Spring 2023

On The Memory Scalability of Spectral Clustering Algorithms

Ran Li
San Jose State University

On The Memory Scalability of Spectral Clustering Algorithms

A Project

Presented to

The Faculty of the Department of Mathematics and Statistics

San José State University

In Partial Fulfillment

of the Requirements for the Degree of

Master of Science

by

Ran Li

May 2023

The Designated Project Committee Approves the Project Titled

On The Memory Scalability of Spectral Clustering Algorithms

by

Ran Li

APPROVED FOR THE DEPARTMENT OF MATHEMATICS AND STATISTICS

SAN JOSÉ STATE UNIVERSITY

May 2023

Dr. Guangliang Chen	Department of Mathematics and Statistics
Dr. Teng Moh	Department of Computer Science
Dr. Tahir Bachar Issa	Department of Mathematics and Statistics

ABSTRACT

On The Memory Scalability of Spectral Clustering Algorithms

by Ran Li

Spectral clustering has lots of advantages compared to previous more traditional clustering methods, such as k-means and Gaussian Mixture Models (GMM), and is popular since it was introduced. However, there are two major challenges, speed scalability and memory scalability, that impede the wide applications of spectral clustering. The first challenge has been addressed recently by Chen [1] [2] in the special setting of sparse or low dimensional data sets. In this work, we will first review the recent study by Chen that speeds up spectral clustering. Then we will propose three new computational methods for the same special setting of sparse or low dimensional data to address the memory challenge when the data sets are too large to be fully loaded into computer memory and when the data sets are collected sequentially. Numerical experiment results will be presented to demonstrate the improvements from these methods. Based on the experiments, the proposed methods show effective results on both simulated and real-world data.

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to Dr. Chen for his invaluable guidance, patience, and feedback. This project would not be possible without his support. Our weekly meeting is one of the most memorable moments during my time at SJSU. Receiving his instructions and working with him give me ideas on how to approach research problems. I also could not have undertaken this journey without the professors in my defense committee, Dr. Issac and Dr. Moh, who generously provided knowledge and expertise. I am grateful that Dr. Moh is my CS274 instructor. His advisory in my CS274 project has better prepared me for my research journey.

I am also grateful to my husband for his unconditional support and sacrifice during my study in the MS data science program. Thanks should also go to my classmates and friends who helped and inspired me.

Lastly, I would like to thank my family for their encouragement throughout my degree program.

TABLE OF CONTENTS

CHAPTER

CHAPTER 1

Background

1.1 Introduction

Clustering is an important technique to help explore the underlying patterns and relationships among data. It has been widely used in many fields, such as computer science and social science, that involve data processing [3]. Traditionally, k-means and Gaussian Mixtures have been the standard methods to perform clustering tasks. But they have their limitations. For example, Gaussian Mixtures need to assume a specific distribution for the data points in each cluster. Also, it can converge to a local minimum even with multiple restarts. And the k-means method imposes spherical shapes in clusters. Spectral clustering is a good alternative without these problems [3]. It was introduced in early 2000 by Shi and Malik [4], Meila and Shi [5], and Ng, Jordan, and Weiss [6]. Since then, it has become a popular method in many applications, such as image segmentation [4] and document clustering [2]. Although there are different variants, spectral clustering normally includes the following basic steps: forming a weight matrix, finding the eigenvectors, and clustering the data based on the eigenvectors [3].

Two practical challenges impede the wide applications of spectral clustering. The first one is computational cost, as it requires decomposing an n×n weight matrix, where n is the number of data points. When n is large, this will be a very expensive computation. Recently, many researchers have come up with different methods to reduce computational costs. Chen's research proposed a computational framework in spectral clustering with cosine similarity to speed up the computation by using low-rank SVD and avoiding the computation of the weight matrix [2] [7]. Later, he extended his work to other similarity functions by converting the raw data into sparse vectors using landmark points [1] [8] [9]. Thus, this challenge has been adequately

addressed.

The second challenge is memory access, which will be the focus of this project. The current spectral clustering research assumes that the full data can be read into the computer memory all at once to use the algorithm. However, with the fast development of the internet, today's data scientists face massive data sets. Oftentimes, we do not have enough computer memory to store the whole data set. And even when we do, reading and writing from computer memory can be computationally expensive. Additionally, these large data are usually collected and distributed in the on-the-fly format. Thus, handling the streaming data is another important issue we need to address.

Our research goal is to reduce the memory requirement of spectral clustering algorithms in special settings where the dataset is either sparse or with low dimensions. In this report, I will first briefly introduce the implementation of the original spectral clustering algorithms. After that, I will introduce a scalable spectral clustering method that speeds up the original algorithm under the special setting of sparse or low-dimensional data. Last but not least, we will introduce three computational methods for the same special setting to address the memory challenge when the data sets are too large to be fully loaded into computer memory. Experiments with real and simulated data are performed to validate the new approaches.

1.2 Spectral Clustering

In a clustering problem, provided a dataset $\mathbf{X} \in \mathbb{R}^{n \times d}$ and a similarity function, we would divide the dataset into disjoint subsets (called clusters) such that points are "similar" (at least to their near neighbors) within every cluster, and points are "dissimilar" between clusters. In spectral clustering, we can reformulate the clustering problem into a graph-cut problem by representing the dataset by an (undirected)

similarity graph $G = (V, E)$. Each vertex $v_i \in V$ represents a data point x_i. Adjacent ("similar") vertices $v_i, v_j \in V$ are connected by an edge, denoted as $e_{i,j} \in E$. The goal is to find a way to partition the graph into disjoint components so that within each component the sub-graph is dense and between the components the connections are sparse.

Spectral clustering normally includes the following basic steps, forming a weight matrix by a similarity graph, finding the graph Laplacian and embedded eigenvectors, and clustering based on the embedding.

1.2.1 Similarity Graph

Given a dataset $\mathbf{X} \in \mathbb{R}^{n \times d}$, we can construct an undirected similarity graph $G = (V, E)$ by a defined similarity function. The weight matrix $\mathbf{W}$ represents the weights of the edges between each vertex in the graph

$$\mathbf{W} = (w_{ij}) \in \mathbb{R}^{n \times n}, \text{ where } w_{ij} = w_{ji} \geq 0.$$

The degree of a vertex x_i is denoted by d_i. The degrees of all vertices are represented by a degree matrix that has $\mathbf{W1}$ in its diagonal:

$$\mathbf{D} = diag(\mathbf{W1}) \in \mathbb{R}^{n \times n}, \text{ where } \mathbf{1} = (1...1)^T \in \mathbb{R}^n.$$

There are different ways to define the weight matrix $W \in \mathbb{R}^{n \times n}$ of the similarity graph, e.g. the ε-neighborhood graph, the k-nearest neighbor graphs, and the fully connected graph with similarity functions such as Gaussian radial basis or Cosine similarity. And there are different choices of similarity functions $K : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}^+ \cup \{0\}$ that can be used to measure the similarity $w_{i,j}$ between two data points x_i, x_j.

1.2.2 Graph Laplacian and Embedding

After forming the similarity graph in spectral clustering, we will use the eigen-decomposition of the graph Laplacian to embed the data points. An unnormalized

graph Laplacian matrix is defined by:

$$\mathbf{L} = \mathbf{D} - \mathbf{W}.$$

There are two normalized graph Laplacian matrices:

$$\mathbf{L_{rm}} = \mathbf{D^{-1}L} = \mathbf{I} - \mathbf{D^{-1}W}$$

and

$$\mathbf{L_{sym}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{L}\mathbf{D}^{-\frac{1}{2}} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{W}\mathbf{D}^{-\frac{1}{2}}$$

The first k eigenvectors corresponding to the $\mathbf{L}$, $\mathbf{L_{sym}}$ and $\mathbf{L_{rm}}$ represent a low dimensional embedding of the original data points.

The spectral embeddings from the un-normalized and normalized graph Laplacian matrices can be derived from the relaxed solutions of RatioCut and NCut problems [3]. In the RatioCut problem, when partitioning the vertices V in the graph into k groups, the goal is to optimize the function of:

$$\min_{A_1..,A_k \in V} RatioCut(A_1, ...A_k) = \sum_{i=1}^{k} \frac{cut(A_i, \bar{A}_i)}{|A_i|}$$

By defining a matrix $H \in \mathbb{R}^{n \times k}$ where each column h_j has the following value:

$$h_{i,j} = \begin{cases} 1/\sqrt{|A_j|} & \text{if } v_i \in A_j \\ 0 & \text{otherwise} \end{cases}, \text{ where } i = 1, \ldots, n \text{ and } j = 1, \ldots, k$$

, it becomes:

$$RatioCut(A_1, ...A_k) = \sum_{i=1}^{k} \frac{cut(A_i, \bar{A}_i)}{|A_i|} = \sum_{i=1}^{k} h_i' \mathbf{L} h_i = Tr(H' \mathbf{L} H)$$

Then we can reconstruct this problem into:

$$\min_{A_1..,A_k \in V} Tr(H' \mathbf{L} H) \text{ subject to } H'H = I, H \text{ as defined above}$$

But this discrete problem is an NP-hard problem. By relaxing the requirement to enable the H to take any real value, the problem becomes solvable: Based on the Rayleigh-Ritz theorem, the optimization was obtained at the first k eigenvectors of $\mathbf{L}$, which leads us to the spectral embedding of the unnormalized graph Laplacian matrix.

Similarly, the NCut problem can be reformulated and relaxed as the following:

$$\min_{A_1..,A_k \in V} Tr(H'\mathbf{L}H) \text{ subject to } H'\mathbf{D}H = I$$

Based on the Rayleigh-Ritz theorem, the optimization was obtained at the first k eigenvectors of $\mathbf{L_{rw}}$. If we substitute the $T = \mathbf{D^{\frac{1}{2}}}H$, then the optimization was obtained when T equals the first k eigenvectors of $\mathbf{L_{sym}}$, which leads to the normalized spectral embedding.

The matrix $\mathbf{P} = \mathbf{D^{-1}W}$ shares the same eigenvectors with $\mathbf{L_{rm}}$. And $\tilde{\mathbf{W}} = \mathbf{D^{-\frac{1}{2}}WD^{-\frac{1}{2}}}$ shares the same eigenvectors with $\mathbf{L_{sym}}$. Their eigenvalues are 1 minus the eigenvalues of $\mathbf{L_{rm}}$ and $\mathbf{L_{sym}}$, repectively. It is also equivalent to using the top k eigenvectors of $\mathbf{P}$, denoted by $\mathbf{U} \in \mathbb{R}^{n \times k}$, or the top k eigenvectors of $\tilde{\mathbf{W}}$, denoted by $\tilde{\mathbf{U}} \in \mathbb{R}^{n \times k}$ as the embedding. It should be noted that $\mathbf{U} = \mathbf{D^{-\frac{1}{2}}}\tilde{\mathbf{U}}$.

1.2.3 Clustering

After embedding the data into the eigenspace of $\mathbf{L_{rm}}$ or $\mathbf{L_{sym}}$, we can use k-means clustering to cluster the embedded data into k clusters. After projecting the data into eigenspace, the clustering pattern becomes more clear and the data are easy to be separated. Algorithm 1 is a common spectral algorithm based on the normalized graph laplacian $\mathbf{L_{sym}}$.

We used four simulated toy datasets to demonstrate the process of the spectral clustering algorithm and how the algorithm embeds the data into eigenspace to enhance the cluster patterns. The four datasets are generated from two moons, three Gaussian distributions, two circles, and three circles patterns respectively. The first

Algorithm 1 Normalized spectral clustering according to Ng, Jordan, and Weiss [6]

Input: Similarity function, data $\mathbf{X} \in \mathbb{R}^{n \times d}$, number of clusters k

Output: Partition of k clusters, $C_1, ..., C_k$

1: Construct a similarity graph by the similarity function. Let $\mathbf{W}$ be its weighted adjacency matrix and $\mathbf{D} = diag(\mathbf{W1})$ the degree matrix.

2: Compute the normalized Laplacian $\mathbf{L_{sym}} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{W}\mathbf{D}^{-\frac{1}{2}}$.

3: Compute the first k eigenvectors $\tilde{\mathbf{U}} \in \mathbb{R}^{\mathbf{n} \times \mathbf{k}}$ of $\mathbf{L_{sym}}$

4: Normalize the rows of $\tilde{\mathbf{U}}$ with the norm equals to 1

5: Partition the rows of the normalized $\tilde{\mathbf{U}}$ into k clusters by k-means

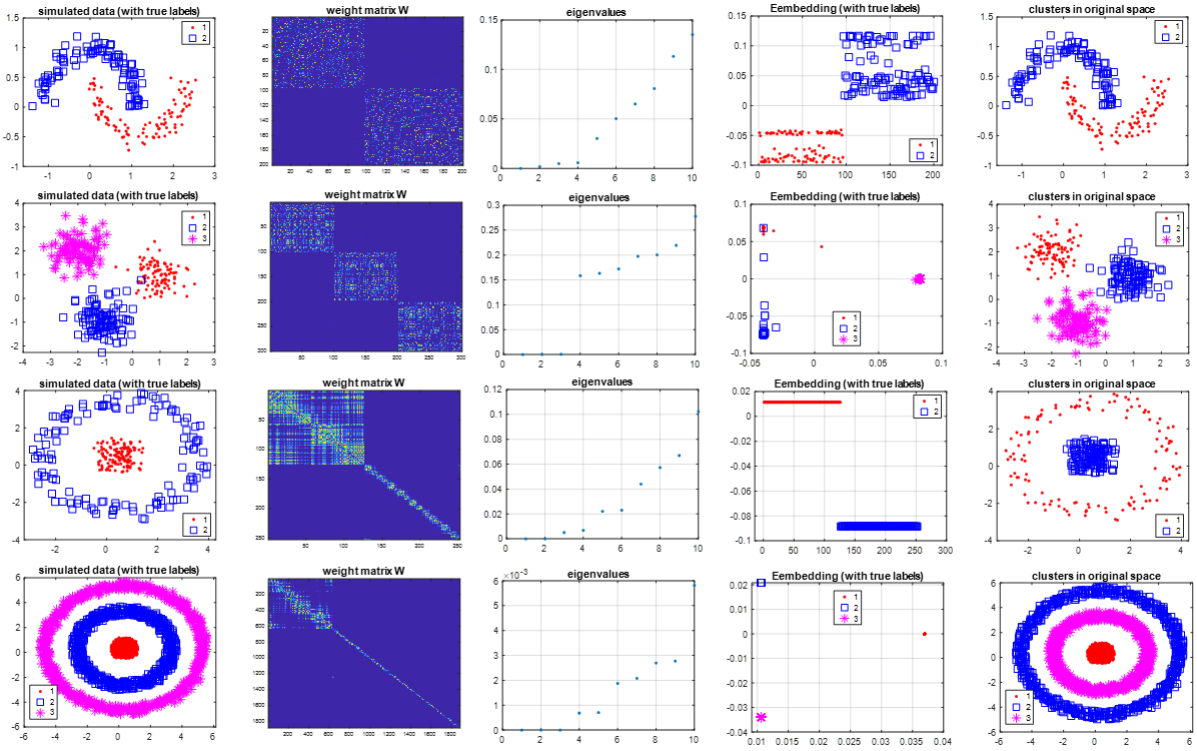


Figure 1: Samples to demonstrate the spectral clustering where the simulated datasets were generated from four different distributions and patterns. First column: scatter plots of the simulated data colored by the true labels. Second column: the heatmap of the weight matrix. Third column: eigenvalues of the L_{sym}. Fourth column: the embedding of the original data by the first k eigenvectors. Fifth column: the scatter plot of the original data colored by the cluster labels.

column in Figure 1 shows the distribution of the original data with their true labels.

The second column shows the heatmap of the weight matrices for these four datasets

constructed from the fully connected similarity graph and the Gaussian similarity function: $s(x_i, x_j) = exp(-|x_i - x_j|^2/(2\sigma^2))$, where $\sigma = 4$. The clusters are shown in the heatmap of the weight matrices by the light color blocks on the diagonals. The third column shows the first 10 eigenvalues of the normalized graph Laplacian matrices. The fourth column shows the embedded data in the k-dimensional eigenspace. For the datasets with 2 clusters, the first two eigenvalues are 0 or close to 0. The 2nd eigenvector functions as an indicator vector to separate the two clusters with a threshold. For the datasets with 3 clusters, the first three eigenvalues are 0 or close to 0. The 2nd and 3rd eigenvectors comprehend the information about the clustering patterns. It is easy to see that in the embedding space, the data points from the same clusters are grouped. Simple k-means can cluster them correctly.

Spectral clustering is easy to implement and can handle non-linear clusters. There are no assumptions about the underlying distributions of the datasets. And it was backed by rich mathematical theories: random walk, graph cut, and perturbation theory. However, the computational cost and the memory requirement of spectral clustering are high. It requires $O(n^2d)$ time to construct the weight matrix from a similarity graph and $O(n^3)$ to perform the eigendecomposition of the graph Laplacian. The total time complexity is $O(n^2d + n^3)$. During the spectral clustering process, it requires storing the weight matrix or the graph Laplacian matrix, which leads to at least $O(n^2)$ space complexity.

1.3 Scalable Spectral Clustering

As we mentioned, spectral clustering has a high computational cost. It requires the decomposition of an n×n weight matrix. When n is large, this is a very expensive computation. Chen's research proposed a landmark-based computation framework in spectral clustering with cosine similarity to speed up the computation by using

low-rank SVD and avoiding the computation on the weight matrix. We called this method the documents model or the scalable cosine similarity model because it refers to the similarity matrix $\mathbf{A}$ as a "document" dataset with sparse rows and clusters them based on the cosine similarity [9][2].

This method assumes the setting of cosine similarity and sparse data. Two kinds of data can be considered sparse. One is that n(number of data points) and d(dimensions) are both large but $\mathbf{X} = \begin{bmatrix} x_1 \\ ... \\ x_n \end{bmatrix} \in \mathbb{R}^{n \times d}$ are row sparse, such as documents under the bag-of-words model. The other is that the data is low dimensional where $n \gg d$, such as image data. If $\mathbf{X}$ has unit l_2 norms, i.e $||x_i|| = 1$. $\mathbf{X}\mathbf{X}^T$ is equivalent to the similarity matrix based on the cosine similarity function. The weight matrix $\mathbf{W}$ can be expressed as:

$$\mathbf{W} = \mathbf{X}\mathbf{X}^T - \mathbf{I}, \text{ where}$$

In this way, as we shall see, the weight matrix can be avoided. First, the degree matrix can be calculated as

$$\mathbf{D} = \mathbf{W}\mathbf{1} = diag((\mathbf{X}\mathbf{X}^T - \mathbf{I})\mathbf{1}) = diag(\mathbf{X}(\mathbf{X}^T\mathbf{1}) - \mathbf{1})$$

The computational cost of the degree matrix becomes $O(nd)$. The normalized weight matrix becomes

$$\tilde{\mathbf{W}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{X}\mathbf{X}^T\mathbf{D}^{-\frac{1}{2}} - \mathbf{D}^{-1} = \tilde{\mathbf{X}}\tilde{\mathbf{X}}^T - \mathbf{D}^{-1}, \text{ where } \tilde{\mathbf{X}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{X}.$$

If the degrees are constant, $\mathbf{D}^{-1} = \frac{1}{\beta}\mathbf{I}$ where $\beta > 0$, the eigenvectors of $\tilde{\mathbf{W}}$ are equal to the left singular vectors of $\tilde{\mathbf{X}}$. Instead of embedding the data points by finding the eigenvectors of the normalized weight matrix, this method embeds the data by finding the left singular vectors of $\tilde{\mathbf{X}}$. Since the dataset is sparse, the SVD of $\tilde{\mathbf{X}}$ is very fast and has a much lower time complexity than the eigendecomposition of $\tilde{\mathbf{W}}$. The algorithm is shown at Algorithm 2.

Algorithm 2 Scalable spectral clustering with cosine similarity [9]

Input: Similarity function, data $\mathbf{X} \in \mathbb{R}^{n \times d}$, number of clusters k, fraction of outliers α

Output: Partition of k clusters, $C_1, ..., C_k$, and a set of outliers C_0

1: Compute the degree matrix $\mathbf{D}$
2: Find the data corresponding to the smallest degrees and remove them as outliers
3: Construct $\tilde{\mathbf{X}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{X}$
4: Use SVD to find the first k left singular vectors of $\tilde{\mathbf{X}}$, $\tilde{\mathbf{U}}$ (can convert $\tilde{\mathbf{U}}$ to $\mathbf{U}$ for Ncut [9])
5: Normalized the rows of $\tilde{\mathbf{U}}$ with norm equal to 1
6: Partition the rows of the normalized $\tilde{\mathbf{U}}$ into k clusters, $C_1, ..., C_k$, by k-means

Algorithm 2 can be generalized to other similarity functions. We first selected a set of l landmark points randomly from the dataset $\mathbf{X}$. A similarity matrix $\mathbf{A} \in \mathbb{R}^{n \times l}$ is constructed between the whole dataset and the selected landmark points. The matrix $\mathbf{A}$ is very sparse with a small number of non-zero terms on the rows. We can consider $\mathbf{A}$ as a document-term matrix and perform Inverse Document Frequency (IDF) weights on it, and then we can apply the spectral clustering with the cosine similarity algorithm to this "document" dataset and cluster the rows of $\mathbf{A}$.

During the steps of the generalized Algorithm 2, a matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$ should be stored, so the space complexity of this algorithm is $O(nd)$ or $O(ns)$ where s is the number of non-zero entry in the rows, depending on the relative size of s and n. The steps in the algorithm include the matrix entrywise manipulation, matrix and vector publication, and low rank SVD of a sparse matrix $\mathbf{A}$, so the time complexity is $O(nsk)$ or $O(ndk)$. Since this algorithm avoids the computation on the matrix $\tilde{\mathbf{W}}$, the computation cost was largely reduced from $O(n^2d + n^3)$ to $O(nsk)$ or $O(ndk)$. However, since it still requires the storage of $\tilde{\mathbf{X}}$, the space complexity is still large if the dataset is large with a very large n, $O(nd)$ or $O(ns)$.

1.4 Software, Data and Evaluation Criteria

In the following sections, we will introduce our new scalable spectral clustering methods along with experiments testing their performance. Here we briefly introduce the software, simulated and real data, and evaluation criteria used in our experiments.

1.4.1 Software

All experiments were conducted in MATLAB R2021b on a desktop computer with 32 GB of RAM and a CPU with 4 cores.

1.4.2 Data

The first data set we used is a simulated data set. The cosine similarity performs best when the underlying clusters are at different angles. The simulated dataset was generated from three different angles with three dimensions representing three clusters each with 1000 data points, a total of 3000 data points. Each data point was added a noise ε generated from a Gaussian with a mean of 3 and a standard deviation of 0.5, illustrated by Figure 2.

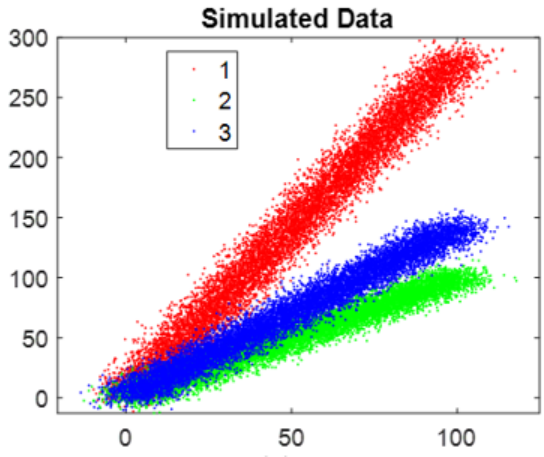


Figure 2: Scatter plots of simulated datasets

Some image data sets are large in quantity but only have moderate dimensions, and cosine similarity is a reasonable choice for quantifying image similarity. Thus,

performing clustering on image data is an appropriate way for testing our algorithms. We used three image data sets, usps, mnist, and pendigits, downloaded from the LIBSVM website[1], They all contained images of the handwriting of numbers 0 to 9, illustrated by Figure 3. The data sets are originally partitioned into training and test parts for classification purposes, but we merged the two parts together for our unsupervised setting.

Figure 3: Sample images from the usps data

Another type of real-world data that we used is document data. The document-term matrix from document data is often large both in size and in dimension, each of which could range from thousands to millions. However, this matrix is typically very sparse, which satisfies our first assumption on the data matrix. Furthermore, cosine similarity is a natural choice for quantifying document similarity. Thus, document clustering is an ideal application for testing our algorithms. The document dataset we considered is the 20 Newsgroups data[2], consisting of 18,774 newsgroup documents that are partitioned nearly evenly across 20 different newsgroups. There are a total of 61,118 unique words (including stopwords) present in the corpus. To preprocess the data, we first converted each text into word counts, and then converted the word counts into binary arrays. We then remove stop words, the words appear in only one document or too many documents (939, the average cluster size). Then we applied the IDF weights to the remaining columns. After these preprocessing steps, the document data can be directly used in the experiment. We also use SVD to project the document

[1] `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`

[2] Available at `http://qwone.com/~jason/20Newsgroups/`.

into the first 100 SVD subspaces. In the experiments, we used both non-projected and SVD-projected data. The information of the data sets is summarized in Table 1.

Table 1: Summary information of the real-world data sets used in our study.

datasets	**#instances**	**#dimensions**	**#classes**
usps	9,298	256	10
pendigit	10,992	16	10
mnist	70,000	184	10
20news	18,768	55,570	20
20news(SVD100)	18,768	100	20

1.4.3 Evaluation

One evaluation metric to compare different clustering solutions is classification rate (accuracy rate), which is given by the total number of agreements divided by the total number of observations. One minus the classification rate is called the misclassification rate (error rate). Since our data all come with true labels, accuracy or error is a reliable and objective measure of the clustering results. Another reason is that our methods are extended from Chen's cosine similarity scalable method and in Chen's paper, they used accuracy and error rates to quantify their method's performance. We continue to use the same metrics to have a fair comparison between the proposed and baseline methods. Last but not least, since the goal of our method is to study new scalable methods, another important measurement is CPU time. We use it to measure the computational time cost in the experiment.

CHAPTER 2

New Approaches

2.1 Approach 1. Subsample

The original spectral clustering and the scalable spectral clustering algorithms mentioned previously assumed that the whole dataset is known and stored in the database. However, real-life data is usually at a large scale which makes it impossible to store at once. Moreover, in a lot of scenarios, the data that needs to be clustered are online data and come in a sequential manner. In that case, it is not possible to see the whole dataset and store them in memory. In our research, we proposed three different approaches to address the memory challenge. The first approach uses a small subset $\mathbf{X}_s \in \mathbb{R}^{s \times d}$ randomly selected from the whole dataset $\mathbf{X} \in \mathbb{R}^{n \times d}$ to approximate the spectral embedding of the whole dataset. We named it the Subsample method.

2.1.1 Methodology

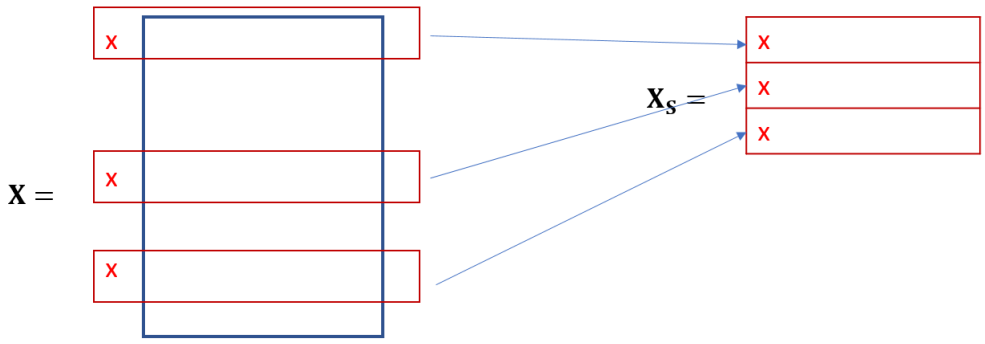


Figure 4: Illustration of subsample method

In this approach, we continue using the setting from the scalable cosine similarity model mentioned in Section 1.3 in which the dataset $\mathbf{X}$ is sparse with few non-zero entries in its rows or is low dimensional data where $n \gg d$ and the similarity function is cosine similarity. We also assume that initially, we only know a random subset

$\mathbf{X}_s = \begin{bmatrix} \mathbf{x}_1^T \\ \vdots \\ \mathbf{x}_s^T \end{bmatrix} \in \mathbb{R}^{s\times d}$ with size s from the whole dataset. And the remaining data points from $\mathbf{X} \in \mathbb{R}^{n\times d}$ come in an online fashion.

The first step of this approach is to randomly sample s data points from the dataset, $\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \vdots \\ \mathbf{x}_n^T \end{bmatrix} \in \mathbb{R}^{n\times d}$. In Section 1.3, we showed that the normalized weight matrix $\tilde{\mathbf{W}} = \tilde{\mathbf{X}}\tilde{\mathbf{X}}^T - \mathbf{D}^{-1},$ where $\tilde{\mathbf{X}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{X}$. If the degrees are constant, $\mathbf{D}^{-1} = \frac{1}{\beta}\mathbf{I}$ where $\beta > 0$, the top k eigenvectors of $\tilde{\mathbf{W}}$ are equal to the top k left singular vectors of $\tilde{\mathbf{X}}$. The key step is to use top singular vectors of $\tilde{\mathbf{X}}$ to approximate the eigenvectors of $\tilde{\mathbf{W}}$. Since the full dataset is unknown, we want to use sampled data to approximate the full dataset $\tilde{\mathbf{X}}\tilde{\mathbf{X}}^T \in \mathbb{R}^{n\times n}$, which is hard to be approximated due to the dimension. But it is possible to use sample data to approximate $\tilde{\mathbf{X}}^T\tilde{\mathbf{X}} \in \mathbb{R}^{d\times d}$:

$$\begin{aligned} \tilde{\mathbf{X}}^T\tilde{\mathbf{X}} &= \sum_{i=1}^{n} \tilde{\mathbf{x}}_i\tilde{\mathbf{x}}_i^T \\ &\approx \frac{n}{s}\sum_{i=1}^{s} \tilde{\mathbf{x}}_i\tilde{\mathbf{x}}_i^T \\ &= \frac{n}{s}\tilde{\mathbf{X}}_s\tilde{\mathbf{X}}_s^T, \text{ where } \tilde{x}_i = d_i^{-1}x_i \end{aligned} \tag{1}$$

This shows that the full degree normalized dataset $\tilde{\mathbf{X}}$ can be approximated by $\tilde{\mathbf{X}}_s$. And if the full SVD of $\tilde{\mathbf{X}}$ is $\tilde{\mathbf{U}}\tilde{\Sigma}\tilde{\mathbf{V}}^T = \tilde{\mathbf{X}}$, where $\tilde{\mathbf{U}} \in \mathbb{R}^{n\times n}$ is the matrix of left singular vectors and $\tilde{\mathbf{V}} \in \mathbb{R}^{d\times d}$ is the matrix of right singular vectors. And the diagonal entries in $\tilde{\mathbf{\Sigma}} \in \mathbb{R}^{d\times d}$ are the singular values. If $\tilde{\mathbf{\Lambda}} = \tilde{\mathbf{\Sigma}}^2 = \begin{bmatrix} \sigma_1^2 & & \\ & \ddots & \\ & & \sigma_d^2 \end{bmatrix} \in \mathbb{R}^{d\times d}$, then:

$$
\begin{aligned}
\tilde{\mathbf{X}}^T\tilde{\mathbf{X}} &= \tilde{\mathbf{V}}\tilde{\mathbf{\Lambda}}\tilde{\mathbf{V}}^T \\
&= \sum_{i=1}^{d} \sigma_i^2 \tilde{\mathbf{v}}_i \tilde{\mathbf{v}}_i^T \\
&\approx \frac{n}{s}\tilde{\mathbf{X}}_s\tilde{\mathbf{X}}_s^T \\
&= \frac{n}{s}\tilde{\mathbf{V}}_s\tilde{\mathbf{\Lambda}}_s\tilde{\mathbf{V}}_s^T, \text{ where } \tilde{\mathbf{X}}_s = \mathbf{D}_s^{-\frac{1}{2}}\mathbf{X}_s \text{ and } \tilde{\mathbf{V}}_s\tilde{\mathbf{\Sigma}}_s\tilde{\mathbf{V}}_s^T = \tilde{\mathbf{X}}_s \\
&= \sum_{i=1}^{d} \sigma_{s,i}^2 \tilde{\mathbf{v}}_{s,i} \tilde{\mathbf{v}}_{s,i}^T, \text{ where } \tilde{\mathbf{V}}_s = \left[\tilde{\mathbf{v}}_{s,1} ... \tilde{\mathbf{v}}_{s,d}\right] \in \mathbb{R}^{d\times d} \text{ and } \tilde{\mathbf{\Sigma}}_s = \begin{bmatrix} \sigma_{s,1} & & \\ & \ddots & \\ & & \sigma_{s,d} \end{bmatrix} \in \mathbb{R}^{d\times d}
\end{aligned}
\tag{2}
$$

This shows that the right singular vectors of $\tilde{\mathbf{X}}_s$ can be used to approximate the left singular vectors of $\tilde{\mathbf{X}}$. Instead of performing the SVD on $\tilde{\mathbf{X}}$ to find the eigen embedding of the full dataset, we can perform SVD on $\tilde{\mathbf{X}}_s$ to first embed the sample dataset using the first k right singular vectors of $\tilde{\mathbf{X}}_s$, denoted by $\tilde{\mathbf{V}}_{s,k}$, then project the rest of the data into the SVD space by the right singular vectors. Withouth knowing the entire dataset, we can approximate the degree matrix of the sample data $\mathbf{D}_s$ by:

$$
\mathbf{D}_s = \mathbf{X}_s(\mathbf{X}^T\mathbf{1}) - 1 \approx \frac{n}{s}(\mathbf{X}_s(\mathbf{X}_s^T\mathbf{1}) - \mathbf{1}) \tag{3}
$$

After finding the degree matrix of the sample data by the above equation, we removed the data points corresponding to the bottom α percent smallest degree among the s data points. Then we perform low-rank SVD of $\tilde{\mathbf{X}}_s$ to find the approximated spectral embeddings of the sample data.

$$
\tilde{\mathbf{U}}_s\tilde{\Sigma}_s\tilde{\mathbf{V}}_s^T \approx \tilde{\mathbf{X}}_s, \text{ where } \tilde{\mathbf{U}}_s \in \mathbb{R}^{n\times k}, \tilde{\mathbf{V}}_s \in \mathbb{R}^{d\times k}, \text{ and } \tilde{\Sigma} \in \mathbb{R}^{k\times k} \tag{4}
$$

We then employed the k-means algorithm to cluster the embedded sample data $\tilde{\mathbf{U}}_{s,k}$ in $\mathbb{R}^k$ into different clusters, C_j, and recorded the centroid of each cluster, $\mathbf{c}_j$, where

$j = 1, ..., k$.

$$\mathbf{c}_j = \frac{1}{n_j} \sum_{i=1}^{n_j} \mathbf{y}_i \tag{5}$$

where $\mathbf{y}_i$ is the ith row vector of $\tilde{\mathbf{U}}_{s,k}$ and n_j is the number of the data in the jth cluster. We assume that the remaining data is online data and processed one by one and the new data point is $\mathbf{x}_0 \in \mathbb{R}^{d\times 1}$. We can approximate the degree of the new data point by:

$$d_0 \approx \frac{n}{s}(\mathbf{x}_0^T(\mathbf{X}_s^T\mathbf{1}) - 1 \tag{6}$$

Then we can find the spectral embedding of the new data point $\mathbf{x}_0$ by project $\tilde{\mathbf{x}}_0$ into the sample SVD space:

$$\tilde{\mathbf{u}}_0 = \tilde{\mathbf{x}}_0\tilde{\mathbf{V}}_s\tilde{\mathbf{\Sigma}}_s^{-1}, \text{ where } \tilde{\mathbf{x}}_0 = d_0^{-\frac{1}{2}}\mathbf{x}_0 \tag{7}$$

Finally, the cluster label j of the new data point was assigned based on the nearest centroid of $\tilde{\mathbf{u}}_0$:

$$\arg\max_j ||\mathbf{c}_j - \tilde{\mathbf{u}}_0|| \tag{8}$$

Algorithm 3 Subsample spectral clustering with cosine similarity

Input: data $\mathbf{X} \in \mathbb{R}^{n\times d}$, number of clusters k, number of samples s, fraction of outliers α

Output: Partition of k clusters, $C_1, ..., C_k$

1: Randomly samples s data points from the given data
2: Approximate the degrees $\mathbf{D}_s$ of the samples and find the outliers
3: Calculate the $\tilde{\mathbf{X}}_s$ for the sample dataset and find its top k left singular vectors by rank-k SVD: $\tilde{\mathbf{U}}_{s,k}\tilde{\mathbf{\Sigma}}_{s,k}\tilde{\mathbf{V}}_{s,k}^T \approx \tilde{\mathbf{X}}_s$.
4: Normalize each row of left singular vectors with a norm equal to 1, and apply k-means clustering to cluster the samples.
5: For each of the new data points that come from the rest of the dataset, approximate the degree of the new data point.
6: Project the data into the sample SVD space by equation (7)
7: Cluster the new data point by finding its nearest centroid.

During the process of clustering the sample dataset, approximating the degree of the sample dataset and calculating the $\tilde{\mathbf{X}}_s$ requires $O(sd)$ time and the rank-k SVD of the $\tilde{\mathbf{X}}_s$ requires $O(ksd)$. During the process of clustering the rest of the incoming data

points, approximating the degrees requires $O((n-s)d + sd) = O(nd)$ and projecting the data point into the approximate SVD space requires $O((n-s)dk)$. So the total time complexity is $O(ndk)$. During the clustering of the sample dataset, $\tilde{\mathbf{X}}_s$ was used while during the clustering of the rest of the data points, the k left singular vectors $\tilde{\mathbf{X}}_{s,k}$ was kept. So the space complexity was $O(sd + kd)$. Compared to the document model, the subsample method has the same time complexity but has a lower space complexity, since instead of the full dataset $\mathbf{X}$, the subsmaple method only needs to work on the sample dataset $\mathbf{X}_s$. And more importantly, the subsampling method was able to handle the sequential data which is very common in modern-day web applications.

2.1.2 Experiment

We applied the Subsample method to the simulated dataset. The goal is to test the performance of the proposed method and to find the smallest sample size for the model to perform well.

Figure 5 shows a sample output from the algorithm when the fraction of outlier α is 0.01 and the fraction of sample size is 0.1. Figure 3(a) shows the embedding of the sample dataset by the first two left singular vectors of $\tilde{\mathbf{X}}_s$. The data were color labeled by their true cluster labels and the centroids of each cluster were outlined. From the figure, we can see the data that embedded into the SVD space was much easier to be separated by k-means than the original data. The cluster results of the sample data were shown in figure 3(b) with different color labels. Figure 3(c) shows how the remaining data were projected into the SVD space of the sample data. It is clear that they were projected into their original clusters. The centroids of the clusters of the sample dataset were still near the centers of the clusters of the remaining data points. From Figures 5(b) and 3(d), the three groups can be easily separated from

this algorithm.

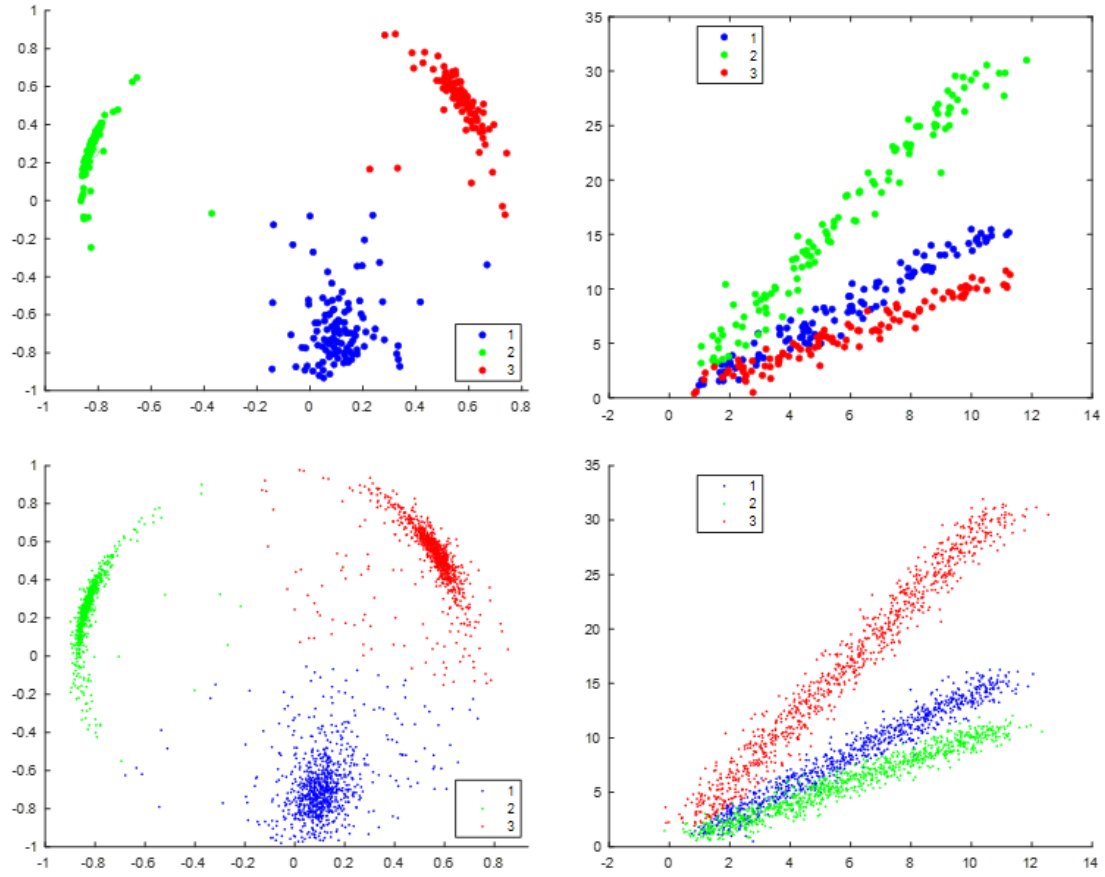


Figure 5: Visualization the embedding and clustering result by the Subsample method in the toy dataset, $\alpha = 0.01$ and $s = 0.1$. (a) Spectral embeddings of the sample data. (b) Clustering result of the sample data. (c) Spectral embeddings of the remaining data. (d) Clustering result of the remaining data

During the process of our algorithm, there is randomness in the step of random sampling and the step of k-means. To have a fair comparison, we repeat the same experiment 5 times under different sample sizes and record the accuracy rate and the CPU time. The rate of outliers α was set to 0.01. From Figure 6, the accuracy rate of the algorithm is around 97.2% when s increase from 0.1 to 0.9. The optimal accuracy was obtained when $s = 0.2$. Overall, the accuracy rates under all s are very

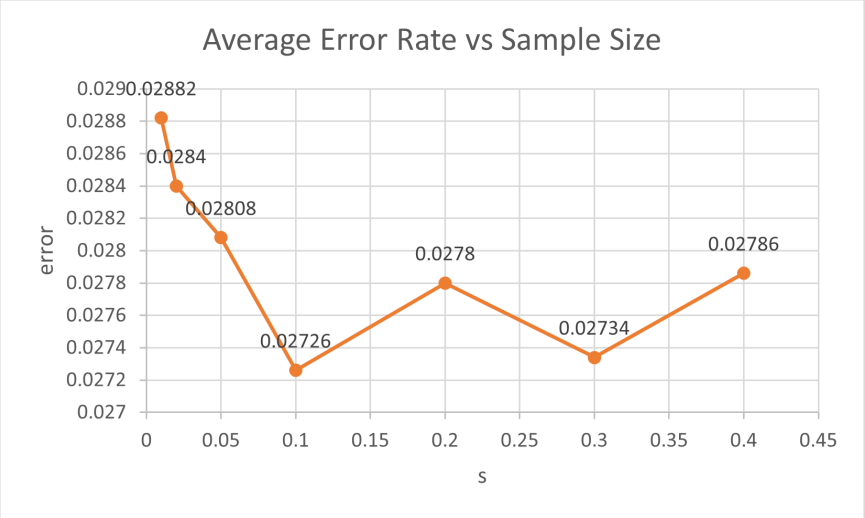


Figure 6: The accuracy rate of the toy dataset with different sample size

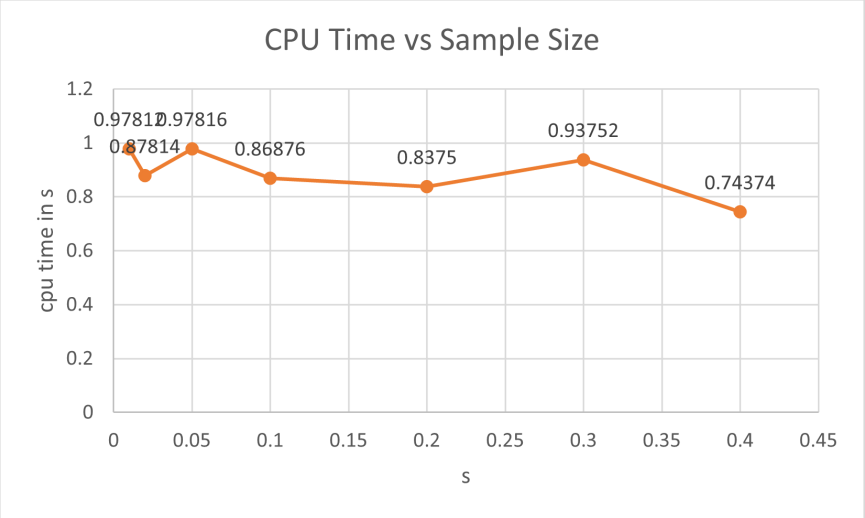


Figure 7: The CPU time in seconds of the toy dataset with different sample size

high. This shows that changing the percentage of the sampling does not make a large difference in the accuracy rate when the sample size is larger than 0.1. Even with a very small sample size, s, the accuracy rate is already very high. From Figure 7, the CPU time decreases with the increase of s. When s = 0.4 the CPU time reaches its minimum of 0.4362 s. Overall the CPU times are all lower than 1 s.

In addition to the simulated dataset, we also applied the algorithm to real-world

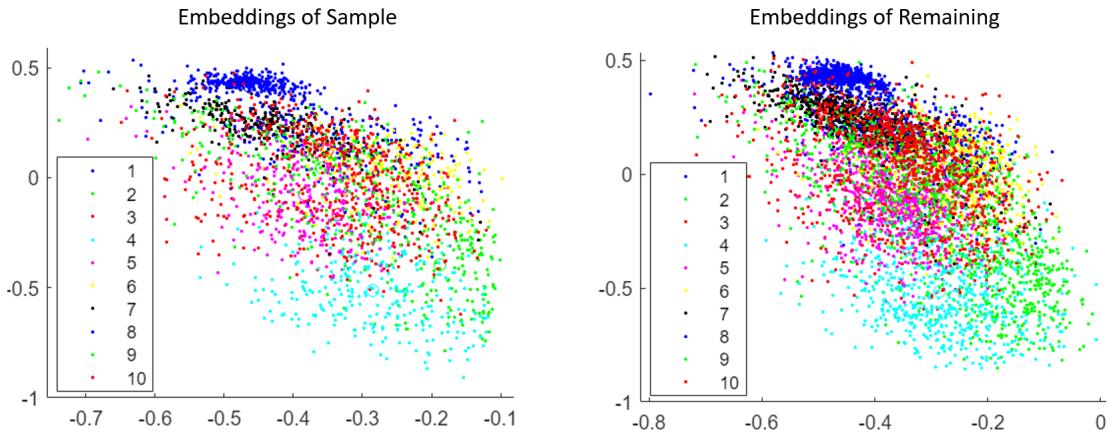
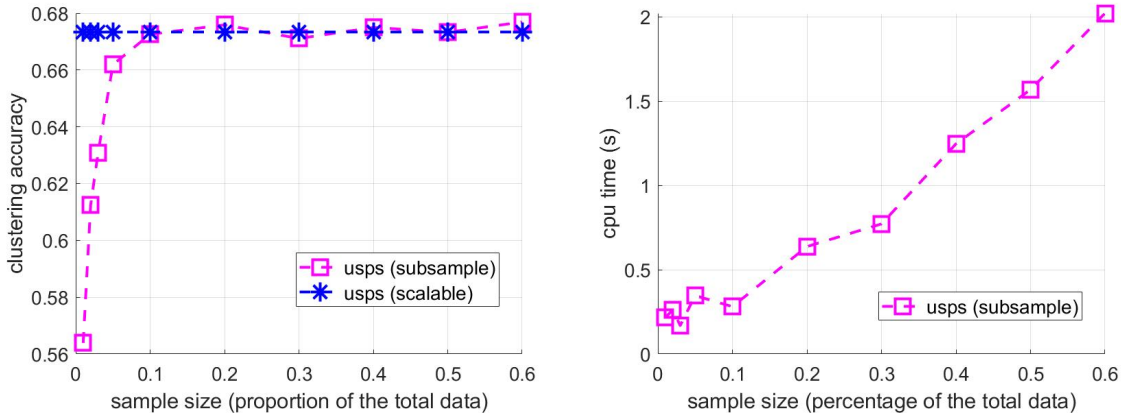


Figure 8: Embeddings of sample data and remaining data points in the USPS dataset by Subsample method with $\alpha = 0.01$ and $s = 0.3$.

datasets to test the effectiveness of the algorithm. We repeated the experiments 5 times with each sample size and recorded the average of the error rates and the CPU times. The fraction of outliers α was set to 0.01. Figure 6 shows the embedding of the sample data and the remaining data of USPS data by the first 2 left singular vectors and colored by the true labels from 0 to 9. Since the USPS dataset has 16×16 dimensions and 10 clusters, the plots of the embedding in 2 dimensions are not very informative.

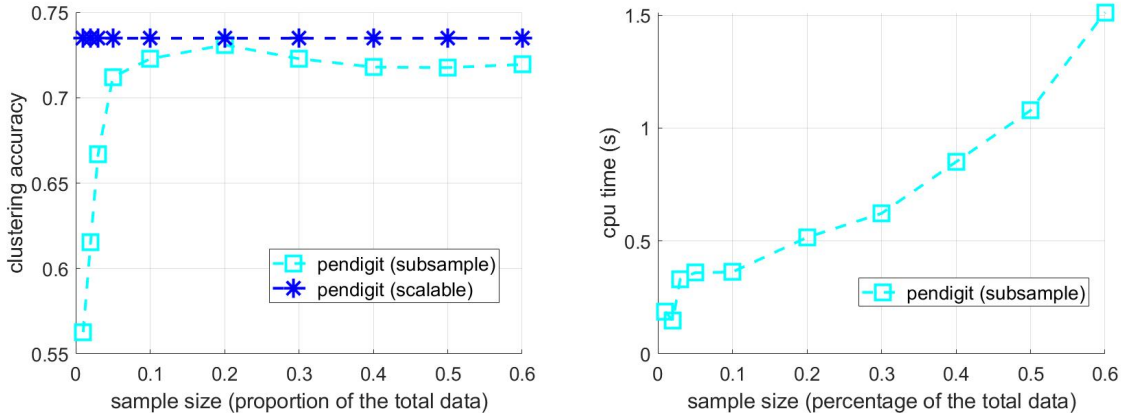
To study the relationship between sample size and the method performance, we plot the average accuracy rate of the three digits data sets under the Subsample model with different sample sizes (proportion of the total dataset). The sample size, s, was chosen in (0.01, 0.02, 0.03, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6). Figure 9a plots the average accuracy rates and the CPU time in seconds with the experiment in the USPS data. The average accuracy rate starts from 0.56 when s = 0.01 and increases with the sample size. It converges to the accuracy rate of the scalable cosine similarity model (scalable) introduced in Section 1.3. The accuracy rate stabilized at 0.65 when s is equal to 0.1. The best error rate, 0.66, was obtained when s equals 0.2. This shows



(a) The accuracy rate of the USPS dataset with different sample size

(b) The CPU time in seconds of the USPS dataset with different sample size

Figure 9: Accuracy & CPU time (s) for USPS Data with Subsample method



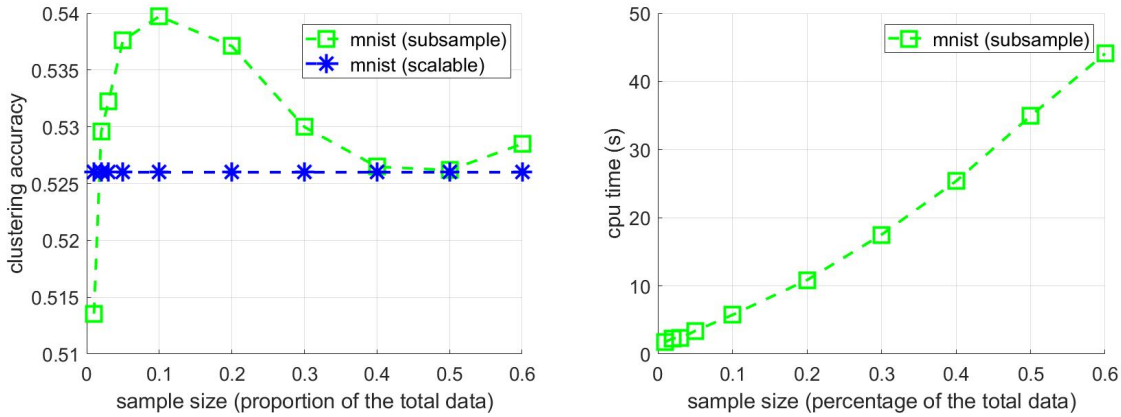
(a) The accuracy rate of the Pendigits data with different sample size

(b) The CPU time in seconds of the Pendigits data with different sample size

Figure 10: Accuracy& CPU time (s) for Pendigits Data with Subsample method, $\alpha = 0.1$

that under a sample size as small as 0.1, the accuracy rate is already very close to the optimum.

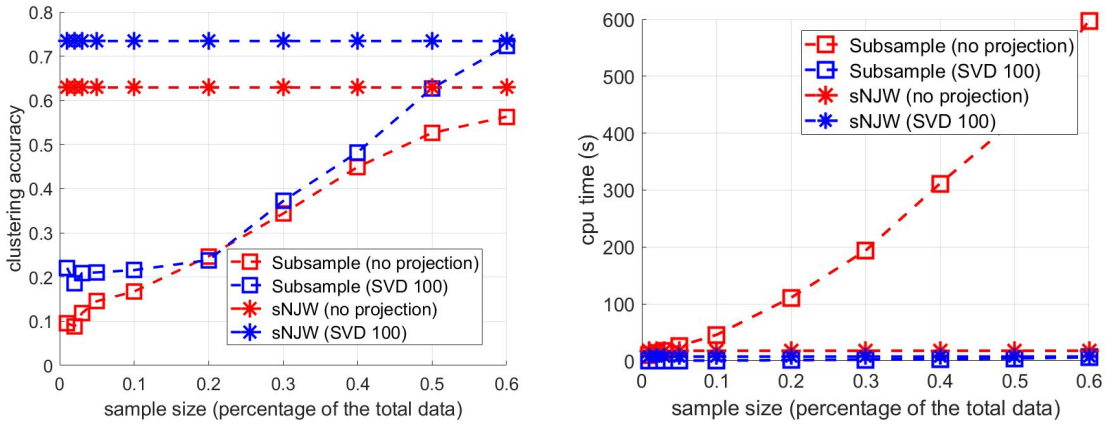
A similar pattern was revealed in the accuracy plots and the cpu plots of the other two data sets, Pendigits, and mnist. From Figure 10a, the accuracy rate of Pendigtits data with the subsample method increases with the increase of the sample



(a) The accuracy rate of the mnist data with different sample size

(b) The CPU time in seconds of the mnist data with different sample sizes

Figure 11: Accuracy& CPU time (s) for mnist Data with Subsample method, $\alpha = 0.1$



(a) The accuracy rate of the 20news data with different sample size

(b) The CPU time in seconds of the 20news data with different sample sizes

Figure 12: Accuracy& CPU time (s) for 20news Data with Subsample method, $\alpha = 0.1$

size and appeared slightly lower than the accuracy rate line from the scalable method. When sample size = 0.1, it has an accuracy rate of 0.72 which is close to the accuracy rate from the scalable method, 0.73. From Figure 11a, the accuracy rate of the mnist data with the subsample method first increases and then decreases with the increase of the sample size. We noticed that mnist has a much larger data size compared to

the other two digit data sets and its sample size = 0.01 (1%) results in 700 sample data points. Because of the large size of the sample data, when the sample size = 0.02, the scalable method already achieved the same accuracy as the scalable method. The CPU time for usps, mnist, and pendigits all increase with the increase of the sample size. We found that the increasing cpu time was caused by the outlier removal steps. For removing the outlier, we need to use the mink() function in Matlab which has $O(n)$ time complexity. The number of iterations is proportional to the number of outliers and the sample size, s.

The same experiments are applied to the document data, shown in Figure 12. We can see that compared to the digit data with 10 classes (0-9 digits), the 20 news document data with 20 classes need more than double the number of sample data points to converge. We suspected that the number of sample data points needed to have a good performance has some relationship with the number of classes. The document data projected to the SVD subspace has a better accuracy rate compared to the original document data. And in the CPU plot, the original document data without projection has a runtime cost with the increase in the sample size. This is due to the large dimension of the original data and the cost of outlier removal steps.

In Table 2, we compared the average accuracy rate and average CPU time of our proposed subsample method with the scalable method, with five replications. The α was set to 0.1. For the digits data, the sample size was set to 0.1, while for the 20 news data, the sample size was set to 0.6. The proposed subsample model achieved a similar accuracy rate and a much smaller run time but with only a small fraction of the data.

dataset	scalable	subsample
usps	67.34% (3.7)	67.52% (1.4)
pendigits	73.48% (2.6)	72.32% (1.0)
mnist	52.60% (33.8)	53.19% (9.3)
20news	62.91% (17.74)	59.22% (13.2)
20news(SVD100)	73.41% (7.5)	70.93% (1.3)

Table 2: Clustering accuracy percentages (and CPU time in seconds) obtained by the scalable and subsample model on the real-world data.

2.2 Approach 2: Sequential Update

In the subsample approach, we use a sample dataset from the full dataset to approximate the spectral embedding of the full dataset and cluster the remaining dataset by projecting them into the approximated embedding space. However, in this way, the information in the remaining dataset was ignored. In our 2nd proposed approach, instead of simply projecting the remaining data into the sample SVD space, we found a way to sequentially update and improve the approximated embedding by the rest of the data.

2.2.1 Methodology

In this approach, we have the same assumptions as the previous approach that the dataset $\mathbf{X}$ is sparse with few non-zero entries on its rows or is low dimensional data where $n \gg d$ and that the similarity function is cosine similarity. Initially, we only know a random subset $\mathbf{X}_s = \begin{bmatrix} \mathbf{x}_1^T \\ \vdots \\ \mathbf{x}_s^T \end{bmatrix} \in \mathbb{R}^{s \times d}$ with size s from the whole dataset. Additionally, the remaining data points from dataset $\mathbf{X} \in \mathbb{R}^{n \times d}$ come in an online fashion.

The first stage of the sequential update algorithm is to find the approximated spectral embedding of the sample data and cluster them by k-means. The process in this stage is the same as the subsample method. We first approximate the $\mathbf{D}_s$ and calculate the $\tilde{\mathbf{X}}_s$. By applying SVD of $\tilde{\mathbf{X}_\mathbf{s}}$, $\tilde{\mathbf{U}_\mathbf{s}}\tilde{\mathbf{\Sigma}_\mathbf{s}}\tilde{\mathbf{V}_\mathbf{s}^\mathbf{T}} = \tilde{\mathbf{X}_\mathbf{s}}$, we embedded the

sample data points into the spectral embedding space by $\tilde{\mathbf{U}}_\mathbf{s}$. Then we apply K-means clustering to $\tilde{\mathbf{U}}_\mathbf{s}$ to cluster the sample data into k clusters. The difference between the sequential method and the subsample method is in how we handle the incoming sequential data in the rest of the dataset.

With each incoming new data point, we can have a better approximation of the full dataset. In the first stage, we used left singular vectors $\tilde{\mathbf{V}}_s$ from $\tilde{\mathbf{X}}_s$ to approximate the $\tilde{\mathbf{V}}$ from $\tilde{\mathbf{X}}$. With the incoming new datapoint, $\tilde{\mathbf{x}}_{s+1}$, the existing $\tilde{\mathbf{V}}_s$ can be updated to $\tilde{\mathbf{V}}_{s+1}$, which will be closer to $\tilde{\mathbf{V}}$. However, to perform rank-k SVD for $\tilde{\mathbf{X}}_{s+1} \in \mathbb{R}^{(n+1)\times d}$ to recalculate the $\tilde{\mathbf{V}}_{s+1}$ for each incoming data point is costly. We want to find a way to update the embedding in constant time for each data point.

$$
\begin{aligned}
\tilde{\mathbf{X}}^T\tilde{\mathbf{X}} &\approx \frac{n}{s+1}\tilde{\mathbf{X}}_{s+1}^T\tilde{\mathbf{X}}_{s+1} \\
&= \frac{n}{s+1}\left([\sum_{i=1}^{s}\frac{1}{d_i}\mathbf{x}_i\mathbf{x}_i^T + \frac{1}{d_{s+1}}\mathbf{x_{s+1}}\mathbf{x_{s+1}}^T]\right), \text{ where } \sum_{i=1}^{s}\frac{1}{d_i}\mathbf{x}_i\mathbf{x}_i^T = \tilde{\mathbf{X}}_s^T\tilde{\mathbf{X}}_s \\
&\approx \frac{n}{s+1}[\tilde{\mathbf{V}}_{s,k}\tilde{\mathbf{\Lambda}}_{s,k}\tilde{\mathbf{V}}_{s,k}^T + \frac{1}{d_{s+1}}\mathbf{x_{s+1}}\mathbf{x}_{s+1}^T] \\
&= \frac{n}{s+1}\tilde{\mathbf{V}}_{s,k}\tilde{\mathbf{\Lambda}}_{s,k}\tilde{\mathbf{V}}_{s,k}^T + \frac{n}{s+1}\frac{1}{d_{s+1}}\mathbf{x_{s+1}}\mathbf{x}_{s+1}^T \qquad (9) \\
&= \begin{bmatrix}\tilde{\mathbf{V}}_{s,k} & \mathbf{x_{s+1}}\end{bmatrix}\begin{bmatrix}\sqrt{\frac{n}{s+1}}\tilde{\mathbf{\Lambda}}_{s,k} & \\ & \sqrt{\frac{n}{(s+1)d_{s+1}}}\end{bmatrix}^2\begin{bmatrix}\tilde{\mathbf{V}}_{s,k}^T \\ \mathbf{x}_{s+1}^T\end{bmatrix} \\
&= \mathbf{B}^T\mathbf{B}, \text{ where } \mathbf{B}^T = \begin{bmatrix}\tilde{\mathbf{V}}_{s,k} & \mathbf{x_{s+1}}\end{bmatrix}\begin{bmatrix}\sqrt{\frac{n}{s+1}}\tilde{\mathbf{\Sigma}}_{s,k} & \\ & \sqrt{\frac{n}{(s+1)d_{s+1}}}\end{bmatrix}
\end{aligned}
$$

From equation (9), we can see that $\mathbf{B} \in \mathbb{R}^{(k+1)\times d}$ can be used to approximate the right singular vectors of $\tilde{\mathbf{X}}$ and $\sqrt{\frac{s+1}{n}}\mathbf{B}$ can be used to approximate $\tilde{\mathbf{X}}_{s+1}$. Since d and k are both constant and small, we can perform a low-rank SVD of $\mathbf{B}$ and find the updated approximate singular values, $\tilde{\mathbf{\Sigma}}_{(s+1),k}$, and right singular vectors, $\tilde{\mathbf{V}}_{(s+1),k}$, of $\tilde{\mathbf{X}}_{s+1}$ in constant time. Low rank SVD of $\mathbf{B}$ is:

$$\mathbf{Y}\tilde{\mathbf{\Sigma}}_{(s+1),k}\tilde{\mathbf{V}}_{(s+1),k}^T = \mathbf{B}, \text{ where } \tilde{\mathbf{\Sigma}}_{(s+1),k} \in \mathbb{R}^{k\times k}, \tilde{\mathbf{V}}_{(s+1),k} \in \mathbb{R}^{d\times k}$$

After we updated the singular vectors and values, we can project the new data point $\mathbf{x}_{s+1}$ into the approximated embedding space and update the embedding of the existing sample points $\mathbf{X}_s$ by:

$$\tilde{\mathbf{U}}_{s+1,k} = \tilde{\mathbf{X}}_{s+1}\tilde{\mathbf{V}}_{s+1,k}\tilde{\mathbf{\Sigma}}_{(s+1),k}, \text{ where } \tilde{\mathbf{X}}_{s+1} = \mathbf{D}_{s+1}^{-\frac{1}{2}}\mathbf{X}_{s+1} \text{ and } \mathbf{X}_{s+1} = \begin{bmatrix} \mathbf{X}_s \\ \mathbf{x}_{s+1} \end{bmatrix} \in \mathbb{R}^{(s+1)\times d} \tag{10}$$

The degree matrix of $\mathbf{X}_{s+1}$, $\mathbf{D}_{s+1}$, can be approximated by:

$$\begin{aligned}
\mathbf{D}_{s+1} &\approx diag(\frac{n}{s+1}(\mathbf{X}_{s+1}\mathbf{X}_{s+1}^T - \mathbf{I})\mathbf{1}) \\
&= diag(\frac{n}{s+1}\begin{bmatrix} \mathbf{X}_s \\ \mathbf{x}_{s+1}^T \end{bmatrix}\begin{bmatrix} \mathbf{X}_s^T & \mathbf{x}_{s+1} \end{bmatrix}\mathbf{1} - \mathbf{1}) \\
&= diag(\frac{n}{s+1}\begin{bmatrix} \mathbf{X}_s \\ \mathbf{x}_{s+1}^T \end{bmatrix}\begin{bmatrix} \mathbf{X}_s^T\mathbf{1} + \mathbf{x}_{s+1} \end{bmatrix} - \mathbf{1}) \\
&= diag(\frac{n}{s+1}\begin{bmatrix} \mathbf{X}_s\mathbf{X}_s^T\mathbf{1} + \mathbf{X}_s\mathbf{x}_{s+1} \\ \mathbf{x}_{s+1}^T\mathbf{X}_s^T\mathbf{1} + \mathbf{x}_{s+1}^T\mathbf{x}_{s+1} \end{bmatrix} - \mathbf{1}) \\
&= diag(\begin{bmatrix} \frac{s}{s+1}\mathbf{d}_s + \frac{n}{s+1}\mathbf{1} + \frac{n}{s+1}\mathbf{X}_s\mathbf{x}_{s+1} - \mathbf{1} \\ \frac{n}{s+1}\mathbf{x}_{s+1}^T\mathbf{X}_s^T\mathbf{1} + \frac{n}{s+1}\mathbf{x}_{s+1}^T\mathbf{x}_{s+1} - 1 \end{bmatrix}) \\
&= diag(\begin{bmatrix} \frac{s}{s+1}\mathbf{d}_s + \frac{n}{s+1}\mathbf{X}_s\mathbf{x}_{s+1} - \frac{n-s-1}{s+1} \\ \frac{n}{s+1}\mathbf{x}_{s+1}^T\mathbf{X}_s^T\mathbf{1} - \frac{n-s-1}{s+1} \end{bmatrix})
\end{aligned} \tag{11}$$

Finally, we cluster $\tilde{\mathbf{U}}_{s+1,k}$ into k clusters through k-means clustering initialized by the centroids, denoted by $\mathbf{C}_s$, that are found in the last iteration. Since the centroids found from the s data points in the last iteration should be very close to the updated centroids found from the $s+1$ data points, the initialized k-mean should be very fast with a constant time. And we recorded the centroids($\mathbf{C}_{s+1}$), singular vectors($\tilde{\mathbf{V}}_{s,k}$) and values($\tilde{\mathbf{\Sigma}}_{s,k}$) found in this iteration and increment s by 1. We then repeat the whole process for the next new data point, until the whole dataset is processed.

The analysis of this algorithm is more complex than the first approach one. Therefore, we use a table to list the major steps' costs to better illustrate the total cost.

Based on Table 2, the total time complexity of the algorithms is $O(ndk + n^2dk -$

Algorithm 4 Sequential update spectral clustering with cosine similarity

Input:data $\mathbf{X} \in \mathbb{R}^{n \times d}$, number of clusters k, number of subsamples s, fraction of outliers α

Output: Partition of k clusters, $C_1, ..., C_k$

1: Randomly sample s data points from the given data
2: Approximate the degrees $\mathbf{D}_s$ of the samples and find the outliers
3: Calculate the $\tilde{\mathbf{X}}_s$ for the sample dataset and find its top k left singular vectors by rank-k SVD: $\tilde{\mathbf{U}}_{s,k}\tilde{\Sigma}_{s,k}\tilde{\mathbf{V}}_{s,k}^T \approx \tilde{\mathbf{X}}_s$.
4: Normalized the rows of $\tilde{\mathbf{U}}_{s,k}$ with a norm equal to 1, and apply k-means clustering to cluster the samples.
5: For each of the new data points coming in the rest of the dataset, construct $\mathbf{B}$ by equation (4) and apply low-rank SVD on it.
6: Approximate the degree of the new data point and update the degree matrix $\mathbf{D}_{s+1}$ by equation (6).
7: Project the data into the updated approximate SVD space by equation (5) and find $\tilde{\mathbf{U}}_{s+1,k}$
8: Clustering the new data point by k-means initialized by the centorids found in the last iteration $\mathbf{C}_s$ based on s datapoints and record the new centorids $\mathbf{C}_{s+1}$
9: Repeat the same process for the next incoming data point.

Stage	each iteration cost	total cost
Cluster sample data	N/A	$O(sd + ksd)$
Construct $\mathbf{B}$	$O(dk)$	$O((n-s)dk)$
Approximate $\mathbf{D}_{s+1}$	$O(sd)$	$O(1/2(n-s)nd)$
SVD of $\mathbf{B}$	$O(k(k+1)d)$	$O((n-s)k(k+1)d)$
Compute $\tilde{\mathbf{U}}_{s+1,k}$	$O((s+1)kd)$	$O(1/2(n+1-s)(n+s)kd)$
Initialized k-means	$O(k)$	$O((n-s)k)$

Table 3: Break-down of the time complexity for each step in Sequential Update. The second column represents the cost of clustering sequential data points. The third column represents the total cost of clustering the whole dataset

s^2dk). If s is very large and close to n, the time complexity is $O(ndk)$. If s is very small and close to 1, the time complexity is $O(n^2kd)$.

Same with the subsample method, this approach solved the problem in the original spectral clustering model and the scalable cosine similarity model which is the clustering algorithm works only when the full dataset is available in memory. It allows us to apply spectral clustering algorithm on the online data elegantly. Unlike

the subsample method that only relies on the approximation quality from the sample data, with more and more data added, this approach updates the approximation and the results will be closer to the embedding of the full dataset. However, we use $\mathbf{X}_s$ when updating the degree matrix for the new datapoint, so we need to keep $\mathbf{X}_s$ in the storage. When s increases and approaches n, the storage needs increase. And since we update the singular vectors for each new coming data, when sample size is small, the time complexity becomes $O(n^2dk)$.

2.2.2 Experiment

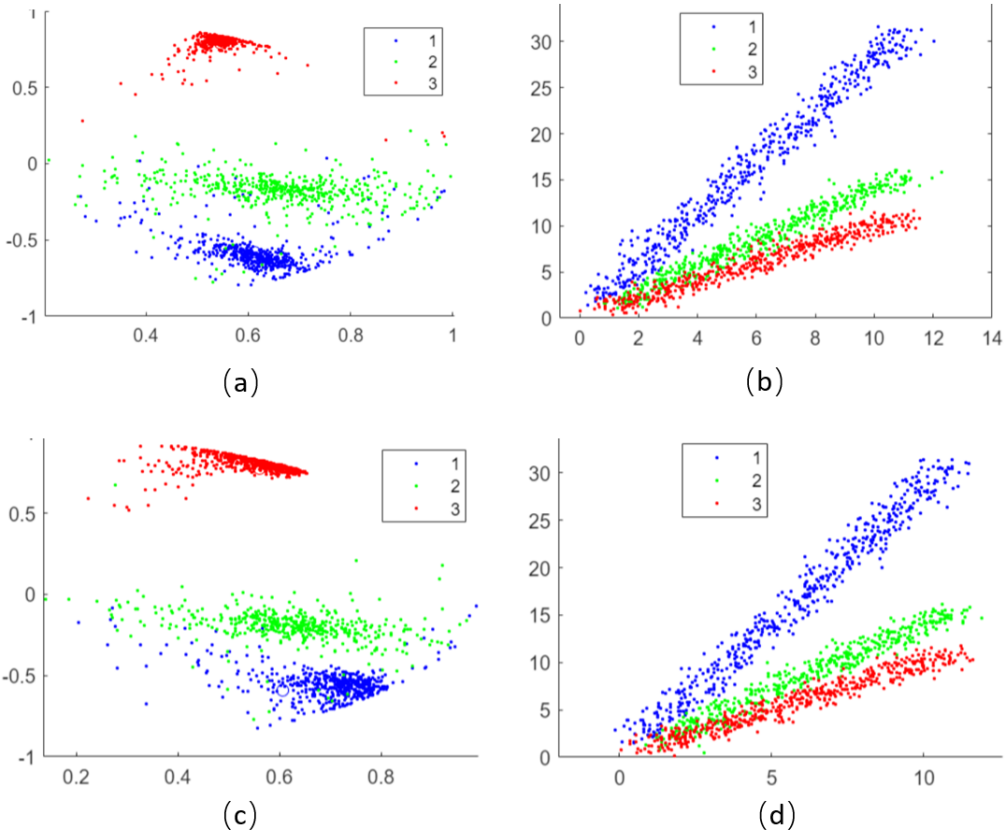


Figure 13: Visualization of the embeddings and clustering result using approach 2 with the toy dataset, $\alpha = 0.01$ and $s = 0.3$. (a) Spectral embeddings of the sample data. (b) Clustering result of the sample data. (c) Spectral embedding of the remaining data. (d) Clustering result of the remaining data

We applied approach 2 to the simulated dataset with 3 clusters. Figure 13 shows the scatter plots of the simulated data with cluster labels generated from approach 2. It also shows the embeddings of the simulated data in the approximated eigenspace.

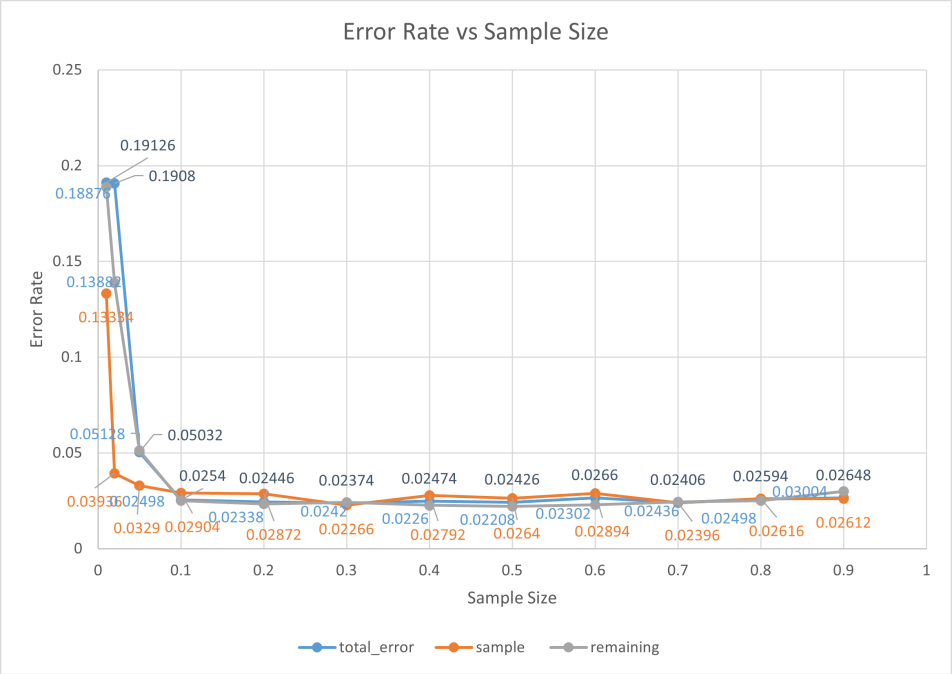


Figure 14: The error rate of the toy dataset with different sample sizes for approach 2

We apply the approach 2 algorithms on the simulated dataset and recorded the error rates of the sample dataset as well as the remaining data points under different sample sizes(α) of 0.01, 0.02, 0.05, 0.1,...0.9. To account for the randomness, the experiments were replicated 5 times. In Figure 10, the blue line represents the average error rates of the total data. The orange line represents the average error rates of the sample set data, and the grey line represents the remaining data. The error rates of the total data, sample data, and remaining data all drop dramatically when the sample size increase from 0.01 to 0.1. And they converge to near 0.0254 when the sample size equals 0.1. The trends of the three lines coincide with each other. There is no obvious difference in the error rates between the sample and the remaining dataset.

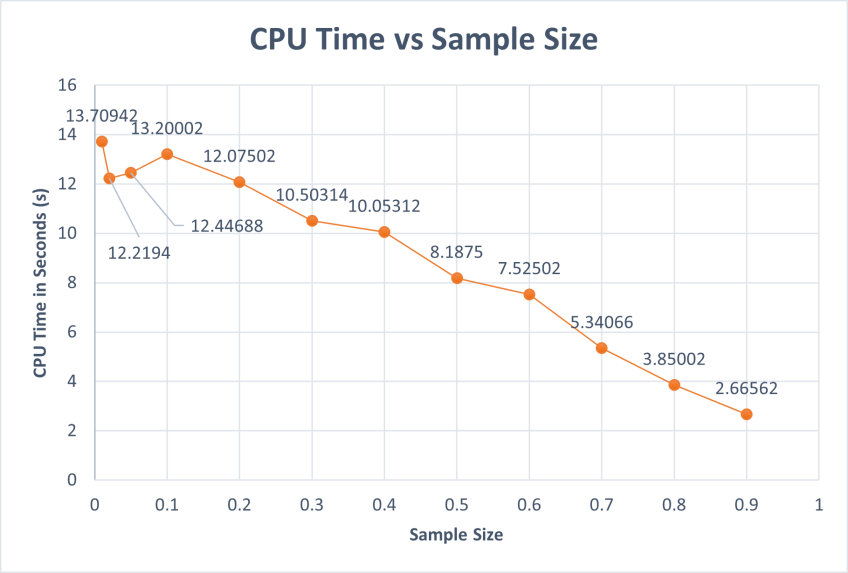


Figure 15: The CPU time of the toy dataset with different sample sizes for approach 2

Based on Figure 15, there is a clear decreasing trend in the CPU time with increasing sample size. The trend coincided with our calculation of time complexity in the big O notation which indicates that the sample size s has a direct linear relationship with the computational time.

To test the algorithm in a real-world dataset, we apply the same experiments to the USPS dataset. The USPS dataset has pictures of handwritten numbers from 0 to 1, which indicates 10 natural clusters. We recorded the error rates of the sample and the remaining data under different sample sizes. In Figure 16, the blue line represents the error rates of the whole dataset. The orange line represents the error rates of the sample data while the gray one represents the remaining sequential data. Overall, the error rates decrease with the increase of the sample size until the sample size reaches 0.1. At the sample size of 0.1, the total error rate, sample error rate, and remaining error rate all reach minimums of 0.031804, 0.024712, and 0.031022, respectively.

From Figure 17, similar to what happened on the simulated dataset, the CPU time of applying the approach 2 algorithms to the USPS dataset decreases with the

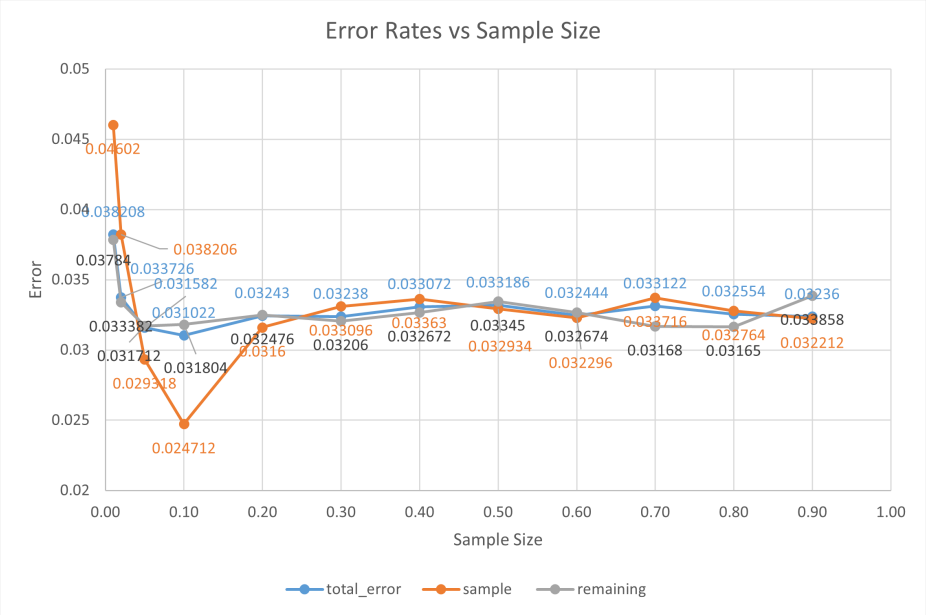


Figure 16: The error rate of the USPS dataset with different sample sizes for approach 2

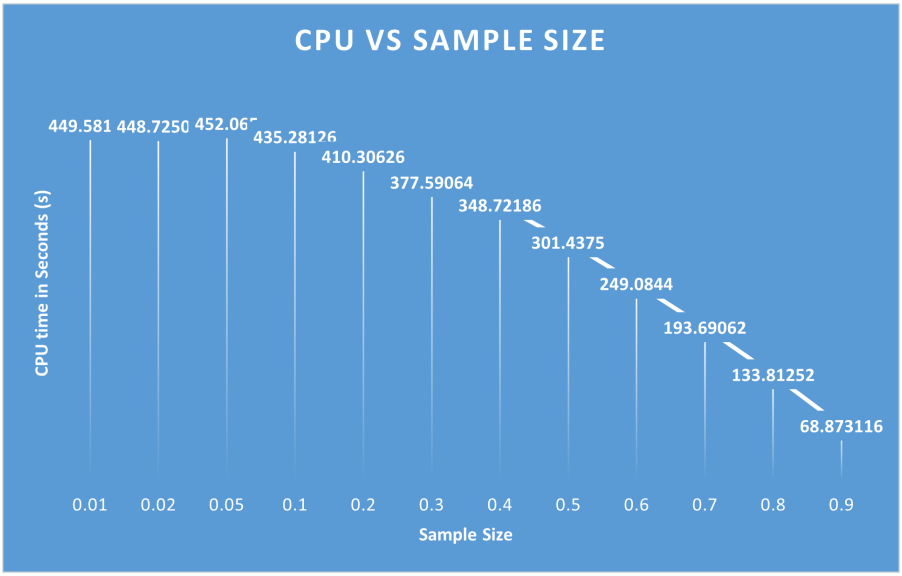


Figure 17: The CPU time of the USPS dataset with different sample sizes for approach 2

increase of the sample size. The computational cost ranges from about 449 s to about 68 s, which is much higher than that of approach 1 which is about 3 seconds.

dataset	scalable	sequential
usps	67.34% (3.7)	68.98% (449.28)
pendigits	73.48% (2.6)	73.58% (352)
mnist	52.60% (33.8)	53.1% (2433)
20news	62.91% (17.74)	61.3% (1256)
20news(SVD100)	73.41% (7.5)	72.9 % (598)

Table 4: Clustering accuracy percentages (and CPU time in seconds) obtained by the scalable and sequential model on the real-world data.

2.3 Approach 3. Batch update

The Sequential Update method solved the problem that spectral clustering cannot handle the online data format. In the batch updates, we propose another method assuming that the incoming data are in a small batch manner. The methodology is similar to the sequential update method except that instead of handling one incoming data point at a time, it handles a small batch with size t at a time.

2.3.1 Methodology

In this approach, we have a similar assumption as the previous approach that the dataset $\mathbf{X}$ is sparse with few non-zero entries on its rows or is low dimensional data where $n \gg d$ and its similarity function is cosine similarity. Initially, we only know a random subset $\mathbf{X}_s = \begin{bmatrix} \mathbf{x}_1^T \\ \vdots \\ \mathbf{x}_s^T \end{bmatrix} \in \mathbb{R}^{s \times d}$ with size s from the whole dataset. Different from approach 2, we assume that the remaining data points from dataset $\mathbf{X} \in \mathbb{R}^{n \times d}$ come in as a small batch with size t at a time, $1 \leq t \leq s$.

The method for handling the sample data $\mathbf{X}_s$ is the same as the Sequential Update method, but the method for handling the rest of the dataset is slightly different. For a new batch t, the dataset become $\mathbf{X}_{s+t} = \begin{bmatrix} \mathbf{X}_s \\ \mathbf{X}_t \end{bmatrix} \in \mathbb{R}^{(s+t) \times d}$. Similar to the Sequential Update, in each new batch, we use a matrix $\mathbf{B}$ to approximate the full dataset by equation 7.

$$
\begin{aligned}
\tilde{\mathbf{X}}^T\tilde{\mathbf{X}} &\approx \frac{n}{s+t}\tilde{\mathbf{X}}_{s+t}^T\tilde{\mathbf{X}}_{s+t} \\
&= \frac{n}{s+t}\begin{bmatrix}\tilde{\mathbf{X}}_s^T & \tilde{\mathbf{X}}_t^T\end{bmatrix}\begin{bmatrix}\tilde{\mathbf{X}}_s \\ \tilde{\mathbf{X}}_t\end{bmatrix} \\
&= \frac{n}{s+t}[\tilde{\mathbf{X}}_s^T\tilde{\mathbf{X}}_s + \tilde{\mathbf{X}}_t^T\tilde{\mathbf{X}}_t] \\
&\approx \frac{n}{s+t}[\tilde{\mathbf{V}}_{s,k}\tilde{\mathbf{\Lambda}}_{s,k}\tilde{\mathbf{V}}_{s,k}^T + \frac{1}{d_{s+t}}(\mathbf{X}_t^T\mathbf{D}_t^{-\frac{1}{2}})(\mathbf{D}_t^{-\frac{1}{2}}\mathbf{X}_t)] \\
&= \begin{bmatrix}\tilde{\mathbf{V}}_{s,k} & \tilde{\mathbf{X}}_t^T\end{bmatrix}\begin{bmatrix}\sqrt{\frac{n}{s+t}}\tilde{\mathbf{\Lambda}}_{s,k} & \\ & \sqrt{\frac{n}{s+t}}\mathbf{I}\end{bmatrix}^2\begin{bmatrix}\tilde{\mathbf{V}}_{s,k}^T \\ \tilde{\mathbf{X}}_t\end{bmatrix} \\
&= \mathbf{B}^T\mathbf{B}, \text{ where } \mathbf{B}^T = \begin{bmatrix}\tilde{\mathbf{V}}_{s,k} & \tilde{\mathbf{X}}_t^T\end{bmatrix}\begin{bmatrix}\sqrt{\frac{n}{s+1}}\tilde{\mathbf{\Sigma}}_{s,k} & \\ & \sqrt{\frac{n}{s+t}}\mathbf{I}\end{bmatrix} \in \mathbb{R}^{d\times(k+t)}
\end{aligned} \tag{12}
$$

After constructing the matrix $\mathbf{B}$, we update the left singular vectors from $\tilde{\mathbf{V}}_{s,k}$ to $\tilde{\mathbf{V}}_{(s+t),k}$ using the SVD of the $\mathbf{B}$ by equation 8.

$$
\mathbf{Y}\tilde{\mathbf{\Sigma}}_{(s+t),k}\tilde{\mathbf{V}}_{(s+t),k}^T = \mathbf{B}, \text{ where } \tilde{\mathbf{\Sigma}}_{(s+t),k} \in \mathbb{R}^{k\times k}, \tilde{\mathbf{V}}_{(s+t),k} \in \mathbb{R}^{d\times k} \tag{13}
$$

Then we projected the $s+t$ data points into the new SVD space by the following equation.

$$
\tilde{\mathbf{U}}_{s+t,k} = \tilde{\mathbf{X}}_{s+t}\tilde{\mathbf{V}}_{s+t,k}\tilde{\mathbf{\Sigma}}_{(s+t),k}, \text{ where } \tilde{\mathbf{X}}_{s+t} = \mathbf{D}_{s+t}^{-\frac{1}{2}}\mathbf{X}_{s+t} \tag{14}
$$

In both steps of constructing the matrix $\mathbf{B}$ and projecting the $s+t$ data points, the degree matrix of the new small batch dataset $\mathbf{D}_t$ and the existing dataset $\mathbf{D}_s$ were used. Since we have t more data points than before, we can update the degree matrix

of the existing dataset $\mathbf{D}_s$ to have a more accurate approximation.

$$\begin{aligned}
\mathbf{D}_{s+t} &\approx \frac{n}{s+t} diag((\mathbf{X}_{s+t}\mathbf{X}_{s+t}^T - \mathbf{I})\mathbf{1}) \\
&= \frac{n}{s+t} diag(\begin{bmatrix}\mathbf{X}_s \\ \mathbf{X}_t\end{bmatrix}\begin{bmatrix}\mathbf{X}_s^T & \mathbf{X}_t^T\end{bmatrix}\mathbf{1} - \mathbf{1}) \\
&= \frac{n}{s+t} diag(\begin{bmatrix}\mathbf{X}_s \\ \mathbf{X}_t\end{bmatrix}\begin{bmatrix}\mathbf{X}_s^T\mathbf{1} + \mathbf{X}_t^T\mathbf{1}\end{bmatrix} - \mathbf{1}) \\
&= \frac{n}{s+t} diag(\begin{bmatrix}\mathbf{X}_s\mathbf{X}_s^T\mathbf{1} + \mathbf{X}_s\mathbf{X}_t^T\mathbf{1} - \mathbf{1} \\ \mathbf{X}_t\mathbf{X}_s^T\mathbf{1} + \mathbf{X}_t\mathbf{X}_t^T - 1\end{bmatrix}) \\
&= \frac{n}{s+t} diag(\begin{bmatrix}\frac{s}{n}\mathbf{d}_s + \mathbf{X}_s\mathbf{X}_t^T\mathbf{1} \\ \mathbf{X}_t\mathbf{X}_s^T\mathbf{1} + \mathbf{X}_t\mathbf{X}_t^T\mathbf{1} - 1\end{bmatrix})
\end{aligned} \tag{15}$$

During updating the approximation of the degrees, we recorded the updated $\mathbf{X}_s^T\mathbf{1}$ and $\mathbf{d}_s$ in each iteration to save some computation time. The computational time cost in this step is from $\mathbf{X}_t(\mathbf{X}_s^T\mathbf{1}) = O(sdt)$, $\mathbf{X}_t(\mathbf{X}_t^T\mathbf{1}) = O(td)$, and $\mathbf{X}_s(\mathbf{X}_t^T\mathbf{1}) = O(sd+td)$

After the process of each small batch t data points, s increment with t, $s = s + t$ and the computational results, such as the new singular values, $\tilde{\mathbf{\Sigma}}_{(s+t),k}$, singular vectors $\tilde{\mathbf{V}}_{(s+t),k}$ and the new approximated degrees $\mathbf{D}_{s+t}$ are recorded and carried on to the calculation of the next small batch, t. The whole process will repeat and continue until the whole data set has been processed.

Stage	each iteration cost	total cost
Cluster sample data	N/A	$O(sd + ksd)$
Construct $\mathbf{B}$	$O(dk)$	$O(\frac{n-s}{t}dk)$
Approximate $\mathbf{D}_{s+t}$	$O(td + sd)$	$O(nd - sd + \frac{n^2}{2t}d - \frac{s^2}{2t}d)$
SVD of $\mathbf{B}$	$O(k(k+t)d)$	$O(\frac{n-s}{t}k(k+t)d)$
Compute $\tilde{\mathbf{U}}_{s+t,k}$	$O((s+t)kd)$	$O(\frac{n^2}{2t}dk - \frac{s^2}{2t}dk)$
Initialized k-means	$O(k)$	$O(\frac{n-s}{t}k)$

Table 5: Break-down of the computational time cost for each step in Batch Update. The second column represents the cost of clustering sequential data points. The third column represents the total cost of clustering the whole dataset

When $t = 1$, the time complexity of Batch Update is the same as the Sequential Update method. When t or s are very large, the time complexity decreases to $O(ndk)$.

Algorithm 5 Small Batch Update spectral clustering with cosine similarity

Input:data $\mathbf{X} \in \mathbb{R}^{n \times d}$, number of clusters k, number of subsamples s, size of small batch t, fraction of outliers α

Output: Partition of k clusters, $C_1, ..., C_k$

1: Randomly samples s data points from the given data
2: Approximate the degrees $\mathbf{D}_s$ of the samples and find the outliers
3: Calculate the $\tilde{\mathbf{X}}_s$ for the sample dataset and find its top k left singular vectors by rank-k SVD: $\tilde{\mathbf{U}}_{s,k}\tilde{\Sigma}_{s,k}\tilde{\mathbf{V}}_{s,k}^T \approx \tilde{\mathbf{X}}_s$.
4: Normalized the row of left singular vectors with norm equals 1, and apply k-means clustering to cluster the samples.
5: For each of the new small batches coming in the rest of the dataset, construct $\mathbf{B}$ by equation 9 and apply low-rank SVD on it.
6: approximate the degree of the new data point and update the degree matrix $\mathbf{D}_{s+t}$ by equation 2.3.1.
7: Project the data into the updated approximate SVD space by equation 7 and find $\tilde{\mathbf{U}}_{s+t,k}$
8: Clustering the new data point by k-means initialized by the centorids found in the last iteration $\mathbf{C}_s$ based on s datapoints and record the new centroids $\mathbf{C}_{s+t}$
9: Increment s = s+t and repeat the same process to the next incoming data point.

When t or s are very small, the time complexity increase to $O(n^2dk)$ because of the computation of $\mathbf{D}_{s+t}$ and $\tilde{\mathbf{U}}_{s+t,k}$ during the process of each batches. This approach enables us to apply spectral clustering on the dataset that is constituted by small batches. It provides a reliable method to gradually approach the true optimal error by processing and updating one batch of data at a time. But like the sequential update method, it still requires storing X_s in the memory to calculate the degree matrix for each batch. With s approaching n, the storage needs and the computational cost increase. This will be a limitation of the application of this method.

2.3.2 Experiment

To test the effectiveness of approach 3, we first apply the algorithm to the simulated toy dataset with three clusters. The scatter plots of this toy dataset and its embedding by approach 3 was shown in figure 18.

To view the process of the batch updates, we recorded and plotted the accumulated

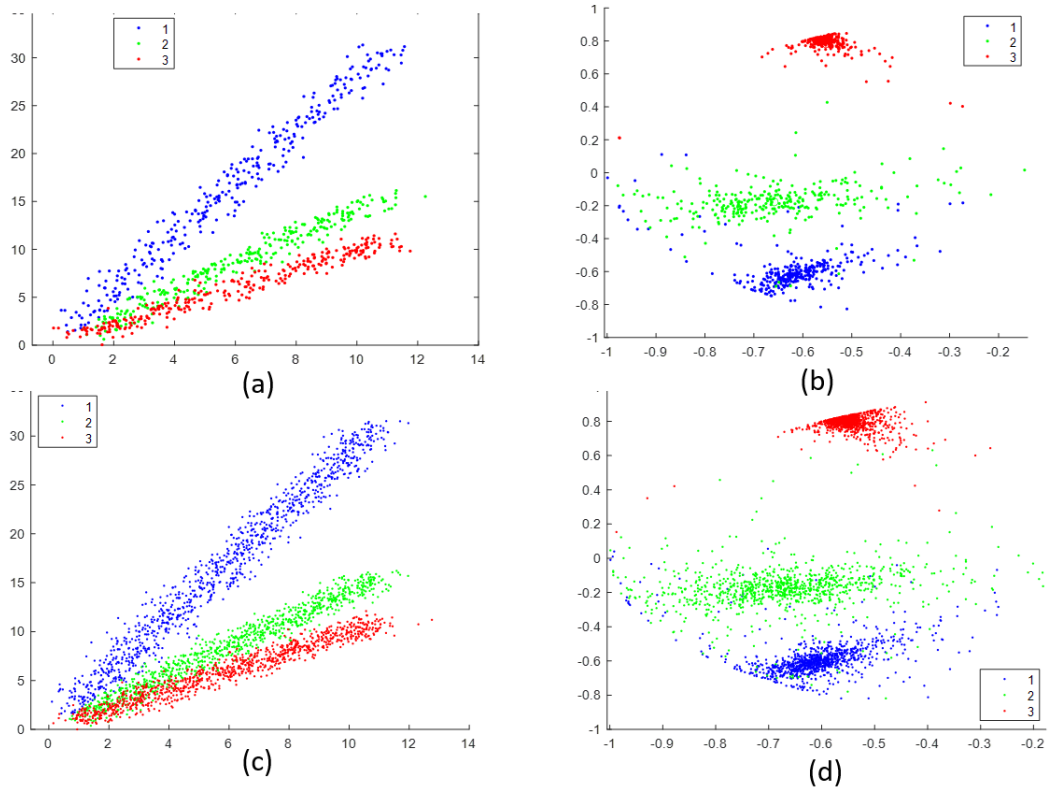


Figure 18: Visualized the embedding and clustering result by Approach 3 in the toy dataset, $\alpha = 0.01$ and $s = 0.3$ and $t = 30$. (a) Scatter plot of the sample with cluster numbers. (b) Spectral embeddings of the sample data. (c) Scatter plot of the remaining data with cluster numbers. (d)Spectral embeddings of the remaining data.

error rates of each added batch data, under the different settings of the batch size. From Figure 19, when batch size equals 30, 50, and 100, the accumulated error rates decrease gradually with the increase of the batches. This indicates that with each added batch, the approximated embedding space was closer to the true embedding space. When batch size equals 200 data points, in the first batch, the algorithm has already been able to produce about 0.025 error rates which are close to the optimum, so the error rates line does not have a decreasing trend.

In the batch update method, initially, we have a sample set. And with the new batch becoming available one at a time, we update the projections by the new batch

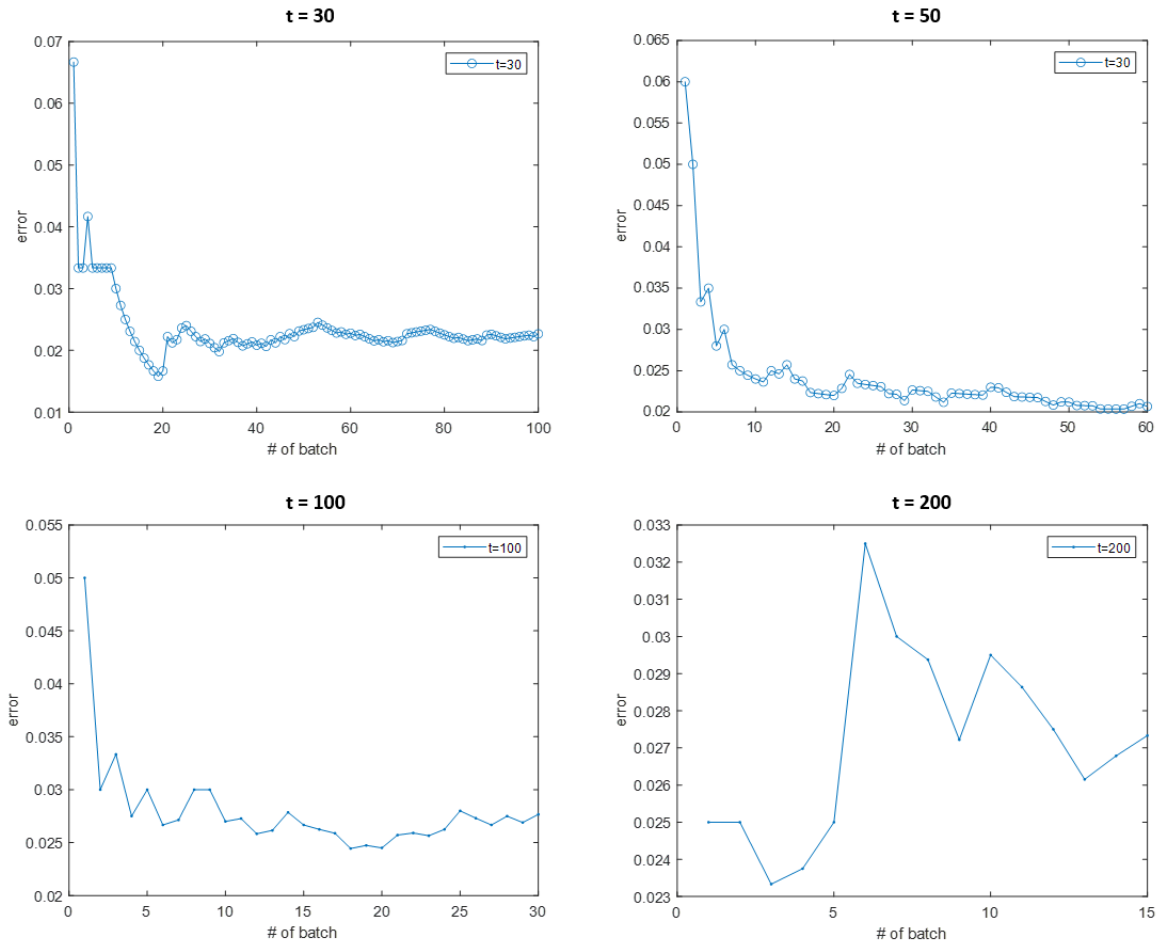


Figure 19: Accumulated error rates of each batch in the toy dataset by approach 3 vs number of batches, sample size s = batch size t

one at a time by modifying the projection matrices. This is equivalent to increasing the sample pool. To study the influence of the size of the initial sample on the algorithm performance, we recorded the total error rates of the toy data with different sizes of the initial sample in Figure 20. The different color lines represent different batch sizes. The yellow line represented batch size = 200 is very volatile. Overall, we found that the initial sample size does not have a significant impact on the overall accuracy of the model in the toy data.

In Figure 21, under the same sample size, a higher batch size leads to a higher

Figure 20: The total error rate of the toy dataset with different sample sizes (s) and different batch size (t) for approach 3

CPU time. Overall, the CPU time increased with the increase in the sample size. However, unlike the CPU time plot in approach 2 which shows a very clear linear relationship between the sample size and CPU time, the plot in figure 19 shows a more complex relationship between the sample size and time. Overall the cost of computational time is much lower in approach 3 compared to approach 2.

After applying the experiment on the simulated toy dataset, we performed the same experiment on the USPS data. Figure 22 shows the accumulated error rates with each batch under different settings of batch size, t. When t is equal to 30, 50, 100, and 200, overall the accumulated error rate decrease when more batches of data points were added to update the embedding.

Figure 23 shows the error rates of the whole dataset corresponding to different initial sample sizes, s, and different batch sizes, t. From the plot, we found that wen batch size is equal to 30, 50, and 100, the initial sample size does not have a large impact on the error rate. Overall, the lines corresponding to a larger batch size, such as 200, are positioned above the lines corresponding to a smaller batch size, which

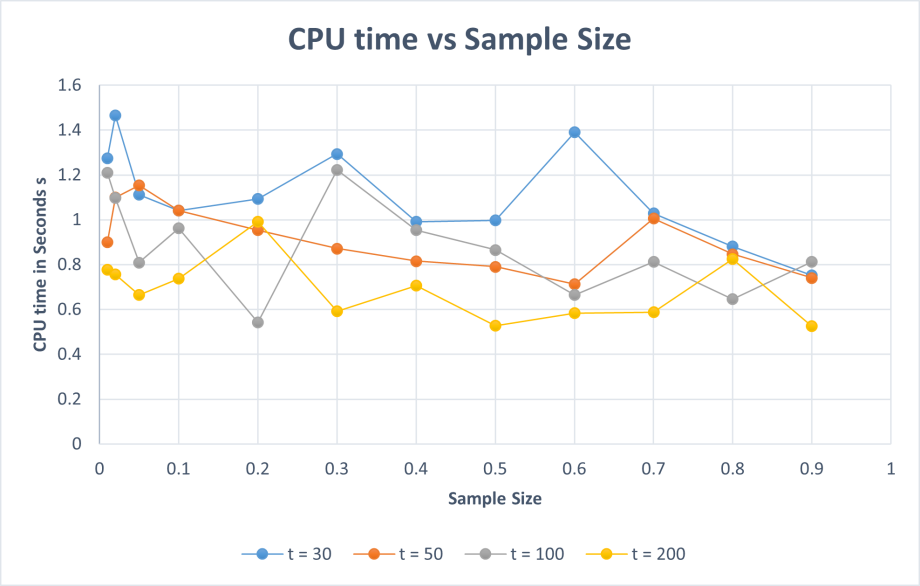


Figure 21: The CPU time of the toy dataset with different sample sizes (s) and different batch size (t) for approach 3

indicates that under the same initial sample size, a setting of a smaller batch size such as 30 is better than a larger batch size, such as 200.

From Figure 24, overall when batch size, t, is equal to 30, 50, and 100, with the increase of the sample size, the computational time decrease. However, this decreasing trend was compensated by the increase of t. When t relative to n is small, such as 30, the behavior of the computational time in approach 3 under different sample sizes, s, is similar to that in approach 2. However, when t is very, the computational cost from $O(ndk + \frac{n^2}{2t}dk - \frac{s^2}{2t}dk)$ reduces to $O(ndk)$. The yellow line representing t = 200 is flat indicating that computational cost does not change with the sample size.

2.3.3 Convergence Criteria

We mentioned that there is a computational limitation in the sequential and batch update methods. One simple way to solve this issue is by monitoring the convergence and stopping updates earlier. Updating the sample embedding by more incoming batches is equivalent to increasing the sample set. With more batch of data

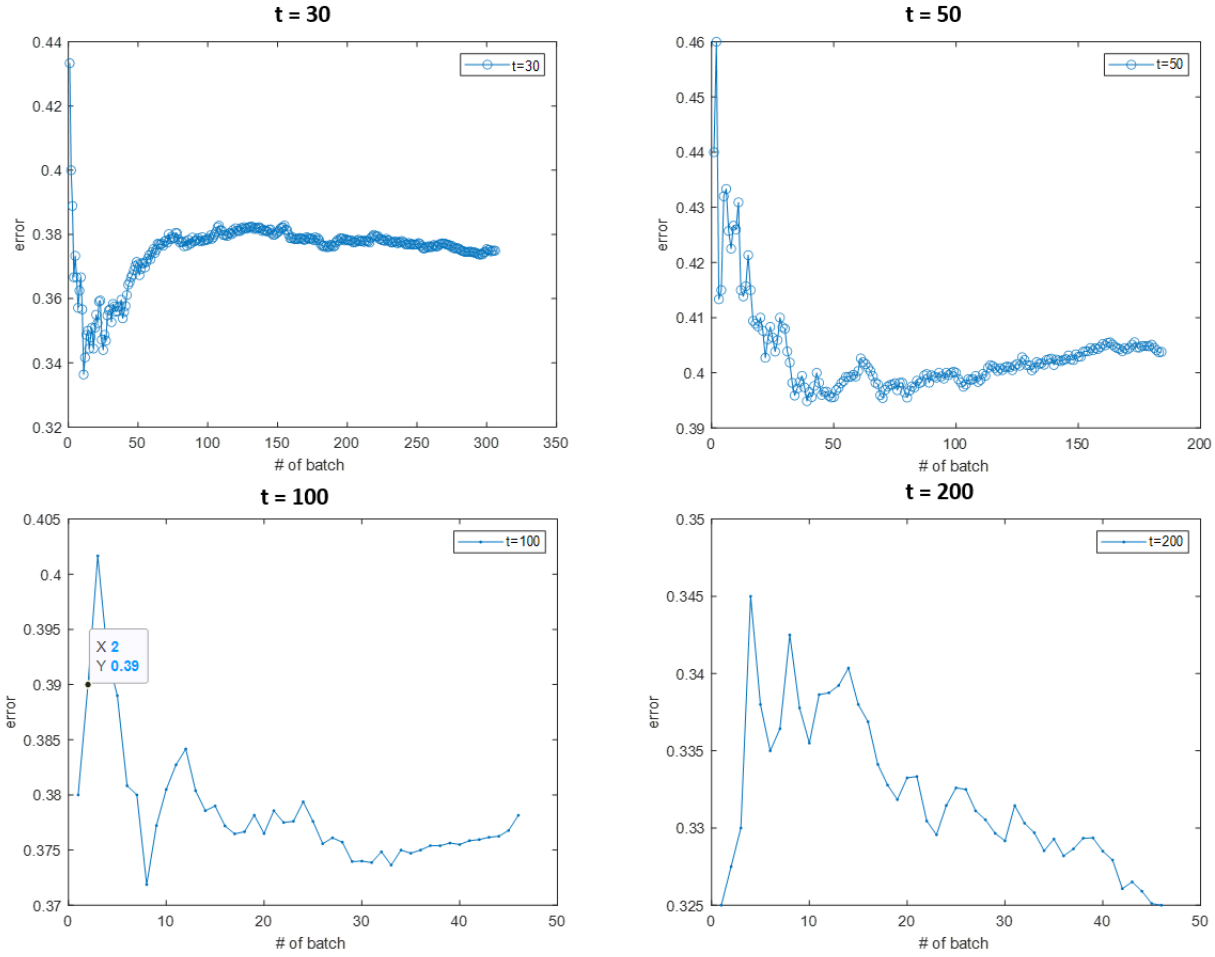


Figure 22: Accumulated error rates of each batch in the USPS dataset by approach 3 vs number of batches, sample size s = batch size t

added, the approximation gets closer to the real embedding, and the error decrease. When the sample size is large enough, the error rate converges and stops decreasing.

The principal angle is a good measurement of the distance between different spaces. It is calculated by the following equation:

$$dist^2(E_k(\mathbf{Z}), E_k(\tilde{\mathbf{Z}})) = 2 \times \sum_{k=1}^{K} sin^2\theta_k \tag{16}$$

We can record the updated maps, $\tilde{\mathbf{V}}_\mathbf{s}$, after each new batch and calculate the principal angles between the current maps and the previous maps. When the principal angle

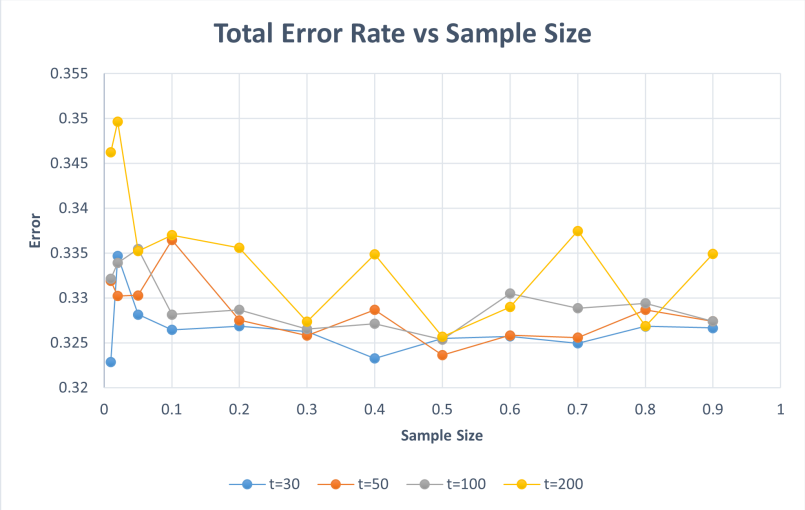


Figure 23: The total error rate of the USPS dataset with different sample sizes (s) and different batch size (t) for approach 3

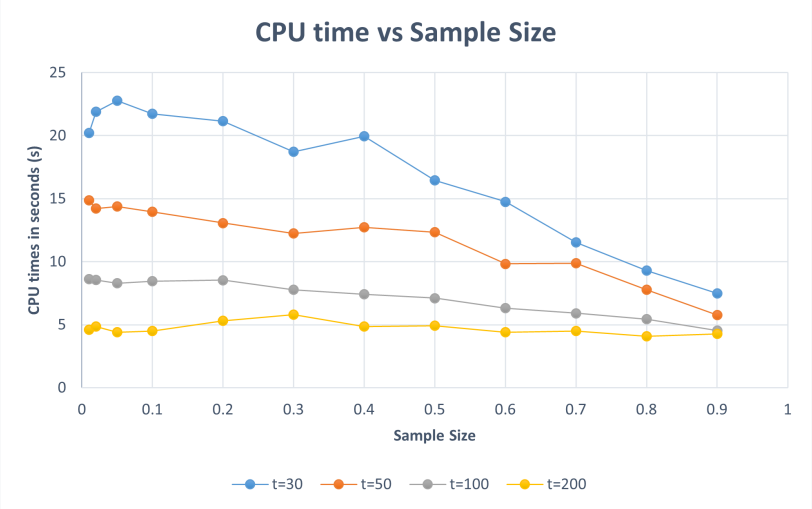
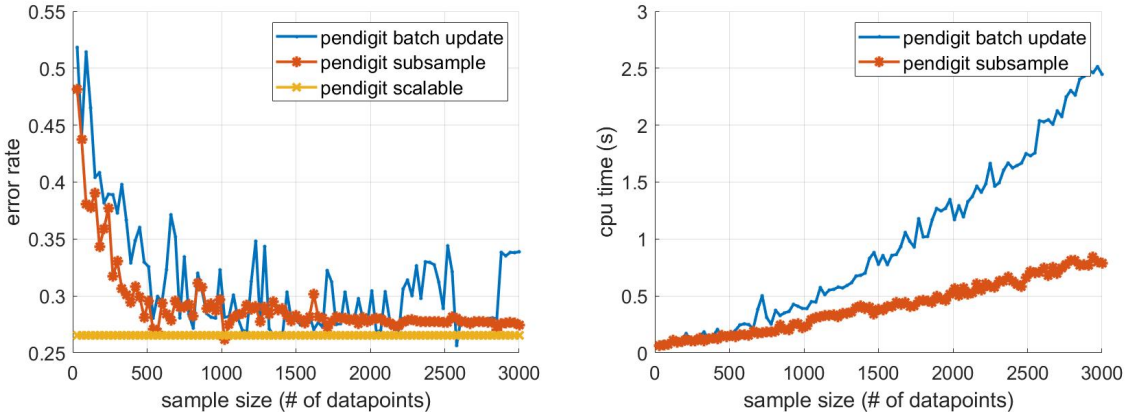


Figure 24: The CPU time of the USPS dataset with different sample sizes (s) and different batch size (t) for approach 3

becomes very small, such as 1 degree, the embedding converges and we stop the update.

Since updating the sample embedding by each new batch is equivalent to increasing the sample size, the batch update method can be compared with the Subsample method. For example, we initially have the 1st batch, 30 sample points. Then with the 2nd

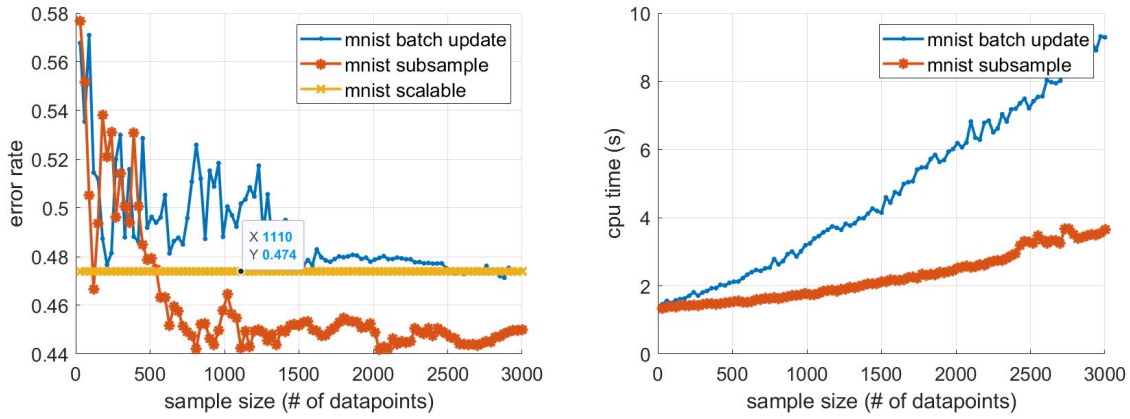
batch of 30 data added, the batch update method can use equation (13) to efficiently update the $\tilde{\mathbf{V}}_\mathbf{s}$, while the Subsample method needs to perform the SVD again on the 60 sample data points to find the new $\tilde{\mathbf{V}}_\mathbf{s}$. We performed experiments on the digit data sets to monitor the convergence with respect to the increasing sample size (1 to 3000) and the number of batches (1 to 100), with a batch size equal to 30. α is set to 0.01. Since there is randomness, we replicate the experiment 5 times and take the average. From Figure 25a 26a and 27a, with the increase of the sample size, both error rates in subsample method and batch update method decrease for all the three data sets and approach the error rates of the scalable method when the sample size is around 1000 to 1500. Overall, the Subsample method produces a smoother curve and converges faster.



(a) The error rates (b) The CPU time

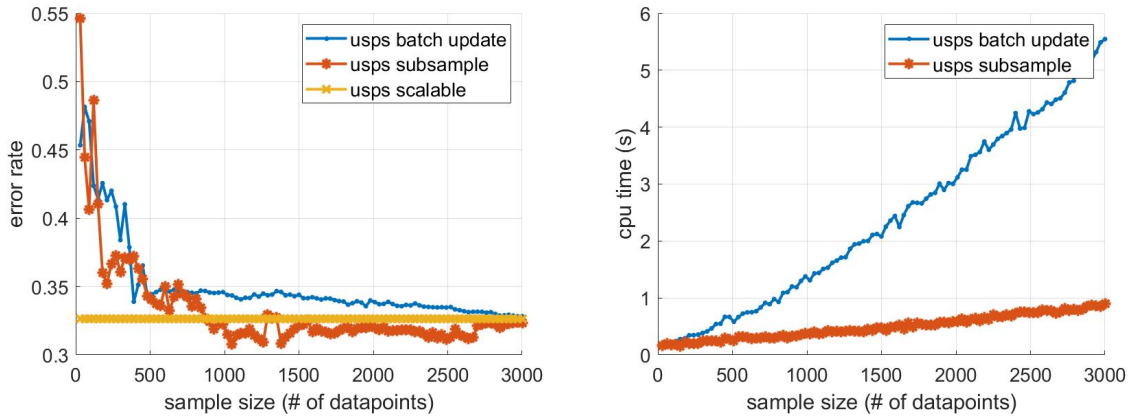
Figure 25: Error rates and CPU times of pendigit dataw ith different sample size

We recorded and monitored the principal angles produced by each batch from the batch update method for the three data sets and found that the convergence in the principal angle plots coincides with the error plots, shown in Figure 28. The red horizontal line represents degree equals 2. This indicates that we can use the principal angle as an indication of where to stop the updates.



(a) The accuracy rate (b) The CPU time

Figure 26: Error rates and CPU times of mnist data with different sample size



(a) The accuracy rate (b) The CPU time

Figure 27: Error rates and CPU times of USPS data with different sample size

dataset	**scalable**	**subsampling**	**sequential**
usps	67.34% (3.7)	67.52% (1.4)	68.98% (1.1)
pendigits	73.48% (2.6)	72.32% (1.0)	72.04% (0.6)
mnist	52.60% (33.8)	53.19% (9.3)	53.62% (4.3)
20news	62.91% (17.74)	59.22% (13.2)	61.02% (11.6)
20news(SVD100)	73.41% (7.5)	70.93% (1.3)	67.51% (0.6)

Table 6: Clustering accuracy percentages (and CPU time in seconds) obtained by the scalable, subsample and batch update on the real-world data.

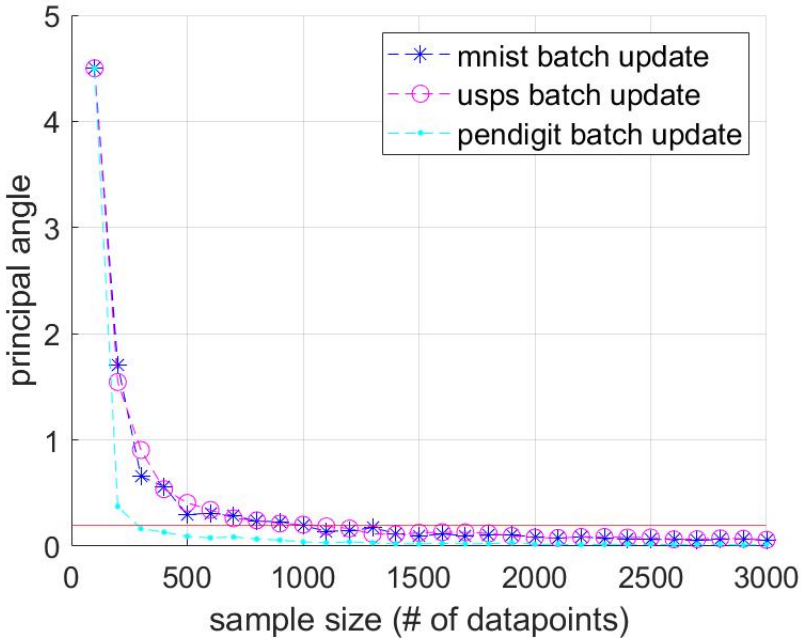


Figure 28: Principal angles from the batch update method

In Table 6, we compared the scalable, subsample, and batch update methods with the cosine similarity on all the real-wolrd datasets and report the performance in terms of clustering accuracy and CPU time. The sample size is set to 1500 for both methods. And for the batch update, batch size is set to 30. We see that for the USPS data, the two proposed memory scalable methods obtained slightly higher accuracy rates than the scalable algorithm while running faster. All the datasets had a comparable or slightly better accuracy rate in subsample method when compared to the scalable method. Among the three algorithms, the batch update has the fastest computation time on all of the datasets. These results suggested that the proposed memory scalable methods further reduce the computation time compared to the scalable cosine similarity algorithm without sacrificing accuracy. Compared to the subsample method, sequentially updating the spectral embedding with each available batch leads to a more efficient procedure with faster computation and higher accuracy.

After using the convergence criteria, the time and space complexity becomes:

- Time complexity: $O(tk + (k + t)d + skd)$
- Space complexity: $O((k + t)d)$

CHAPTER 3

Conclusion

In this report, we introduced the necessary background of our research, such as the spectral clustering algorithm and the scalable spectral clustering with cosine similarity. We then proposed three new approaches to reduce the space requirement in spectral clustering and extend the application of spectral clustering to handle the sequential data. Moreover, we applied the proposed algorithm on both a simulated dataset and several real-world datasets to test the effectiveness of the proposed approaches. Based on the experimental result, the new approaches achieved a performance as good as the baseline approach with much less space needed to initiate the algorithm. From the experiment, the approaches helped us successfully apply the spectral clustering algorithm to the online data.

Our research goal is to propose scalable spectral clustering methods to handle the situation that the data is too large to be fully loaded into the memory and the data is distributed in an online format. To achieve this goal, we first introduced approach 1, the subsample method, which is the fastest method with a well enough performance. However, in this approach, the quality of the sample is crucial since it only uses the sample set to approximate the full data set. To address this problem we proposed approach 2, sequential, which performs better than approach 1 in accuracy rate because it improves approach 1 by updating the embedding space with each incoming new data point. However, since it is required to approximate the degree and do low-rank SVD for each incoming data, it has a much higher computational cost. The last method, approach 3, further extends the approach to enable us to apply spectral clustering on data coming in batches. In this method, we also solve the computational cost problem in approach 2 by introducing convergence measurement criteria. Our approach 3, the batch update method only requires a small fraction

of the data and achieves a very good performance in the experiments. Our current assumption is that the similarity function is cosine similarity. In the future, we will continue to work on generalizing this method to handle other similarity functions.

LIST OF REFERENCES

[1] G. Chen, ''A scalable spectral clustering algorithm based on landmark embedding and cosine similarity,'' in *IAPR Joint International Workshops on Statistical Techniques in Pattern Recognition (SPR 2018) and Structural and Syntactic Pattern Recognition (SSPR 2018)*, Fragrant Hill, Beijin, 2018.

[2] G. Chen, ''Scalable spectral clustering with cosine similarity,'' in *The 24th International Conference on Pattern Recognition (ICPR)*, Beijin, China, 2018.

[3] U. von Luxburg, ''A tutorial on spectral clustering,'' *Statistics and Computing*, vol. 17, no. 4, p. 395–416, 2007.

[4] K. Briggs, ''English and latin digram and trigram frequencies,'' http://keithbriggs.info/documents/english_latin.pdf, 2013, (Accessed on 07/28/2014).

[5] M. Meila and J. Shi, ''A random walks view of spectral segmentation,'' in *8th International Workshop on Artificial Intelligence and Statistics (AISTATS).*, 2001.

[6] M. J. A. Ng and Y. Weiss, ''On spectral clustering: Analysis and an algorithm,'' *Advances in Neural Information Processing Systems 14*, pp. 849--856, 2001.

[7] G. Chen, ''Matlab implementation details of a scalable spectral clustering algorithm with cosine similarity,'' in *The 2nd Workshop on Reproducible Research in Pattern Recognition (RRPR 2018)*, Beijing, China, August 2018.

[8] K. Pham and G. Chenn, ''Large-scale spectral clustering using diffusion coordinates on landmark-based bipartite graphs,'' in *The 12th Workshop on Graph-based Natural Language Processing (TextGraphs-12)*, New Orleans, Louisiana, June 2018.

[9] G. Chen, ''A general framework for scalable spectral clustering based on document models,'' *Pattern Recognition Letters, 125*, pp. 488--493, 2019.