IMPROVING CLUSTERING WITH GRAPH SMOOTHING

Senior Honors Thesis

submitted by

Charlie Colley

in

Mathematics

TUFTS UNIVERSITY

May 2016

Advisors: Xiaozhe Hu, Lenore Cowen

Abstract

Clustering is a data mining technique which offers insight into datasets based off of similarities of elements to each other. Spectral clustering uses decompositions of matrix representations of graphs to find appropriate clusters in a data set. In this thesis we propose two new methods to improve spectral clustering, Smoothed Embedding clustering and Graph Smoothed clustering which involve perturbing the spectral embedding of a graph and replacing each vertex with a clique respectively. We prove that the process graph smoothing in Graph Smoothing clustering decreases the magnitude of the smallest non-zero eigenvalue for a connected weighted graph, and finish with empirical investigation applying to the algorithms to the Fisher Iris data set.

Dedicated to my younger brother Alex

Acknowlegements

There's no way that I ever would've made it as far as I have this year without the help of my advisors Lenore and Xiaozhe. I would like to thank them both for their patience, guidance, and support. Working with them has been a privilege.

Contents

List of Tables **viii**

List of Figures **ix**

1 Introduction **2**

1.1 Clustering 2

1.2 Partition Objective Functions 3

1.2.1 Min-Cut 4

1.2.2 Ratio Cut 4

1.2.3 Normalized Cut 5

1.3 Spectral Clustering 5

1.3.1 Cheeger's Inequality 6

1.3.2 Spectral Embedding 7

1.4 The Symmetric Eigenvalue Problem 7

1.4.1 Problem Statement 7

1.4.2 Schur Form 8

1.4.3 Rayleigh Quotients 8

1.4.4 Stability 9

1.4.5 Shifts 9

2 Algorithms **10**

2.1 Graph Building 10

2.1.1 k nearest 10

2.1.2 ϵ neighborhood 11

2.1.3 fully connected 11

2.2 Implemented Algorithms . . . 12
2.2.1 Benchmark Algorithms . . . 12
2.2.1.1 K means . . . 13
2.2.1.2 Ng, Jordan, Weiss Spectral Clustering (NJW) . . . 13
2.2.1.3 Shi and Malik Spectral Clustering (SM) . . . 14
2.2.2 Proposed Algorithms . . . 15
2.2.2.1 Smoothed Embedding Clustering . . . 15
2.2.2.2 Graph Smoothed Clustering . . . 17
2.3 Computing Eigenvectors . . . 18

3 Proofs **21**

4 Experiments **30**

4.1 Parameters . . . 30
4.1.1 Graph Building Parameters . . . 30
4.1.2 Kernel Parameters . . . 31
4.1.3 Proposed Algorithm Parameters . . . 31
4.2 Data . . . 31
4.3 Tests . . . 32
4.3.1 Evaluating Success . . . 32
4.3.2 Clustering Comparison . . . 32
4.3.3 Clique Size . . . 32
4.3.4 Clique Edge Weight . . . 33
4.3.5 Separation of λ_2 . . . 33

5 Results **34**

5.1 Fisher Iris . . . 34
5.2 Altering Internal Edge Weights . . . 36
5.3 Altering Clique Sizes . . . 38
5.4 Embedding Plot . . . 40

6 Conclusion **43**

Bibliography **45**

List of Tables

List of Figures

5.1 Comparing all implemented clustering algorithms against one another. Graph built using ϵ neighborhood method where ϵ = .2, smoothing methods used 5 times the number of points. Results averaged over 10 tests.This figure shows the Graph Smoothing algorithm greatly outperforming other methods 34

5.2 Comparing all implemented clustering algorithms against one another. Graph built using k nearest method where $k = 3$, smoothing methods used 5 times the number of points. Results averaged over 10 tests. Results showing subpar performance from Graph Smoothing method, k means and Shi Malik Spectral clustering offering best results 35

5.3 Comparing all implemented clustering algorithms against one another. Graph built using the fully connected method. Smoothing methods used 5 times the number of points, results averaged over 10 tests. Results showing best results coming from Graph Smoothed Clustering 35

5.4 Performance of the Graph Smoothed Clustering algorithm as clique edge weight varies.Averaged over 2 tests, clique size is 4. Figure shows that performance is similar for edge weights less than .5, success starts to diminish as edge weights become larger than .5. . . . 36

5.5 Performance of the Graph Smoothed Clustering algorithm as clique edge weight varies.Clique size set to 4. Figure shows small dip from edge weights ranged 2-5. Edge weights >= 1 still don't perform as well as edge weights < 1. 37

5.6 Orange Line is the original separation of λ_2 for the normalized Laplacian of G. The blue line is the separation of λ_2 for the normalized Laplacian of $\tilde{G}$. Edges weights vary from .01 to 1, the larger the edge weight the smaller the separation. 37

5.7 Orange Line is the original separation of λ_2 for the normalized Laplacian of G. The blue line is the separation of λ_2 for the normalized Laplacian of $\tilde{G}$. The edges range from 1 to 10, and for large edge weight we see smaller separation. 38

5.8 This figure compares the results of Graph Smoothed Clustering as the size of the cliques is altered. Clique sizes range from 2-11. The best results are present for clique sizes 4-6. 39

5.9 Orange line is separation of λ_2 of normalized Laplacian of G. Blue line is the separation of λ_2 of the normalized Laplacian of the smoothed graph $\tilde{G}$. Clique sizes range from 2-11. 39

5.10 Graph Smoothing applied on graph built with ϵ neighborhood with tolerance of .2 40

5.11 Graph Smoothing applied on graph built with fully connected method. 41

5.12 Graph Smoothing applied on graph built with 3 nearest neighbors. This embedding plots a lot of vertices to the same place, the clusters are not clear in either figure. This corresponds with the poor clustering performance when tested on the Iris data. 42

Improving Clustering with Graph Smoothing

Chapter 1

Introduction

1.1 Clustering

Consider the problem of having a collection of genetic information for individuals in a population, and looking to group people with the most similar genes [13]. Perhaps instead, a private tutoring software that uses data from students responding to questions and performance measures to identify what type of students use the program, and how to optimize student learning [1,16] . One could also try to accelerate catching criminals by looking at the spatial information and metadata associated with each crime, and narrow down which were performed by the same person, and where they may next act [11]. These are all problems that can be solved with the help of clustering.

Clustering is a data mining problem, that looks to extract information about a data set from its structure. Clustering is the problem of finding subsets of vertices in graphs or points in an arbitrary mathematical space that are more similar to each other than to any other elements in the set. This concept of similarity is ambiguous and different algorithms and problem contexts measure the similarity differently. This ambiguity provides cluster analysis with power because of its applications in various fields.

Jain [8] offers the following mathematical definition of clustering. Given a representation of n objects, find K groups based on a similarity function f such that the similarity of objects in the same group is high, while the similarity of objects in different groups is low.

Clustering problems can be considered hard or soft (also know as fuzzy) depending on the constraints for how elements can be clustered. Hard clustering only allows

for vertices to exist in one cluster as a time, soft clustering allows for vertices to partially belong to multiple clusters. The distinction is the idea that data points can only belong to one cluster or to many clusters. In a program looking to partition vertices to recursively call an algorithm on that subset, a hard clustering program would be appropriate. For a problem looking to analyze friend groups within a social network, a softer approach would be called for as individuals can belong to many social circles.

Clustering can be solved with supervised and unsupervised algorithms. A supervised algorithm will use a data set of initial categorized information, and strive to put new vertices or data points in with the clusters provided. Unsupervised algorithms will cluster the graph or data from its mathematical structure, providing the challenge of ensuring that the method will converge to appropriate clusters. Unsupervised methods are typically used when there is no ground truth data set to compare the results of clustering. Because there‘s no ground truth, unsupervised algorithms can be difficult to decide upon which subsets are appropriate cluster candidates, and which parts of the mathematical structure should be exploited to produce clusters that correlate with the properties desired for applications.

In this thesis, we will investigate hard clustering, and all algorithms implemented will be unsupervised. Even with this narrowed scope, there are still many problems to be overcome in searching for an optimal cluster. One such example is how to properly evaluate how appropriate a subset of vertices or elements is as a cluster candidate. For this we may turn to objective functions.

1.2 Partition Objective Functions

When considering the quality of clusters in a dataset, the quality of an assignment will be judged based on various possible objective functions (as defined in the next section), its domain is a set of subsets of vertices, and the codomain is the set of real numbers. To find the most suitable clusters in a set, the objective function is maximized or minimized, and the sets that achieve the local or global optimum, are

returned as the solution. Finding clusters through this method is called the discrete optimization approach to clustering [19]. Objective functions all have their own drawbacks and benefits, as certain functions are more suited to specific problems. One of the simplest, and most thoroughly studied is the minimum cut.

1.2.1 Min-Cut

The intuition of clustering is to find subsets of vertices that have very small edge weights between the vertices in defined clusters and large edge weights for vertices within the same clusters . The first condition can be solved for with the help of the min cut. The minimum cut is a fundamental graph problem, and can be solved efficiently for the case of two subsets, using variations of Karger's Algorithm [9], or maximum flow algorithms. The problem can even be generalized to finding the minimum cut between k subsets of vertices. Minimum cut has been thoroughly studied, and there's a lot of material on the subject, making it a great starting place to finding clusters.

For a graph $G = (V, E, \omega)$, using the notation that $W(A, B) = \sum_{i \in A, j \in B} \omega_{ij}$ and the complement of a set A is $\bar{A}$. The problem of the minimum cut of a graph for $k = 2$ can be written as looking for the subset of vertices $A \subset V$ that minimizes $\frac{1}{2}W(A, \bar{A})$. Though a recursive min cut algorithm seems to solve the problem of clustering, Von Luxburg [18] discusses the drawbacks of min-cut in her tutorial. She explains that min-cut can produce trivial clusters containing just a single vertex. Min-cut also requires that k, the number of desired clusters, be presented as an external input to the algorithm, which can be problematic in some unsupervised problem settings where the number of desired clusters is not a known priori. To remedy this, generalizations of min-cut are defined, requiring a more balanced partition of the vertices.

1.2.2 Ratio Cut

To find a cut that minimizes the weight of the edges while ensuring that the number of vertices in each cut isn't some small subset, the sum of the weights between subsets

can be divided by a factor that increases as the size of the partitions increase.The Ratio cut is defined as

$$RatioCut(A) := \frac{W(A, \bar{A})}{|A|}$$

The Ratio Cut tries to rectify the trivial clusters problem by dividing by the size of the set. Thus a solution of just one vertex won't be optimal; this is true for the general case of finding k partitions. However one thing to consider is the Ratio Cut does not taking into consideration the internal edges weights. The denominator is just the size of the partition, so while we're not getting trivial solutions, the internal edges weights will not be maximized.

1.2.3 Normalized Cut

$$NormalizedCut(A) := (\frac{W(A, \bar{A})}{Vol(A)})$$

where the Vol(A) is defined as the sum of the weighted degrees of the vertices in A. $Vol(A) := \sum_{v \in A} d(v)$. This quantity divides the minimized cut by all of the edge weights within the cluster of the set. The optimal solution of the normalized cut will not only seek to minimize the weight of the cut, but will also maximize the sum of the internal edges within the cluster. This tackles the problem of trivial solutions, while also providing clusters with vertices that are more likely to be similar to each other. The normalized cut is proportional to the objective function conductance, some sources define them to be the same. Conductance is defined as

$$Conductance(A) := \frac{W(A, \bar{A})}{min\{Vol(A), Vol(\bar{A})\}}$$

The conductance of a graph $\phi(G)$, is defined the be the minimum conductance over all possible subsets of vertices in the graph.

1.3 Spectral Clustering

Spectral graph theory uses matrix representations of the graphs to solve approximations to graph problems like min max cuts, coloring, graph embeddings, and more.

The approximations algorithms come from re-writing a optimization function with the Rayleigh quotient of a matrix. This turns the problem into an eigenvalue matrix problem. The matrix representations of a graph $G = (V, E, \omega)$ are the adjacency A, the Laplacian L, and the normalized Laplacian matrix N and are defined as

$$A = \begin{cases} \omega_{ij} & i \sim j \\ 0 & \text{else} \end{cases}, \quad L = \begin{cases} \deg(v_i) & i = j \\ -\omega_{ij} & i \sim j \\ 0 & \text{else} \end{cases}, \quad N = \begin{cases} 1 & i = j \\ -\frac{\omega_{ij}}{\sqrt{\deg v_i \deg v_j}} & i \sim j \\ 0 & \text{else}s \end{cases}$$

1.3.1 Cheeger's Inequality

Cheeger's inequality is the most important theorem in spectral clustering as it relates the eigenvalues of the normalized Laplacian to the optimal conductance of a graph. If we denote the eigenvalues of the normalized Laplacian of a connected unweighted graph to be $0 = \lambda_0 < \lambda_1 \leq \lambda_2 \leq ... \leq \lambda_{n-1}$, then the optimal conductance on the graph $\phi(G)$ is related to λ_1 by

$$2\phi(G) \geq \lambda_1 \geq \frac{\phi(G)^2}{2}$$

a proof of this can be found in [3]. This inequality can be generalized to encompass positively weighted graphs. Friedland and Nabben [5] show that for any positive diagonal matrix $D = \text{diag}(d_1, ..., d_n)$, $\lambda_1 \geq \min_i \frac{\delta_i}{2d_i}\phi(G)^2$, where δ_i is the weighted degree of vertex i. By choosing D to be the weighted degree matrix of the graph, we can show that bounds for positively weighted graphs still hold.

$$2\phi(G) \geq \lambda_1 \geq \min_i \frac{\delta_i}{2\delta_i}\phi(G)^2 = \frac{\phi(G)^2}{2}$$

With these inequalities one can see that the corresponding eigenvector is a 2-approximation to the optimal conductance of a graph.The eigenvector $\mathbf{v}$ determines which vertex belongs in which partition by assigning vertex i to cluster 1 if the value of the $\mathbf{v}_i > 0$ and to cluster 2 else. Shi and Malik showed that the smallest eigenvectors of the generalized eigenvalue problem $Lx = \lambda Dx$ are relaxations of the discrete optimization

of the normalized cut [14]. Von Ulrike shows in her tutorial on spectral graph theory that the smallest non-constant eigenvectors of the graph Laplacian approximate the optimal ratio cut [18].

1.3.2 Spectral Embedding

One application of spectral graph theory is the use of the non-constant eigenvectors of the graph Laplacian as an embedding of the graph. In 1970 Kenneth Hall published a method of placing n points or nodes into an r dimensional space subject to constraints on the scale of the embedding [7]. Hall showed that you could reduce the problem to that of solving for the eigenvectors corresponding to the r smallest positive eigenvalues of the graph Laplacian. Ng et al. showed that r smallest eigenvectors of the normalized Laplacian will map the vertices onto an r dimensional hypersphere, and that points that should be clustered together typically are mapped close to one another [12]. Spectral clustering algorithms use the embeddings as a preprocessing step before using an unsupervised clustering algorithm on the embedding.

1.4 The Symmetric Eigenvalue Problem

1.4.1 Problem Statement

Eigenvalue problems are matrix problems where for a matrix A you must find a non-zero vector x such that $Ax = \lambda x$. The Symmetric Eigenvalue Problem is the special case where $A^T = A$ or A is symmetric, or when A is a matrix over the complex field then we say $A^H = A$ where A^H is the conjugate transpose of A also known as A being Hermitian. I will only consider the case when A is symmetric, because we are dealing with real matrix representations of the graphs, but know that the theory can be generalized to accommodate complex valued matrices. The symmetry of the graph Laplacian and variants can be enforced by only considering undirected graphs.

1.4.2 Schur Form

When A is symmetric, A has properties that make the eigendecomposition a lot easier to compute. First of all, we're guaranteed to have n linearly independent eigenvectors of A.

Definition 1.4.1 (Unitary and Orthogonal Matrices [4]) *A square complex matrix U is **unitary** if*

$$U^H U = UU^H = I$$

*if U is real then U is **orthogonal** if $U^T U = UU^T = I$*

***note:** $U^{-1} = U^T$ and the product of two orthogonal matrices is orthogonal*

Theorem 1.4.2 (Real Schur Form [4]) *If A is an $n \times n$ matrix then there exists a unitary matrix such that*

$$U^H AU = T$$

Where T is a triangular matrix with the eigenvalues $\lambda_1, \lambda_2, ..., \lambda_n$ as the diagonal entries.

The Real Schur form can be used to show that a symmetric matrix is diagonalizable, and thus a full basis of orthogonal eigenvectors exist. This existance guarantee is needed for the proofs in Chapter 3.

1.4.3 Rayleigh Quotients

When the A is symmetric, the eigenvalues can be considered optimal solutions to a minimization or maximization of a term called the Rayleigh quotient. The Rayleigh quotient is defined as $R(A, x) := \frac{x^T Ax}{x^T x}$

Theorem 1.4.3 (Minimax characterization(Courant-Fischer) Theorem [4]) *Let $\lambda_1 \leq \lambda_2 \leq ... \leq \lambda_n$ be the eigenvalues of a symmetric matrix A. Then*

$$\lambda_i = \min_S \max_{0 \neq x \in S} \frac{x^T Ax}{x^T x}$$

This theorem is important because it can be used to bound any Rayleigh quotient of a symmetric matrix by the eigenvalues of from above or below. This theorem will play a role in the analysis of the Graph Smoothed Clustering algorithm in section 3.

1.4.4 Stability

Symmetric Eigenvalue problems also enjoy the benefit of being well conditioned. The Bauer-Fike Theorem [4] shows that if the matrix has some small perturbations in the form of a matrix E, the the eigenvalues will only be altered by at most the $\|E\|_2$ which is the largest eigenvalue of E. All problems solved in finite precision will have these perturbation matrices added to the problems. However the entries of these perturbation matrices will be on the order of machine-epsilon, so the eigenvalues will be trustworthy if the algorithm solving the problem is stable. Coupled with the fact that we know that there won't be any degenerate eigenvalues,because there is a full basis of eigenvectors, the eigenvectors will be straightforward to solve for.

1.4.5 Shifts

Though one thing to note about solving for the eigenvectors of the graph Laplacian. Because the graph Laplacian is singular, its condition number will be very large. To prevent this from being an issue, spectral clustering algorithms employ a small shift in the form of a matrix to alter the eigenvalues, but not the eigenvectors. For the normal eigenvalue problem $Ax = \lambda x$, a shift would be replacing the matrix A with the $B = A + \alpha I$. If λ is an eigenvalue of A, the $\lambda + \alpha$ is an eigenvalue of B, when solving the generalized eigenvalue problem $Ax = \lambda Bx$, then the shifted matrix will be $B = A + \alpha B$. All of the programs solve a shifted matrix to get better convergence and stability properties.

Chapter 2

Algorithms

2.1 Graph Building

All of the algorithms in this section other than k means can be implemented to take in either a set of data points or a graph. In cluster analysis, a large part of the problem, and where a lot of difficulty lies is building a graph from a dataset in order to run these spectral methods on. There are many methods to build graphs from geometric data, and some datasets may require specialized methods in order to produce a graph that would be of any use. Some standard methods are outlined, along with some of their drawbacks and benefits. Each method uses a distance function that may be a norm or some sort of kernel that is associated with the data set in question.

2.1.1 k nearest

Input: A set of n points $X = x_1, ..., x_n$ from a d dimensional real space $x_i \in \mathbb{R}^d$ for $i \in 1, ..., n$, some metric $d : X \times X \to \mathbb{R}$ to compare points with, a positive integer $k \in \mathbb{Z}^+$ to denote the number of closest neighbors.

Returns: An unweighted or weighted directed graph. The k nearest method of building a graph is for each vertex, identifying k data points that have the smallest value from a metric to it, and connecting it to each of these k other vertices in the graph.

This method is excellent for creating sparse networks, and is a good starting point when attempting to cluster a data set for the first time. Because the method is only looking for the k nearest, it doesn't pay very much attention to the scale of the data set. If you have two clusters where one is dense, and the other is sparse, then both clusters will most likely be connected. There may be connections between the two

clusters. Also, as k increases then the graph is only more likely to become connected. One concern with the k nearest method is the fact that it will produce a directed graph, but that can be resolved by replacing directed edges with an undirected edges. All of the programs implemented force the graph to be undirected when using the k nearest method. This method can also produce a weighted graph by assigning the value of the metric to the edge, but implementations of the k nearest method return unweighted graphs in all experiments conducted.

2.1.2 ϵ neighborhood

Input: A set of n points $X = x_1, ..., x_n$ from a d dimensional real space $x_i \in \mathbb{R}^d$ for $i \in 1, ..., n$,some metric $d : X \times X \to \mathbb{R}$ to compare points with, a positive real number $\epsilon \in \mathbb{R}^+$ to denote the threshold of when to add an edge.

Returns: A weighted undirected graph. The ϵ neighborhood method is to compare all of the data points in the space to each other, and add an edge between vertices if their distance function is greater than or less than a threshold ϵ.choosing to include an edge because the distance is less than or greater than depends on the co-domain of the distance function being used. This graph building method is a lot more sensitive to the scales of the data points and the distance function being used to compare. The method usually produces a highly connected graph, but may not be suitable if there exist clusters that have different densities. Different choices of ϵ may result in one cluster being a connected component in the graph, while the other less dense cluster is only sparse pairs or small dis-connected components. This method will produce an undirected graph, as distance functions are typically symmetric. In the experiments run, the values of the distance function for each pair of vertices were used as edge weights.

2.1.3 fully connected

Input: A set of n points $X = x_1, ..., x_n$ from a d dimensional real space $x_i \in \mathbb{R}^d$ for $i \in 1, ..., n$,some metric $d : X \times X \to \mathbb{R}$ to compare points with.

Returns: A weighted undirected clique of size n. The fully connected method

is very similar to the ϵ neighborhood method. Again we compare every data point to each other, and use the distance of the points as the edge weight in the graph. But instead of a threshold, every vertex is connected to every other vertex in the graph. This forms a clique, which from a computational viewpoint isn't ideal as the resulting graph Laplacian will be a dense matrix.

2.2 Implemented Algorithms

To find more suitable approximations for the normalized cut, we propose two algorithms called the Smoothed Embedding and Graph Smoothed clustering algorithms. The Smoothed Embedding algorithm is based off of the intuition of the approach to improve discrete optimization clustering by embedding points into a Euclidean space and introducing more points by sampling a Gaussian distribution of appropriate dimension.The Graph Smoothed algorithm follows the same intuition, but never actually embeds the vertices into an Euclidean space. The Graph Smoothed algorithm has theoretical results showing it results in a graph that has a smaller λ_2, which because of the bounds on the conductance of graph leads to better clustering.

When implementing all of the algorithms, there is a lot of leeway in deciding what what kind of kernel the points will be compared against with, and what kind of final clustering algorithm will be used on the embedded graph. For the sake of consistency, the radial basis function or Gaussian kernel is used in building the graph.

2.2.1 Benchmark Algorithms

To understand how the algorithms perform, the results of the algorithms were compared to other spectral clustering methods. The essence of spectral clustering is to use the k-means algorithm on an embedding of the graph. Spectral methods typically use the eigenvectors of a matrix representation of the graph like the graph Laplacian or the normalized Laplacian. These can be shown to offer approximation guarantees on sparse cuts in the graph. A brief overview of each of the algorithms

will be presented, and followed by the algorithms being analyzed in this thesis.

2.2.1.1 K means

Each spectral clustering method can use any unsupervised clustering algorithm on the embedded vertices, but in our experiments we use k means exclusively. The k means algorithm was a developed in 1957 by Stuart Loyd at Bell Labs, but wasn't published until 1982. k means is a standard clustering algorithm and in practice offers good results for well separated data sets.

K means [10]

Input: Data set of $\mathbb{R}^d$ data points, k $\in \mathbb{Z}^+$ number of clusters

Initialize: choose k random centroids in $\mathbb{R}^d$

repeat until labels no longer change from iteration to iteration

1. compute distance of each element in the data set to each centroid and label corresponding to closest centroid
2. compute k new centroids from average of all points associated with each centroid.

This implementation of k means may converge to the wrong clusters when the randomly selected points are close to one another. This addressed in the k++ means algorithm [2] which chooses initial centroids to be well separated.

2.2.1.2 Ng, Jordan, Weiss Spectral Clustering (NJW)

In Ng et al.'s paper on analyzing spectral clustering performance and robustness they analyze the normalized spectral clustering algorithm with matrix perturbation theory.

NJW Clustering [12]

Input: a dataset $X = \{x_1, ..., x_n\}$ of $\mathbb{R}^d$ data points, k $\in \mathbb{Z}^+$ number of clusters, variance $\sigma \in \mathbb{R}^+$

1. Build the affinity matrix $A \in \mathbb{R}^{nxn}$ where $A_{ij} = exp(\frac{-\|x_i - x_j\|^2)}{2\sigma^2})$ if $i \neq j$ and $A_{ii} = 0$

2. Let D be the diagonal matrix, such that $D_{ii} = \sum_{j=1}^{n} A_{ij}$, and build the matrix $L = D^{\frac{-1}{2}} A D^{\frac{-1}{2}}$

3. Find $v_1, v_2, ..., v_k$, the k largest eigenvectors of L, made to be orthogonal. Form the matrix $V \in \mathbb{R}^{nxk}$ where the columns of V are the vectors v_i.

4. Make W by normalizing the rows of V to have 2-norm = 1. $W_{ij} = V_{ij} / (\sum_j V_{ij}^2)^{\frac{1}{2}}$

5. Let each row of W be an embedded vertex in $\mathbb{R}^k$ and cluster them with any clustering algorithm (K means will be used in experiments).

6. Assign data point x_i to cluster j if row i of W was assigned to cluster j

Their paper outlines conditions for when spectral clustering should lead to proper clusters on a hypersphere. These results have been used by a lot of the spectral clustering community. To cluster they use a variation of the normalized Laplacian, but their algorithm can be adjusted to use the standard graph Laplacian matrix. In line 2, $L = D - A$ and in line 3 the vectors will be the smallest eigenvectors of L. In the experiments conducted, the normalized and unnormalized versions were used. Note that in line 3, they use $L = D^{\frac{-1}{2}} A D^{\frac{-1}{2}}$. This is a shifted an negated normalized Laplacian, by the properties of the spectrum, you can find the largest eigenvectors of this matrix to find the smallest eigenvectors of the graph Laplacian.

2.2.1.3 Shi and Malik Spectral Clustering (SM)

This spectral clustering algorithm uses the generalized eigenvalue problem to find an approximate solution to the normalized cut. Shi and Malik proposed this algorithm in 2000 for the purpose of image segmentation. The main difference between this algorithm and the normalized NWJ method, is that it finds the solution by solving a generalized eigenvector problem.

SM Clustering [14]

Input: a dataset $X = \{x_1, ..., x_n\}$ of $\mathbb{R}^d$ data points, k $\in \mathbb{Z}^+$ number of clusters, variance $\sigma \in \mathbb{R}^+$

1. Build the affinity matrix $A \in \mathbb{R}^{nxn}$ where $A_{ij} = exp(\frac{-\|x_i - x_j\|^2)}{2\sigma^2})$ if $i \neq j$ and $A_{ii} = 0$

2. Let D be the diagonal matrix, such that $D_{ii} = \sum_{j=1}^{n} A_{ij}$, and build the matrix $L = D - A$

3. Find $v_1, v_2, ..., v_k$, the k smallest generalized eigenvectors of $Lv = \lambda Dv$, made to be orthogonal. Form the matrix $V \in \mathbb{R}^{nxk}$ where the columns of V are the vectors v_i.

4. Let each row of V be an embedded vertex in $\mathbb{R}^k$ and cluster them with any clustering algorithm (K means will be used in experiments).

5. Assign data point x_i to cluster j if row i of W was assigned to cluster j

2.2.2 Proposed Algorithms

2.2.2.1 Smoothed Embedding Clustering

The algorithm is an additional step to the standard spectral clustering methods. And works for either m geometric points in some $\mathbb{R}^n$ space or a graph $G = (V, E)$. The basic intuition for the algorithm is the idea that clustering vertices in graphs has two trivial solutions. Either we can consider V to be one big cluster or we can consider each v_i its own independent cluster. This is an issue that is commonly present in clustering methods that rely on optimizing an objective function over a set of vertices. Typically solutions of small clusters are found by the method, but these smaller clusters would be better suited grouped with a larger cluster candidates. Heuristics can be employed to minimize, but not eliminate this problem.

This issue arises because this is a discrete optimization problem, so we can put boundaries around the individual clusters and trivially solve the clustering problem, but if we moved to a continuous functional space, then these boundaries wouldn't be as well defined. By sampling points from continuous functions that are modeled with probability density functions we can add in extra dummy data into the problem that is "similar" to the original data points. We can then run our clustering algorithm on

this new data set and remove the dummy data that we introduced, and the clustering algorithm should produce cluster candidates free of trivial clusters.

Since the points are being sampled from the probability distribution function, as number of points that we sample approaches infinity,we would approach a continuous space. Fortunately this method doesn't require that we sample an infinite number of points. We can add in the points one at a time from a distribution that is previously determined. For all the experiments the programs all found points from an n-dimensional Gaussian distribution

let $f : \mathbb{R}^n \mapsto \mathbb{R}$ be the n dimensional Gaussian distribution. Then if data set $X = \{x_1, ..., x_m\}$ where $x_i \in \mathbb{R}^n$ for $i = 1, .., m$ has mean μ and covariance matrix $\sigma \in \mathbb{R}^{m \times m}$ then

$$f(\vec{v}, \sigma, \mu) = \frac{1}{\sqrt{(2\pi)^k |\sigma|}} exp(-\frac{1}{2}(\vec{v} - \mu)^T \sigma^{-1}(\vec{v} - \mu))$$

Smoothed Embedding Clustering

Input:a dataset $X = \{x_1, ..., x_n\}$ of $\mathbb{R}^d$ data points or an undirected Graph $G = (V, E)$ with or without edge weights, k $\in \mathbb{Z}^+$ number of clusters, a covariance matrix σ, a positive integer $m \in \mathbb{R}^+$ which denotes the number of points to introduce for each vertex.

1. If provided a data set, then build a graph with one of the previously discussed methods.

2. Build the normalized Laplacian $N = D^{-\frac{1}{2}} L D^{-\frac{1}{2}}$

3. Find $v_1, v_2, ..., v_k$, the k smallest eigenvectors of N, made to be orthogonal. Form the matrix $V \in \mathbb{R}^{nxk}$ where the columns of V are the vectors v_i.

4. Make W by normalizing the rows of V to have 2-norm = 1. $W_{ij} = V_{ij}/(\sum_j V_{ij}^2)^{\frac{1}{2}}$

5. for each row in the matrix W, use the row as the expected value of a Gaussian distribution $f(x, \sigma, W^T_{j,i,...,k})$, and for the defined function randomly sample m points $\in \mathbb{R}^k$ and treat them as new rows in the matrix W.

6. form a new graph G' using the the rows of W as points with the same graph building method in step 1.

7. form the normalized graph Laplacian of the new "smoothed" graph G', $N' = D^{\frac{-1}{2}} L' D^{\frac{-1}{2}}$

8. Find $v_1, v_2, ..., v_k$, the k smallest eigenvectors of N', made to be orthogonal. Form the matrix $V \in \mathbb{R}^{nmxk}$ where the columns of V are the vectors v_i.

9. Make W by normalizing the rows of V to have 2-norm = 1. $W_{ij} = V_{ij} / (\sum_j V_{ij}^2)^{\frac{1}{2}}$

10. Let each row of W be an embedded vertex in $\mathbb{R}^k$ and cluster them with any clustering algorithm (K means will be used in experiments).

11. Assign data point x_i to cluster j if row i of W was assigned to cluster j

One thing to note about the implementations of this algorithm is that all experiments used the identity matrix as the covariance matrix of the Gaussian distribution.

2.2.2.2 Graph Smoothed Clustering

The Smoothed Embedding Clustering Algorithm works for either a graph or a set of geometric data points, but there exist graphs that don't necessarily come from some geometric data set. The principal of graph smoothing is trying to capture the ideas of replacing a set of discrete points with continuous functions, by expanding each vertex into a clique. Each vertex in the clique is connected to each other, and all of the neighbors of the original vertex are connected to the clones.

Graph Smoothed Clustering

Input: a dataset $X = \{x_1, ..., x_n\}$ of $\mathbb{R}^d$ data points or an undirected Graph $G = (V, E)$ with or without edge weights, k $\in \mathbb{Z}^+$ number of clusters, size of the clique $C \in \mathbb{Z}^+$, and the clique edge weights $a \in \mathbb{R}^+$

1. If provided a data set, then build a graph with one of the previously discussed methods.

2. If the graph is unweighted, then impose weights onto the graph, where each edge weight is 1.

3. expand each vertex in the vertex set of G into a clique of size C. Assign the edge weight of each edge in any clique to have weight a.

4. connect the new vertices in the clique to the neighbors of their original vertex v with edge weight scaled by $\frac{1}{2*C-1}$

5. form the normalized graph Laplacian of the new "smoothed" graph G', $N' = D^{\frac{-1}{2}} L' D^{\frac{-1}{2}}$

6. Find $v_1, v_2, ..., v_k$, the k smallest eigenvectors of N', made to be orthogonal. Form the matrix $V \in \mathbb{R}^{nxk}$ where the columns of V are the vectors v_i.

7. Make W by normalizing the rows of V to have 2-norm = 1. $W_{ij} = V_{ij} / (\sum_j V_{ij}^2)^{\frac{1}{2}}$

8. Let each row of W be an embedded vertex in $\mathbb{R}^k$ and cluster them with any clustering algorithm (K means will be used in experiments).

9. Assign data point x_i to cluster j if row i of W was assigned to cluster j

A few comments on this algorithm. In step 2, the edge weights are imposed so that the internal edge weights of the clique will be distinguished from the edge weights of neighbors. In step 4, the edge weights are all scaled by the term $\frac{1}{2*C-1}$ so that when the matrix is multiplied by the restriction operators in the proof, the result is the original graph Laplacian, not a scaled version. This is not a necessary step for the algorithm, but it's implemented for consistency with the proofs of the method.

2.3 Computing Eigenvectors

The power of spectral clustering doesn't come from the quality of the partitions that are provided, but the fact that it can be formulated into a eigenvector problem. From a complexity perspective, the largest asymptotic component of the algorithm is the computation of the eigensystem or select eigenvectors. A standard iterative

method for computing the eigensystem such as the QR method can cost a hefty $O(n^4)$ steps to compute, which doesn't bode well for scalability. Fortunately there are ways around this that all work well with spectral methods.

First off, if the graph that is being analyzed is sparse, the number of edges m = O(n), then it can be stored with a sparse matrix data type. Using these formats, matrix vector multiplication can be computed in $O(n)$ steps instead of $O(n^2)$. This greatly reduces the complexity of the eigensystem problem, because all eigenvalue problems for matrices on the order of 5 and larger must be solved iteratively.

Another thing to consider is the fact that we only need a small number of eigenvectors for the embeddings. When computing all the eigenvectors of a symmetric matrix, methods typically cost $O(n^3)$ operations like the Tridiagonal QR iteration, Divide-and-conquer, and Jacobi methods [4] (an exception is the Bisection method which requires $O(n^2)$).The Inverse Power method calculates the smallest eigenvectors with the inverse of the matrix. The Laplacian and it's variants are singular, and so one can shift the eigenvalues so that the Power Method will converge to the smallest eigenvectors. It's then possible to deflate the problem and solve for other eigenvectors of the matrix. For a small number of eigenvectors this isn't a problem, but continuing the process of deflation leads to stability issues, and the eigenvectors may be incorrect. Also the convergence may be slow for this method because it will be mathematically equivalent to running the inverse power method. Also if the eigenvalue is not well separated (definition in section 4.3.5) the convergence may be slow.

Krylov projection methods like the Symmetric Lancoz algorithm offer a way to compute a small subset of the the eigenvectors quickly for sparse matrices. Krylov methods are iterative procedures that increase in size at each iteration, but produce subproblems much smaller than the original matrix problems. To the find the eigensystem of a large sparse matrix, Krylov methods can use the Ritz vectors to approximate the eigenspace of a matrix, instead of solving for the exact eigenvectors. The Symmetric Lancoz method can find k eigenvectors in $O(kn)$ time when using a sparse matrix datatype [15].

It's also possible to compute the smallest non-constant eigenvector though multi-grid methods. This class of methods use restriction and prolongation operators to solve smaller matrix problems at each iteration. Multi-grid methods are scalable and can find the eigenvector within O($nlogn$) operations [17].

Chapter 3

Proofs

This chapter contains the proofs of the behavior of the second smallest eigenvalue when the graph is smoothed. The proofs of these section are for specific versions of graph smoothing where either all vertices are expanded into cliques of the same size, and when only one vertex is expanded. Before mathematically defining the process of graph smoothing, we will show a few lemmas that explain the choice of Rayleigh quotient, and properties of the eigenvectors found. We can show that the eigenvalues of the normalized Laplacian are the same as the random walk matrix. The benefit of doing so is being able to use the generalized Rayleigh quotient which makes showing the eigenvalues decrease simpler.

Lemma 3.0.1 *The Normalized Laplacian is similar to the random walk matrix given by* $W = D^{-1}L$.

Proof: The Normalized Laplacian is given by $D^{-1/2}LD^{-1/2}$, W is similar to N if there exists a matrix X such that $W = X^{-1}NX$. Let $X = D^{1/2}$, then $W = X^{-1}NX = D^{-1/2}D^{-1/2}LD^{-1/2}D^{1/2} = D^{-1}LI = W$ □

furthermore if the eigenvalue, eigenvector pair(λ_2, ϕ_2) satisfy $W\phi_2 = \lambda_2\phi_2$ then they also satisfy the generalized eigenvalue problem $L\phi_2 = \lambda_2 D\phi_2$.

Now we want to ensure that there is a full basis of orthogonal eigenvectors to compute for the matrix. However the random walk matrix is not symmetric using the standard euclidean inner product. By using a different inner product space, we can show that the matrix is symmetric. By finding the eigenvectors using this new inner product space, there will exist be a full basis of orthogonal eigenvectors.

Lemma 3.0.2 *The random walk matrix* W *is symmetric with respect to* $\langle\cdot,\cdot\rangle_D$*, given by* $\langle x, y\rangle_D = \langle x, Dy\rangle$ *for* $x, y \in \mathbb{R}^n$.

Proof: W is symmetric with respect to $\langle\cdot,\cdot\rangle_D$ iff $\forall x, y \in \mathbb{R}^n$ $\langle x, Wy\rangle_D = \langle Wx, y\rangle_D$.

Then $\langle x, Wy\rangle_D = \langle x, D^{-1}Ly\rangle_D = x^T DD^{-1}Ly = x^T LD^{-1}Dy = \langle D^{-1}Lx, Dy\rangle = \langle D^{-1}Lx, y\rangle_D$ so $\langle x, Wy\rangle_D = \langle Wx, y\rangle_D \forall x, y \in \mathbb{R}^n \square$

Now for a formal statement of what graph smoothing is. Intuitively it is the process of replacing each vertex with a clique of size k, and connecting each clique to the rest of the graph. This is a preliminary definition, as later in the chapter, I will discuss how the eigenvalues change for just expanding one vertex into a clique.

Definition 3.0.3 (Graph Smoothing) *Let $G = (V, E, \omega)$ be a weighted graph with vertex set V, Edge set E, and a function $\omega : E \mapsto \mathbb{R}^+$. Let the sizes of V, E be denoted by $|V| = n$ $|E| = m$. Consider replacing each vertex v in the graph with a clique C of size k, where every edge in the clique has weight a. now for each C, let all the vertices $\tilde{v} \in C$ be neighbors to the neighbors of v. When connecting the clones to the neighbors of the original point, the edge weights need to be scaled such that the $\sum_{\tilde{v} \in C} deg_{\tilde{G}}(\tilde{v}) = deg_G(v)$. Call this new graph $\tilde{G}$ the smoothed graph of G.*

To relate the new graph Laplacian to the old graph Laplacian, we can consider a contraction matrix P. If the ordering of the vertices for the graph Laplacian of $G = (\tilde{V}, \tilde{E}, \tilde{\omega})$ is $1, .., n$, then let the ordering of each the k vertices in the ith clique be determined by index$(v_{ik}^{\sim}) = (k-1)n + i$. Consider the contraction matrix $P = \begin{bmatrix} I \\ \vdots \\ I \end{bmatrix}$ where $P \in \mathbb{R}^{kn \times n}$ is k copies of the identity matrix. P will contract each vertex in the ith clique C_i into the original vertex $v_i \in V$. Then the new graph Laplacian $\tilde{L}$ can be related with the old Laplacian by $L = P^T \tilde{L} P$. A proof that the contracted graph Laplacian remains a graph Laplacian can be found in [17].

With this we can now prove that the smallest non-zero eigenvalue of a smoothed graph will be strictly less than the smallest non-zero eigenvalue of the original graph it came from.

Theorem 3.0.4 *Using the inner product space $\langle \cdot, \cdot \rangle_D$, let (ϕ_2, λ_2) be the second smallest eigenvalue eigenvector pair of the random walk matrix $D^{-1}L$ of a connected graph G. Then the second smallest eigenvalue of the normalized Laplacian $\tilde{N}$ of the*

smoothed graph $\tilde{G}$, $\tilde{\lambda_2}$ is related to λ_2 by

$$\tilde{\lambda_2} \leq \lambda_2 \frac{1}{1 + \frac{2\omega(G)(ak(k-1)\phi_2^T\phi_2)}{\tilde{\mathbb{1}}^T\tilde{D}\tilde{\mathbb{1}}}} < \lambda_2$$

where k is the number of vertices in each clique,$\omega(G)$ is the sum of the edge weights in G, and a is the edge weight of every edge in each clique.

Proof:

First take note of the fact that $\tilde{D}$ can be decomposed into $\tilde{D} = \tilde{D}_C + \tilde{D}_R$. Where $\tilde{D}_C$ represents the part of the degree matrix that is the sum of weights of edges between vertices in a clique from the graph smoothing process, and $\tilde{D}_R$ represents the part of the degree matrix that has the weights of the edges connecting to all the other vertices in the graph.

We can show that the contraction operators applied to $\tilde{D}_R$ will return it to D. Because we split the weights of the edges such that $\sum_{\tilde{v}\in C} deg_{\tilde{G}}(\tilde{v}) = deg_G(v)$ we can show that $P^T\tilde{D}_RP = \begin{bmatrix} I, ..., I \end{bmatrix} \begin{bmatrix} \tilde{D}_{R1,...,n} & & \\ & \ddots & \\ & & \tilde{D}_{R(k-1)n+1,...,kn} \end{bmatrix} \begin{bmatrix} I \\ \vdots \\ I \end{bmatrix} = \begin{bmatrix} I, ..., I \end{bmatrix} \begin{bmatrix} \tilde{D}_{R1,...,n} \\ \vdots \\ \tilde{D}_{R(k-1)n+1,...,kn} \end{bmatrix} =$
$\tilde{D}_{R1,...,n} + ... + \tilde{D}_{R(k-1)n+1,...,kn} = D$.

We can also simplify the contracted $\tilde{D}_C$, because $\tilde{D}_C = a(k-1)\tilde{I}$ where $\tilde{I}$ is the identity matrix of $\mathbb{R}^{kn}$, and $P^TP = kI$ where I is the identity matrix of $\mathbb{R}^n$. Then $P^T\tilde{D}_CP = a(k-1)P^T\tilde{I}P = a(k-1)kI$.

Let $\tilde{\phi_2} \in \mathbb{R}^{kn}$ and let the constant vector of $\mathbb{R}^{kn}$ be denoted by $\tilde{\mathbb{1}}$. To make $\tilde{\phi_2}$ orthogonal to $\tilde{\mathbb{1}}$ with respect to $\langle\cdot,\cdot\rangle_{\tilde{D}}$, let $\tilde{\phi_2} = P\phi_2 - \alpha\tilde{\mathbb{1}}$, where $\alpha = \frac{\langle P\phi_2, \tilde{\mathbb{1}}\rangle_{\tilde{D}}}{\langle \mathbb{1}, \mathbb{1}\rangle_{\tilde{D}}}$. The eigenvalues of the normalized Laplacian are the same as the random walk matrix, so by the min-max characterization theorem $\tilde{\lambda_2} = \min_{x\perp\tilde{\mathbb{1}}} \frac{x^T\tilde{L}x}{x^T\tilde{D}x}$. Which implies

$$\tilde{\lambda_2} \leq \frac{\tilde{\phi_2}^T\tilde{L}\tilde{\phi_2}}{\tilde{\phi_2}^T\tilde{D}\tilde{\phi_2}} = \frac{(P\phi_2 - \alpha\tilde{\mathbb{1}})^T\tilde{L}(P\phi_2 - \alpha\tilde{\mathbb{1}})}{\tilde{\phi_2}^T\tilde{D}(P\phi_2 - \alpha\tilde{\mathbb{1}})} = \frac{\phi_2^TP^T\tilde{L}P\phi_2 - \alpha\tilde{\mathbb{1}}^T\tilde{L}P\phi_2 - \alpha\phi_2^TP^T\tilde{L}\tilde{\mathbb{1}} + \alpha^2\tilde{\mathbb{1}}^T\tilde{L}\tilde{\mathbb{1}}}{\tilde{\phi_2}^T\tilde{D}(P\phi_2 - \alpha\tilde{\mathbb{1}})}$$

This can be simplified by noticing that $\tilde{L}\tilde{\mathbb{1}} = \tilde{L}\tilde{\mathbb{1}}^T = \tilde{\mathbb{1}}^T\tilde{L} = 0$ because $\tilde{\mathbb{1}} \in \text{null}(\tilde{L})$.Also

from construction $\tilde{\phi_2}$ is orthogonal to $\tilde{\mathbb{1}}$ in the $\tilde{D}$ inner product, so $\tilde{\phi_2}^T \tilde{D} \tilde{\mathbb{1}} = 0$

$$= \frac{\phi_2^T P^T \tilde{L} P \phi_2 - 0 - 0 + 0}{\tilde{\phi_2}^T \tilde{D} P \phi_2} = \frac{\phi_2^T P^T \tilde{L} P \phi_2}{\tilde{\phi_2}^T \tilde{D} P \phi_2}$$

recall that $L = P^T \tilde{L} P$

$$= \frac{\phi_2^T L \phi_2}{\tilde{\phi_2}^T \tilde{D} P \phi_2} = \frac{\phi_2^T D \phi_2}{\phi_2^T D \phi_2} \frac{\phi_2^T L \phi_2}{\tilde{\phi_2}^T \tilde{D} P \phi_2} = \lambda_2 \frac{\phi_2^T D \phi_2}{\tilde{\phi_2}^T \tilde{D} P \phi_2}$$

Because the random walk matrix is symmetric in the D inner product space, we can find eigenvectors that are all orthogonal to each other in that space, and thus we can find a ϕ_2 such that $\phi_2^T D \phi_2 = 1$

$$= \lambda_2 \frac{1}{\tilde{\phi_2}^T \tilde{D} P \phi_2}$$

Now we wish to show that the denominator is greater than 1.

$$= \lambda_2 \frac{1}{\phi_2^T P^T \tilde{D} P \phi_2 - \alpha \tilde{\mathbb{1}}^T \tilde{D} P \phi_2} = \lambda_2 \frac{1}{\phi_2^T P^T \tilde{D}_R P \phi_2 + \phi_2^T P^T \tilde{D}_C P \phi_2 - \alpha \tilde{\mathbb{1}}^T \tilde{D} P \phi_2}$$

$$= \lambda_2 \frac{1}{\phi_2^T D \phi_2 + \phi_2^T P^T (a(k-1)\tilde{I}) P \phi_2 - \alpha \tilde{\mathbb{1}}^T \tilde{D} P \phi_2} = \lambda_2 \frac{1}{1 + a(k-1)\phi_2^T (kI) \phi_2 - \alpha \tilde{\mathbb{1}}^T \tilde{D} P \phi_2}$$

Now focusing on the denominator, because there is a 1, we want to show that

$$a(k-1)\phi_2^T (kI) \phi_2 - \alpha \tilde{\mathbb{1}}^T \tilde{D} P \phi_2 > 0$$

Note the fact that $\tilde{\mathbb{1}}^T \tilde{D}_R P = \mathbb{1}^T D$ Where $\mathbb{1}$ is the constant vector in $\mathbb{R}^n$.

$$a(k-1)\phi_2^T \phi_2 - \alpha \tilde{\mathbb{1}}^T \tilde{D}_R P \phi_2 - \alpha \tilde{\mathbb{1}}^T \tilde{D}_C P \phi_2 = a(k-1)\phi_2^T \phi_2 - \alpha \mathbb{1}^T D \phi_2 - \alpha \tilde{\mathbb{1}}^T \tilde{D}_C P \phi_2$$

Because ϕ_2 is an eigenvector of $D^{-1}L$ and $\mathbb{1} \in \text{null}(L)$ and we're finding the eigenvectors with the $\langle \cdot, \cdot \rangle_D$ inner product, then we know that $\langle \mathbb{1}, \phi_2 \rangle_D = \mathbb{1}^T D \phi_2 = 0$

$$= a(k-1)\phi_2^T \phi_2 - 0 - \alpha \tilde{\mathbb{1}}^T \tilde{D}_C P \phi_2 = a(k-1)\phi_2^T \phi_2 - \alpha \tilde{\mathbb{1}}^T (a(k-1)\tilde{I}) P \phi_2 = a(k-1)\phi_2^T \phi_2 - \alpha a(k-1)(k \mathbb{1}^T) \phi_2$$

Now let's have a closer look at the value of α

$$\alpha = \frac{\langle P\phi_2, \tilde{\mathbb{1}}\rangle_{\tilde{D}}}{\langle \tilde{\mathbb{1}}, \tilde{\mathbb{1}}\rangle_{\tilde{D}}} = \frac{\phi_2^T P^T \tilde{D}\tilde{\mathbb{1}}}{\tilde{\mathbb{1}}^T \tilde{D}\tilde{\mathbb{1}}} = \frac{\phi_2^T P^T (\tilde{D}_R + \tilde{D}_C)\tilde{\mathbb{1}}}{\tilde{\mathbb{1}}^T \tilde{D}\tilde{\mathbb{1}}} = \frac{\phi_2^T P^T \tilde{D}_R\tilde{\mathbb{1}} + \phi_2^T P^T \tilde{D}_C\tilde{\mathbb{1}}}{\tilde{\mathbb{1}}^T \tilde{D}\tilde{\mathbb{1}}}$$

$$= \frac{\phi_2^T D\mathbb{1} + \phi_2^T (a(k-1)k\mathbb{1})}{\tilde{\mathbb{1}}^T \tilde{D}\tilde{\mathbb{1}}} = \frac{a(k-1)k\phi_2^T\mathbb{1}}{\tilde{\mathbb{1}}^T \tilde{D}\tilde{\mathbb{1}}} =$$

Substituting this back into the original equation we get.

$$ak(k-1)\phi_2^T\phi_2 - \alpha ak(k-1)\mathbb{1}^T\phi_2 = ak(k-1)\phi_2^T\phi_2 - (\frac{a(k-1)k\phi_2^T\mathbb{1}}{\tilde{\mathbb{1}}^T \tilde{D}\tilde{\mathbb{1}}})ak(k-1)\mathbb{1}^T\phi_2$$

$$ak(k-1)\phi_2^T\phi_2 - (\frac{(a(k-1)k\phi_2^T\mathbb{1})^2}{\tilde{\mathbb{1}}^T \tilde{D}\tilde{\mathbb{1}}})$$

Now by the Cauchy Schwartz Bunyakovsky inequality we can show that the quantity

$$(\phi_2^T\mathbb{1})^2 = \langle\phi_2, \mathbb{1}\rangle^2 \leq \langle\phi_2, \phi_2\rangle\langle\mathbb{1}, \mathbb{1}\rangle = n\phi_2^T\phi_2$$

using this inequality to bound the previous line in the equation

$$ak(k-1)\phi_2^T\phi_2 - (\frac{(a(k-1)k\phi_2^T\mathbb{1})^2}{\tilde{\mathbb{1}}^T \tilde{D}\tilde{\mathbb{1}}}) \geq ak(k-1)\phi_2^T\phi_2 - (\frac{(a(k-1)k)^2 n\phi_2^T\phi_2}{\tilde{\mathbb{1}}^T \tilde{D}\tilde{\mathbb{1}}})$$

$$= \frac{ak(k-1)\phi_2^T\phi_2(\tilde{\mathbb{1}}^T \tilde{D}\tilde{\mathbb{1}}) - (a(k-1)k)^2 n\phi_2^T\phi_2}{\tilde{\mathbb{1}}^T \tilde{D}\tilde{\mathbb{1}}} = \frac{(\tilde{\mathbb{1}}^T \tilde{D}\tilde{\mathbb{1}} - (a(k-1)k)n)(ak(k-1)\phi_2^T\phi_2)}{\tilde{\mathbb{1}}^T \tilde{D}\tilde{\mathbb{1}}}$$

let $\omega(G)$ represent the sum of all the weights of a graph.

$$= \frac{(2\omega(\tilde{G}) - (a(k-1)k)n)(ak(k-1)\phi_2^T\phi_2)}{\tilde{\mathbb{1}}^T \tilde{D}\tilde{\mathbb{1}}} = \frac{(na(k-1)k + 2\omega(G) - (a(k-1)k)n)(ak(k-1)\phi_2^T\phi_2)}{\tilde{\mathbb{1}}^T \tilde{D}\tilde{\mathbb{1}}}$$

$$= \frac{2\omega(G)(ak(k-1)\phi_2^T\phi_2)}{\tilde{\mathbb{1}}^T \tilde{D}\tilde{\mathbb{1}}} > 0$$

this is greater than 0 because the edge weight $a > 0$, $k > 1$, and because both G and $\tilde{G}$ are positively weighted graphs. Substituting this inequality back into the inequality

relating the eigenvalues we get

$$\tilde{\lambda_2} \leq \lambda_2 \frac{1}{\tilde{\phi_2}^T \tilde{D} P \phi_2} \leq \lambda_2 \frac{1}{1 + \frac{(na(k-1)k+2\omega(G))(ak(k-1)\phi_2^T \phi_2)}{\tilde{\mathbb{1}}^T \tilde{D} \tilde{\mathbb{1}}}} < \lambda_2 \square$$

This proof makes use of the Cauchy Swartz Bunyakovsky inequality to bound the terms in the denominator. However this is not a tight bound, as at the step when we use the CSB, when only expanding one vertex we can show equality. We will now show the results of expanding one vertex into a clique.

W.l.o.g. assume that the $v_1 \in V$ is the only vertex that is expanded into a clique of size k. Then let the ordering of the vertices in the clique other than v_1 be $n+1, ..., n+k-1$. To related the old Laplacian to the new Laplacian, the contraction operator will be

$$P = \begin{bmatrix} 1 & 0 & ... & 0 \\ 0 & 1 & ... & 0 \\ \vdots & & \ddots & \\ 0 & & ... & 1 \\ 1 & 0 & ... & 0 \\ \vdots & 0 & ... & 0 \\ 1 & 0 & ... & 0 \end{bmatrix} = \begin{bmatrix} e_1 & I_{2,..n} \\ \mathbb{1} & \mathbf{0} \end{bmatrix}$$

Where $\mathbb{1}$ is the constant vector in $\mathbb{R}^{k-1}$,e_1 is a standard basis vector of $\mathbb{R}^n$,$I_{2,..,n}$ are columns 2 through n of the identity matrix of $\mathbb{R}^{n \times n}$, and $\mathbf{0}$ is an all zero matrix in $\mathbb{R}^{(k-1)\times(n-1)}$. Now with the ordering that we've defined, one can show that the old graph Laplacian L is related to the new Laplacian $\tilde{L}$ by $L = P^T \tilde{L} P$

Theorem 3.0.5 *Using the inner product space $\langle \cdot, \cdot \rangle_D$, let (ϕ_2, λ_2) be the second smallest eigenvalue eigenvector pair of the random walk matrix $D^{-1}L$ of a connected graph G. Then the second smallest eigenvalue of the normalized Laplacian $\tilde{N}$ of the smoothed graph $\tilde{G}$, $\tilde{\lambda_2}$ is related to λ_2 by*

$$\tilde{\lambda_2} \leq \lambda_2 \frac{1}{1 + \frac{(2\omega(G))(ak(k-1)\phi_{2,1})}{\tilde{\mathbb{1}}^T \tilde{D} \tilde{\mathbb{1}}}}$$

where k is the number of vertices in each clique,$\omega(G)$ is the sum of the weights of all edges in G, and a is the edge weight of every edge in each clique.

Proof:

Just like before, $\tilde{D}$ can be decomposed into $\tilde{D} = \tilde{D}_C + \tilde{D}_R$. Where $\tilde{D}_C$ represents the part of the degree matrix that is the sum of weights of edges between vertices in the clique from the graph smoothing process, and $\tilde{D}_R$ represents the part of the degree matrix that has the weights of the edges connecting to all the other edges in the graph.

We can show that the contraction operators applied to $\tilde{D}_R$ will return it to D. Because we split the weights of the edges such that $\sum_{\tilde{v}\in C} deg_{\tilde{G}}(\tilde{v}) = deg_G(v)$ we can show that $P^T\tilde{D}_R P = D$. We can also simplify the contracted $\tilde{D}_C$.

$$\tilde{D}_C = \begin{bmatrix} a(k-1) & & & & & & & \\ & 0 & & & & & & \\ & & \ddots & & & & & \\ & & & 0 & & & & \\ & & & & a(k-1) & & & \\ & & & & & \ddots & & \\ & & & & & & a(k-1) \end{bmatrix}$$

. Then

$$P^T\tilde{D}_C P = P^T \begin{bmatrix} a(k-1) & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \vdots & 0 & \dots & 0 \\ a(k-1) & 0 & \dots & 0 \\ \vdots & 0 & \dots & 0 \\ a(k-1) & 0 & \dots & 0 \end{bmatrix} = \begin{bmatrix} ak(k-1) & & & \\ & 0 & & \\ & & \ddots & \\ & & & 0 \end{bmatrix}$$

.

Let $\tilde{\phi_2} \in \mathbb{R}^{n+(k-1)}$ and let the constant vector of $\mathbb{R}^{n+(k+1)}$ be denoted by $\tilde{\mathbb{1}}$. To make $\tilde{\phi_2}$ orthogonal to $\tilde{\mathbb{1}}$ with respect to $\langle\cdot,\cdot\rangle_{\tilde{D}}$, let $\tilde{\phi_2} = P\phi_2 - \alpha\tilde{\mathbb{1}}$, where $\alpha = \frac{\langle P\phi_2, \tilde{\mathbb{1}}\rangle_{\tilde{D}}}{\langle \tilde{\mathbb{1}}, \tilde{\mathbb{1}}\rangle_{\tilde{D}}}$.

To reduce the rayleigh quotient to the form below, you simply must follow the

same steps as the previous proof, so we will start from here.

$$\tilde{\lambda_2} \leq \lambda_2 \frac{1}{\tilde{\phi_2}^T \tilde{D} P \phi_2}$$

Now we wish to show that the denominator is greater than 1.

$$= \lambda_2 \frac{1}{\phi_2^T P^T \tilde{D} P \phi_2 - \alpha \tilde{\mathbb{1}}^T \tilde{D} P \phi_2} = \lambda_2 \frac{1}{\phi_2^T P^T \tilde{D}_R P \phi_2 + \phi_2^T P^T \tilde{D}_C P \phi_2 - \alpha \tilde{\mathbb{1}}^T \tilde{D} P \phi_2}$$

Note that the contracted $\tilde{D}_C$ matrix is zero other than the $1,1$ position. So $\phi_2^T P^T \tilde{D}_R P \phi_2 = a(k-1)k\phi_{2,1}^2$ where $\phi_{2,1}$ is the first entry of ϕ_2.

$$= \lambda_2 \frac{1}{\phi_2^T D \phi_2 + a(k-1)k\phi_{2,1}^2 - \alpha \tilde{\mathbb{1}}^T \tilde{D} P \phi_2} = \lambda_2 \frac{1}{1 + a(k-1)k\phi_{2,1}^2 - \alpha \tilde{\mathbb{1}}^T \tilde{D} P \phi_2}$$

Now focusing on the denominator, because there is a 1, we want to show that

$$a(k-1)k\phi_{2,1}^2 - \alpha \tilde{\mathbb{1}}^T \tilde{D} P \phi_2 > 0$$

Note the fact that $\tilde{\mathbb{1}}^T \tilde{D}_R P = \mathbb{1}^T D$ where $\mathbb{1}$ is the constant vector in $\mathbb{R}^n$.

$$a(k-1)k\phi_{2,1}^2 - \alpha \tilde{\mathbb{1}}^T \tilde{D}_R P \phi_2 - \alpha \tilde{\mathbb{1}}^T \tilde{D}_C P \phi_2 = a(k-1)k\phi_{2,1}^2 - \alpha \mathbb{1}^T D \phi_2 - \alpha \tilde{\mathbb{1}}^T \tilde{D}_C P \phi_2$$

Because ϕ_2 is an eigenvector of $D^{-1}L$ and $\mathbb{1} \in \text{null}(L)$ and we're finding the eigenvectors with the $\langle \cdot, \cdot \rangle_D$ inner product, then we know that $\langle \mathbb{1}, \phi_2 \rangle_D = \mathbb{1}^T D \phi_2 = 0$

$$= a(k-1)k\phi_{2,1}^2 - 0 - \alpha \tilde{\mathbb{1}}^T \tilde{D}_C P \phi_2 = a(k-1)k\phi_{2,1}^2 - \alpha(a(k-1)k\phi_{2,1})$$

Now let's have a closer look at the value of α

$$\alpha = \frac{\langle P\phi_2, \tilde{\mathbb{1}} \rangle_{\tilde{D}}}{\langle \tilde{\mathbb{1}}, \tilde{\mathbb{1}} \rangle_{\tilde{D}}} = \frac{\phi_2^T P^T \tilde{D} \tilde{\mathbb{1}}}{\tilde{\mathbb{1}}^T \tilde{D} \tilde{\mathbb{1}}} = \frac{\phi_2^T P^T (\tilde{D}_R + \tilde{D}_C) \tilde{\mathbb{1}}}{\tilde{\mathbb{1}}^T \tilde{D} \tilde{\mathbb{1}}} = \frac{\phi_2^T P^T \tilde{D}_R \tilde{\mathbb{1}} + \phi_2^T P^T \tilde{D}_C \tilde{\mathbb{1}}}{\tilde{\mathbb{1}}^T \tilde{D} \tilde{\mathbb{1}}}$$

$$= \frac{\phi_2^T D \mathbb{1} + a(k-1)k\phi_{2,1}}{\tilde{\mathbb{1}}^T \tilde{D} \tilde{\mathbb{1}}} = \frac{a(k-1)k\phi_{2,1}}{\tilde{\mathbb{1}}^T \tilde{D} \tilde{\mathbb{1}}}$$

Substituting this back into the original equation we get.

$$a(k-1)k\phi_{2,1}^2 - \alpha(a(k-1)k\phi_{2,1}) = a(k-1)k\phi_{2,1}^2 - (\frac{a(k-1)k\phi_{2,1}}{\tilde{\mathbb{1}}^T\tilde{D}\tilde{\mathbb{1}}})a(k-1)k\phi_{2,1}$$

$$a(k-1)k\phi_{2,1}^2 - (\frac{(a(k-1)k\phi_{2,1})^2}{\tilde{\mathbb{1}}^T\tilde{D}\tilde{\mathbb{1}}}) = \frac{(\tilde{\mathbb{1}}^T\tilde{D}\tilde{\mathbb{1}})(a(k-1)k\phi_{2,1}^2) - ((a(k-1)k\phi_{2,1})^2}{\tilde{\mathbb{1}}^T\tilde{D}\tilde{\mathbb{1}}}$$

$$= \frac{(\tilde{\mathbb{1}}^T\tilde{D}\tilde{\mathbb{1}} - a(k-1)k)(a(k-1)k\phi_{2,1}^2)}{\tilde{\mathbb{1}}^T\tilde{D}\tilde{\mathbb{1}}} = \frac{(a(k-1)k + 2\omega(G) - a(k-1)k)(a(k-1)k\phi_{2,1}^2)}{\tilde{\mathbb{1}}^T\tilde{D}\tilde{\mathbb{1}}}$$

$$= \frac{2\omega(G)(a(k-1)k\phi_{2,1}^2)}{\tilde{\mathbb{1}}^T\tilde{D}\tilde{\mathbb{1}}}$$

substituting this back into the inequality of the eigenvalues.

$$\tilde{\lambda_2} \leq \lambda_2 \frac{1}{1 + \frac{2\omega(G)(a(k-1)k\phi_{2,1}^2)}{\tilde{\mathbb{1}}^T\tilde{D}\tilde{\mathbb{1}}}}$$

The fraction will be less than 1 when $a > 0$, $k > 1$, and $\phi_{2,1} \neq 0$.

We can now use Cheeger's inequality to show how the optimal conductance of the graph G and the fully smoothed graph $\tilde{G}$ are related.

$$2\phi(G) \geq \lambda_2 > \tilde{\lambda_2} \geq \frac{\phi(\tilde{G})^2}{2}$$

Chapter 4

Experiments

To get a better idea for how the parameters of the algorithm effect the performance of the clustering technique, MATLAB scripts were written to test all of the clustering methods against each other. The proofs in section 3 imply that larger edge weights for the edges in the clique, and larger sized cliques may be optimal. However the bounds on the eigenvalues are not tight, and the proof shows that the eigenvalues will decrease, but cannot directly prove that the optimal solutions to the Ratio Cut, Normalized Cut, or Conductance will decrease. One thing to consider about the problem of spectral clustering, is the fact that the method is not an excellent black box solver. The success of spectral clustering relies heavily upon how the graph is constructed. Choices of kernels, number of clusters, graph building algorithms, and their specific parameters make setting the up graph a difficult task. In the general case, there are few results on how to automate the process. Though there has been some success in identifying the number of clusters.

4.1 Parameters

For all of the experiments there are different types of parameters for the subroutines of the programs written.

4.1.1 Graph Building Parameters

- type - Which graph building method used, could either be fully connected, ϵ neighbors, or k nearest.
- threshold - For the ϵ neighborhood method, this quantity reflects how large or small the distance function must be for an edge to be put into a graph. For the k nearest method, this quantity is the choice of k.

4.1.2 Kernel Parameters

For the experiments the Gaussian kernel (or Radial Basis Function) was used to for measuring the distance of the data points. The Gaussian kernel is defined as

$$RBF(x, y, \sigma, p) := exp(\frac{-\|x - y\|_p^2}{2\sigma^2})$$

- variance - controls the variance of the Gaussian kernel.
- normtype - the p value of the p norm to compute the distance between points(typically 2 norm was used).

4.1.3 Proposed Algorithm Parameters

- radius- controls the variance of the data points sampled from the n dimensional Gaussian distribution in the Smoothed Embedding Clustering method.
- ratio increase - a positive integer that chooses how many copies of data points should be sampled, or what the size of the cliques should be.

4.2 Data

Fisher's Iris Flower data set is a set of measurements of three species of Iris flower;Iris setosa,Iris Virginica, and Iris Versicolor. The data set was published in 1936 by Donald Fisher to test linear discriminant analysis. The data set is interesting because when you plot the measurements in $\mathbb{R}^4$ there are only two clear clusters, two species of flower form a tight group together. Because this dataset is not linearly separable in this space, k-means typically does an unsatisfactory job, and mis-clusters the data. It can be shown that when the data is projected onto its non-linear branching principal components it becomes separable [6].

4.3 Tests

4.3.1 Evaluating Success

Even though a ground truth is provided for a dataset, it may not be so straight forward to compare the results of a clustering algorithm against the ground truth. Because of the random nature of unsupervised clustering algorithms, the integers labeling each cluster may differ over different tests. Furthermore because the clusters may be of the incorrect size, its difficult to know which cluster is which by comparing the sizes to the ground truth cluster size. To evaluate success of each algorithm, each cluster is compared to its ground truth, any cluster with the most matches is considered the true cluster, and the number of matched vertices is returned.

4.3.2 Clustering Comparison

The first test on the data conducted is comparing all of the clustering algorithms against one another. All of the Algorithms in Chapter 2 all assign the clusters to each vertex or data point, and the results are all compared against the ground truth of the data. Experiments are conducted using different graph building methods.

4.3.3 Clique Size

The next test is to identify optimal clique size for the Graph Smoothing Clustering algorithm. Because the proof bounds can still be improved, it is unclear what is the true optimal size of the cliques.There is also th e matter that the original edge weights are proportional to the $\frac{1}{\text{clique size}}$, so as the cliques get larger and larger they become more and more like disconnected components. The script runs the Graph Smoothing Clustering algorithm multiple times with different clique sizes, and averages the results of correctly clustered vertices.

4.3.4 Clique Edge Weight

With the same reasoning as before, it is unclear what the best value for the edge weights inside the cliques is. The Graph Smoothing Clustering algorithm is run multiple times, and each test reports the average number of correctly clustered vertices. In the proofs, the largest edge weight would be optimal. However from a random walker perspective of spectral clustering, this would lead to the random walker being much more likely to only walk around the clique instead of the rest of the network.

4.3.5 Separation of λ_2

The last experiment is to see how well separated λ_2 is from the rest of the eigenvalues.The separation of an eigenvalue λ, and a square matrix A is defined as

$$sep(\lambda, A) := \|(A - \lambda I)^{-1}\|^{-1} \text{ for a consistent norm } \|\cdot\|$$

The quantity is the smallest distance between that eigenvalue and the rest of the eigenvalues for the matrix A. The separation of the eigenvalue will determine how easy it is to compute the corresponding eigenvector.A large separation for an eigenvalue usually means that iterative eigensystem solvers will converge on that eigenvector, eigenvalue pair quickly with high accuracy. The separation of the eigenvalue will be tested for the same values of the internal edge weights and the clique sizes.

Chapter 5

Results

5.1 Fisher Iris

The figures in this section compare each of the clustering methods against one another. The results are in the form of a bar graph labeling how many vertices were mislabeled in the clusters provided by each method. The first figure is for the ϵ neighborhood graph construction method

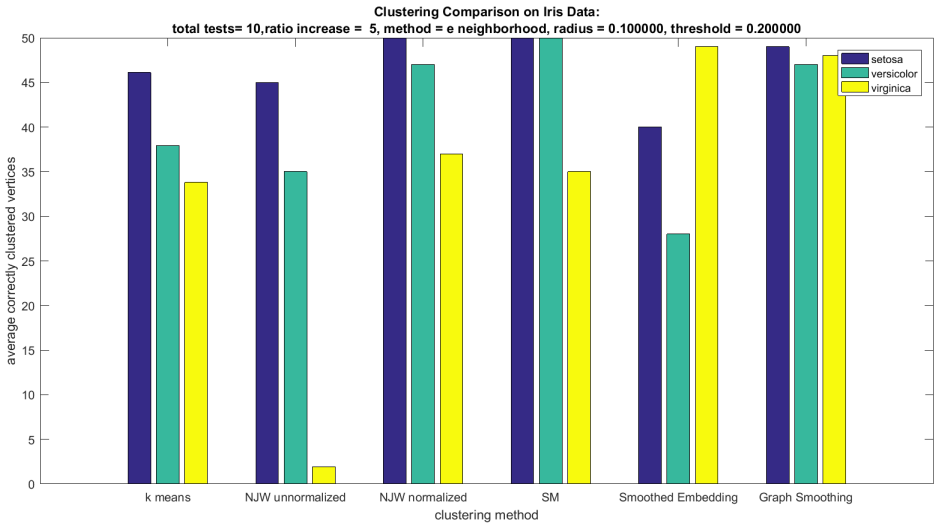

Figure 5.1: Comparing all implemented clustering algorithms against one another. Graph built using ϵ neighborhood method where ϵ = .2, smoothing methods used 5 times the number of points. Results averaged over 10 tests.This figure shows the Graph Smoothing algorithm greatly outperforming other methods

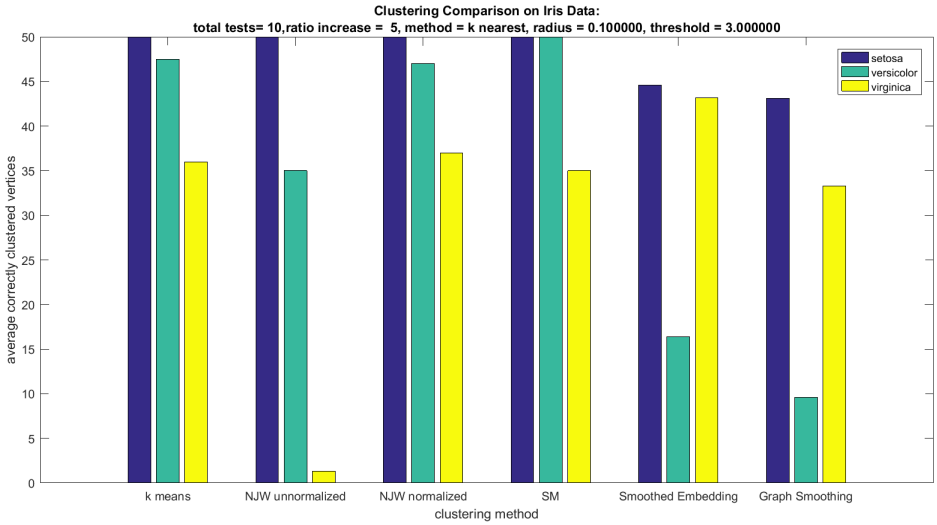

Figure 5.2: Comparing all implemented clustering algorithms against one another. Graph built using k nearest method where $k = 3$, smoothing methods used 5 times the number of points. Results averaged over 10 tests. Results showing subpar performance from Graph Smoothing method, k means and Shi Malik Spectral clustering offering best results

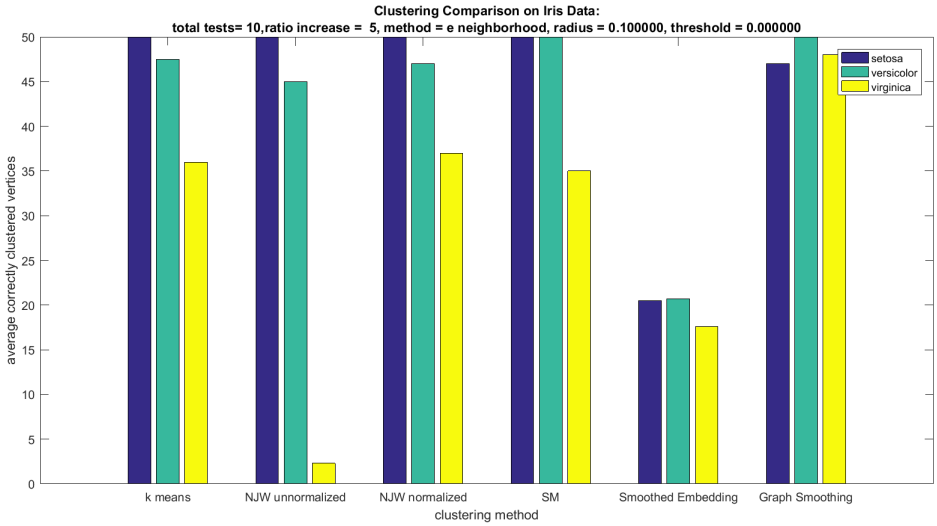

Figure 5.3: Comparing all implemented clustering algorithms against one another. Graph built using the fully connected method. Smoothing methods used 5 times the number of points, results averaged over 10 tests. Results showing best results coming from Graph Smoothed Clustering

5.2 Altering Internal Edge Weights

The follow figures are the results of altering the edge weights in the graph smoothing procedure.All experiments build the graph from the Iris data with a fully connected method.

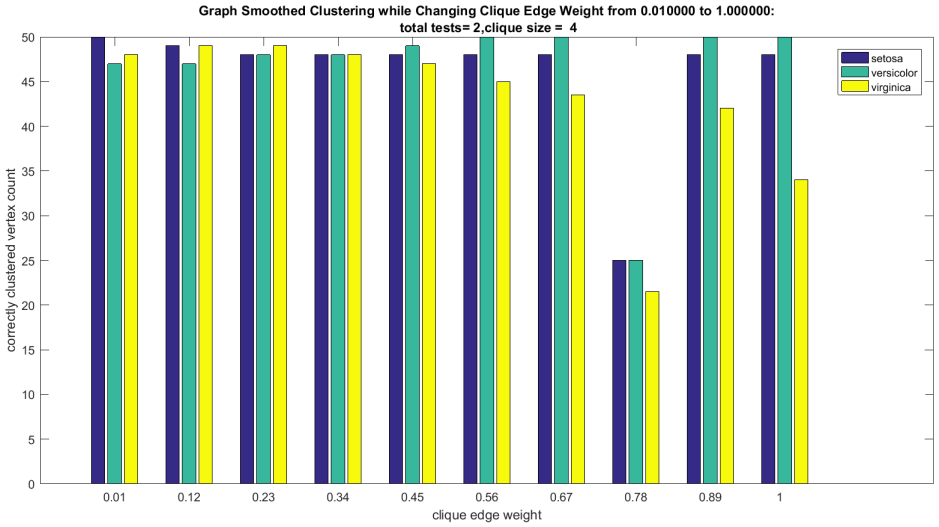

Figure 5.4: Performance of the Graph Smoothed Clustering algorithm as clique edge weight varies.Averaged over 2 tests, clique size is 4. Figure shows that performance is similar for edge weights less than .5, success starts to diminish as edge weights become larger than .5.

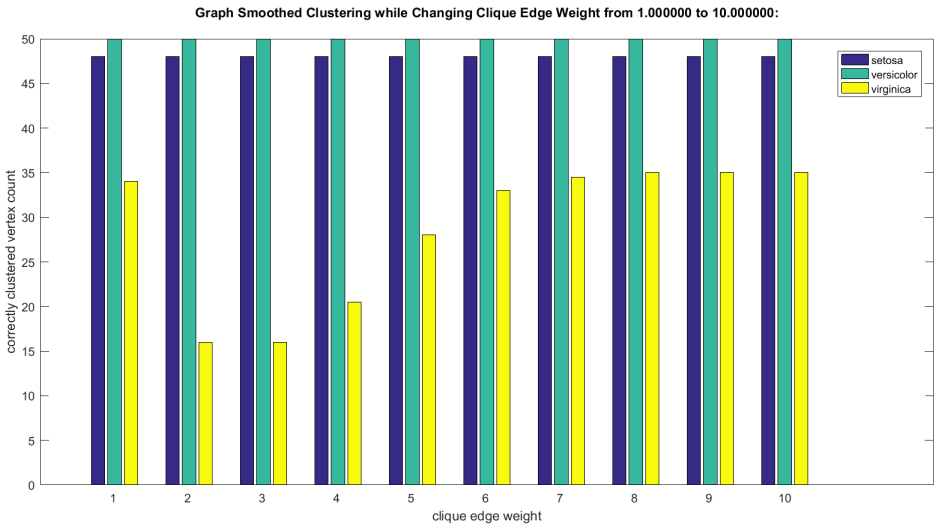

Figure 5.5: Performance of the Graph Smoothed Clustering algorithm as clique edge weight varies.Clique size set to 4. Figure shows small dip from edge weights ranged 2-5. Edge weights $>= 1$ still don't perform as well as edge weights < 1.

The following figures measure how the separation smallest non-zero eigenvalue of the Normalized Laplacian changes. The constant value in each plot corresponds to the original separation of λ_2.

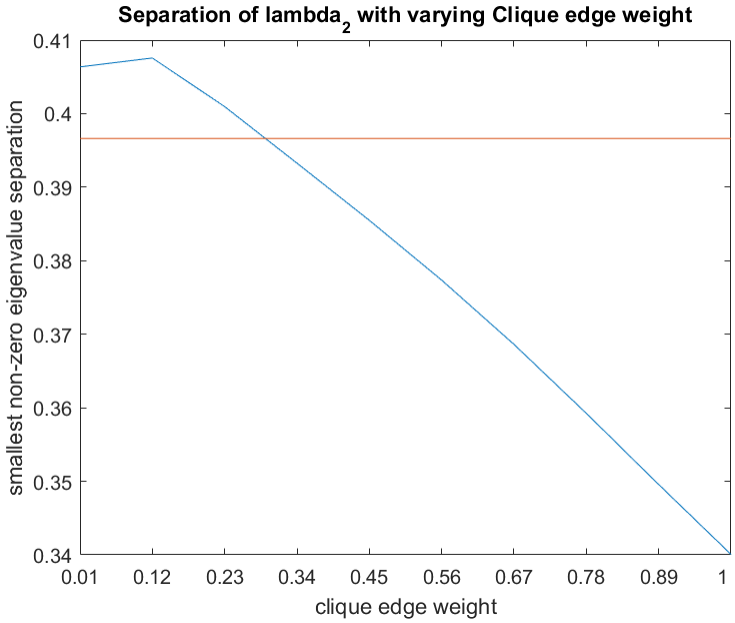

Figure 5.6: Orange Line is the original separation of λ_2 for the normalized Laplacian of G. The blue line is the separation of λ_2 for the normalized Laplacian of $\tilde{G}$. Edges weights vary from .01 to 1, the larger the edge weight the smaller the separation.

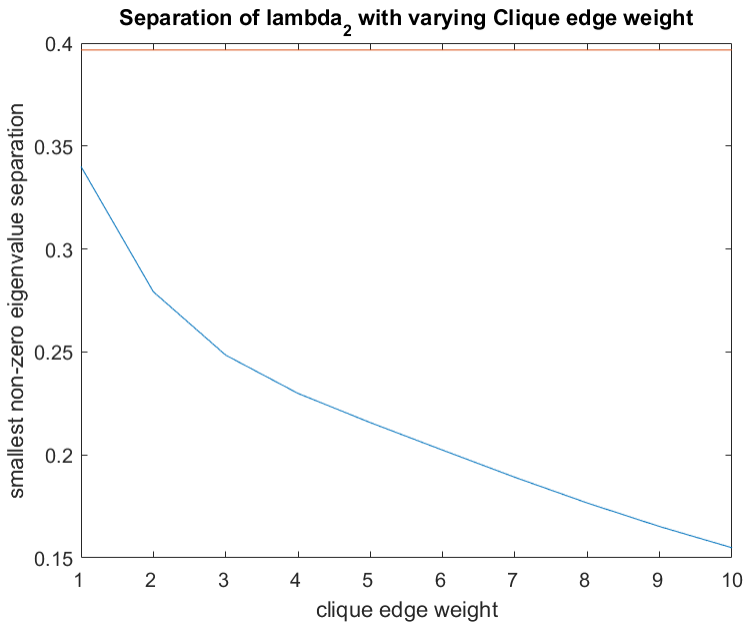

Figure 5.7: Orange Line is the original separation of λ_2 for the normalized Laplacian of G. The blue line is the separation of λ_2 for the normalized Laplacian of $\tilde{G}$. The edges range from 1 to 10, and for large edge weight we see smaller separation.

5.3 Altering Clique Sizes

The figure on the left shows how the graph smoothed clustering algorithm performs when the size of the cliques are increase.For these experiments the graphs were built using the fully connected method. The internal edge weight in all experiments stays constant at $a = .2$.

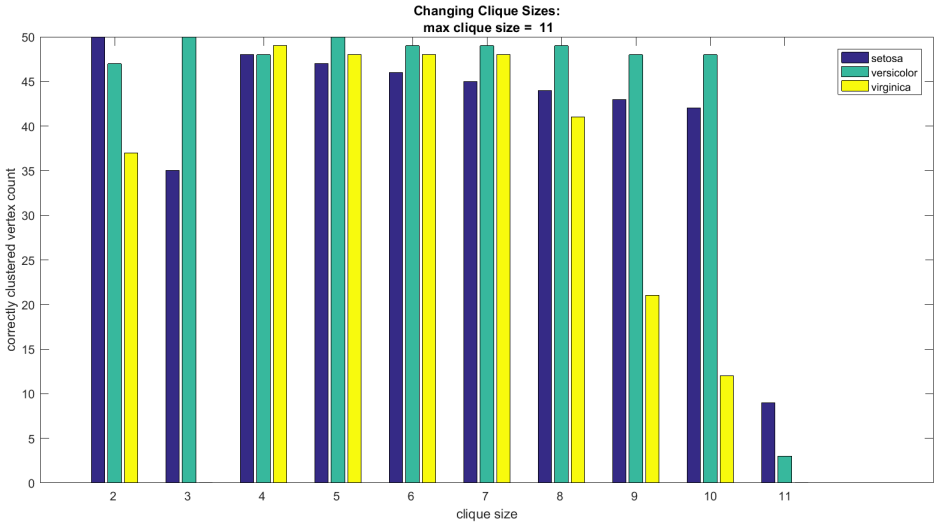

Figure 5.8: This figure compares the results of Graph Smoothed Clustering as the size of the cliques is altered. Clique sizes range from 2-11. The best results are present for clique sizes 4-6.

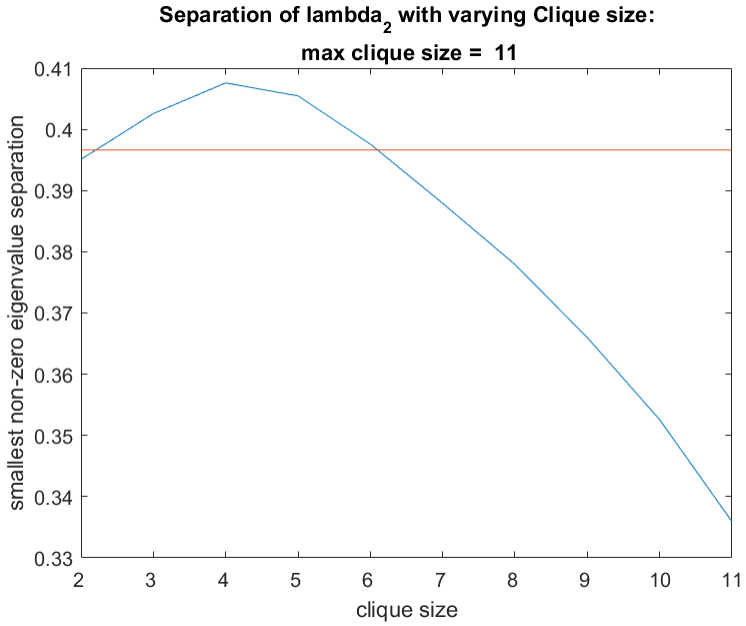

Figure 5.9: Orange line is separation of λ_2 of normalized Laplacian of G. Blue line is the separation of λ_2 of the normalized Laplacian of the smoothed graph $\tilde{G}$. Clique sizes range from 2-11.

5.4 Embedding Plot

This section has plots of the 3 dimensional embeddings from the normalized Laplacian before and after applying the graph smoothing process. Each plot uses a different graph building method. Figures on the left are the original embedding from the normalized Laplacian where each color is a different cluster from the ground truth of the Iris data. Figures on the right are the embeddings of the smoothed graph where the colors are the same clusters, but darker x's represent vertices added from the graph smoothing process. Each smoothing process uses a clique of 5 vertices for the smoothing process.

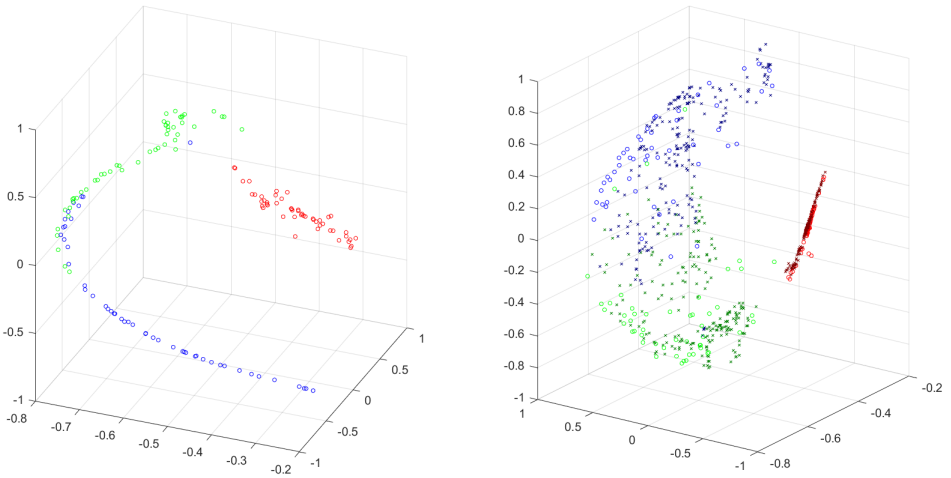

Figure 5.10: Graph Smoothing applied on graph built with ϵ neighborhood with tolerance of .2

Figure 5.10 shows how the process of graph smoothing changes the embeddings from the smallest non-constant eigenvectors of the normalized Laplacians. The image on the left is the original embedding, which plots all of the points in thin arcs. We can see a separated cluster, and another cluster of points of two groups with a small overlap. The plot on the right shows a much more spread out embedding after applying the graph smoothing process. There is significant overlap between the same two desired clusters like before, however most of the points overlapping are new vertices introduced by the smoothing process. The number of overlapping

vertices from the original graph is much less than before, which explains the improved performance of the graph smoothing algorithm. One thing to note about the graph, was that the ϵ threshold was quite low, meaning that the graph tested was dense.

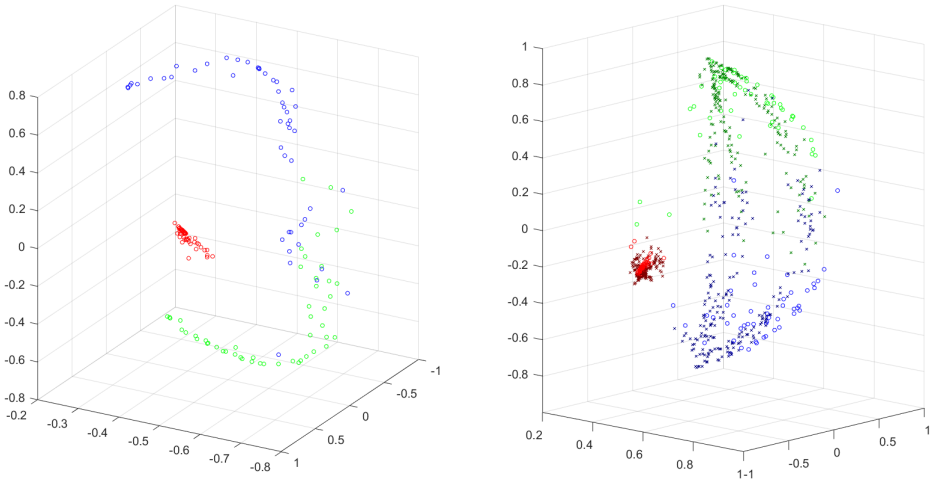

Figure 5.11: Graph Smoothing applied on graph built with fully connected method.

These figures are of the same comparison as before, but on a complete graph. Note that the clusters are grouping tighter than the ϵ neighborhood graph, with less overlap between the green and blue clusters. When running the graph smoothing process, the original vertices have significantly less overlap in the new embedding than the original embedding, even improving upon the ϵ neighborhood graph results in figure 5.10. This seems to indicate that the denser the graph is, the better the graph smoothing clustering results will be.

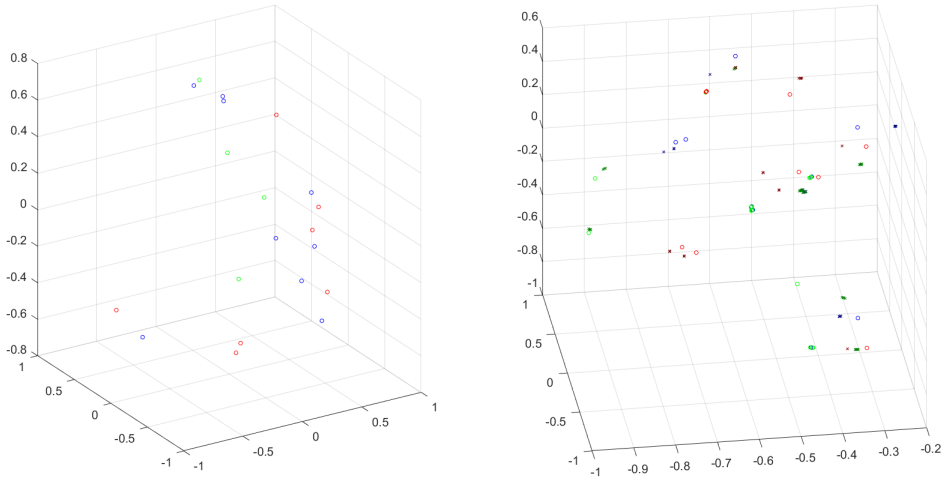

Figure 5.12: Graph Smoothing applied on graph built with 3 nearest neighbors. This embedding plots a lot of vertices to the same place, the clusters are not clear in either figure. This corresponds with the poor clustering performance when tested on the Iris data.

Figure 5.12 shows why the spectral clustering performs so poorly on k nearest graphs with low degree vertices. The embeddings are poorly separated and no distinct clusters can be identified in either plot. Many points have been clustered to the same point in space, giving the appearance that there are fewer vertices in the graph. This is disappointing, because in practice we desire graphs with few edges. Sparse graphs have sparse matrix representations, and eigenvector computation scales well as the number of vertices increase. This also supports the idea that the graph smoothing process works better with denser graphs.

Chapter 6

Conclusion

All of the tests for the dense graph building methods show that Graph Smoothing clustering outperforms the rest of the algorithms. However the Smoothed Embedding algorithm did not perform very well for any of the graphs built. This is most likely due to the fact that the perturbation doesn't account for the fact that the points are embedded onto a hypersphere, and that the new graph built does not use a kernel that measures the distance of points on the surface of a hypersphere. The reason why graph smoothing only performs well on a dense graph is not fully understood. It may have to do with the fact that the implementations of the k nearest graph building did not include any edge weights. Because the graph smoothing algorithm scales the values of the edge weights, when there are no weights each edge is assigned an edge weight of 1 and the scaling is applied to this weighted graph. This scaling may prematurely lead to cliques simulating disconnected components, especially in a sparse network produced by the k nearest method.

The proofs in chapter 3 for the full graph smoothing procedure make use of the Cauchy Schwartz Bunyakovsky inequality, which suggests that there are tighter bounds that can be found. In the next proof where only one vertex is smoothed that step is replaced with an equality, confirming the suspicion. Future work should revolved around tightening these bounds as they may offer a theoretical basis for how we should choose the clique sizes and internal edge weights to minimize the new smallest non-zero eigenvalue of the normalized Laplacian produced. In addition to finding a tighter bound, theoretical results for how the eigenvalue will change when using a less homogeneous smoothing method would be useful. Perhaps cliques of different sizes could be used for different vertices in the graph. Also, instead of every edge in the clique being the same, different edge weights could be chosen. Currently

every vertex being expanded into a clique with the same edge weight resembles a uniform distribution of transitioning to new vertices, but perhaps different distributions could improve clustering if the graph had a special structure.

Bibliography

[1] Pavlo D Antonenko, Serkan Toy, and Dale S Niederhauser, *Using cluster analysis for data mining in educational technology research*, Educational Technology Research and Development, 60 (2012), pp. 383–398.

[2] David Arthur and Sergei Vassilvitskii, *k-means++: The advantages of careful seeding*, in Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035.

[3] Fan RK Chung, *Spectral graph theory*, vol. 92, American Mathematical Soc., 1997.

[4] Biswa Nath Datta, *Numerical Linear Algebra and Applications, Second Edition*, SIAM, 2nd ed., 2010.

[5] Shmuel Friedland and Reinhard Nabben, *On cheeger-type inequalities for weighted graphs*, Journal of Graph Theory, 41 (2002), pp. 1–17.

[6] Alexander N Gorban, Neil R Sumner, and Andrei Yu Zinovyev, *Topological grammars for data approximation*, Applied Mathematics Letters, 20 (2007), pp. 382–386.

[7] Kenneth M Hall, *An r-dimensional quadratic placement algorithm*, Management science, 17 (1970), pp. 219–229.

[8] Anil K Jain, *Data clustering: 50 years beyond k-means*, Pattern recognition letters, 31 (2010), pp. 651–666.

[9] David R Karger, *Global min-cuts in rnc, and other ramifications of a simple min-cut algorithm.*

[10] Stuart P Lloyd, *Least squares quantization in pcm*, Information Theory, IEEE Transactions on, 28 (1982), pp. 129–137.

[11] Shyam Varan Nath, *Crime pattern detection using data mining*, in Web Intelligence and Intelligent Agent Technology Workshops, 2006. WI-IAT 2006 Workshops. 2006 IEEE/WIC/ACM International Conference on, IEEE, 2006, pp. 41–44.

[12] Andrew Y Ng, Michael I Jordan, Yair Weiss, et al., *On spectral clustering: Analysis and an algorithm*, Advances in neural information processing systems, 2 (2002), pp. 849–856.

[13] David Reich, Alkes L Price, and Nick Patterson, *Principal component analysis of genetic data*, Nature genetics, 40 (2008), pp. 491–492.

[14] Jianbo Shi and Jitendra Malik, *Normalized cuts and image segmentation*, Pattern Analysis and Machine Intelligence, IEEE Transactions on, 22 (2000), pp. 888–905.

[15] Gilbert W Stewart, *Matrix algorithms volume 2: eigensystems*, vol. 2, Siam, 2001.

[16] Shubhendu Trivedi, Zachary A Pardos, Gábor N Sárközy, and Neil T Heffernan, *Spectral clustering in educational data mining.*, in EDM, Citeseer, 2011, pp. 129–138.

[17] John C Urschel, Xiaozhe Hu, Jinchao Xu, and Ludmil T Zikatanov, *A cascadic multigrid algorithm for computing the fiedler vector of graph laplacians*, arXiv preprint arXiv:1412.0565, (2014).

[18] Ulrike Von Luxburg, *A tutorial on spectral clustering*, Statistics and computing, 17 (2007), pp. 395–416.

[19] Ulrike Von Luxburg, Sébastien Bubeck, Stefanie Jegelka, and Michael Kaufmann, *Consistent minimization of clustering objective functions*, in Neural Information Processing Systems, 2007.