasing = TRUE)
46 the.order <- the.order[1:top.var]
47
48 if(min(the.label) == 0){
49 top.var.list[[i+1]] <- variable.importance[the.order,]
50 } else {
51 top.var.list[[i]] <- variable.importance[the.order,]
52 }
53 }
54
55 cluster.interpret <- data.frame(top.var.list)
56
```

```
return(top.var.list)
}
```

A.3.4 Classification of Outliers

```
#Nearest Centroid Outlier Classification########
#Takes the Scalable Spectral Clustering result labels and
#classify the outliers (0's) as one of the clusters by
#using the nearest centroid classifier.
#################################################################

##function name
#outlier.classify

##arguments
#the.data - the original data matrix
#the.cluster - output from scalable spectral clustering (including
 ↪  the 0's for outliers)

##outputs
#The cluster label, where the outliers have been classified into the
 ↪  clusters.

##packages
#Matrix - for sparse matrices
#Matrix.utils - for sparse matrices
#wordspace - for fast row normalization with "normalize.rows"

##function
outlier.classify <- function(the.data, the.cluster){
##############################################
library(Matrix) #for sparse matrices
library(Matrix.utils) #for sparse matrices
library(wordspace) #for fast row normalization with "normalize.rows"
##############################################


#sparse matrix
norm.data <- as(the.data, "sparseMatrix" )

#normalize the data
```

```
35 norm.data <- normalize.rows(norm.data)
36
37 #create training set with the non-outliers
38 #subset with outlier index
39
40 outlier.index <- which(the.cluster == 0)
41 if(length(outlier.index)==0) return(the.cluster)
42
43 trimmed.data <- norm.data[-outlier.index, ]
44 trimmed.cluster <- the.cluster[-outlier.index]
45
46 outlier.data <- norm.data[outlier.index, ]
47
48 #average each of the k clusters
49 the.k <- length(unique(trimmed.cluster))
50
51 #sum each cluster
52 cluster.col.sum <-aggregate.Matrix(
53 x=trimmed.data,
54 groupings = trimmed.cluster,
55 fun = "sum"
56 )
57 #divide by cluster sizes
58 cluster.counts <-as.numeric(table(trimmed.cluster))
59 training.set <- cluster.col.sum/cluster.counts
60
61
62 #K nearest neighbor with the centroids
63 training.label <- 1:length(unique(trimmed.cluster))
64 test.set <- Matrix(outlier.data, sparse = TRUE)
65
66 similarities <- test.set %*% t(training.set)
67
68 knn.cluster <-max.col(similarities)
69
70
71
72 #replace the 0 labels with nearest centroid labels
73 new.cluster.labels <- cbind(outlier.id = outlier.index, knn.label =
    ↪ knn.cluster)
```

```
entire.cluster <- the.cluster
entire.cluster[new.cluster.labels[,1]]<- new.cluster.labels[,2]


return(entire.cluster)
}
```

A.4 Team 2 Code

A.4.1 LSC Code

```
LSC <- function(data,                # input feature matrix
      k,                             # number of clusters
      p = NULL,                      # number of landmarks
      r = NULL,                      # number of nearest landmarks
      t = 0,                         # time-step parameter
      method = "random",             # landmark selection method
      similarity = "cosine",         # similarity measure
      clustering = "traditional", # clustering method
      seed = 0,                      # initial seed
      iter.max = 100,                # max iterations for k-means
      nstart = 10,                   # sets of initializations for
       ↪ k-means
      reclassify = TRUE,             # reclassify outliers or give zero
       ↪ label
      alpha1 = 0,                    # data outlier removal
      alpha2 = 0,                    # landmark  outlier removal
      beta = 0.6,                    # variance tuning parameter
      knn = 1) {                     # nearest neighbors used for knn

  # Set seed
  set.seed(seed)

  # Make sure sparse storage is used if applicable
  data <- Matrix(data)

  # Convert alpha1 and alpha2 from percentage to proportion
  alpha1 <- alpha1/100
  alpha2 <- alpha2/100


```

```r
  # Some rules of thumb to pick p and r
  if (is.null(p)) {
    p <- floor(sqrt(nrow(data) * k))
  }

  if (is.null(r)) {
    r <- floor(p / 10)
  }

  if (method == "random") {
    # Sampling p landmarks randomly
    pindex <- sample(nrow(data), p)
    pmatrix <- data[pindex, ]
    # Cosine similarity by default. Gaussian otherwise.
    if (similarity == "cosine") {
      # Compute affinity matrix with cosine similarity
      A <- (data / sqrt(rowSums(data ^ 2))) %*% t(pmatrix /
       ↪  sqrt(rowSums(pmatrix ^ 2)))
    } else {
      # Compute distance matrix for p landmarks for estimating sigma
      p_dist <- as.matrix(dist(pmatrix, method = "euclidean"))
      keepr.small <- function(x, m) {
        return(sort(x)[1:m + 1])
      }
      m <- 2
      p_dist <- t(apply(p_dist, 1, keepr.small, m = m)) / 2
      # Estimate sigma using mean distance between two closest
       ↪  landmarks
      g_sigma <- beta * mean(p_dist)
      # Calculate euclidean distance between points and landmarks
      A <- as.matrix(pdist(data, pmatrix))
      A <- 1 / exp((A ^ 2) / (2 * (g_sigma ^ 2)))
    }

  } else {
    # Sampling p landmarks with kmeans
    if (similarity == "cosine") {
      # Suppress warnings because failure to converge is irrelevant
      suppressWarnings(pmatrix <- kmeans(data / sqrt(rowSums(data ^
       ↪  2)), centers = p)$centers)
```

```
66      } else {
67        suppressWarnings(pmatrix <- kmeans(data, centers = p)$centers)
68      }
69      if (similarity == "cosine") {
70        # Compute affinity matrix with cosine similarity
71        A <- (data / sqrt(rowSums(data ^ 2))) %*% t(pmatrix /
          ↪  sqrt(rowSums(pmatrix ^ 2)))
72      } else {
73        # Compute affinity matrix with gaussian similarity
74        pfit <- kmeans(data, centers = p)
75        pmatrix <- pfit$centers
76        # Use total within sum of squares to estimate sigma
77        g_sigma <- beta * sqrt(pfit$tot.withinss / (nrow(data) - p))
78        A <- as.matrix(pdist(data, pmatrix))
79        A <- 1 / exp((A ^ 2) / (2 * (g_sigma ^ 2)))
80      }
81    }
82
83    # Landmark outlier removal. Outliers defined as having small column
      ↪  sums.
84    if ((alpha2 * p) >= 1) {
85      colsums <- colSums(A)
86      landmark_quantile <- quantile(colsums, alpha2)
87      A <- A[, colsums >= landmark_quantile]
88      if (method == "random") {
89        pindex <- pindex[colsums >= landmark_quantile]
90      }
91    }
92
93    # Check for rowSums == 0. Give 0 label if there are any.
94    zerooutliers <- FALSE
95    if (any(rowSums(A) == 0)) {
96      zerooutliers <- TRUE
97      zoutliers <- rowSums(A) == 0
98      znum <- sum(zoutliers)
99      warning(znum, " row(s) of zeros in affinity matrix. Will give
        ↪  zero label.")
100     A <- A[!zoutliers,]
101   }
102
```

```
103   # Data outlier removal. Outliers defined as having small row sums.
104   outliers <- FALSE
105   if ((alpha1 * nrow(A)) >= 1) {
106     outliers <- TRUE
107     rowsums <- rowSums(A)
108     obs_quantile <- quantile(rowsums, alpha1)
109     outlier_index <- rowsums < obs_quantile
110
111     if (method == "random" & clustering != "traditional") {
112       # If random sampling and landmark clustering, outliers can't be
          ↪ landmarks
113       outlier_index[which(outlier_index)[which(outlier_index) %in%
          ↪ pindex]] <- FALSE
114       # Shift pindex for splitting into test and training sets later
115       porder <- order(order(pindex))
116       pindex <- which((1:length(outlier_index) %in%
          ↪ pindex)[!outlier_index])
117       pindex <- pindex[porder]
118     }
119
120     outlier_similarity <- A[outlier_index, ]
121     A <- A[!outlier_index,]
122
123     # Can give outliers zero label or reclassify with knn
124     if (reclassify == TRUE) {
125       knnindex <- function(x, length, knn) {
126         if (knn == 1) {
127           return(which.max(x))
128         } else {
129           return(order(x, decreasing = TRUE)[1:knn])
130         }
131       }
132       # Construct nearest neighbor matrix for classification of
          ↪ outliers
133       oknnmatrix <- Matrix(t(apply(outlier_similarity, 1, knnindex,
          ↪ length = ncol(A), knn = knn)))
134     }
135   }
136
137   if (clustering == "traditional") {
```

```
138       keepr <- function(x, length, r) {
139         # Keep r largest entries in each row
140         x[order(x)[1:(length - r)]] <- 0L
141         return(x)
142       }
143       # Keep r nearest landmark distances
144       A <- Matrix(t(apply(A, 1, keepr, length = ncol(A), r = r)))
145     } else {
146       keeprknn <- function(x, length, r, knn) {
147         # Keep r largest entries in each row
148         sorted <- order(x)
149         x[sorted[1:(length - r)]] <- 0L
150         return(c(sorted[length - ((knn - 1):0)], x))
151       }
152       A <- Matrix(t(apply(A, 1, keeprknn, length = ncol(A), r = r, knn
          ↪  = knn)))
153       knnmatrix <- A[, 1:knn]
154       A <- A[, -(1:knn)]
155     }
156
157     # Remove columns with colSums == 0
158     if (any(colSums(A) == 0)) {
159       colzeroindex <- colSums(A) != 0
160       A <- A[, colzeroindex]
161       if (method == "random") {
162         pindex <- pindex[colzeroindex]
163       }
164     }
165
166     if (clustering == "traditional") {
167       if (t > 0) {
168         A1  <- A / rowSums(A)  # Calculate A1
169         A2 <- t(t(A1) / sqrt(rowSums(t(A1))))  # Calculate A2
170         svdresult <- svds(A2, k = k, nu = k, nv = 0)
171         sigma <- svdresult$d
172         U <- svdresult$u
173         U <- t(t(U) * sigma^t)
174         U <- U / sqrt(rowSums(U ^ 2))
175         labelsout <- kmeans(U, centers = k, iter.max = iter.max, nstart
            ↪  = nstart)$cluster
```

```
176     } else {
177       A1  <- A / rowSums(A)  # Calculate A1
178       A2 <- t(t(A1) / sqrt(rowSums(t(A1))))  # Calculate A2
179       # Take top k left singular vectors of A2
180       U <- svds(A2, k = k, nu = k, nv = 0)$u
181       # L2 normalize rows of U
182       U <- U / sqrt(rowSums(U ^ 2))
183       # Kmeans on U
184       labelsout <- kmeans(U, centers = k, iter.max = iter.max, nstart
          ↪  = nstart)$cluster
185     }
186   } else {
187     A1  <- t(t(A) / (rowSums(t(A))))  # Calculate A1
188     A2  <- A1 / sqrt(rowSums(A1))  # Calculate A2
189     # Take top k right singular vectors of A2
190     V <- svds(A2, k = k, nu = 0, nv = k)$v
191     # L2 normalize rows of V
192     V <- V / sqrt(rowSums(V ^ 2))
193     # Kmeans on V
194     landmarkfit <- kmeans(V, centers = k, iter.max = iter.max, nstart
        ↪  = nstart)$cluster
195
196     if (method == "random") {
197       #Split test and train sets
198       fullindex <- 1:nrow(A2)
199       trainindex <- pindex
200       testindex <- fullindex[-trainindex]
201       if (knn == 1) {
202         knnmatrix <- knnmatrix[-trainindex]
203       } else {
204         knnmatrix <- knnmatrix[-trainindex, ]
205       }
206       votematrix <- matrix(landmarkfit[as.vector(knnmatrix)], nrow =
          ↪  length(testindex), ncol = knn)
207       #Do knn on data
208       if (knn == 1) {
209         knnresults <- votematrix
210       } else {
211         getmode <- function(x) {
212           tab <- table(x)
```

```
213           max <- names(which(tab == max(tab)))
214           if (length(max) > 1) {
215             max <- sample(max, 1)
216           }
217           return(as.integer(max))
218         }
219         knnresults <- apply(votematrix, 1, getmode)
220       }
221       # Recombine labels
222       predictedlabels <- rep(NA, length.out = nrow(A2))
223       predictedlabels[testindex] <- knnresults
224       predictedlabels[trainindex] <- landmarkfit
225       labelsout <- predictedlabels
226     } else {
227       votematrix <- matrix(landmarkfit[as.vector(knnmatrix)], nrow =
          ↪ nrow(A), ncol = knn)
228       if (knn == 1) {
229         labelsout <- votematrix
230       } else {
231         getmode <- function(x) {
232           tab <- table(x)
233           max <- names(which(tab == max(tab)))
234           if (length(max) > 1) {
235             max <- sample(max, 1)
236           }
237           return(as.integer(max))
238         }
239         labelsout <- apply(votematrix, 1, getmode)
240       }
241     }
242   }
243
244   # Reclassify outliers or give zero label
245   if (outliers == TRUE) {
246     if (reclassify == TRUE) {
247       ovotematrix <- matrix(labelsout[as.vector(oknnmatrix)], nrow =
          ↪ nrow(oknnmatrix), ncol = knn)
248       if (knn == 1) {
249         predicted.o.label <- ovotematrix
250       } else {
```

```
251           getmode <- function(x) {
252             tab <- table(x)
253             max <- names(which(tab == max(tab)))
254             if (length(max) > 1) {
255               max <- sample(max, 1)
256             }
257             return(as.integer(max))
258           }
259           predicted.o.label <- apply(ovotematrix, 1, getmode)
260         }
261         finallabels <- rep(NA, length.out = length(rowsums))
262         finallabels[outlier_index] <- predicted.o.label
263         finallabels[!outlier_index] <- labelsout
264         labelsout <- finallabels
265       } else {
266         finallabels <- rep(NA, length.out = length(rowsums))
267         finallabels[outlier_index] <- 0
268         finallabels[!outlier_index] <- labelsout
269         labelsout <- finallabels
270       }
271     }
272
273     if (zerooutliers == TRUE) {
274       finallabels <- rep(NA, length.out = nrow(data))
275       finallabels[zoutliers] <- 0
276       finallabels[!zoutliers] <- labelsout
277       labelsout <- finallabels
278     }
279
280     # Check for missing clusters. Can only occur with kmeans landmark
        ↪  selection and
281     # landmark clustering.
282     if (length(unique(labelsout[labelsout != 0])) != k) {
283       warning("The final number of clusters is not equal to k")
284     }
285
286     return(as.vector(labelsout))
287   }
```

A.5 Team 3 Code

A.5.1 LBDM Code

```
function [label, kept_idx, U, reps] = LBDM(fea, k, r, s, t, affinity, varargin)
%LARGE SCALE SPECTRAL CLUSTERING USING DIFFUSION COORDINATE ON
%LANDMARK-BASED BIPARTITE GRAPH

%REQUIRED:
%fea: the data in row-major order (i.e each datapoint is a row)
%k: desired number of clusters
%affinity: currently supports cosine and radial basis function (gaussian)
%r: number of representatives (d. 100)
%s: number of nearest landmarks to keep
%t: diffusion time step
%affinity:
                %'gaussian':
                %'cosine': will normalize features first

%PARAMETER:
%remove_outlier: remove a subset of outliers based on point's total
%distances to other points
%select_method:
                %'random': pick landmarks uniformly random
                %'++': pick landmarks using kmeans++ weighting
                %'kmeans': pick landmarks as centers of a kmeans run
%embed_method:
                %'landmark': use right singular vector
                %'direct': use left singular vector
                %'coclustering': use both
%cluster_method: algorithms to partition embeddings
                %'kmeans':
                %'discretize:
% initRes/finalRes: number of restarts for initial/final Kmeans (d. 1/10)
% initIter/finalIter: number of maximum iterations for initial/final

%OPTS
%sigma: scaling factor for gaussian kernel. Default is computed as
```

```
35 %mean(mean(distance_matrix))
36
37 [n,m] = size(fea);
38
39 if (~exist('opts','var'))
40    opts = [];
41 end
42
43 %% input parser
44 p = inputParser;
45 addParameter(p,'initIter',10);
46 addParameter(p,'initRes',1);
47 addParameter(p,'finalIter',100);
48 addParameter(p,'finalRes',10);
49 addParameter(p,'select_method','kmeans');
50 addParameter(p,'embed_method','landmark');
51 addParameter(p,'cluster_method','kmeans');
52 addParameter(p,'remove_outlier',[1,1]);
53 addParameter(p,'fileid',1);
54 parse(p,varargin{:});
55
56 initIter = p.Results.initIter;
57 initRes = p.Results.initRes;
58 finalIter = p.Results.finalIter;
59 finalRes = p.Results.finalRes;
60 select_method = p.Results.select_method;
61 embed_method = p.Results.embed_method;
62 cluster_method = p.Results.cluster_method;
63 remove_outlier = p.Results.remove_outlier;
64 fileid = p.Results.fileid;
65
66
67 %% affinity
68
69 %1 cosine
70
71 if strcmp(affinity, 'cosine')
72     fprintf(fileid,'using cosine affinity\n');
73     fprintf(fileid,'normalizing features...\n');
74     fea = fea ./ sqrt(sum(fea.^2, 2));
```

```
75
76      %remove outlier
77      if ((0 < remove_outlier(1)) && (remove_outlier(1) < 1)) || ((0 <
        ↪  remove_outlier(2)) && (remove_outlier(2) < 1))
78          tic;
79          fprintf(fileid,'removing outliers by finding row-sums...\n');
80          d = fea * (fea' * ones(n,1));
81          [val, amax] = sort(d, 'descend');
82          remove_high = remove_outlier(1);
83          remove_low = remove_outlier(2);
84          if (0 < remove_high) && (remove_high < 1)
85              high = floor(n * remove_high);
86          else
87              high = 1;
88          end
89          if (0 < remove_low) && (remove_low < 1)
90              low = n - ceil(n * remove_low);
91          else
92              low = n;
93          end
94          kept_idx = amax(high:low);
95          no_kept = size(kept_idx,1);
96          plot(1:n,val);
97          fprintf(fileid,'remove %.0f%% of dataset...\n',(1 - no_kept /
            ↪  n) *100);
98          fprintf(fileid,'removing outliers done in %.2f seconds\n',
            ↪  toc);
99      else
100         no_kept = n;
101         kept_idx = 1:n;
102
103     end
104
105     %select landmarks
106     fprintf(fileid,'selecting landmarks using ');
107     tic;
108     if strcmp(select_method, 'kmeans')
109         fprintf(fileid,'kmeans...\n');
110         [lb, reps] = litekmeans(fea(kept_idx,:), r, 'Distance',
            ↪  'cosine', 'MaxIter', initIter, 'Replicates', initRes,...
```

```
111             'Start', 'cluster', 'clustermaxrestart', 10,
                ↪  'clustermaxiter', 100, 'clustersample', 0.1);
112         lbcount = hist(lb, 1:r); %#ok<NASGU>
113     elseif strcmp(select_method, 'uniform')
114         fprintf(fileid,'random sampling...\n');
115         reps = fea(kept_idx(randsample(no_kept, r, false)), :);
116     else
117         error('unsupported mode');
118     end
119
120     W = fea * reps';
121     fprintf(fileid,'done in %.2f seconds\n', toc);
122
123     %construct A
124     fprintf(fileid,'constructing sparse A...\n');
125     tic;
126     if s > 0
127         dump = zeros(n,s);
128         idx = dump;
129         for i = 1:s
130             [dump(:,i),idx(:,i)] = max(W,[],2);
131             temp = (idx(:,i)-1)*n+(1:n)';
132             W(temp) = 1e-100;
133         end
134         Gidx = repmat((1:n)',1,s);
135         Gjdx = idx;
136         W = sparse(Gidx(:),Gjdx(:),dump(:),n,r);
137         %remove outliers
138         W = W(kept_idx,:);
139     fprintf(fileid,'done in %.2f seconds\n',toc);
140     elseif s <= 0 % default to dense matrix
141         fprintf(fileid,'Default to dense martix\n');
142         if strcmp(embed_method, 'landmark')
143             [~,idx] = min(W,[],2);
144             %remove outliers
145             W = W(kept_idx,:);
146         end
147     end
148
149 %2 gaussian
```

```
150 elseif strcmp(affinity, 'gaussian')
151     fprintf(fileid,'using gaussian affinity\n');
152     %remove outlier
153     if ((0 < remove_outlier(1)) && (remove_outlier(1) < 1)) || ((0 <
        ↪  remove_outlier(2)) && (remove_outlier(2) < 1))
154         tic;
155         fprintf(fileid,'removing outliers by finding row-sums...\n');
156         An = sum(fea.^2, 2);
157         Aminus = An - mean(An); %to avoid overflow
158         AA = Aminus * n;
159         AB = fea * (fea' * ones(n,1));
160         d = AA  - 2 * AB;
161         [val, amax] = sort(d, 'descend');
162         remove_high = remove_outlier(1);
163         remove_low = remove_outlier(2);
164         if (0 < remove_high) && (remove_high < 1)
165             high = floor(n * remove_high);
166         else
167             high = 1;
168         end
169         if (0 < remove_low) && (remove_low < 1)
170             low = n - ceil(n * remove_low);
171         else
172             low = n;
173         end
174         kept_idx = amax(high:low);
175         no_kept = size(kept_idx,1);
176         plot(1:n,val);
177         fprintf(fileid,'remove %.0f%% of dataset...\n',(no_kept / n)
            ↪  *100);
178         fprintf(fileid,'removing outliers done in %.2f seconds\n',
            ↪  toc);
179     else
180         no_kept = n;
181         kept_idx = 1:n;
182     end
183
184     %select landmarks
185     fprintf(fileid,'selecting landmarks using ');
186     tic;
```

```
187     if strcmp(select_method, 'kmeans')
188         fprintf(fileid,'kmeans...\n');
189 %         warning('off', 'stats:kmeans:FailedToConverge')
190         [lb, reps, ~, VAR] = litekmeans(fea(kept_idx,:), r,
            ↪ 'MaxIter', initIter, 'Replicates', initRes,...
191             'Start', 'cluster', 'clustermaxrestart', 10,
                ↪ 'clustermaxiter', 100, 'clustersample', 0.1);
192         lbcount = hist(lb, 1:r);
193 
194     elseif strcmp(select_method, 'uniform')
195         fprintf(fileid,'random sampling\n');
196         reps = fea(kept_idx(randsample(no_kept, r, false)), :);
197 
198     elseif strcmp(select_method, '++')
199         fprintf(fileid,'D2 weight sampling\n');
200         [~, reps] = kmeans(fea_kept, r, 'MaxIter',0,'Replicates',1);
201     else
202         error('unsupported mode');
203     end
204     fprintf(fileid,'done in %.2f seconds\n',toc);
205     W = EuDist2(fea, reps, 0);
206 
207     %determine sigma
208     if isfield(opts, 'sigma')
209         sigma = opts.sigma;
210     elseif strcmp(select_method, 'kmeans')
211         sigma = mean(sqrt(VAR ./ lbcount));
212 
213     elseif strcmp(select_method, 'random') || strcmp(select_method,
        ↪ '++')
214         error('method random and ++ require sigma')
215     end
216 
217     fprintf(fileid,'using sigma = %.2f\n',sigma);
218 
219     %sparse representation
220     fprintf(fileid,'constructing sparse A...\n');
221     tic;
222     if s > 0
223         dump = zeros(n,s);
```

```
224         idx = dump;
225         for i = 1:s
226             [dump(:,i),idx(:,i)] = min(W,[],2);
227             temp = (idx(:,i)-1)*n+(1:n)';
228             W(temp) = 1e100;
229         end
230 
231         Gidx = repmat((1:n)',1,s);
232         Gjdx = idx;
233         W = sparse(Gidx(:),Gjdx(:),dump(:),n,r);
234         %remove outlier
235         W = W(kept_idx,:);
236 
237     elseif s <= 0 % default to dense matrix
238         fprintf(fileid,'default to dense matrix\n');
239         if strcmp(embed_method, 'landmark')
240             [~,idx] = min(W,[],2);
241         end
242         W = exp(-W/(2.0*sigma^2));
243         %remove outlier
244         W = W(kept_idx,:);
245     end
246     fprintf(fileid,'done in %.2f seconds\n',toc);
247 end
248 
249 %% compute laplacian
250 
251 fprintf(fileid,'Computing Laplacian and diffusion map...\n');
252 tic;
253 
254 d1 = sum(W, 2);
255 d2 = sum(W, 1);
256 d1 = max(d1, 1e-15);
257 d2 = max(d2, 1e-15);
258 D1 = sparse(1:no_kept,1:no_kept,d1.^(-0.5));
259 D2 = sparse(1:r,1:r,d2.^(-0.5));
260 L = D1*W*D2;
261 [U,S,V] = svds(L, k);
262 
263 if t > 0
```

```
264     U = D1 * U * S.^t;
265     V = D2 * V * S.^t;
266 elseif t == 0
267     U = D1 * U;
268     V = D2 * V;
269 end
270 fprintf(fileid,'Done in %.2f seconds\n',toc);
271
272 %% cluster embeddings
273
274 fprintf(fileid,'Clustering result embeddings...\n');
275 tic;
276 if strcmp(embed_method, 'landmark')
277     if strcmp(cluster_method, 'kmeans')
278         V(:,1) = [];
279         V = V ./ sqrt(sum(V .^2, 2));
280         reps_labels = litekmeans(V, k, 'Distance', 'cosine',
              ↪ 'MaxIter', finalIter, 'Replicates',finalRes);
281     elseif strcmp(cluster_method, 'discretize')
282         reps_labels = discretize(V);
283     end
284     label = zeros(n, 1);
285     for i = 1:n
286         label(i) = reps_labels(idx(i));
287     end
288
289 elseif strcmp(embed_method, 'direct')
290     if strcmp(cluster_method, 'kmeans')
291         U(:,1) = [];
292         U = U ./ sqrt(sum(U .^2, 2));
293         label = litekmeans(U, k, 'Distance', 'cosine', 'MaxIter',
              ↪ finalIter, 'Replicates',finalRes);
294     elseif strcmp(cluster_method, 'discretize')
295         label = discretize(U);
296     end
297
298 elseif strcmp(embed_method, 'coclustering')
299     W = [U;V];
300     if strcmp(cluster_method, 'kmeans')
301         W(:,1) = [];
```

```
302         W = W ./ sqrt(sum(W .^2, 2));
303         all_label = litekmeans(W, k, 'Distance', 'cosine', 'MaxIter',
             ↪ finalIter, 'Replicates',finalRes);
304     elseif strcmp(cluster_method, 'discretize')
305         all_label = discretize(W);
306     end
307     label = all_label(1:n);
308 end
309 fprintf(fileid,'Done in %.2f seconds\n', toc);
310
311 end
```