8-16-2018

Robust Path-based Image Segmentation Using Superpixel Denoising

Renee T. Meinhold
rtm9271@rit.edu

Robust Path-based Image Segmentation Using Superpixel Denoising

by

Renee T. Meinhold

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Science in Applied and Computational Mathematics
School of Mathematical Sciences, College of Science

Rochester Institute of Technology
Rochester, NY
August 16, 2018

Committee Approval:

R·I·T
College of
SCIENCE

Dr. Nathan Cahill Date
School of Mathematical Sciences
Thesis Advisor

Dr. Nathaniel Barlow Date
School of Mathematical Sciences
Committee Member

Dr. Kara Maki Date
School of Mathematical Sciences
Committee Member

Dr. John Hamilton Date
School of Mathematical Sciences
Committee Member

Dr. Matthew Hoffman Date
School of Mathematical Sciences
Director of Graduate Programs

Abstract

Clustering is the important task of partitioning data into groups with similar characteristics, with one category being spectral clustering where data points are represented as vertices of a graph connected by weighted edges signifying similarity based on distance. The longest leg path distance (LLPD) has shown promise when used in spectral clustering, but is sensitive to noisy data, therefore requiring a data denoising procedure to achieve good performance. Previous denoising techniques have involved identifying and removing noisy data points, however this is not a desirable pre-clustering step for data sets with a specific structure like images. The process of partitioning an image into regions of similar features known as image segmentation can be represented as a clustering problem by defining the vector of intensity and spatial information at each pixel as data point. We therefore propose the method of pre-cluster denoising to formulate a robust LLPD clustering framework. By creating a fine clustering of approximately equal-sized groups and averaging each, a reduced number of data points can be defined that represent the relevant information of the original data set by locally averaging out noise influence. We can then construct a smaller graph representation of the data based on the LLPD between the reduced data points, and identify the spectral embedding coordinates for each reduced point. An out-of-sample extension procedure is then used to compute spectral embedding coordinates at each of the original data points, after which a simple (k-means) clustering is performed to compute the final cluster labels. In the context of image segmentation, computing superpixels provides a nice structure for performing this type of pre-clustering. We show how the above LLPD framework can be carried out in the context of image segmentation, and show that a simple computationally efficient spatial interpolation procedure can be used instead to extend the embedding in a way that yields better segmentation performance with respect to ground truth on a publicly available data set. Similar experiments are also performed using the standard Euclidean distance in place of the LLPD to show the proficiency of the LLPD for image segmentation.

Contents

List of Figures

List of Tables

I. Introduction

Clustering, the process of partitioning a data set into groups with similar characteristics, is an important automated task with uses in many fields. Many techniques have been created to solve this problem, with one of the most popular being spectral clustering [2]. A spectral clustering method represents the data as a graph where vertices represent data points and edges show the similarity between those data points, and then computes a new representation, or embedding, of each data point based on the eigendecomposition of an affinity matrix representing graph edges. The popular k-means clustering algorithm can then be applied to the embedded coordinates to compute the final point labels. Although k-means is limited in capability for data on a manifold in a high-dimensional space, the embedding produced by the spectral clustering procedure gives a basis for a lower-dimensional representation that preserves local relationships, allowing for this simpler clustering procedure to perform well in the embedding space.

An important component of spectral clustering methods is how the affinity matrix is defined. A common method employs a Gaussian kernel using the Euclidean distance between data points; however, this often performs poorly for elongated or non-convex clusters where data points may be separated by a large Euclidean distance but are connected by a high density of points. Therefore, for data sets composed of high density clusters with low density noise, a path-based distance metric between points can overcome these challenges. The longest-leg path distance (LLPD) between vertices on a graph is the minimum edge weight such that all unique paths between the two vertices contain an edge of at least this weight, and has been shown to perform well in spectral clustering, where edges are weighted by Euclidean distance to compute the LLPD, under certain assumptions about the data model [1]. The LLPD, however, is sensitive to noise and outlying data points, so denoising before computing an LLPD spectral clustering is necessary. Denoising schemes usually rely on identifying and removing noise points before computing the spectral clustering.

The removal of outlying points from a data set may be difficult or unreasonable for certain data sets, like images, that have a specific structure. The clustering problem applied to images is known as image segmentation, and it involves partitioning an image into regions containing similar pixels, where similarity is based on both spatial and feature components. Removing image pixels from an image data set thus interrupts the spatial continuity of the final segmentation as well as the spatial similarity between two image pixels. However, keeping all NM pixels of an $N \times M$ image can be problematic from a computation point of view since this would require storing and estimating

eigenvectors of an $NM \times NM$ matrix, and even an average sized image can contain hundreds of thousands of pixels.

In this thesis, we propose a new method for robustly performing LLPD spectral clustering. This method first denoises the data set by performing a pre-clustering of the data on a fine scale, meaning the number of pre-clusters is much larger than the final number of data clusters desired, but much less than the total number of data points. All data points in a pre-cluster are then averaged to form a reduced data set. Under the assumption that noise points are of a much lower density than cluster points, a large enough pre-cluster will contain mostly cluster data points with only a few noise points. Averaging all point in a pre-cluster will largely reflect the relevant cluster data, making the effect of noise much smaller. The spectral clustering algorithm is applied to the reduced data set, yielding an embedding of the reduced data set that is robust to noise of the full data set. The embedding is extended back to the full data set using an out-of-sample extension technique [26] to get the final clustering.

For an image, pre-clusters can be computed by oversegmenting an image into a few hundred or thousand small spatial regions of similar pixels called superpixels [16]. These superpixels are then represented by the mean color or feature and spatial location of all component pixels and an embedding is computed based on a graph whose vertices are defined over the set of superpixels. Since image pixels are defined at a known spatial locations, the embedding can be extended to each pixel location by a simple spatial interpolation, as opposed to the more general out-of-sample extension technique. The spatial interpolation method is much less computationally expensive for LLPD spectral clustering, but it lacks the mathematical foundation of the out-of-sample extension. Both methods have the advantage over other techniques that the noise does not have to be identified specifically, and significantly reduces the size of the eigendecomposition problem.

The goal of this thesis is to enable robust image segmentation via approximate LLPD spectral clustering, and show the advantage of this method employing path-based distances over the use of the standard euclidean distance. This thesis is organized as follows: Section 2 provides an introduction to several clustering techniques pertinent to this research and an introduction to basic concepts of image segmentation. Section 3 explains various aspects of the LLPD including its formal definition, prior uses in clustering, computation, and specific advantages. Section 4 introduces our new denoising method providing specific motivating examples, mathematical background, schematic for computing, and optional spatial smoothing. We also discuss the simpler spatial interpolation method, and its advantages and weaknesses. We then perform a series of

image segmentation tests on the publicly available BSDS-500 [21] data set in Section 5, comparing L2 and LLPD spectral clustering, and comparing out-of-sample extension to spatial interpolation methods. The results are discussed both qualitatively and quantitatively in the context of well known segmentation performance measures [14]. Section 6 concludes the thesis by summarizing our main results and discussing future directions this research can take.

II. Introduction to Clustering and Image Segmentation

Given a set of data points $X = \{x_1, x_2, \ldots, x_n\}$ with $x_i \in \mathbb{R}^m$, the process of partitioning X into a set of k groups with similar characteristics $\mathbb{C} = \{C_1, C_2, \ldots, C_k\}$ is called *clustering*, with $\mathbb{C}$ often referred to as *a clustering of* X. We assume for this thesis that $\mathbb{C}$ is a strict partitioning of X, meaning it is exhaustive and all clusters are mutually exclusive, however there is a separate body of work known as "fuzzy" clustering allowing each data point to belong to more than one cluster. Clustering is an unsupervised learning technique, which, unlike supervised learning techniques, assumes that there are no response variables $Y = \{y_1, y_2, \ldots, y_n\}$ corresponding to each data point. This is one of the most fundamental problems of machine learning because of its applicability to problems in a wide variety of fields, such as pattern recognition, bioinformatics, finance, and image processing. As such, many different approaches have been proposed to solve this problem. The subgroups of solution techniques that are most relevant to this research are centroid-based, hierarchical, and graph-based clustering, each of which are summarized below.

II.1 Centroid-based clustering

Centroid-based clustering represents each cluster C_j with a central vector $\hat{C}_j$, then creates k clusters by minimizing the sum of the squared Euclidean distances between $\hat{C}_j$ and each data point $x \in C_j$. The most popular centroid-based clustering algorithm is the k-means algorithm where the central vector is the average of the data points in its cluster,

$$\hat{C}_j = \bar{x}_j = \frac{1}{|C_j|} \sum_{x \in C_j} x. \tag{II.1}$$

The optimization problem of the k-means algorithm can then be written formally as

$$\mathbb{C}^* = \arg\min_{\mathbb{C}} \sum_{j=1}^{k} \sum_{x \in C_j} \|x - \bar{x}_j\|_2^2, \tag{II.2}$$

where $\mathbb{C}^*$ is the $k-$means clustering. Finding the exact solution of Equation II.2 is NP-hard, but the solution can be easily and quickly approximated with an iteration scheme, initialized by k random points in $\mathbb{R}^m$ representing the k cluster vectors. Each data point is then assigned to the closest cluster vector, and the cluster vector is recomputed using the newly formed clusters. This

process is then repeated until the central vectors converge. An outline of this process is displayed in Figure 1.

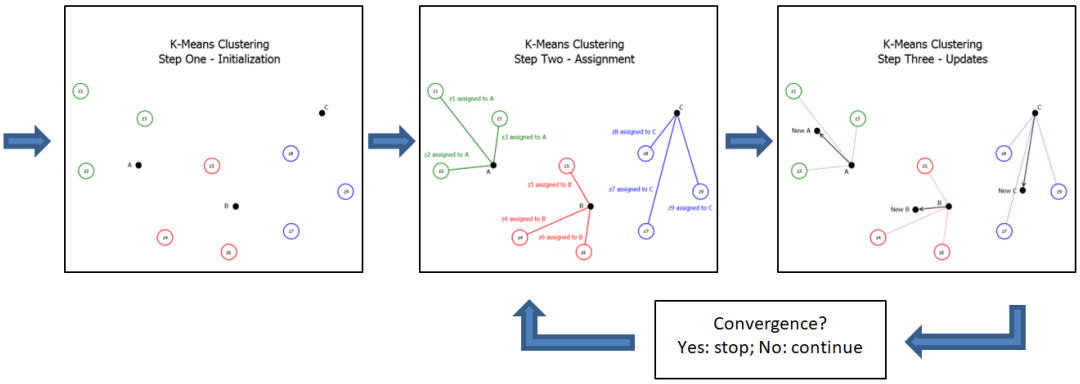


Figure 1: Outline of k-means iteration scheme. This process is initialized by assigning k random central vectors shown in Step One, then each point is then assigned to its closest central vector in Step Two. The k central vectors are then recomputed as the average of all points assigned to it, and Steps Two and Three are repeated until the central vectors have converged to a point within a user specified tolerance. Visualizations from http://www.turingfinance.com

This approximate solution, however, can be dependent on initialization since the iteration scheme may converge to a local minimum. Therefore, computing the clustering for different initializations may be necessary. A limitation of k-means clustering is that it has been shown to perform inadequately for data sets composed of poorly separated, noisy, and/or non-spherical clusters. Because of these limitations, a variety of more sophisticated techniques (including spectral clustering) first compute or define a new representation of the data in which these problems are lessened, followed by k-means clustering on the new representation.

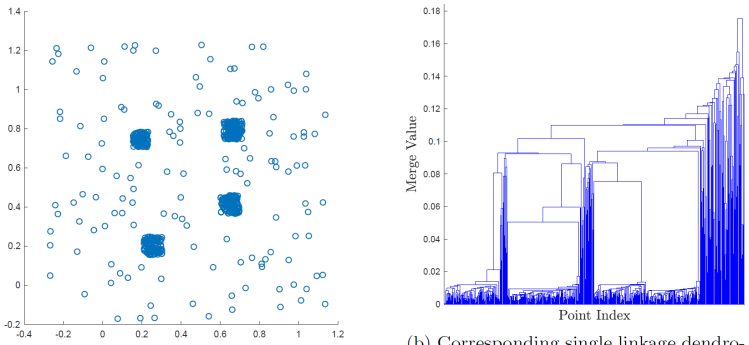
II.2 Hierarchical clustering

Hierarchical clustering methods form a set of n nested clusterings of a data set X, each containing a different number of clusters k ranging from 1 to n. These techniques can be categorized as either agglomerative or divisive. An agglormerative method initially considers each data point as a cluster and iteratively merges the two most similar clusters, while a divisive algorithm initially considers the entire data set as a cluster and iteratively breaks in two the cluster with the highest

dissimilarity.

Deciding which regions to merge or break defines specific hierarchical clustering methods. This is done by defining a distance between pairs (potential pairs) of clusters, and choosing the clusters with the minimum (maximum) of those distances to merge (break). Most technical research has been focused on agglomerative methods, of which complete linkage, group average, and single linkage are some of the most well known. Complete linkage clustering defines the distance between two clusters as the maximum distance of the distances between two points drawn from separate clusters. Group average clustering takes the average distance of all inter-cluster data pairs as the cluster distance. Single linkage clustering defines cluster distance as the minimum distance of all inter-cluster data pairs. Single linkage clustering tends to produce a "chaining" effect, causing unbalanced clusters by retaining single outlying data points as clusters until the final iterations [6].

All these cluster distances obey the utlrametric property, which guarantees that once two clusters are merged or a cluster is broken, they cannot be separated or merged again in future iterations. The ultrametric property will be discussed in the next section in detail. This allows for the creation of a hierarchical structure that can be represented by a dendrogram, of which the height represents the value at which each cluster merge or break is made. An example of a data set and its corresponding single linkage dendrogram is shown in Figure 2. It is then up to the user to decide at which height to retrieve a clustering, defining how many clusters are appropriate. This is not always obvious however, especially with noisy data. For example, it is not readily seen from the dendrogram of Figure 2 that the data set is composed of four clusters.



(a) Data to cluster

(b) Corresponding single linkage dendrogram

Figure 2: (a) Two-dimensional data set with four high density clusters surrounded by low density noise, (b) Corresponding single linkage dendrogram for data set of (a). Figure from [1].

II.3 Graph-based spectral clustering

Graph-based clustering relies on representing the data set X as a weighted undirected graph with vertices representing the data points and weighted edges the similarity between the data points. Edges with high weight signify that the two connected data points are very similar and should likely be placed in the same group. Dissimilar data points are connected by edges with low or zero weight, and likely should be placed in different groups.

Formally, let $\mathcal{G} = (V, E)$ be an undirected graph consisting of the set of vertices $V = X$ and edges E. To specify the weight of each edge, we define an $n \times n$ weighted adjacency matrix W of $\mathcal{G}$, where W_{ij} holds the weight of the edge between vertex i and j. W is symmetric since $\mathcal{G}$ is undirected, and it contains zeros along the diagonal since self-edges are not considered. A common technique to define the adjacency matrix is using the Gaussian or heat kernel, defined as

$$W_{ij} = \begin{cases} e^{-\frac{\|x_i - x_j\|^2}{\sigma^2}}, & i \neq j \\ 0, & i = j \end{cases} \tag{II.3}$$

where $\sigma > 0$ is a user-specified a bandwidth parameter. Note that any type of norm may be used in the exponential, and that $W_{ij} \in (0, 1]$. As σ is increased, all weights tend toward a value of 1, meaning points that are further from each other will be considered more similar. Likewise, as σ is decreased, all weights tend toward 0, meaning only points very close together are considered related. Therefore, varying σ controls the scale at which points are considered similar. The user may choose to compute the weight of edges between all data points creating a complete graph, threshold the weights at a small value by assigning all weights below the threshold to zero, or only keep each point's K-nearest neighbors' edge weights, assigning the rest weight zero.

Next, define the diagonal *degree* matrix D, whose entries are the degrees of each vertex, given by the row sums of W, that is, $D_{ii} = \sum_{j=1}^{n} W_{ij}$. The graph Laplacian matrix is then given by $L = D - W$, but often is normalized to obtain the symmetric graph Laplacian $L_S = I - D^{-\frac{1}{2}} W D^{-\frac{1}{2}}$.

A clustering can now be performed by "cutting" or removing all edges of the graph in an optimal way, yielding a set of connected subgraphs, with each connected subgraph defining one of the clusters. Let $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_k$ be the k smallest eigenvalues of L_S, $v_1, v_2, \ldots, v_k$ be the corresponding k eigenvectors, and $V = [v_1, v_2, \ldots, v_k]^T \in \mathbb{R}^{k \times n}$ be the matrix containing those

eigenvectors as rows. Next a matrix Y is formed from V by normalizing the columns of V as $Y_{ij} = V_{ij}/\sqrt{\sum_i V_{ij}^2}$. Each column in Y, which we will denote by y_i, is now treated as a point in $\mathbb{R}^k$, and k-means is applied to cluster these points into k clusters. It is assumed that y_i represents the i^{th} data point x_i of X, and the label assigned to y_i is then assigned to x_i giving the final cluster label assignments [2].

Although k-means clustering is applied to find the final cluster labels, the power in this method comes from the new representation y_i of each point x_i. In this lower dimensional space, the new data points are tightly grouped in a Euclidean sense and clusters can be easily identified. This method therefore keeps the relevant information about the high dimensional data set X when representing the data as an embedding Y in a lower dimensional space where the clustering is calculated. To gain perspective on why this is true, this embedding can be related to a well known field in machine learning called dimensionality reduction.

The Laplacian Eigenmaps data reduction technique [11], which assumes that the data lies on a low-dimensional manifold embedded in a high-dimensional space, has a related solution. This technique then attempts to recover the low-dimensional coordinates that still capture the relevant information of the high-dimensional input data. The low-dimensional Laplacian Eigenmaps embedding is given by the $k \times n$ matrix $Y = [y_1, \ldots, y_n]$ solving the constrained minimization problem

$$\begin{aligned} \min_{Y} \quad & \sum_{i,j} W_{ij} \|y_i - y_j\|_2^2 \\ \text{subject to} \quad & YDY^T = I, \\ & YD\mathbb{1} = 0 \end{aligned} \tag{II.4}$$

where y_i is the k-dimensional representation of the i^{th} vertex of X and the i^{th} column of Y, and $\mathbb{1}$ is the $n \times 1$ vector of ones. The first constraint enforces orthogonality to ensure the embedding Y is nontrivial, while the second constraint avoids the trivial eigenvalue. Although spectral clustering does not avoid the trivial eigenvalue, it always corresponds to the eigenvector $\sqrt{D}1$ if G is connected. Therefore LE ignores this constant eigenvector. The solution Y of Equation II.4 is given by the k eigenvectors corresponding to the k smallest nontrivial eigenvalues of the generalized eigenvector problem

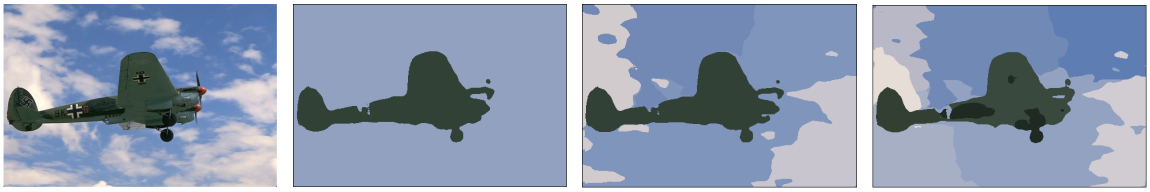
$$(D - W)y = \lambda Dy, \tag{II.5}$$

which will gives nearly the same coordinates as the spectral clustering algorithm if using L_S,

which are the k eigenvectors corresponding to the strictly k smallest eigenvalues of Equation II.12. Thus, in performing spectral clustering, we are reducing the dimension of data set in a way that preserves the structure and better separates distinct groups of data points where clustering can be performed easier.

II.4 Image segmentation

The clustering problem applied to an image is known as image segmentation and involves dividing an image into k regions of similar characteristics. Segmenting an image into two regions typically identifies the foreground and background of the image, while more regions identifies distinct image regions. Often these regions represent physical objects and can therefore be used as a preprocessing step in many computer vision tasks like tracking, detection, and recognition. Note though that a region does not have to be spatially contiguous, and so may be composed of non-adjacent pixels. For example, in an image of a person, pixels displaying the person's arms and legs may be one region even though they are not spatially contiguous. An example of an image and its segmentation into $k = 2, 5, 10$ regions is shown in Figure 3. In this figure, for each k, the segmentation is represented as a color image with each pixel of a group labeled with the mean color of its assigned region. This is a standard technique for visual and qualitative representation of a segmentation.



(a) **(b)** **(c)** **(d)**

Figure 3: Example of segmented image. (a) Original image, (b) $k = 2$, (c) $k = 5$, (d) $k = 10$.

II.4.1 Defining the Weight Matrix for Graph-based Image Segmentation

Image segmentation involves grouping pixels based on their color or feature characteristics, but it also has the added complexity of being dependent on spatial location. Spatially close pixels are more likely to come from the same physical object and should be more likely to be grouped together. A naive way to incorporate both spatial and feature data is to concatenate the normalized

spatial and feature data vectors; however, this does not allow control over how much each type of data influences the distance between pixels. It is also dependent on the ratio of the spatial to feature dimension, which is dependent on the type of image. Instead, it is suggested in [10] for graph-based image segmentation to compute the Gaussian weight matrix with the feature coordinates multiplied by a Gaussian spatial window,

$$W_{ij} = e^{-\frac{\|f_i - f_j\|_2^2}{\sigma_f^2}} \cdot \begin{cases} e^{-\frac{\|s_i - s_j\|_2^2}{\sigma_s^2}}, & \|s_i - s_j\|_2 < r \\ 0, & \text{otherwise} \end{cases} \tag{II.6}$$

where $\{f_1, f_2, \ldots, f_n\}$ represents the normalized feature data of each pixel with $f_i \in \mathbb{R}^w$ and $\{s_1, s_2, \ldots, s_n\}$ represents the normalized spatial data of each pixel with $s_i \in \mathbb{R}^2$, and r is a user inputted distance. The feature data are normalized by the largest feature value, and the pixel coordinates are normalized by the largest image dimension, ensuring that elements of f_i and s_i are between 0 and 1. Note that the information included in the feature vector f_i need not be restricted to color/intensity values, but may include information like texture signatures, gradients, and results of filtering techniques. The method of Equation II.6 captures the local relationships between pixels by only considering points within an r radius of each pixel as related. A more global approach is to define a single distance between pixels p_i and p_j to then use in the Gaussian kernel of Equation II.3; i.e.,

$$d(p_i, p_j) = \sqrt{\alpha \|s_i - s_j\|_2^2 + (1 - \alpha)\|f_i - f_j\|_2^2}, \tag{II.7}$$

for $\alpha \in [0, 1]$. This technique allows for control over the amount of spatial versus feature information desired with the adjustment of α while not restricting distances to a spatial domain. Defining a data vector $x_i = [\sqrt{\alpha}\, s_i^T, \sqrt{1-\alpha}\, f_i^T]^T$ for each pixel p_i allows the data set $X = \{x_i\}_{i=1}^n$, $x_i \in \mathbb{R}^{w+2}$, to represent our image, with distances between points in X equivalent to Equation II.7. This also gives the user the option of computing a full or k-nearest neighbors graph where only the distances to k closest neighbors of each point are kept in the weight matrix. We will represent the image data as the above described data set X when computing Gaussian affinity matrices in future sections.

II.4.2 Image Segmentation Algorithms

Although image segmentation can be treated as a clustering problem, many algorithms have been proposed and applied specifically to the image segmentation problem. Many of the fundamental image segmentation algorithms are graph-based, and although they are very similar to the spectral clustering methods of Section II.3, they are derived from a different perspective. Perhaps the most well known is the Normalized Cut algorithm.

Let the image be represented as a undirected weighted graph $\mathcal{G} = (V, E)$ where the vertices represent image pixels and the edges the similarity between the pixels, whose weights contained in the affinity matrix W. To build the intuition behind the Normalized Cut algorithm, first consider the case where $k = 2$; that is, we wish to partition the graph into two groups of vertices C_1 and C_2 by cutting edges between these groups. The *cut cost*, or cost for creating this partitioning is defined as the sum of edge weights W_{ij} of edges between the two groups:

$$\text{cut}(C_1, C_2) = \sum_{x_i \in C_1, x_j \in C_2} W_{ij}. \tag{II.8}$$

The minimum cut is the partitioning of $\mathcal{G}$ that minimizes Equation II.8; however, this often produces undesirable results by cutting a singleton point away from $\mathcal{G}$ [31]. To overcome this problem, the Normalized Cut cost

$$\text{NCut}(C_1, C_2) = \frac{\text{Cut}(C_1, C_2)}{\text{Assoc}(C_1, V)} + \frac{\text{Cut}(C_1, C_2)}{\text{Assoc}(C_2, V)} \tag{II.9}$$

can be minimized instead, where

$$\text{Assoc}(C, V) = \sum_{x_i \in C, x_j \in V} W_{ij} \tag{II.10}$$

is the *association cost*, or total degree of C. Normalizing by the total degree of each cluster makes singleton partitions no longer optimal and leads to more balanced cluster sizes. The solution to

the minimum of the above NCut cost has been shown to be equivalent to

$$\begin{aligned} \min_{y} \quad & \frac{y^T(D-W)y}{y^TDy} \\ \text{subject to} \quad & y_i \in \{1, -\beta\},\ i = 1, 2, \ldots, n, \\ & y^TD1 = 0, \end{aligned} \tag{II.11}$$

where D is the diagonal degree matrix containing the row sums of W, $d_i = D_{ii} = \sum_j W_{ij}$, $\beta = (\sum_{z_i>0} d_i)/(\sum_{z_i<0} d_i)$, $y = (1+z)/2 - \beta(1-z)/2$, and z is an n-dimensional indicator vector where $z_i = 1$ if vertex x_i is in C_1 and $x_i = -1$ if x_i is in C_2. Finding the solution to Equation II.11 is NP-hard, but by relaxing the binary constraint on y by allowing $y \in \mathbb{R}^n$, the solution of this relaxed problem becomes equivalent to solving the generalized eigenvector problem

$$(D-W)y = \lambda Dy \tag{II.12}$$

for the generalized eigenvector corresponding to the smallest non-trivial eigenvalue. Since we allow components of y to take on continuous values, a clustering algorithm such as $k-$means must be applied to the eigenvector to assign a discrete labeling, however y is close enough to the solution of Equation II.11 that this can now be done easily.

Remembering back to Section II.3, the solution Y of the Laplacian Eigenmaps objective function of Equation II.4 is given by the k eigenvectors corresponding to the k smallest nontrivial eigenvalues of Equation II.12 as well. This implies that for $k = 2$, the solution to the Laplacian Eigenmaps embedding problem is identical to that of the relaxed version of the NCut problem to optimally cut a data set in two. This provides a natural extension for partitions into k clusters $\mathbb{C}$, where we can define the multiway cut and normalized cut [24] respectively as

$$\text{Cut}(\mathbb{C}) = \frac{1}{2}\sum_{l=1}^{k} \text{cut}(C_l, V \backslash C_l), \tag{II.13}$$

$$\text{NCut}(\mathbb{C}) = \frac{1}{2}\sum_{l=1}^{k} \frac{\text{cut}(C_l, V \backslash C_l)}{\text{Assoc}(C_l, V)}. \tag{II.14}$$

The minimization of Equation II.14 can be relaxed into a form that is equivalent to the k-dimensional Laplacian Eigenmaps problem, which has a solution that can be found from the k eigenvectors of Equation II.12 corresponding to the k smallest nontrivial eigenvalues. The

k-clustering is then found by applying $k-$means clustering to the k-dimensional embedding Y, just as in the spectral clustering algorithm of Section II.3.

II.4.3 Superpixel Representation

Image segmentation is also complicated by the vast number of pixels in most images. Since each image pixel is considered a vertex in the graph representation, if n is the number of image pixels then spectral clustering requires computing an $n \times n$ graph Laplacian matrix with potentially n^2 nonzero elements and then performing an eigendecomposition. This becomes computationally prohibitive with medium to large images for most computers. Even if a K-nearest neighbors graph is used, guaranteeing a Laplacian matrix having $\mathcal{O}(kn)$ nonzero entries, sparse solvers may still have problems for large images with hundreds of thousands or millions of pixels. To simplify the complexity of the problem, small spatial regions of similar featured pixels called *superpixels* can be identified and used to form vertices of a smaller graph that can be used for clustering. Letting $\mathcal{S} = \{S_1, S_2, \ldots, S_b\}$ be the partitioning of X into superpixels, the reduced superpixel feature data set becomes $F_S = \{f_{S_i}\}$ and spatial data set $S_S = \{s_{S_i}\}$, where

$$f_{S_i} = \frac{1}{|S_i|} \sum_{x_j \in S_i} f_j, \tag{II.15}$$

$$s_{S_i} = \frac{1}{|S_i|} \sum_{x_j \in S_i} s_j. \tag{II.16}$$

Since these regions contain similar pixels, these sets of mean feature and spatial vectors of each superpixel are still a good representation of the original data set. Similarly to Section II.4.1, we can represent this as a reduced data set $X_r = \{x_{r_i}\}_{i=1}^{b}$ with $x_{r_i} = [\sqrt{\alpha}\, s_{S_i}^T, \sqrt{1-\alpha}\, f_{S_i}^T]^T$ so that Euclidean distances between points in X_r are equivalent to the desired feature-spatial distance of Equation II.7. In the context of images, X_r will represent the superpixel data set and in later sections will represent the reduced data set for any type of data set.

Superpixels have been applied successfully to target detection [25], anomaly detection [20], and image segmentation [17], as well as many other problems. The power of superpixels is the ability to represent an image by a few thousand superpixels instead of hundreds of thousands of pixels which allows for reasonable computation of these complex computer vision tasks. Superpixels are also intimately related to image segmentation as the approximately equal sized superpixels can be thought of as an "oversegmentation" of the image.

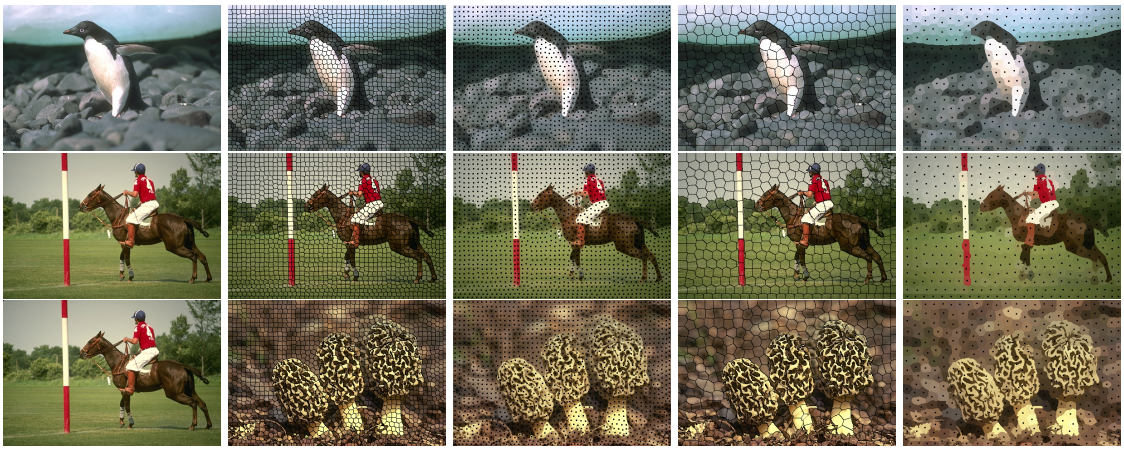


Figure 4: Examples of superpixel calculation using SLIC algorithm. By columns, Left: Original image, Middle left: Outlines of region size 10 superpixels, Middle: Region Size 10 superpixels displayed with average color and mean position as black dot, Middle right: Outlines of region size 20 superpixels, Right: Region Size 20 superpixels displayed with average color and mean position as black dot.

A fast and robust algorithm for computing superpixels is the Simple Linear Iterative Clustering (SLIC) [16] method, which creates image superpixels based on user-provided size and shape parameters. The region size is specified as the approximate side length of the desired superpixels, so for example a region size of 10 gives superpixels that contain around 100 pixels each. An implementation of this method for use in MATLAB is publicly available in the VLfeat Toolbox [23]. Examples of images, their SLIC superpixel regions, and the mean representation for different region sizes are shown in Figure 4. The first column shows the original RGB image, the second and fourth columns shows the outlines of each superpixel for a region size of 10 and 20 respectively, and the third and fifth column displays the image with each superpixel assigned its mean color, with a black dot representing the mean position of that superpixel for region size 10 and 20 respectively.

When computing a segmentation using superpixels, the data embedding is created for the reduced superpixel data set instead of the full image data set. Since the desired output is a segmentation of the original image, labels must be assigned to all pixels using only the knowledge of the reduced data embedding. The simplest method of doing this would be to perform k-means clustering on the superpixel data embedding, and assign each pixel the label of its containing superpixel. This however gives very chunky regions since superpixels do not perfectly conform to object boundaries, as shown in Figure 5 for the images of Figure 4 with a superpixel Region Size of

10.

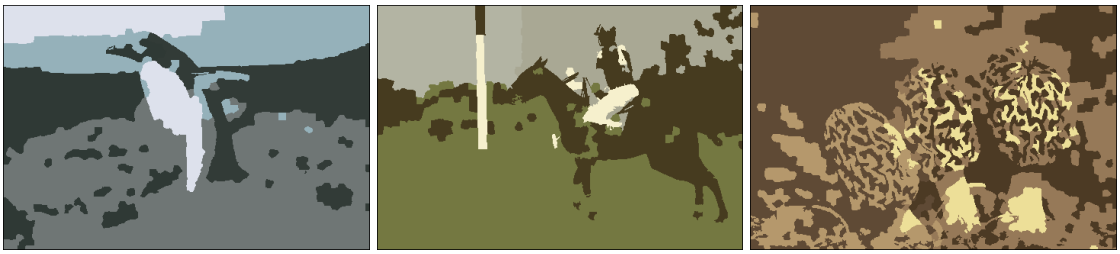


Figure 5: Normalized cut image segmentations using superpixels for $k = 5$, with each pixel being assigned the label of its superpixel.

III. Longest-Leg Path Distance Ultrametric

III.1 Definition

Let $\mathcal{G} = (V, E)$ be an undirected graph consisting of the set of vertices V and edges E, and let $u, v \in V$. Let $\mathcal{P}_{u,v}$ be the set consisting of all paths p_i connecting u and v in $\mathcal{G}$, where $i \in \{1, \cdots, L\}$ with L representing the total number of unique paths. Note that since $\mathcal{G}$ is undirected, $\mathcal{P}_{u,v} = \mathcal{P}_{v,u}$. Each path p_i consists of a series edges $p_i = \{e_{i,1}, e_{i,2}, ...e_{i,K_i}\}$ representing the edges traversed by that path, with $e_{i,j} \in E$. The longest-leg path distance (LLP-distance or LLPD) between vertices u and v is defined as the minimum of the maximum weight edges of all paths connecting u and v,

$$d_{ll}(u, v) = \min_{p_i \in \mathcal{P}_{u,v}} \max_{e_j \in p_i} w(e_j) \tag{III.1}$$

Thus, no matter which path is chosen between u and v, or equivalently between v and u, an edge of weight at least $d_{ll}(u, v)$ must be traversed. A similar concept in graph theory is the Bottleneck Edge Query (BEQ) problem which involves finding the maximum of the minimum edge weights of paths of between two vertices. Since the graph adjacency matrix is computed from a Gaussian Kernel, finding the LLP-distances on a graph $G' = (V, E')$ with the same vertices as G and edges weighted based on Euclidean distance is equivalent to solving the Bottleneck Edge Query (BEQ) problem on the graph adjacency matrix followed by a conversion via a logarithm.

An important quality of the LLP-distance is that it is an ultrametric, meaning it satisfies a stronger version of the triangle inequality given by

$$d_{ll}(u, v) \leq \max\{d_{ll}(u, w), d_{ll}(w, v)\}, \tag{III.2}$$

with $u, v, w \in V$. This provides the foundation for many theoretical guarantees for the outcomes of unsupervised graph clustering using the LLP-distance as the metric in the exponent of the Gaussian Kernel of Equation II.3 [1].

Ultrametrics naturally induce hierarchies because of this stronger version of the triangle inequality, and hierarchies naturally induce an ultrametric [13]. The distances discussed in Section II.2 on hierarchical clustering all obey this property, and actually the LLP-distance is closely related to single linkage clustering. The LLP-distance between two points can be though of as the single-linkage distance at a level of the hierarchy where those points are in separate clusters.

III.2 Prior research

The LLP-distance was first introduced for use in clustering problems by Fischer et. al. [4], creating the field of path-based clustering. The LLP-distance is used to define a clustering cost function which is then minimized by an Iterated Conditional Mode algorithm using multi-scale techniques to improve computational performance. This algorithm was shown to outperform other concurrent top agglomeration methods on artificial data sets as well as segmentation of textured images. This algorithmic framework was later modified to include automatic outlier detection by inclusion of an outlier class, which was shown to perform well for edgel grouping and textured data [3].

In a later publication, Fischer et. al. proved that the LLP-distance is an ultrametric and thus the matrix of LLP-distances was shown to induce a Mercer's Kernel [5]. This allows for an approximation of the path-based clustering result by applying k-means clustering to a lower dimensional embedding of path-based distances using Kernel Principle Components Analysis. This technique has the advantage of de-noising the hierarchy created by their previous agglomerative method, leading to a more robust method. This also lead the way for path-based spectral clustering.

Since path-based distances have the disadvantage of being heavily affected by noisy data points, a robust path-based similarity measure was introduced in [8] based on M-estimation which lessens the effect of outliers. This similarity measure was then used to define the affinity matrix for both supervised and unsupervised spectral clustering. This method was tested on difficult artificial data sets, image grouping of hand written digits and faces, and image segmentation of color images, with promising results reported.

Theoretical guarantees of LLP-distance based spectral clustering have been proven under the low-dimensional large noise (LDLN) data model, which assumes clusters are high density sets separated by lower density regions of noise or outliers [1]. In this model, points with large LLP-distance to their K^{th} nearest LLPD nearest neighbor are considered noise and are removed. LLP-distances are then recomputed on the denoised data set before spectral clustering. It was shown that given this data model, the largest eigengap of the symmetric LLPD Laplacian correctly estimates the number of appropriate clusters as the number of eigenvalues before this gap. It was also proved that the embedding of the data according to the symmetric LLPD Laplacian followed by k-means clustering correctly labels most data points [1]. This brings improvement over using the standard Euclidean distance between data points to create the affinity matrix which does not usually estimate the correct number of clusters with the eigengap.

III.3 Computation

In order to compute a spectral clustering, an $n \times n$ matrix of distances is required, where n is the total number of data points. When employing the LLP-distance, this is referred to as the All Points Path Distance (AAPD) problem. The AAPD problem has been solved previously with the algorithm of Floyd with $\mathcal{O}(n^3)$ complexity [4], and can be solved using bottleneck spanning trees using a modified SLINK algorithm with $\mathcal{O}(n^2)$ complexity [1]. There exist theoretical methods using bottleneck spanning trees with complexity $\mathcal{O}(n \log n)$, but numerical implementations of these methods are currently not publicly available. There is however an easily implemented algorithm to approximate the LLP-distance for a set of high dimensional data introduced in [1] with $\mathcal{O}(n \log n)$ complexity. Due to the flexibility and efficiency of this approximate method, we employ a modified version of this algorithm to calculate LLP-distance matrices in the experiments of this thesis.

This fast approximate method represents the data points theoretically as vertices on a complete graph $\mathcal{G}$ with edge weights given by the Euclidean (L2) distances between vertices. Since computing the complete graph would be computationally expensive for most data sets, a spanning tree $\tilde{\mathcal{G}}$ of the complete graph is created instead by computing the edges corresponding to the K_1-nearest L2 neighbors of each vertex. It is assumed that K_1 is chosen large enough to induce a spanning tree of $\mathcal{G}$, with it being sufficient for this algorithm to choose K_1 only large enough to induce a minimum spanning tree.

Next, a set of m thresholds $t_1 < t_2 < \cdots < t_m$ are chosen between the maximum and minimum edge weights of $\tilde{\mathcal{G}}$, which are used to create a series of subgraphs of $\tilde{\mathcal{G}}$, denoted $\tilde{\mathcal{G}}_{t_s}$, containing only edges of weight less than t_s. The LLP-distance between two vertices $u, v \in V$ can then be approximated by finding the threshold t_s at which two separate path connected components C_1 and C_2, with $u \in C_1$ and $v \in C_2$, merge. That is, we assign $d_{ll}(u, v) = t_s$, which is approximating the minimum weight edge separating these two clusters. Since all intra-cluster edges at this stage have edges of weight less than or equal to t_s, the clusters are connected and any further edges joining the two clusters will have weight greater than or equal to t_s. It thus intuitively makes sense that t_s represents the minimum of the maximum edges separating any $u \in C_1$ and $v \in C_2$. The algorithm is set up to find a vertex's K-LLPD nearest neighbors by computing and sorting a representation of each thresholded graph's connected components.

We use a small generalization of this algorithm by approximating the LLP-distances using an arbitrary L2 distance matrix D^{L2} instead of the K_1-nearest neighbors graph. This allows the full

L2-distance matrix to be inputted if the size of the data set allows, and a knn-graph if the data set is too large. Pseudo-code for our modified fast approximate LLPD algorithm is shown in Algorithm 1. This algorithm works by computing an $n \times 1$ list of connected components for each of subgraph described above and placing in the columns of an $n \times m$ matrix CC. The rows of CC are then sorted by column from left to right, which simulates a hierarchical clustering by organizing all rows in the same connected component together at each stage. Therefore starting at some row and traversing the first column of CC_{sorted} up or down will show if those points are in the same connected component. If they are, this t_s is the approximate minimum weight separating those two data points. If they are not, then these points are still not in the same cluster, so we look at the next row representing a higher threshold t_{s+1} and check if this point is in the same cluster. This process is continued until K nearest neighbors are found. Note that if $K = n$ a full matrix of approximate LLP-distances will be returned.

Algorithm 1 Modified Fast Computation of Approximate LLPD

Input: $X, D^{L2}, \{t_s\}_{s=1}^{m}, K$
Output: $n \times n$ K approximate LLPD-nearest neighbors matrix $\hat{D}^{ll}$

Allocate $n \times m$ matrix CC.
for $s = 1 : m$
  Form matrix D^{t_s} containing elements of D^{L2} less than t_s, zeros elsewhere.
  Compute connected components of D^{t_s} storing in s^{th} column of CC.
end
Sort the rows of CC by column from right to left to create CC_{sorted} and let $\pi(i)$ denote the corresponding point order.
for $i = 1 : n$
  NN = 1 (number of nearest neighbors found)
  $i_{\text{up}} = 1$, $i_{\text{down}} = 1$
  for $s = 1 : m$
    while $CC_{\text{sorted}}(i_{\text{up}}, s) = CC_{\text{sorted}}(i_{\text{up}} - 1, s)$ and $NN < K$ and $i_{\text{up}} > 1$
      $i_{\text{up}} = i_{\text{up}} - 1$
      $\hat{D}^{ll}_{\pi(i),\pi(i_{\text{up}})} = t_s$
      $NN = NN + 1$
    end
    while $CC_{\text{sorted}}(i_{\text{down}}, s) = CC_{\text{sorted}}(i_{\text{down}} - 1, s)$ and $NN < K$ and $i_{\text{down}} > n$
      $i_{\text{down}} = i_{\text{down}} + 1$
      $\hat{D}^{ll}_{\pi(i),\pi(i_{\text{down}})} = t_s$
      $NN = NN + 1$
    end
  end
end

III.4 Advantages and disadvantages for use in spectral clustering

Many clustering algorithms, including spectral clustering, have the disadvantage of poor performance on non-convex and highly elongated clusters. This is due to points in the same cluster being considered far apart in the Euclidean sense even if there is a high density of points between signifying they belong to the same cluster. The LLPD is able to overcome this problem, since points in high density clusters have small LLP-distances no matter the shape of the cluster.

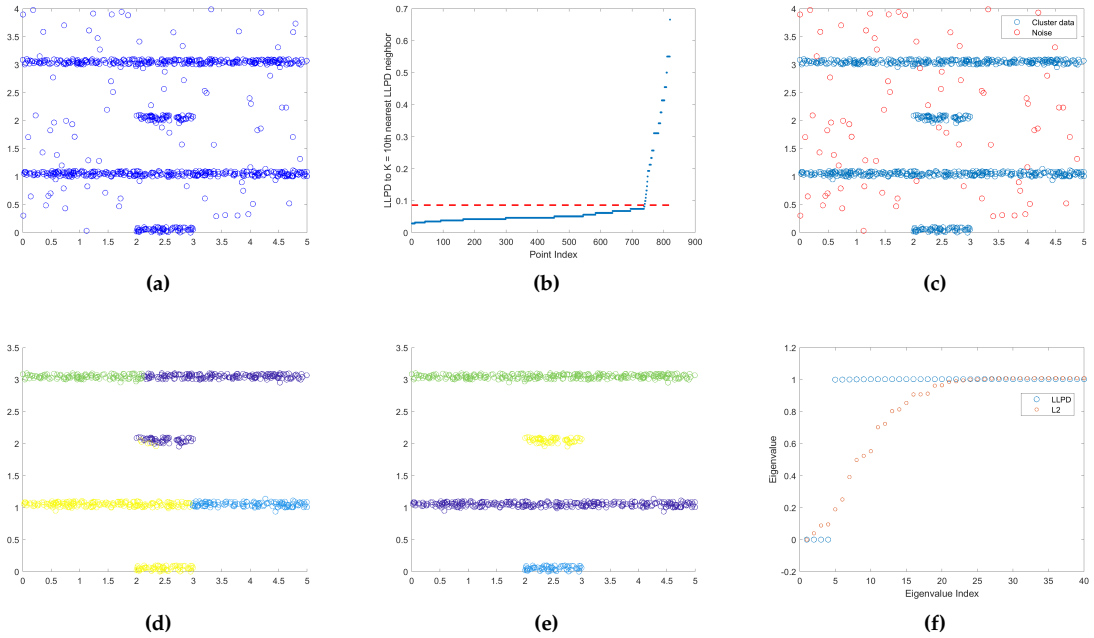
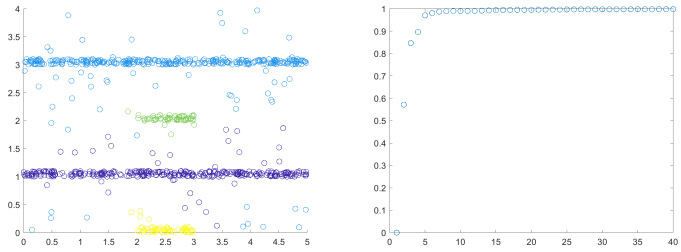


Figure 6: (a) Four Lines Data Set, (b) Plot of each point's $K = 10^{\text{th}}$ LLPD nearest neighbor in ascending order, points with K^{th} LLPD nearest neighbor distance above the red line are considered noise and removed, (c) Plot of data with noise as red circles and cluster data as blue circles, (d) L2 clustering results, (e) LLPD clustering results, (f) 40 smallest eigenvalues for L2 and LLPD embedding.

For example, consider the Four Lines data set from [1] in Figure 6, where points in two-dimensional space are positioned in four elongated lines with low density noise between. Computing each point's $K = 10^{\text{th}}$ nearest LLPD-neighbor and finding the elbow point when plotted in ascending order gives an estimate of which points are outliers. These points are removed from the data set, then the clustering is done on the remaining data. Using L2-distances to create the Gaussian weight matrix, the resulting spectral clustering divides the long clusters into multiple pieces

instead of keeping the high density clusters together. Spectral clustering using the LLPD Gaussian weight matrix however yields a more accurate clustering. The LLPD clustering also has the advantage that the correct number of clusters is clearly shown by the number of eigenvalues before largest eigengap, whereas for the L2 clustering there is no large eigengap. This process is outlined in Figure 6.

If instead clustering is performed on the Four Lines data set without denoising, the four lines are still pulled out as separate clusters with noise getting assigned to one of the four clusters, shown in Figure 7a. Although the correct clusters are identified, the largest eigengap no longer suggests the correct number of clusters, shown in Figure 7b. Therefore the denoising process is still useful for this data set in order to guarantee the eigengap property, and would be more necessary in data sets with distinct outliers versus low density noise as with this data set.

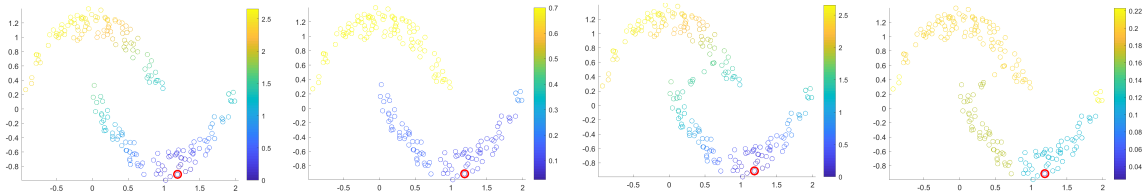


(a) **(b)**

Figure 7: (a) LLPD clustering results without denoising, (b) 40 smallest eigenvalues for LLPD embedding without denoising.

The LLP-distance has the disadvantage of sensitivity to noise and outliers [29], especially structured noise and clusters that are much denser than others [1]. For example, consider the data set in Figure 8, composed of two boomerang shaped clusters. Choosing one data point, the LLPD and L2 distances from this point to all other points are calculated and plotted by color in Figures 8a and 8b respectively. The LLP-distances alone suggest two clusters, while the L2 distances do not provide such a crisp boundary. Adding ten points along the line $y = x + \epsilon$ where $x \in [.1, .7]$ and $\epsilon \in [-.02, .02]$ are randomly chosen, the same two plots are created in Figures 8c and 8d. With these few added points of structured noise, the scale of the LLPD changes dramatically, and causes the two clusters to no longer be distinctly separable visually, showing that the addition of just a few well placed data points can affect the LLPDs of the entire data set. Although all sets of random noise chosen along this line may not give such drastic changes in the LLP-distances of the data set, change in the LLP-distances is characteristic of adding structured noise to a data set.

An advantage of the L2 distance is that the scale of distances and distances to non-noise points remains the same with the addition of the noise, so only 10 new distances need to be calculated, whereas all LLPDs must be recalculated.



(a) (b) (c) (d)

Figure 8: Comparison of L2 to LLPD from the point $p = (1.1910, 0.2114)$ in an artifical data set with and without noise. (a) L2 distances from p without noise, (b) LLPD from p without noise, (c) L2 distances from p with 10 added data points of noise, (d) LLPD distances from p with 10 added data points of noise.

IV. Path-based Clustering Denoising by Pre-cluster Averaging

We propose a novel LLPD-based clustering scheme that reduces the influence of noise on the spectral data embedding as well as the size of the eigendecomposition problem while maintaining accuracy.

IV.1 Motivation

The noise removal method introduced in [1] relies on identifying and removing potential outliers before computing the spectral clustering. This method been shown to be effective on a number of artificially created data sets as well as the DrivFace data set [12]. These data sets are all composed of a set of discrete data points, however for other types of data the strict removal of data points becomes more problematic. For example, consider a data set of pixels of an RGB or hyperspectral image where both a spectral as well as a spatial component are important to segmenting relevant regions of the image. Removing data points in this instance amounts to removing pixels in the image which complicates the spatial component of similarity as well as the continuity of the final segmentation. This is also quite computationally expensive since images generally contain hundreds of thousands of pixels. We thus are looking for a way of negating the effect of noise on a path-based distance metric without removing the noise.

Assuming the data set is composed of high density clusters with relatively uniform low density noise, an initial coarse clustering $C_{init} = \{c_1, c_2, \ldots, c_b\}$ of approximately *equal-sized* partitions of the data should each contain about the same number of noise points, with $k \ll b \ll n$. These "pre-clusters" c_i will be mostly composed of non-noise data points with a few noise data points each due to the low density of noise points. The average of the data points of each initial cluster $\bar{c}_i = (1/|c_i|) \sum_{x_j \in c_i} x_j$ will thus largely reflect the data points of the desired clusters and not the noise. We then can define these averages of the initial clusters as points in a new reduced data set $X_r = \{\bar{c}_1, \bar{c}_2, \ldots, \bar{c}_b\}$ which reflect the relevant characteristics of the full data set, and calculate the embedding $Y_r \in \mathbb{R}^{b \times k}$ on this smaller data set. Using this embedding, we can then transfer this information back to compute an embedding $Y \in \mathbb{R}^{n \times k}$of the full data set.

We illustrate this with three artificial data sets in Figure 9 and show that, using this alternative denoising technique, the appropriate clusters are identified. The first data set is the Four Lines data set of Figure 6, the second is the Nine Gaussians data set introduced in [1] composed of nine sets of Gaussian distributed points, with the four corner Gaussians having larger standard

deviation, and the last is composed of three intertwined arcs of data points. All data sets have uniform low density random noise inserted. Each data set is reduced by performing k-means clustering and merging clusters until all are of approximately equal size, shown in the second column of Figure 9. An LLPD spectral clustering is then applied to each reduced data set seeking the appropriate number of clusters, that is $k = 4, 9, 3$ respectively, shown in the third column of Figure 9. We then extend this clustering to the full data set by naively assuming that all $x_j \in c_i$ should be assigned the same label as c_i, shown in the last column of Figure 9. From the results of these three toy problems we can see that this method yields the correct clusters, assigning the noise to a nearby cluster.

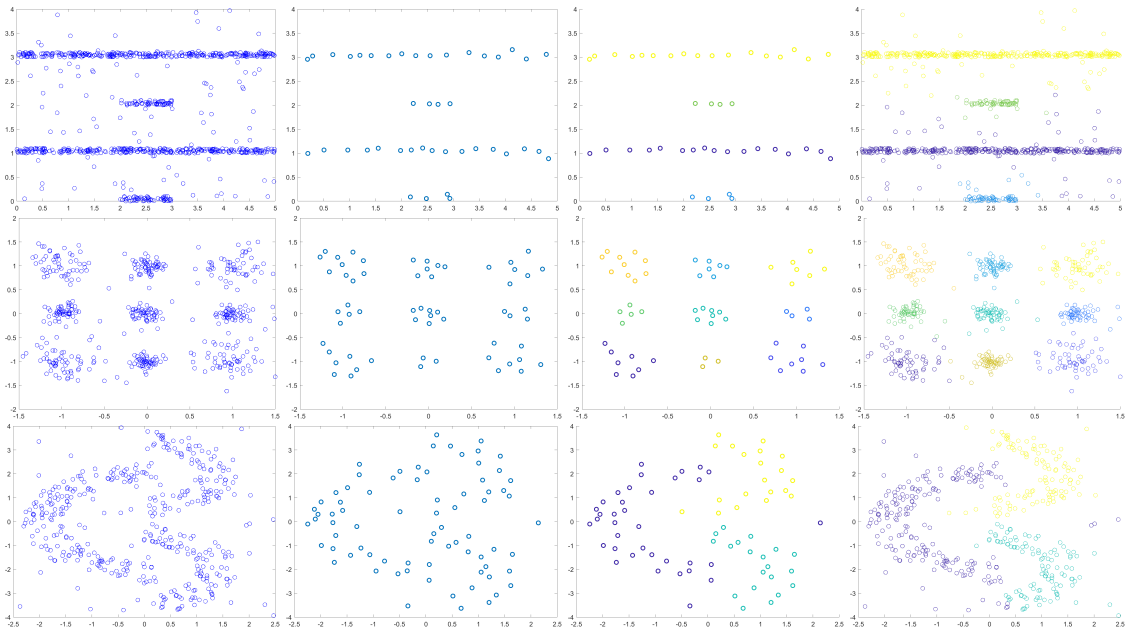
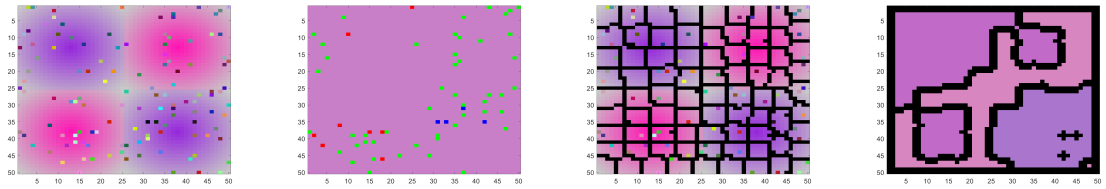


Figure 9: Clustering by preclustering on three artificial data sets: Four Lines (Row 1), Nine Gaussians (Row 2), and Three Arcs (Row 3). Columns left to right: Original data set, pre-clustering reduced data set, LLPD clustering of reduced data, corresponding LLPD clustering of original data.

For an image, we can decompose the hundreds of thousands of uniformly spaced pixels into a few hundred or thousand superpixels, represented by the mean color and spatial location of all containing pixels. The superpixel segmentation will be used as the preclustering step for our new denoising method described above. We test this on the 50×50 pixel artificially created RGB image of Figure 10a. The image is composed of four square regions of Gaussian shaped pink or purple coloring with 125 points of random RGB noise. Ideally for $k = 2$ segments, the pink and

purple squares should be identified, and with $k = 4$ segments the four separate squares should be identified. Performing LLPD spectral clustering with $k = 4$ on the full image without any noise reduction causes three of the segments to be composed of noise, represented by red, green, and blue in Figure 10b for clarity, and the fourth segment containing all other pixels, represented by their average color in Figure 10b. This is very far from the desired segmentation of the four square regions, and shows how affected by noise segmentation, and especially path-based segmentation can be. The image is then divided into approximately 6×6 superpixels, visualized in Figure 10c with black lines showing the outlines of the superpixels. A new segmentation is then calculated from the set of averaged superpixel data, and all pixels in a superpixel are given the label of that superpixel. This is displayed in Figure 10d by giving each segment its average color and outlining the segment boundaries in black. The segmentation in Figure 10d is still not ideal, but is able to capture the main structure of the image with only one class getting lost to noise.



(a) **(b)** **(c)** **(d)**

Figure 10: (a) Original Image, (b) LLPD segmentation of full image without noise reduction for $k = 4$ segments, with the background class represented by average color and three noise classes represented by red, green, and blue for clarity (c) Superpixel visualization of approximately 6×6 pixels each, (d) LLPD segmentation using superpixels of (c) for $k = 4$ segments. Using superpixels allows for the main structure of the image to be visible after segmentation whereas segmentation of the full image identifies noisy pixels as clusters.

These simple artificial examples show that pre-cluster averaging can be a viable way to compute an embedding robust to noise. In these examples we made the simple assumption that all points in a pre-cluster should be assigned the same label as the pre-cluster data point in the reduced model. For more complex and noisy data sets, this simple assumption will give coarse and unsophisticated segmentations, such as very blocky region borders in images (see Figure 5). This also does not guarantee that the embedding at each image pixel will be the same embedding coordinates as the full non-noisy image computation. Therefore we explore various methods of extending the reduced embedding that have a more rigorous mathematically backing to give the

appropriate clustering of the full data set.

IV.2 Out-of-sample extension of a Laplacian-Eigenmaps embedding

A large burden of spectral graph-based clustering methods is the necessary computation of a large square weight matrix and its eigendecomposition. For large data sets this becomes computationally prohibitive quickly as the size of the data set grows. Therefore, methods that make use of a smaller data set for the bulk of the computations, then extend this information back to the full data set are needed. This type of method is also useful for data sets that are continually growing, so that with each new data input a new model does not have to be created.

An out-of-sample extension for unsupervised graph-based spectral techniques was introduced in [26]. This method based on the Nyström extension formula [7] can be used to extend the results from Multi-Dimensional Scaling (MDS) [19], Laplacian Eigenmaps (LE) [11], Isomap [9], and Local Linear Embeddings (LLE) [18]. These methods are all based on computing a low-dimensional representation of a set of data points from the eigenvectors of a symmetric matrix, with each using a different matrix. The steps and notation for this common framework can be described as

1. Given a data set $X = \{x_1, x_2, \ldots, x_n\}$ with $x_i \in \mathbb{R}^m$, construct an $n \times n$ similarity (adjacency) matrix M, with each entry $M_{ij} = K_X(\cdot, \cdot)$ defined by a symmetric function $K_X : (\mathbb{R}^m \times \mathbb{R}^m) \to \mathbb{R}$,

2. Compute the matrix $V = [v_1, \ldots, v_k]^T$ containing the eigenvectors of M corresponding to the k largest positive eigenvalues $\lambda_1, \ldots, \lambda_k$ and define the vector $\lambda_{vec} = [\lambda_1, \ldots, \lambda_k]^T$,

3. Let y_i represent the i^{th} column of the matrix V. The embedding $e_i \in \mathbb{R}^k$ of data point $x_i \in X$ is $e_i = y_i$ for LE and LLE, and $e_i = \lambda_{vec}^{1/2} \odot y_i$ for MDS and Isomap where $\odot$ represents pointwise multiplication.

Note that in Section II.3, the LE embedding solution was given by the eigenvectors corresponding to the k smallest nontrivial eigenvalues of a generalized eigenvector problem. This is an equivalent result up to a componentwise scaling to the solution described by the above framework using the normalized adjacency matrix [28].

Now consider a point $x \in \mathbb{R}^m$, with $x \notin X$, that we would like to embed as a new point $y \in \mathbb{R}^k$. Ideally this should be done so that the relation between x and all $x_i \in X$ is captured in the relation between y and all y_i. As more and more data are added, each eigenvector of the embedding will

converge to an eigenfunction. Therefore, a linear operator K_ρ operating on functions in a Hilbert space $\mathcal{H}_\rho$ of density $\rho(x)$ can be associated with kernel K_X for $g \in \mathcal{H}_\rho$ as

$$(K_\rho g)(x) = \int K_X(x,y)g(y)\rho(y)dy. \tag{IV.1}$$

The actual density $\rho(x)$ is unknown given our limited data set, so IV.1 must be approximated using the empirical distribution $\hat{\rho}$ given by the data in X. Using these ideas, it is shown in [27] that using the empirical distribution $\hat{\rho}$, the $k \times 1$ embedding coordinate y for LE and LLE of the new point x is given componentwise by

$$e_j = y_j = \frac{1}{\lambda_j} \sum_{i=1}^{n} V_{ji} K_X(x, x_i), \tag{IV.2}$$

which gives a single real valued number, and is calculated for $j = 1$ to $j = k$ then stacked to form the embedding coordinate $y == [y_1, y_2, \ldots, y_k]^T$. For MDS and Isomaps, the embedding coordinate componentwise is $e_j = \sqrt{\lambda_j} y_j$ as before.

IV.3 Spatial-smoothing techniques for out-of-sample image segmentation

Denoising by superpixel averaging causes the superpixel embedding to be relatively unaffected by noise, so that when extended to non-noise image points via the out-of-sample extension, the "correct" embedding coordinates are calculated. When the superpixel embedding is extended to noise points however, the resulting embedding of those points are still noisy since the superpixel embedding will not represent them well. This may seem counterintuitive since we are claiming this method is robust to noise, however the distinction between this method and previous is that the *underlying embedding* to be extended is robust to noise, whereas performing the embedding with noisy points results in the entire embedding being tainted, as shown earlier in Figure 10. Our new technique therefore allows the underlying embedding of the superpixels to be true to the main features of the image.

Therefore, since noise points are not embedded well by the extension, the final segmentation will be composed of spatially incoherent segments. Desired segments however are generally composed of spatially neighboring pixels representing physical objects in the image. To overcome this, the image or its final embedding can be smoothed spatially with a Gaussian filter. This is a

two-dimensional convolution operator that is used to smooth or "blur" an image, given pixelwise by

$$\tilde{p}_i = \frac{\sum_{p \in \Omega_i} \hat{f}\, G(p, p_i)}{\sum_{p \in \Omega_i} G(p, p_i)} \tag{IV.3}$$

where $\hat{f}$ represents the vector pixel p (either the feature vector for smoothing the image or the embedding vector for smoothing the final embedding), Ω_i is a square spatial window around pixel p_i, and $G(p, p_i)$ is the two-dimensional Gaussian distribution given by

$$G(p, p_i) = \frac{1}{2\pi\sigma^2} e^{-\frac{(x-x_i)^2+(y-y_i)^2}{2\sigma^2}}. \tag{IV.4}$$

where x and y represent the spatial location of pixel p, and x_i and y_i represent the location of pixel p_i.

IV.4 General algorithm for image segmentation

In order to calculate the out-of-sample extension of Equation IV.2, the value of the LLPD Gaussian kernel must be computed between each point of X and X_r. This could be computed by first finding the LLP-distances using Algorithm 1 for the data set $\{X, X_r\}$, however this would be computationally expensive and only distances between the two groups of data points are needed. Instead, we can simply use the CC matrix computed using X_r in Algorithm 1 by adding a row for the new data point of X being considered, connect this point to each graph representation of X_r at level t_s with any new edges, and merge any distinct connected components that have been joined by new edges. The same method of finding nearest neighbors as Algorithm 1 is then employed for the newly added point only. This method is outlined in Algorithm 2 with pseudocode. Combining this with the ideas of the previous three sections, we present the framework for computing an image segmentation using the LLPD and superpixel denoising with extension method. The overall process is shown by the flow chart of Figure 11, with each step written in detail below:

1. Input $p \times q \times w$ image I and calculate pixel data set $X = \{x_i\}$ with $x_i = [\sqrt{\alpha}\, s_i^T, \sqrt{1-\alpha}\, f_i^T]^T$. Optional: Perform pre-spatial smoothing on input image I.

2. Calculate the b SLIC superpixels $\mathcal{S} = \{S_1, S_2, \ldots, S_b\}$ of I and create the reduced data set $X_r = \{x_{r_i}\}_{i=1}^{b}$ containing the mean feature and spatial information of each superpixel,

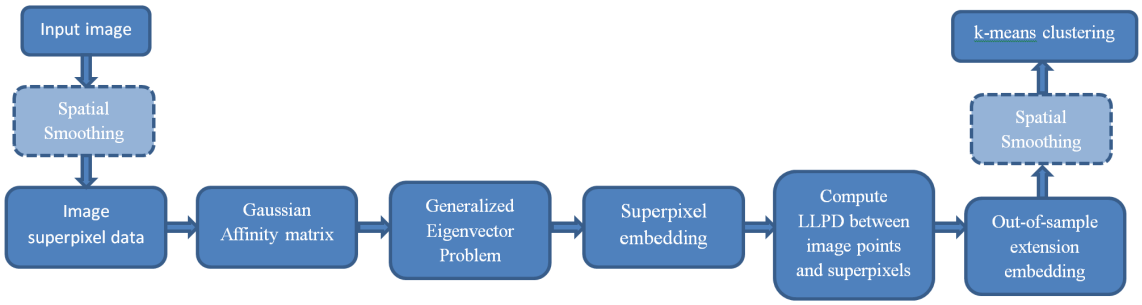


Figure 11: Flowchart of superpixel denoising with extension method.

$x_{r_i} = [\sqrt{\alpha}\, s_{S_i}^T, \sqrt{1-\alpha}\, f_{S_i}^T]^T$.

3. Create adjacency matrix W of X_r using Gaussian kernel, $W_{ij} = e^{-\frac{\|x_i - x_j\|^2}{\sigma^2}}$. Use to form diagonal degree matrix $D_{ii} = \sum_j W_{ij}$.

4. Solve the generalized eigenvector problem $(D-W)v = \lambda Dv$ for the k eigenvectors $v_1, v_2, \ldots, v_k$ corresponding to the k smallest eigenvalues $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_k$.

5. Form the spectral embedding of X_r given by the columns of $V = [v_1, v_2, \cdots, v_k]^T$.

6. Calculate the LLPD of each $x_i \in X$ to each $x_{r_j} \in X_r$ using Alg. 2. That is b distances $d_{ll}(x_i, x_{r_j})$ for each x_i. Form kernel from each distance $K_X(x_i, x_{r_j}) = \exp(-\frac{d_{ll}(x_i, x_{r_j})^2}{\sigma^2})$.

7. Use out-of-sample extension to calculate the $k \times 1$ embedding y of each $x \in X$ given componentwise by $y_i = \frac{1}{\lambda_i} \sum_{j=1}^{b} V_{ij} K(x, x_{r_j})$.

8. Compute $k-$means clustering on final image embedding y_i to compute segmentation. Optional: Perform post-spatial smoothing.

Algorithm 2 Compute LLP-distances to new point in reduced data set

Input: $CC, x, X_r, \{t_s\}_{s=1}^{m}$ **Output:** List of distances to new point x, $D^{ll,x}$

Add zero row to bottom of CC.
Calculate $b \times 1$ vector $D_{n,x}$ of L2 distances of x to all $x_i \in X_r$.
for $s = 1 : m$
  Calculate indicator vector $\tilde{D}_{n,x}$ of entries in $D_{n,x} < t_s$.
  if $\tilde{D}_{n,x}$ contains nonzero entry(s)
    Find which connected component each nonzero entry corresponds to: nbrs= $CC(s, \tilde{D}_{n,x})$.
    Merge all components of nbrs into one component
    Add merge component number to last row of column s.
  else
    $CC(s, \text{end}) = \max(CC(s, :)) + 1$.
  end
end
Calculate CC_{sorted} by sorting rows of CC by column, with $\pi(i)$ denoting the corresponding point order.
startingPoint= $\pi(n+1)$ % row of CC corresponding to the new point
Compute lines 7-21 of Alg. 1 for i=startingPoint, $K = n + 1$, storing distances in $D^{ll,x}$.
end

IV.5 Linear interpolation of superpixel embedding

A less sophisticated method of extending a superpixel embedding to the full pixel grid is to perform a linear interpolation of the superpixel embedding over the spatial coordinates of all pixels. That is, let S_i be the i^{th} superpixel and N_i be the set of superpixels that are adjacent spatially to superpixel S_i as well as S_i. A linear interpolation F_i is then created such that for $S \in N$,

$$F_i(s_S) = y_S \tag{IV.5}$$

where s_S is the average spatial location of superpixel S and y_S represents the embedding of superpixel S. Now that this function has been created, for every pixel $p_i \in S$, the embedding of that pixel is given by $F_i(s_{p_i})$ where s_{p_i} is the spatial location of p_i. A different interpolation function F_i is then created for each superpixel S_i, giving an embedding coordinate for every pixel in the image.

This method allows us to assign each pixel an embedding coordinate that transitions smoothly between superpixels. Now k-means clustering can be performed on the entire image embedding,

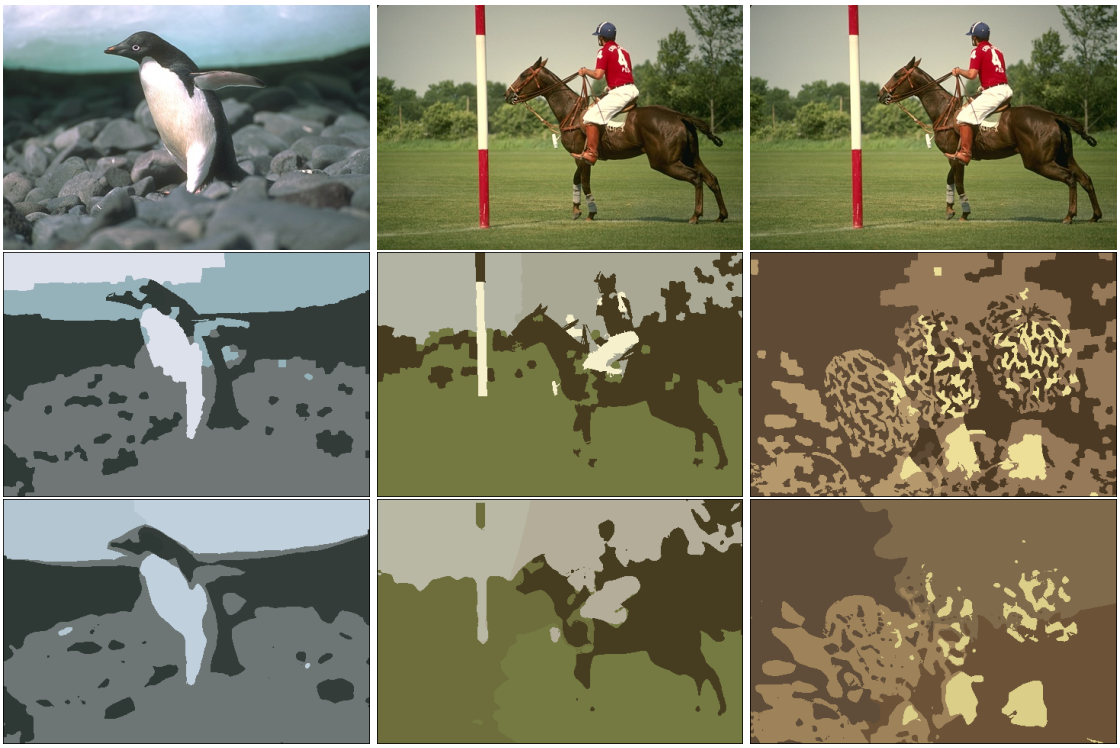


Figure 12: Normalized cut image segmentations using superpixels for $k = 5$. First row: original image, Second row: each pixel is assigned the label of its superpixel, Third Row: Interpolating of the superpixel embedding over the spatial coordinates of the image.

yielding segmentations with smoother boundaries compared to assigning each pixel the label of its superpixel, as seen in the third column of Figure 12. Even using this interpolation method, object boundaries are not captured with high accuracy since the interpolated embedding is not equal to the true embedding of the image.

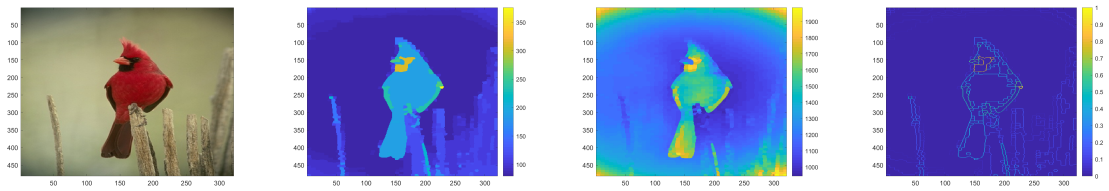
Unlike the out-of-sample extension method described in the previous sections, the interpolation method does not guarantee the convergence of the embedding coordinates as the number of samples grow. This method does however still benefit from the denoising by superpixel averaging outlined in Section IV.1, and is much less computationally expensive. Therefore we provide it as a viable alternative and will compare the more mathematically founded out-of-sample extension method with this linear interpolation method in later experiments.

V. Image Experiments and Results

V.1 LLPD in images

Image segments are separated by boundaries called contours, often representing the transition from one object, material, or region to another. Since images are discrete representations of continuous objects, these contours are not straightforward to detect automatically, and is one of the reasons image segmentation is not a trivial matter. One of the advantages of using the LLPD, as discussed earlier, is that for any two points $x \in C_1$ and $y \in C_2$ that the LLP-distance $d_{ll}(x, y)$ should be the same due to the ultrametric property. We can gain perspective of why this is true in images by thinking about contours. The "longest-leg" of any path between x and y can be thought of as a representation the strongest contour or boundary between the points x and y.

We can visualize this by plotting the degree of each pixel as a color image, which is the sum of the distances from one pixel to all other pixels. Since any two pixels within a segment should have the same distance to other segments, the summation of all distances should be very similar within regions. For example, consider the image of a cardinal displayed in Figure 13a. Computing the LLPD distance matrix and summing along the rows to get the degree of all pixels, we get the single valued image displayed in Figure 13b, which shows regions of similar pixels having similar color and therefore degree. If we compare this to the L2 degree displayed in Figure 13c, we see that the regions of similar pixels are not so tightly grouped by degree as with the LLPD. The LLPD separates the regions by degrees so well that we can create an image of the major contours of the image by indicating the pixels where the LLPD degree changes. Larger changes in degree are indicative of stronger contours, displayed in Figure 13d.



(a) **(b)** **(c)** **(d)**

Figure 13: (a) Original Image, (b) Color representation of LLPD pixel degree, (c) Color representation of LLPD pixel degree , (d) Contours of image given by sharp changes in LLPD pixel degree

We can also explore this concept by choosing an image pixel, and plotting the each image pixel with the value of the distance to that pixel. We do this for four image superpixels, one on the body of the bird, one in the black colored facial area, one in the lower branches, and one in the upper background, and display the L2 results in top row of Figure 14 and the LLPD results in the bottom row. From this experiment also, we can see that LLP-distances tend to have very similar values for pixels of the same region, even if they are spatially far apart, whereas the L2 distance is much more affected by spatial distance.

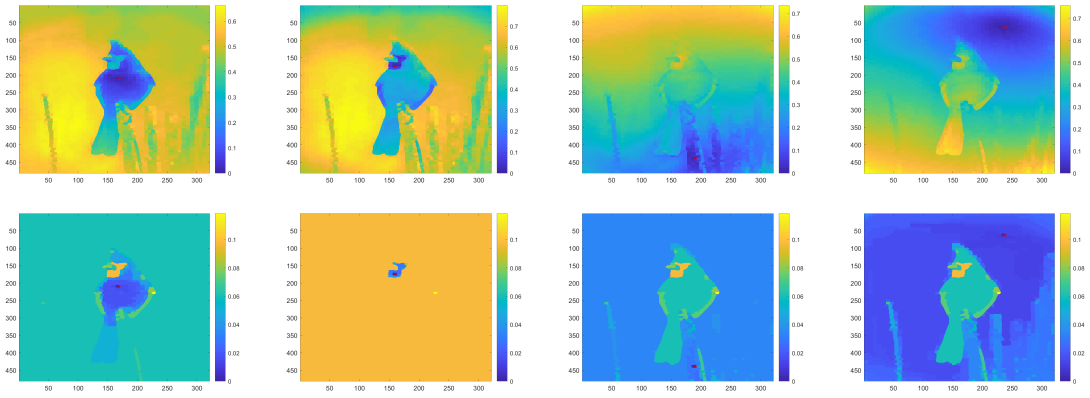


Figure 14: L2- (top row) and LLP- (bottom row) distances to all superpixels from chosen superpixel outlined in red. By column left to right, the chosen superpixel is located on the body of the bird, the black colored facial area, the lower branches, and the upper background.

V.2 Segmentation evaluation criteria

Segmentation experiments are conducted on images from the publicly available BDSD-500 data set [21]. This data set contains contains 200 test, 200 training, and 100 validation RGB images, all $481 \times 321 \times 3$ or $321 \times 481 \times 3$. Each image additionally has been segmented by an average of five different human subjects providing ground truth segmentations to compare against those generated by unsupervised techniques. To evaluate quantitatively how well a computed segmentation S compares against the ground truth segmentation G, we use the three measures described in [14], namely Segmentation Covering (covering), Probabilistic Rand Index (PRI), and Variation of Information (VI).

Segmentation covering is a measure of the overlap between S and G, defined as

$$\mathcal{C}(S \to G) = \frac{1}{N} \sum_{R \in S} |R| \cdot \max_{R' \in G} \mathcal{O}(R, R') \tag{V.1}$$

where N is the total number of image pixels and the overlap $\mathcal{O}(\cdot, \cdot)$ between segmentation regions R and R' is defined as

$$\mathcal{O}(R, R') = \frac{|R \cap R'|}{|R \cup R'|}. \tag{V.2}$$

In the case of multiple ground truth segmentations $\{G_i\}$, the covering is given by the average covering of all ground truth segmentations. A greater segmentation covering indicates a better match between S and G since this indicates better overlap between the two segmentations.

The Probabilistic Rand Index is a measure of the compatibility between S and $\{G_i\}$ given by

$$PRI(S, \{G_i\}) = \frac{1}{T} \sum_{i<j} [c_{ij} p_{ij} + (1 - c_{ij})(1 - p_{ij})], \tag{V.3}$$

where T is the total number pixel pairs, p_{ij} is the probability of the event c_{ij} that pixels i and j have the same label. As PRI increases the segmentation S becomes closer to the ground truth segmentation G indicating better performance.

The Variation of Information metric measures the distance between two segmentations based on their average conditional entropy, and is calculated as

$$VI(S, G) = H(S) + H(G) - 2I(S, G) \tag{V.4}$$

where H represents the entropy and I represents the mutual information between S an G. Opposite to covering and PRI, a lower value of VI indicates better segmentation performance. The segmentation measures are calculated using publicly available code from the UC Berkeley Computer Vision Group [22].

Following the experiments of [30], we evaluate segmentation results with both a *fixed* and *dynamic* scheme. In the fixed scheme, a segmentation is computed for each value of k represented in a ground truth segmentation and report the average PRI, VI, and covering segmentation measures. In the dynamic scheme, a segmentation is produced for each $k \in \{5, 10, 15, 20, 25\}$ and only the measures for the best segmentation are reported for each image.

V.3 Image segmentation experiments

The goal of this thesis is to investigate the use of the LLPD in graph-based image segmentation and to propose a new framework for extending a superpixel embedding to a full image embedding based on the idea of denoising by superpixel averaging. As such, we calculate several segmentations for each image using the following techniques:

- L2 and LLPD after superpixel interpolation, denoted L2I and LLPDI respectively,
- L2 and LLPD after superpixel out-of-sample extension, denoted L2E and LLPDE respectively,
- L2 and LLPD with spatial smoothing of the superpixel out-of-sample extension embedding using a Gaussian Filter, denoted L2E-sa and LLPDE-sa respectively,
- L2 and LLPD after superpixel out-of-sample extension, calculated on an image spatially smoothed with a Gaussian Filter before computations, denoted L2E-sb and LLPDE-sb respectively.

Each of these segmentations are computed on the 200 images of the test set of the BSDS-500 data set, and the corresponding PRI, VI, and covering values are computed. We use a superpixel region size of 10 pixels, yielding approximately 1600 superpixels per image, each containing around 100 pixels each. For spatial smoothing, the size of the window of the Gaussian filter is chosen as the same size as the superpixel region size. To compute the pixel and superpixel data set, we use a value of $\alpha = 0.5$ to equally weight feature and spatial information. When computing the LLPD distance matrix, the full L2 distance matrix is inputted in Algorithm 1, meaning K is set as the number of superpixels.

Since the LLPD represents only the distance of one leg of a path between two points, the scale of L2- and LLP-distances are very different. Therefore choosing a specific value of σ for use in creating the graph adjacency matrix of II.3 will not give a fair comparison between the two methods. Therefore we define as $\sigma = p\, d_{max}$ were d_{max} is the maximum distance of two data points using the appropriate metric. This allows the user to choose the value of p and ensures similar bandwidth for both L2 and LLPD adjacency matrices.

On a personal laptop, computing the region size 10 superpixel data set takes approximately 12 seconds, computing the matrix of LLPDs of the superpixel data set takes around 15 seconds, and solving the eigenvalue problem takes around 2 seconds. The bottleneck of computing these methods come during the embedding extension, which takes around 30-60 seconds for both the

LLPD and L2 interpolation methods depending on the value of k, and around 30 seconds for the L2 and 45 minutes for the LLPD the out-of-sample extension method. The out-of-sample extension method is very computationally expensive because Algorithm 2 must be run for every pixel of the image.

V.4 Results

Table 1 displays the performance measures for each method averaged over the 200 test images of the BSDS-500 data set. From these results, we can draw several conclusions. First is that the LLPD version of all methods outperforms the L2 distance version for all segmentation measures. This indicates that the LLPD may a better metric of the distance between clusters of points than the L2 distance.

Secondly is that extension without any spatial smoothing does not perform very well with these given measures, especially with the VI measure, and that smoothing after computing the extension embedding gives approximately the same covering and PRI but slightly better VI measures. Smoothing before computing the embedding however shows significant improvement over extension with no spatial smoothing, with this method giving the top PRI values of all methods.

Even with the technique of spatially smoothing the image before computing the out-of-sample extension of the superpixel embedding, these results are approximately the same as that of a simple linear interpolation over the spatial pixel coordinates. There are a few reasons why this could be the case. The superpixel grid is still on a small enough scale that the interpolation provides a good approximation of the actual embedding. The use of interpolation between superpixels could also be interpreted as its own secondary denoising scheme as noisy pixels are assigned a smooth value between two already denoised values. The good interpolation results could also be due to the human segmented ground truth provided with the BSDS-500 data set, as human segmentations generally tend to give coarser region boundaries than in actuality, while the extension method is able to give more exact region boundaries.

To explore the differences between these types of segmentations qualitatively, several images from the BSDS-500 test set and their corresponding segmentations are shown in Figure 15. Since the L2 out-of-sample extension methods did not provide good average results, they are not included in Figure 15. From this figure, we can see that using the LLPD allows for large homogeneous regions to not get broken up as is a normality using the L2 distance. We can also see that both the L2 and

	covering		PRI		VI	
method	fixed	dynamic	fixed	dynamic	fixed	dynamic
L2I	0.32	0.46	0.75	0.83	2.80	2.24
LLPDI	**0.39**	**0.57**	0.77	0.86	**2.65**	**1.91**
L2E	0.25	0.40	0.74	0.82	3.49	2.66
LLPDE	0.38	0.52	0.78	0.86	3.01	2.15
L2E-sa	0.25	0.41	0.74	0.82	3.41	2.59
LLPDE-sa	0.37	0.55	0.78	0.86	2.94	1.99
L2E-sb	0.26	0.42	0.75	0.83	3.30	2.52
LLPDE-sb	**0.39**	0.56	**0.79**	**0.87**	2.75	1.96

Table 1: Segmentation measure results average across images of the BDSD-500 data set.

Figure 15: Segmentations for images in the BSDS-500 data set, by column from left to right: Original Image, L2 interpolation, LLPD interpolation, LLPD out-of-sample extension, LLPD out-of-sample extension with post-spatial smoothing, LLPD out-of-sample extension with pre-spatial smoothing

LLPD interpolation segmentations are composed of spatially simpler regions with smooth borders, whereas the out-of-sample extension method almost looks like a copy of the original image even though only a small number of colors are used. We attempt to overcome this level of detail by

applying the Gaussian image filters to spatially smooth the image, but the image regions are still very specific, as can be seen in the last two columns of Figure 15.

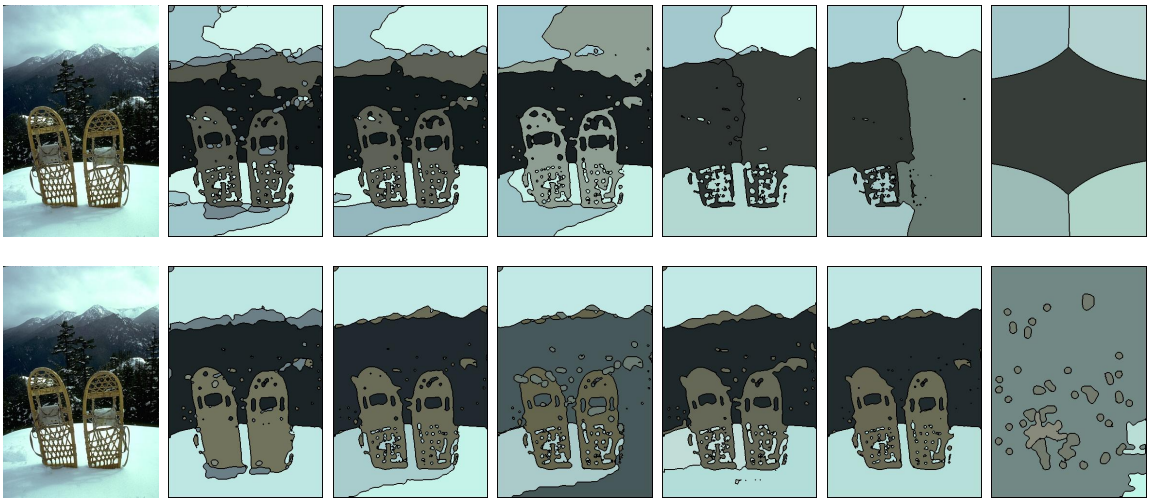
V.5 Varying of parameters for interpolation segmentation

We now explore the result of changing the α and k parameter values for L2 and LLPD interpolation segmentation, and compare visually the advantages of LLPD over L2 segmentation in more detail using the two images displayed below.

The parameter α controls the percentage of spatial information considered in the distance between pixels, with $\alpha = 0$ representing all feature data and $\alpha = 1$ all spatial data. Figure 16 shows the LLPD and L2 interpolation segmentations for an image for $\alpha = 0, 0.2, 0.4, 0.6, 0.8, 1$. This shows that L2-distance segmentations are much more affected by increasing α than the LLPD segmentations, with all values of $\alpha < 1$ showing similar results with LLPD. This implies that increasing the percentage of spatial information does not take away from the importance of feature information when using the LLPD, but it does when using the L2 distance. Our hypothesis on why this is the case is that the minimum of the maximum graph edges on a path between two pixels is an edge between spatially close points, most likely on the boundary between regions where color transitions are sharpest. This would account for this behavior because for this edge the feature distance would be very large and spatial distance very small, so even if the former is weighted low and latter is weighted high, the former still dominates. Thus, approximately the same segmentation is given until $\alpha \approx 1$.

Although a value of $\alpha = 1$ does not make sense to segment the image since feature information is needed in some manner, it reveals some interesting characteristics of the LLPD vs L2-distance. The LLPD on a uniform grid of spatial coordinates is the same for any two points, namely the distance between any two neighboring points. Since we are employing superpixels and the spatial location is given by the average location of all containing pixels, the superpixels are not on a uniform grid. Therefore, for $\alpha = 1$ we get a random looking pattern instead of an image of one color. The L2 distance on the grid of spatial coordinates give a symmetric looking pattern of approximately equal sizes.

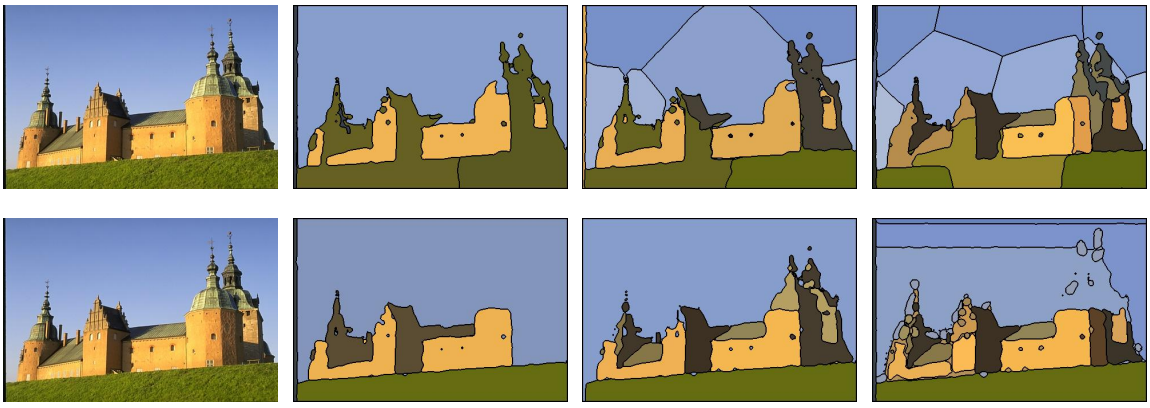
We also look at how varying the number of segments k differs for L2 and LLPD segmentation in Figure 17 below, with L2 segmentations on the top row and LLPD segmentations on the bottom row for $k = 5, 10, 20$. As k increases, the L2 segmentations start to divide large homogeneous regions like the sky and the grass into approximately equal pieces, while the LLPD segmentation



(a) Image **(b)** $\alpha = 0$ **(c)** $\alpha = 0.2$ **(d)** $\alpha = 0.4$ **(e)** $\alpha = 0.6$ **(f)** $\alpha = 0.8$ **(g)** $\alpha = 1$

Figure 16: Segmentations with varying α values for $k = 10$, with the first row showing L2 segmentations and the second row showing LLPD segmentations. By column, (a) Original image, (b) $\alpha = 0$, (c) $\alpha = 0.2$, (d) $\alpha = 0.4$, (e) $\alpha = 0,6$, (f) $\alpha = 0.8$, (g) $\alpha = 1$.

generally keeps the homogeneous regions as one segment. When $k = 20$, the sky gets divided into two pieces using the LLPD, but in a manner that makes sense from looking at the original image versus the blocky pieces that using the L2-distance divides the sky into. This may imply that using LLPD segmentation requires smaller values of k to achieve detection of the major regions of an image.



(a) Image **(b)** $k = 5$ **(c)** $k = 10$ **(d)** $k = 20$

Figure 17: Segmentations with varying k values for $\alpha = 0.5$, with the first row showing L2 segmentations and the second row showing LLPD segmentations. By column, (a) Original image, (b) $k = 0$, (c) $k = 10$, (d) $k = 20$.

VI. Conclusion and Future Directions

VI.1 Conclusion

The creation of a reduced data set using the average of a pre-clustering of the data allows for a spectral embedding that represents only the relevant information of the data by locally averaging out the effect of outlying and noisy data. This scheme is also computationally efficient as the eigendecomposition of spectral clustering only needs to be computed on the reduced data set, and then extended to a full data set embedding. We described how this process can be used for image segmentation using superpixels calculated by the SLIC algorithm, but also laid out the notion of superpixels for general data as small pre-clusters of similar size by merging clusters created by k-means.

This process allows LLPD clustering to give more accurate results on the full data set without the influence of noise. We showed that with this scheme, LLPD graph-based image segmentation outperformed L2 graph-based image segmentation for all segmentation schemes tested on a publically available data set. These experiments also showed that although the out-of-sample extension method is mathematically rigorous in that the eigenvectors will converge to the eigenfunctions of the continuous data set as more data points are added to the model, the simpler scattered data interpolation method performed similarly in experiments on a large publicly available data set. The interpolation of the LLPD created embedding is also much faster than the out-of-sample extension of the LLPD because of the unique way of calculating path-based distances. The LLPD interpolation scheme therefore is the overall best out of the various combinations of extension and distances tested.

We then further investigated the use of LLPD interpolation versus L2 interpolation by varying the value of the k and α parameters. We found that LLPD segmentation generally keeps homogeneous sections of an image as one region even as the number of segments k increases. This implies that smaller values of k may be able to locate the relevant regions of an image when using the LLPD versus the L2. We also found that LLPD is less affected by the amount of spatial information included in the distance between data points, given by α. This leads to the conclusion that the shortest longest leg between two given pixels occurs on a small spatial range. This also means that spatial information is not as necessary for LLPD segmentation as it is for L2 segmentation since the LLPD segmentation varied little as α is varied between 0 and 1, whereas the L2 segmentation varied greatly. This can also be interpreted as the LLPD identifying the strongest contour between

two groups of pixels, and therefore any two pixels one in each group should theoretically have the same LLPD since they are all separated by the same strongest contour.

VI.2 Future directions

VI.2.1 More sophisticated methods of smoothing final embedding

The final image embedding when created with the out-of-sample extension still has noisy points, since the reduced data model does not account for these. To remedy this, we discussed applying a Gaussian filter to the final embedding, which sums over all points in a spatial region of each pixel multiplied by a Gaussian. This works well for smoothing noise points within a spatial region by averaging out the noise over a spatial region. Along the borders of objects though this method will mix the embedding coordinates of two distinct regions, creating a third category of similar embedding coordinates. When performing the final k-means clustering, this tends to cause a thin segments consisting of that border between two distinct regions, which is not desirable. Not respecting object boundaries is a well known phenomenon of Gaussian blurring.

Instead we could implement anisotropic diffusion as a smoothing filter [15], which aims to respect object boundaries by adding a conduction coefficient c that varies in space and smoothing range t, written as

$$I_t = c(x, y, t)\nabla^2 I + \nabla c \cdot \nabla I \tag{VI.1}$$

where x, y represent the spatial coordinates of the image, $I(x, y, t)$ represents the feature of the image at spatial location (x, y) with smoothing parameter t and $I(x, y, 0) = I_0(x, y)$ where I_0 is the original image. The conduction coefficient $c(x, y, t)$ is chosen in such that it has a value of 1 on the interior of regions and a value of 0 on region boundaries, causing smoothing to only occur within each region with no undesirable mixing. The presence of an edge can be estimated by a feature gradient between pixels, and Equation VI.1 can be approximated using a descritzed square lattice. Note that $c(x, y, t) = 0$ will give Gaussian filtering.

VI.2.2 Application to hyperspectral images for target detection

The experiments shown previously indicate that objects of similar features have similar LLPD to other regions, allowing for homogeneous regions to be labeled as one segment. This characteristic could potentially be useful in target detection, which aims to find regions of a specific feature composition, since this metric is more uniform over similar regions than the standard Euclidean distance. Target detection is generally performed on hyperspectral images, which have tens to hundreds of spectral or feature bands instead of the three RGB bands of color images. The segmentation framework described above can be naturally be used for hyperspectral images due to its generality when discussing feature vectors. The process described in [25] for target detection using biased normalized cuts could easily be adjusted to use this LLPD framework, and may provide detection improvement.

VI.2.3 Computational improvement

The LLPD out-of-sample extension is quite computationally intensive due to the unique computation of the LLPD because of its path-based nature. When performing the LLPD out-of-sample extension using Algorithm 2, we are able to make some simplifications to Algorithm 1 because each time we are only adding one point to an already constructed graph. In each use of Algorithm 2 though we still need to sort the rows of the matrix CC by column which is a computationally expensive procedure. Since for each use of Algorithm 2, all rows of CC remain the same besides for the last row, there may be a simpler way to sort CC since the majority of the matrix remains the same. Our implementation currently sorts the entire matrix using MATLAB's built in function **sortrows**.

VI.2.4 Impact of the Bandwidth Parameter

When performing the experiments of Section 5, the bandwidth parameter σ was kept constant with $p = 1$, meaning $\sigma = pd_{\max} = d_{\max}$, for both LLPD and L2. This value of p showed decent results for both types of distances, and in order to focus on comparing LLPD and L2, and the various types of extension the value of σ was kept constant. A further study of the impact of varying σ could therefore be of use, and may result in finding an optimal σ value that gives improved segmentations.

VII. Acknowledgments

I would like to thank my thesis adviser Dr. Nathan Cahill for providing guidance on this thesis and many other research projects over the last three years. I also want to thank my thesis committee members Dr. Nathaniel Barlow, Dr. Kara Maki, and Dr. John Hamilton for taking the time to look over my research, as well Tyler Hayes for being an awesome student mentor and role model over my years at RIT.

Thank you to James Murphy for useful conversations and brainstorming for the main concepts of this research.

I would like to say thank you to my family and friends, especially my parents, for their support in all parts of my college career. A special thanks goes to Brian for always believing in me and providing love and support, even when times get tough.

References

[1] A. Little, M. Maggioni, J. Murphy. Path-based Spectral Clustering: Guarantees, Robustness to Outliers, and Fast Algorithms. *ArXiv Pre-print*, Dec. 2017

[2] A. Y. Ng, M.I. Jordan, Y. Weiss. On Spectral Clustering: Analysis and an algorithm. *Advances in neural information processing systems*, 2002.

[3] B. Fischer, and J. M. Buhmann. Path-based clustering for grouping of smooth curves and texture segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25.4: 513-518, 2003.

[4] B. Fischer, T. Zöller, and J. M. Buhmann. Path based pairwise data clustering with application to texture segmentation. *International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition, Springer*, Berlin, Heidelberg, 2001.

[5] B. Fischer, V. Roth, and J. M. Buhmann. Clustering with the connectivity kernel. *Advances in Neural Information Processing Systems,* 2004.

[6] B. S. Everitt, et al. Hierarchical clustering. *Cluster Analysis*, 5th Edition: 71-110, 2011.

[7] C. Baker. The numerical treatment of integral equations. *Clarendon Press, Oxford*, 1977.

[8] H. Chang, and D. Yeung. Robust path-based spectral clustering. *Pattern Recognition,* 41.1:191-203, 2008.

[9] J. B. Tenenbaum, V. Silva, J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *science* 290.5500:2319-2323, 2000.

[10] J. Shi, and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence* 22.8:888-905, 2000.

[11] M. Belkin, and P. Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. *Advances in neural information processing systems*, 2002.

[12] M. Lichman. UCI machine learning repository, 2013. URL http://archive.ics.uci.edu/ ml.

[13] P. Arbelaez. Boundary extraction in natural images using ultrametric contour maps. *Computer Vision and Pattern Recognition Workshop*, CVPRW'06. Conference on. IEEE, 2006.

[14] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik. Contour detection and hierarchical image segmentation. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 33(5):898-916, May 2011.

[15] P. Perona, T. Shiota, and J. Malik. Anisotropic diffusion. *Geometry-driven diffusion in computer vision,* Springer, Dordrecht, 73-92, 1994.

[16] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Susstrunk, SLIC superpixels compared to state-of-the-art superpixel methods *IEEE Trans. Pattern Analysis and Machine Intelligence* 34, 2274-2282, November 2012.

[17] S. E. Chew, and N.D. Cahill. Semi-supervised normalized cuts for image segmentation. *Proceedings of the IEEE International Conference on Computer Vision*, 2015.

[18] S. T. Roweis, L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *science* 290.5500:2323-2326, 2000.

[19] T. Cox, M. Cox. Multidimensional Scaling. *Chapman & Hall, London*, 1994.

[20] T. Doster and C. C. Olson. Building robust neighborhoods for manifold learning-based image classification and anomaly detection. *Algorithms and Technologies for Multispectral, Hyperspectral, and Ultraspectral Imagery XXII*, Vol. 9840. International Society for Optics and Photonics, 2016.

[21] The Berkeley Segmentation Dataset and Benchmark. www.cs.berkeley.edu/projects/vision/grouping/segbench/.

[22] UC Berkeley Computer Vision Group, Source Code (for Linux/Mac, 32/64 bits), https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/resources.html

[23] VLfeat Toolbox *http://www.vlfeat.org/*

[24] X. Yu Stella, and J. Shi. Multiclass spectral clustering. *null. IEEE*, 2003.

[25] X. Zhang, et al. Biased normalized cuts for target detection in hyperspectral imagery. *Algorithms and Technologies for Multispectral, Hyperspectral, and Ultraspectral Imagery XXII*, Vol. 9840, International Society for Optics and Photonics, 2016.

[26] Y. Bengio, et al. Out-of-sample extensions for lle, isomap, mds, eigenmaps, and spectral clustering. *Advances in neural information processing systems*, 2004.

[27] Y. Bengio, et al. Spectral clustering and kernel PCA are learning eigenfunctions. *CIRANO*, Vol. 1239, 2003.

[28] Y. Weiss. Segmentation using eigenvectors: a unifying review. *Proceedings IEEE International Conference on Computer Vision*, 1999.

[29] Y. Yang, Y. Wang, and X. Xue. A novel spectral clustering method with superpixels for image segmentation. *Optik-International Journal for Light and Electron Optics* 127.1: 161-167, 2016.

[30] Y. Yu, C. Fang, and Z. Liao. Piecewise flat embedding for image segmentation. *Proc. International Conference on Computer Vision*, pages 1368-1376, 2015.

[31] Z. Wu and R. Leahy. An optimal graph theoretic approach to data clustering: theory and its application to image segmentation. *Pattern Analysis and Machine Intelligence*, IEEE Trans. 15(11): 1101-1113, Nov 1993.