A Unified View of Graph Partitioning and Weighted Kernel k-means

Inderjit S. Dhillon, Yuqiang Guan, and Brian Kulis
University of Texas at Austin
Department of Computer Sciences
Austin, TX 78712

UTCS Technical Report #TR-04-25

June 30, 2004

Abstract

The last few years have seen a surge of interest in advanced methods for clustering data. A drawback with several popular clustering algorithms, such as k-means, is that they can only find clusters that are linearly separable in input space. To overcome this problem, two seemingly different approaches have been proposed: kernel k-means and spectral clustering using graph partitioning. Despite significant research, these methods have remained only loosely related. In this paper, we give an explicit theoretical connection between weighted kernel k-means and graph partitioning. We show the generality of the weighted kernel k-means objective function, and derive various graph partitioning objectives as special cases. We are then able to re-interpret nearly all spectral clustering algorithms in terms of the weighted kernel k-means objective function. Given a graph, our results lead to novel weighted kernel k-means algorithms that monotonically improve the normalized cut, ratio cut, or ratio association in the graph, as well as an incremental kernel k-means algorithm for the Kernighan-Lin objective. Our results have important implications: a) eigenvector-based algorithms, which can be computationally prohibitive, are not essential for optimizing various cut criteria, b) due to the monotonicity of iterative algorithms, the output of spectral clustering algorithms may be further optimized using their appropriate iterative counterpart. Since kernel methods have significant memory overheads, we show how to scale our algorithms both in terms of speed and memory requirements. Finally, we present several experimental results, including showing that normalized cut image segmentation can successfully be performed without calculating eigenvectors.

1 Introduction

Clustering has received a significant amount of attention in the last few years as one of the fundamental problems in data mining. It has been applied to a number of different problems in data mining applications, as well as other areas such as image segmentation and circuit layout.

One of the most popular algorithms for clustering is the k-means algorithm, which has been in use for decades. In this algorithm, local optima of a squared loss function are obtained by iteratively reassigning points to their closest clusters and computing the distance from points to every cluster center. Recent research has generalized the algorithm in many ways; for example, similar algorithms for clustering can be obtained using arbitrary Bregman divergences as the distortion measure [2].

Other improvements include local search to improve the clustering results [7], and better cluster initialization [3].

A major drawback to k-means is that it cannot separate clusters that are not linearly separable in input space. A recent approach that has emerged for tackling such a problem is kernel k-means. The data is first mapped to a higher-dimensional feature space using a nonlinear function, and then kernel k-means partitions the points by linear separators in the new feature space. The linear separator in feature space corresponds to a non-linear separator in the input space.

Spectral clustering using graph partitioning is another proposed technique that allows non-linear separation of clusters. These algorithms use the eigenvectors of an affinity (or closely related) matrix to obtain a clustering of the data - the nodes of the graph are the data points, and edges represent similarity. Such an approach has proven useful for a number of clustering problems. Popular objective functions used in spectral clustering are to minimize the ratio cut [4] or normalized cut [22].

On the surface, kernel k-means and graph partitioning appear to be completely different approaches. In this paper we first unite these two forms of clustering under a single framework. By generalizing the k-means objective function to use both weights and kernels, we show how the two approaches to clustering are related. Specifically, we can rewrite the weighted kernel k-means objective function as a trace maximization problem whose relaxation can be solved with eigenvectors. The result shows how particular kernel and weight schemes are related to a number of different spectral clustering objectives. The advantage to our approach is that we can generalize the clustering algorithm to use arbitrary kernels and weights.

In particular, we show that by choosing the weights and kernels in certain ways, the weighted kernel k-means objective function is the same (up to a constant) as the normalized cut, ratio cut, ratio association, or Kernighan-Lin objectives, all graph partitioning problems. Thus far, only eigenvector-based algorithms have been employed to optimize the normalized cut, ratio cut, or ratio association objectives in spectral clustering and image segmentation. However, software to compute eigenvectors of large sparse matrices (often based on the Lanczos algorithm) can have substantial computational overheads, especially when many eigenvectors are to be computed. In such situations, our equivalence has an important implication: we can use k-means-like iterative algorithms for optimizing these graph cut objectives. Using our derivations, we can re-interpret most spectral clustering algorithms, and provide their weighted k-means counterparts. Furthermore, we can show how an incremental kernel k-means algorithm will allow us to optimize the Kernighan-Lin objective.

An issue in using spectral clustering algorithms is that we must transform the eigenvector matrix into a discrete clustering of the points. We generalize the result in [1] to our weighted k-means objective function. This leads to a general method for forming a discrete partitioning of points from the eigenvectors. Finally, our iterative algorithms can be used to further refine the spectral objectives: after running the spectral algorithms and obtaining a discrete partitioning, one can use the iterative algorithms that we introduce to monotonically improve the graph cut objectives.

Spectral methods can be impractical for huge databases; we show how our iterative algorithms can be scaled to very large datasets. We combine two different approaches for scaling kernel k-means, one based on organizing the affinity matrix by blocks, and the other on an acceleration scheme that exploits the triangle inequality.

We show the usefulness of our approach to the application of clustering gene expression data. In bioinformatics applications, we often want to group genes that have strong positive as well as negative correlations to each other, since both forms of correlations imply functional similarity. In order to do so, we can use a quadratic kernel (squared correlation) to obtain an appropriate objective function, and then use spectral relaxation for computing such clusters. Our kernel k-means algorithm also provides a non-spectral approach to such a task. We also illustrate the scalability of our algorithms in terms of computation time by applying it to a large handwriting recognition data set. Finally, we perform image segmentation using an iterative normalized cut algorithm. This approach, which

Polynomial Kernel	$\kappa(\mathbf{a}, \mathbf{b}) = (\mathbf{a} \cdot \mathbf{b} + c)^d$
Gaussian Kernel	$\kappa(\mathbf{a}, \mathbf{b}) = \exp(-\|\mathbf{a} - \mathbf{b}\|^2/2\sigma^2)$
Sigmoid Kernel	$\kappa(\mathbf{a}, \mathbf{b}) = \tanh(c(\mathbf{a} \cdot \mathbf{b}) + \theta)$

Table 1: Examples of popular kernel functions

does not involve computing any eigenvectors, may be quite useful for large images, or in situations where eigenvector computation is prohibitive.

A word about our notation. Capital letters such as A, X, Y and Φ denote matrices; lower-case bold letters such as $\mathbf{a}, \mathbf{b}$ denote column vectors; script letters such as $\mathcal{A}, \mathcal{B}, \mathcal{V}, \mathcal{E}$ represent sets; $\|\mathbf{a}\|$ denotes the L^2 norm of a vector; $\mathbf{a} \cdot \mathbf{b}$ represents the inner product between vectors; and $\|X\|_F$ denotes the Frobenius norm of a matrix, given by $\|X\|_F = (\sum_{i,j} X_{ij}^2)^{1/2}$.

2 The Essentials

In this section, we separately summarize the seemingly different approaches of weighted kernel k-means and graph partitioning.

2.1 Weighted Kernel k-means

The k-means clustering algorithm can be enhanced by the use of a kernel function, a nonlinear mapping from the original (input) space to a higher-dimensional feature space. By using an appropriate mapping, one can extract clusters that are non-linearly separable in input space.

Let us denote clusters by π_j, and a partitioning of points as $\{\pi_j\}_{j=1}^k$. Using the non-linear function ϕ, the k-means objective function using Euclidean distance becomes

$$\begin{aligned} \mathcal{D}(\{\pi_j\}_{j=1}^k) &= \sum_{j=1}^{k} \sum_{\mathbf{a} \in \pi_j} \|\phi(\mathbf{a}) - \mathbf{m}_j\|^2, \\ \text{where} \qquad \mathbf{m}_j &= \frac{1}{|\pi_j|} \sum_{\mathbf{a} \in \pi_j} \phi(\mathbf{a}). \end{aligned}$$

To compute inner products of the form $\phi(\mathbf{a}) \cdot \phi(\mathbf{b})$, we use the kernel representation $K(\mathbf{a}, \mathbf{b})$, where $K(\mathbf{a}, \mathbf{b}) = \phi(\mathbf{a}) \cdot \phi(\mathbf{b})$. This allows us to compute the dot product without having to compute the mapping ϕ. The matrix of inner products K is often called the *kernel matrix* (it is also called the Gram matrix). See Table 1 for examples of popular kernel functions.

Using the above objective function, we can derive an algorithm analogous to standard k-means. Furthermore, by careful manipulation, it becomes possible to run the entire computation for the kernel k-means algorithm using only the entries of the kernel matrix.

We can generalize the kernel k-means objective function by discussing a weighted variant. As we shall see later, this generalization is powerful and encompasses various well known spectral clustering formulations. We introduce a non-negative weight for data point $\mathbf{a}$, denoted by $w(\mathbf{a})$. The modified objective function is:

$$\mathcal{D}(\{\pi_j\}_{j=1}^k) = \sum_{j=1}^{k} \sum_{\mathbf{a} \in \pi_j} w(\mathbf{a}) \|\phi(\mathbf{a}) - \mathbf{m}_j\|^2.$$

ALGORITHM 1: Weighted Kernel k-means.

WEIGHTED_KERNEL_KMEANS(K, k, w, $C_1, ..., C_k$)
Input: K: kernel matrix, k: number of clusters, w: weights for each point
Output: $C_1, ..., C_k$: partitioning of the points
1. Initialize the k clusters: $C_1^{(0)}, ..., C_k^{(0)}$.
2. Set $t = 0$.
3. For each point $\mathbf{a}$, find its new cluster index as

$$c^*(\mathbf{a}) = \text{argmin}_c \|\phi(\mathbf{a}) - \mathbf{m}_c\|^2, \text{ using (1).}$$

4. Compute the updated clusters as

$$C_c^{t+1} = \{\mathbf{a} \ : \ c^*(\mathbf{a}) = c\}.$$

5. If not converged, set $t = t + 1$ and go to Step 3; Otherwise, stop.

The cluster representative of π_j is

$$\mathbf{m}_j = \frac{\sum_{\mathbf{b} \in \pi_j} w(\mathbf{b}) \phi(\mathbf{b})}{\sum_{\mathbf{b} \in \pi_j} w(\mathbf{b})}.$$

Note that this is the "best" cluster representative since

$$\mathbf{m}_j = \text{argmin}_{\mathbf{z}} \sum_{\mathbf{a} \in \pi_j} w(\mathbf{a}) \|\phi(\mathbf{a}) - \mathbf{z}\|^2.$$

Clearly, we can obtain the original (or unweighted) kernel k-means objective function by setting all weights to be equal to one. We must compute the distance from each (weighted) point to every cluster representative. For cluster π_j, this is given by

$$\left|\left| \phi(\mathbf{a}) - \frac{\sum_{\mathbf{b} \in \pi_j} w(\mathbf{b}) \phi(\mathbf{b})}{\sum_{\mathbf{b} \in \pi_j} w(\mathbf{b})} \right|\right|^2,$$

which can be expanded as

$$\phi(\mathbf{a}) \cdot \phi(\mathbf{a}) - \frac{2 \sum_{\mathbf{b} \in \pi_j} w(\mathbf{b}) \phi(\mathbf{a}) \cdot \phi(\mathbf{b})}{\sum_{\mathbf{b} \in \pi_j} w(\mathbf{b})} + \frac{\sum_{\mathbf{b}, \mathbf{d} \in \pi_j} w(\mathbf{b}) w(\mathbf{d}) \phi(\mathbf{b}) \cdot \phi(\mathbf{d})}{(\sum_{\mathbf{b} \in \pi_j} w(\mathbf{b}))^2}. \quad (1)$$

As stated before, the dot products $\phi(\mathbf{a}) \cdot \phi(\mathbf{b})$ are contained in the kernel matrix K. All computation is in the form of such inner products, hence we can replace all inner products by entries of the kernel matrix.

To obtain an iterative algorithm for kernel k-means, we must compute the distances as in (1). However, all other aspects k-means remain the same, as shown in Algorithm 1.

Assuming we are able to store the whole affinity matrix in main memory, we can analyze the time complexity of Algorithm 1. It is clear that the bottleneck is Step 3, i.e., the computation of distances. The first term in (1), $\phi(\mathbf{a}) \cdot \phi(\mathbf{a})$, need not be computed since it is a constant for $\mathbf{a}$ and thus does not affect the assignment of $\mathbf{a}$ to clusters. The second term is calculated once per data point, and costs $O(n)$ each time it is computed, leading to a cost of $O(n^2)$ per iteration. For the third term, notice that $\frac{\sum_{\mathbf{b}, \mathbf{c} \in \pi_j} w(\mathbf{b}) w(\mathbf{c}) \phi(\mathbf{b}) \cdot \phi(\mathbf{c})}{(\sum_{\mathbf{b} \in \pi_j} w(\mathbf{b}))^2}$ is fixed for cluster j, so in each iteration it is computed

once and stored. Thus the complexity is $O(n^2)$ scalar operations per iteration. Initially, we must compute the kernel matrix K, which usually takes time $O(n^2 m)$, where m is the dimension of the original points. If the total number of iterations is τ, then the time complexity of Algorithm 1 is $O(n^2(\tau + m))$.

2.2 Spectral Clustering and Graph Partitioning

Spectral clustering has emerged recently as a popular method for clustering data that uses eigenvectors of a matrix derived from the data. Several algorithms have been proposed in the literature [14, 16, 17, 22], each using the eigenvectors in slightly different ways. There have been a few papers comparing different algorithms, including some analysis of their properties [23, 24].

Nearly all spectral clustering algorithms can be viewed as optimizing a graph partitioning objective. Several different graph partitioning objectives have been proposed in the literature and used in various applications. In this section, we review a few of the most prominent ones. We will see later that by writing graph partitioning objectives as trace maximization problems, we can obtain iterative and spectral algorithms for the objectives.

We are given a graph $G = (\mathcal{V}, \mathcal{E}, A)$, where $\mathcal{V}$ is the set of vertices, $\mathcal{E}$ is the set of edges, and A is an edge affinity matrix. We assume that the matrix A is both symmetric (G is undirected) and nonnegative. For two subsets $\mathcal{A}$ and $\mathcal{B}$ of $\mathcal{V}$, we define the links between $\mathcal{A}$ and $\mathcal{B}$ to be the sum of the affinity weights from $\mathcal{A}$ to $\mathcal{B}$. In other words,

$$\text{links}(\mathcal{A}, \mathcal{B}) = \sum_{i \in \mathcal{A}, j \in \mathcal{B}} A(i, j).$$

We define the degree of a set $\mathcal{A}$ to be the total links from $\mathcal{A}$ to all nodes in $\mathcal{V}$, i.e., $\text{degree}(\mathcal{A}) = \text{links}(\mathcal{A}, \mathcal{V})$. We can now define a number of graph partitioning objectives.

Kernighan-Lin Objective. The Kernighan-Lin graph partitioning algorithm [15] is a local search procedure that maintains two equally sized partitions while trying to minimize the cut between the partitions. We can generalize their objective function to k partitions (for ease in presentation, we assume that the total number of vertices is divisible by k), and we say that the objective function is

$$\text{minimize} \sum_{i=1}^{k} \frac{\text{links}(\mathcal{V}_i, \mathcal{V} \setminus \mathcal{V}_i)}{|\mathcal{V}_i|}, \quad \text{subject to } |\mathcal{V}_i| = |\mathcal{V}|/k, \quad \text{for all } i = 1, \ldots, k.$$

Ratio Cut. For the ratio cut objective [4], we are intertested in optimizing the following:

$$\text{minimize} \sum_{i=1}^{k} \frac{\text{links}(\mathcal{V}_i, \mathcal{V} \setminus \mathcal{V}_i)}{|\mathcal{V}_i|}.$$

Ratio Association. This maximization problem is similar to the ratio cut problem (though not identical). We are interested in optimizing the following:

$$\text{maximize} \sum_{i=1}^{k} \frac{\text{links}(\mathcal{V}_i, \mathcal{V}_i)}{|\mathcal{V}_i|}.$$

Normalized Cut. For normalized cut [16, 1, 25], instead of dividing by the size of the partition, we use the degree of the partition:

$$\text{minimize} \sum_{i=1}^{k} \frac{\text{links}(\mathcal{V}_i, \mathcal{V} \setminus \mathcal{V}_i)}{\text{degree}(\mathcal{V}_i)}.$$

Note that minimizing the normalized cut is equivalent to the corresponding normalized association problem, a maximization problem.

3 The Spectral Connection

At first glance, weighted kernel k-means and spectral clustering using graph partitioning appear to be quite different. After all, spectral clustering uses eigenvectors to help determine the partitions, whereas eigenvectors do not appear to figure in kernel k-means. In this section, we show how we can express weighted kernel k-means as a trace maximization problem, and later we will see how to express each of the graph partitioning objectives as trace maximizations as well, leading to iterative algorithms for the graph partitioning objectives. On the other hand, relaxations for the trace maximization problems lead to spectral clustering algorithms, which allows us to optimize weighted kernel k-means using spectral methods. This will connect the two methods of clustering.

For ease in presentation, let us denote the "distortion" of a cluster π_j to be $d(\pi_j) = \sum_{\mathbf{a}\in\pi_j} w(\mathbf{a})\|\phi(\mathbf{a}) - \mathbf{m}_j\|^2$. Then we have that $\mathcal{D}(\{\pi_j\}_{j=1}^k) = \sum_{j=1}^k d(\pi_j)$. Moreover, let us denote, for a cluster π_j, the sum of the w weights of the points in π_j to be s_j; in other words, $s_j = \sum_{\mathbf{a}\in\pi_j} w(\mathbf{a})$. Finally, let us denote W to be the diagonal matrix of all the w weights, and W_j to be the diagonal matrix of the weights in π_j.

It is easy to see that we can rewrite the mean vector $\mathbf{m}_j$ as

$$\mathbf{m}_j = \Phi_j \frac{W_j \mathbf{e}}{s_j},$$

where Φ_j is the matrix of points associated with cluster π_j (after the ϕ mapping), i.e., $\Phi = [\phi(\mathbf{a}_1, \phi(\mathbf{a}_2), \ldots, \phi(\mathbf{a}_n)]$, and $\mathbf{e}$ is the vector of all ones.

We can rewrite the distortion of cluster π_j to be:

$$\begin{aligned}
d(\pi_j) &= \sum_{\mathbf{a}\in\pi_j} w(\mathbf{a})\|\phi(\mathbf{a}) - \mathbf{m}_j\|^2 \\
&= \sum_{\mathbf{a}\in\pi_j} w(\mathbf{a})\|\phi(\mathbf{a}) - \Phi_j \frac{W_j\mathbf{e}}{s_j}\|^2 \\
&= \|(\Phi_j - \Phi_j \frac{W_j\mathbf{e}\mathbf{e}^T}{s_j})W_j^{1/2}\|_F^2 \\
&= \|(\Phi_j W_j^{1/2}(I - \frac{W_j^{1/2}\mathbf{e}\mathbf{e}^T W_j^{1/2}}{s_j})\|_F^2.
\end{aligned}$$

Using the fact that $\mathrm{trace}(AA^T) = \mathrm{trace}(A^TA) = \|A\|_F^2$, and noting that $I - \frac{W_j^{1/2}\mathbf{e}\mathbf{e}^T W_j^{1/2}}{s_j} = P$ is an orthogonal projection, i.e. $P^2 = P$ since $s_j = \mathbf{e}^T W_j \mathbf{e}$, we get that

$$\begin{aligned}
d(\pi_j) &= \mathrm{trace}\left(\Phi_j W_j^{1/2}\left(I - \frac{W_j^{1/2}\mathbf{e}\mathbf{e}^T W_j^{1/2}}{s_j}\right)\left(I - \frac{W_j^{1/2}\mathbf{e}\mathbf{e}^T W_j^{1/2}}{s_j}\right)W_j^{1/2}\Phi_j^T\right) \\
&= \mathrm{trace}\left(\Phi_j W_j^{1/2}\left(I - \frac{W_j^{1/2}\mathbf{e}\mathbf{e}^T W_j^{1/2}}{s_j}\right)W_j^{1/2}\Phi_j^T\right) \\
&= \mathrm{trace}(W_j^{1/2}\Phi_j^T\Phi_j W_j^{1/2}) - \frac{\mathbf{e}^T W_j}{\sqrt{s_j}}\Phi_j^T\Phi_j\frac{W_j\mathbf{e}}{\sqrt{s_j}}.
\end{aligned}$$

If we represent the full matrix of points as $\Phi = [\Phi_1, \Phi_2, \ldots, \Phi_k]$, then we have that

$$\mathcal{D}(\{\pi_j\}_{j=1}^k) = \text{trace}(W^{1/2}\Phi^T\Phi W^{1/2}) - \text{trace}(Y^T W^{1/2}\Phi^T\Phi W^{1/2} Y),$$

where

$$Y = \begin{bmatrix} \frac{W_1^{1/2}\mathbf{e}}{\sqrt{s_1}} & & & \\ & \frac{W_2^{1/2}\mathbf{e}}{\sqrt{s_2}} & & \\ & & \cdots & \\ & & & \frac{W_k^{1/2}\mathbf{e}}{\sqrt{s_k}} \end{bmatrix}.$$

Note that Y is an $n \times k$ *orthonormal* matrix, i.e., $Y^T Y = I$.

Since $\text{trace}(\Phi W \Phi^T)$ is a constant, we see that we have derived an equivalent formulation for the weighted kernel k-means objective function. In particular, we rewrite the minimization of the objective function as a maximization of $\text{trace}(Y^T W^{1/2}\Phi^T\Phi W^{1/2} Y)$. The matrix $\Phi^T\Phi$ is simply the kernel matrix K of the data, so we can rewrite it as the maximization of $\text{trace}(Y^T W^{1/2} K W^{1/2} Y)$.

A standard result in linear algebra [11] provides a global solution to a relaxed version of this problem. By allowing Y to be an arbitrary orthonormal matrix, we can obtain the optimal Y by taking the top k eigenvectors of $W^{1/2}KW^{1/2}$. Similarly, the sum of the top k eigenvalues of $W^{1/2}KW^{1/2}$ gives the optimal trace value.

4 Implications

The previous section shows that the weighted kernel k-means problem can be written as a trace maximization problem. We now show how each of the graph partitioning objectives can also be written as trace maximizations, leading to new iterative algorithms for the objectives. We also discuss a postprocessing method for obtaining a discrete clustering from the eigenvector matrix when using spectral methods.

4.1 Normalized Cuts using Weighted Kernel k-means

As discussed in [25], the normalized cut problem can be recast as a trace maximization problem.

Let $\mathbf{x}_j$ denote the indicator vector for partition j, i.e., $\mathbf{x}_j(i) = 1$ if cluster $\mathcal{V}_j$ contains the data point i. Also, let D be the degree matrix for A: D is a diagonal matrix whose entries correspond to the sum of the rows of A. Notice that $\text{degree}(\mathcal{V}_j) = \mathbf{x}_j^T D \mathbf{x}_j$ and $\text{links}(\mathcal{V}_j, \mathcal{V}_j) = \mathbf{x}_j^T A \mathbf{x}_j$. We noted earlier that the normalized cut problem is equivalent to the normalized association problem; i.e., the problem can be expressed as:

$$\text{maximize} \left\{ \sum_{j=1}^k \frac{\text{links}(\mathcal{V}_j, \mathcal{V}_j)}{\text{degree}(\mathcal{V}_j)} = \sum_{j=1}^k \frac{\mathbf{x}_j^T A \mathbf{x}_j}{\mathbf{x}_j^T D \mathbf{x}_j} = \sum_{j=1}^k \tilde{\mathbf{x}}_j^T A \tilde{\mathbf{x}}_j \right\},$$

where $\tilde{\mathbf{x}}_j = \mathbf{x}_j / (\mathbf{x}_j^T D \mathbf{x}_j)^{1/2}$.

If we let $\tilde{X}$ be the matrix of all $\tilde{\mathbf{x}}_j$ vectors, then the above expression may be re-written as $\text{trace}(\tilde{Y}^T D^{-1/2} A D^{-1/2} \tilde{Y})$, where $\tilde{Y} = D^{1/2}\tilde{X}$, and is orthonormal.

We now show a simple relationship between the trace maximizations of the normalized cut and kernel k-means problems. If we set $W = D$ and $K = D^{-1}AD^{-1}$, the trace maximization problem of weighted kernel k-means is to maximize $\text{trace}(Y^T D^{-1/2} A D^{-1/2} Y)$, which is *equivalent* to the trace maximization for normalized cut. Thus, with this choice of weights, we can use the weighted kernel k-means procedure in order to minimize the normalized cut. However, this choice of K is only

ALGORITHM 2: NCut_Kernel_k-means.

NCUT_KERNEL_KMEANS(A, k, $C_1, ..., C_k$)
Input: A: edge affinity matrix, k: number of clusters
Output: $C_1, ..., C_k$: partitioning of the points
1. Compute D, the diagonal matrix whose entries are the sum of the rows of A.
2. Let the w weight for each point $\mathbf{a}_i$ be D_{ii} and set $K = pD^{-1} + D^{-1}AD^{-1}$.
3. Initialize the k clusters: $C_1^{(0)}, ..., C_k^{(0)}$.
4. Set $t = 0$.
5. For each point $\mathbf{a}_i$ corresponding to column i of K, find its new cluster index as

$$c^*(\mathbf{a}_i) = \text{argmin}_c f(\mathbf{a}_i, c),$$

where $f(\mathbf{a}_i, c)$ equals

$$K(i,i) - \frac{2\sum_{\mathbf{b}_j \in \pi_c} w(\mathbf{b}_j)K(i,j)}{\sum_{\mathbf{b}_j \in \pi_c} w(\mathbf{b}_j)} + \frac{\sum_{\mathbf{b}_j, \mathbf{d}_l \in \pi_c} w(\mathbf{b}_j)w(\mathbf{d}_l)K(j,l)}{(\sum_{\mathbf{b}_j \in \pi_c} w(\mathbf{b}_j))^2}.$$

6. Compute the updated clusters as

$$C_c^{t+1} = \{\mathbf{a}_i \ : \ c^*(\mathbf{a}_i) = c\}.$$

7. If not converged, set $t = t + 1$ and go to Step 5; Otherwise, stop.

guaranteed to monotonically decrease the normalized cut objective if K is positive definite. This is because if K is positive definite, then it can be viewed as $\Phi^T\Phi$, and thus as inner products of a function ϕ. With this property, we can prove convergence of the kernel k-means algorithm, but we will have no such guarantee for arbitrary K.

For the case that K is not positive definite, we instead define $K = pD^{-1} + D^{-1}AD^{-1}$, where p is chosen to be large enough that K is positive definite. Here we use the fact that a matrix is positive definite if and only if all of its eigenvectors are positive, so we can easily find a p value such that K is positive definite. Consider running weighted kernel k-means on this matrix K, with $W = D$. Then the trace maximization can be written as:

$$\begin{aligned} & \text{trace}(Y^T D^{1/2} K D^{1/2} Y) \\ = \ & \text{trace}(Y^T D^{1/2} pD^{-1} D^{1/2} Y) + \text{trace}(Y^T D^{-1/2} AD^{-1/2} Y) \\ = \ & pk + \text{trace}(Y^T D^{-1/2} AD^{-1/2} Y) \end{aligned}$$

Hence, the maximization problem is equivalent to the trace maximization problem for the normalized cut of A, for *any* symmetric A.

By the monotonicity property of kernel k-means, it can be shown that Algorithm 2 has the following property:

Property 1: Each iteration of NCut_Kernel_k-means (Algorithm 2) decreases the k-way normalized cut.

Since K is positive definite by construction, we can interpret K as inner products of a kernel function [5], as is done in kernel k-means.

4.2 Ratio Cuts and Kernighan-Lin using Weighted Kernel k-means

Now we extend our analysis for the ratio cut problem. Again, let $\mathbf{x}_j$ denote the indicator vector for partition j. Then we can rewrite the ratio-cut minimization problem as

$$\text{minimize} \sum_{j=1}^{k} \frac{\mathbf{x}_j^T (D - A)\mathbf{x}_j}{\mathbf{x}_j^T \mathbf{x}_j},$$

since $\mathbf{x}_j^T D\mathbf{x}_j - \mathbf{x}_j^T A\mathbf{x}_j = \text{degree}(\mathcal{V}_j) - \text{links}(\mathcal{V}_j) = \text{links}(\mathcal{V}_j, \mathcal{V} \setminus \mathcal{V}_j)$, and $\mathbf{x}_j^T \mathbf{x}_j = |\mathcal{V}_j|$.

The matrix $D - A$ is the Laplacian, which we will write as L, and it can be proven that this matrix is positive definite. It is easy to see that the above problem is equivalent to:

$$\text{minimize} \left\{ \sum_{j=1}^{k} \frac{\mathbf{x}_j^T L\mathbf{x}_j}{\mathbf{x}_j^T \mathbf{x}_j} = \sum_{j=1}^{j} \tilde{\mathbf{x}}_j^T L\tilde{\mathbf{x}}_j = \text{trace}(\tilde{X}^T L\tilde{X}) \right\},$$

with $\tilde{\mathbf{x}}_j = \mathbf{x}_j/(\mathbf{x}_j^T \mathbf{x}_j)^{1/2}$, and $\tilde{X}$ the matrix of $\tilde{\mathbf{x}}_j$ vectors.

The difficulty with this formulation is that we have written k-means in Section 3 as a trace maximization, not a minimization. Though one can perform trace minimization by taking the smallest eigenvectors of L, the corresponding k-means objective function is a maximization problem. Hence, it is not immediately clear how to relate the minimization of the k-means objective function to the above trace minimization.

In a similar fashion to the normalized cut problem, consider the matrix $K = pI - L$ for unweighted kernel k-means, with p sufficiently large such that K is positive definite. Then we have that the objective function is equivalent to

$$\begin{aligned} & \text{trace}(pI - L) - \text{trace}(Y^T(pI - L)Y) \\ = \; & \text{trace}(pI) - \text{trace}(L) - p\text{trace}(Y^T Y) + \text{trace}(Y^T LY) \\ = \; & p(n - k) - \text{trace}(L) + \text{trace}(Y^T LY). \end{aligned}$$

Hence, the minimization of the objective function for kernel k-means with $K = pI - L$ is equivalent to the minimization of $\text{trace}(Y^T LY)$, which is the ratio cut of A since $Y = \tilde{X}$.

There are several simple methods for choosing the value p. One approach is to use the fact that the norm of a matrix is greater than or equal to the largest eigenvalue. Hence, we can calculate a simple upper bound on the largest eigenvalue by determining $\|L\|_\infty = \max_i \sum_{j=1}^{n} |L_{ij}|$.

Our approach, therefore, is simply to run kernel k-means with all weights equal to one, using the kernel matrix $pI - L$. We can verify the following property of the algorithm:

Property 2: Each iteration of Ratio-Cut Kernel k-means decreases the k-way ratio cut.

Again, this follows directly from the fact that the k-means objective function decreases monotonically at every iteration, and our analysis from the previous sections.

The Kernighan-Lin graph partitioning objective follows easily from the ratio cut objective. For the case of K-L partitioning, we maintain equally sized partitions, and hence the only difference between the ratio cut and K-L partitioning is the fact that the X_j indicator vectors are constrained to be of size $|\mathcal{V}|/k$. If we start with equally sized partitions, an incremental weighted kernel k-means algorithm (where we only consider swapping points, or chains of points, that improve the objective function) can be run to simulate the Kernighan-Lin algorithm.

Graph Partitioning Objective	Weights	Kernel
Ratio Association	$w(\mathbf{a}) = 1$ for all $\mathbf{a}$	$K = pI + A$
Ratio Cut	$w(\mathbf{a}) = 1$ for all $\mathbf{a}$	$K = pI - L$
Kernighan-Lin	$w(\mathbf{a}) = 1$ for all $\mathbf{a}$	$K = pI - L$
Normalized Cut	$w(\mathbf{a}) =$ degree of $\mathbf{a}$	$K = pD^{-1} + D^{-1}AD^{-1}$

Table 2: Popular graph partitioning objectives and corresponding weights and kernels given affinity matrix A

4.3 Ratio Association using Weighted Kernel k-means

Finally, the related maximization problem of ratio association can also easily be added into our framework. In the ratio association problem for an affinity matrix A, we are interested in maximizing

$$\sum_{j=1}^{k} \frac{\text{links}(\mathcal{V}_j, \mathcal{V}_j)}{|\mathcal{V}_j|}$$

If we use the same indicator vector that was used in ratio cuts, the maximization can be written as

$$\text{maximize} \sum_{j=1}^{k} \frac{\mathbf{x}_j^T A \mathbf{x}_j}{\mathbf{x}_j^T \mathbf{x}_j}$$

This can be rewritten as the maximization of trace$(Y^T AY)$, where Y is orthonormal (we set column j of Y as $\mathbf{x}_j / (\mathbf{x}_j^T \mathbf{x}_j)^{1/2}$). It is easy to verify that this is equivalent to the unweighted kernel k-means objective function, using the affinity matrix as the kernel matrix. As before, the matrix A does not necessarily have to be positive definite, as we can substitute $pI + A$ for sufficiently large p.

In Table 2, the weights and kernels for each graph objective are summarized.

4.4 Re-interpreting Spectral Algorithms

Nearly all spectral clustering algorithms attempt to optimize one of the three spectral clustering objectives that we have just discussed (normalized cut, ratio cut, or ratio association). Consequently, we now have the ability to re-interpret most spectral clustering algorithms in terms of the weighted kernel k-means objective function.

As an example, consider the algorithm of Pothen, Simon, and Liou [19]. This algorithm is recursive, and uses the second smallest eigenvector of the Laplacian to partition the points into two clusters. The algorithm partitions all of the data into two clusters, then recurses on the two clusters. Using the result from the ratio cut section, if we consider the associated k-means objective, with $k = 2$, then we find that the spectral relaxation involves using the two smallest eigenvectors of the Laplacian to partition the points. However, the smallest eigenvector of the Laplacian is all ones, and does not contribute. Therefore, only the second smallest eigenvector is used to partition the points. Since the eigenvectors of $pI - L$ are equal to the eigenvectors of L, their algorithm can be reformulated as a spectral relaxation to a weighted kernel k-means objective function, where the kernel is $pI - L$ and all weights are one.

As another example, consider the Ng, Jordan, and Weiss algorithm [17]. Their algorithm first computes the kernel matrix K, where the kernel that is used is the Gaussian Kernel. They compute a diagonal matrix D such that the diagonal entries of D are the sums of the rows of K. Then they form $D^{-1/2}(I - K)D^{-1/2}$, compute the eigenvectors of this matrix (which are equal to the eigenvectors of $D^{-1/2}KD^{-1/2}$), and form a discrete clustering using these eigenvectors.

Algorithm	Spectral Objective
Perona and Freeman [18]	Recursive Ratio Association
Scott and Longuet-Higgins [21]	k-way Ratio Association
Pothen, Simon, and Liou [19]	Recursive Ratio Cut
Chan, Schlag, and Zien [4]	k-way Ratio Cut
Shi and Malik [22]	Recursive Normalized Cut
Meila and Shi [16]	k-way Normalized Cut
Yu and Shi [25]	k-way Normalized Cut
Ng, Jordan, and Weiss [17]	k-way Normalized Cut

Table 3: Spectral Clustering Algorithms and their Associated Objectives

Hence, we see that the NJW algorithm can be viewed as either a spectral relaxation to the weighted kernel k-means objective function or as a normalized cut problem. The connection to normalized cuts is clear: we view the affinity matrix K in the spectral algorithm as defining the edge weights of a graph, and their algorithm attempts to minimize the normalized cut in this graph. It follows then that this algorithm can be viewed as a spectral relaxation to weighted kernel k-means, using the weights and kernel from normalized cut.

In Table 3, we list several spectral algorithms and their associated objectives. All of these algorithms can be interpreted in terms of weighted kernel k-means and their objectives can be locally optimized by iterative algorithms.

One advantage to our use of an iterative algorithm for these graph problems is that we can use different improvement methods, such as local search, to increase the quality of the results. In situations where eigenvector computation is difficult, for example, when the affinity matrix is large and sparse, and many eigenvectors are desired, our iterative algorithm is particularly useful.

Another major advantage is that we can improve the output of most spectral clustering algorithms using our iterative approach. After running spectral clustering, we obtain a partitioning of the points that relates to a relaxed version of a spectral objective function. If we then run the corresponding iterative clustering algorithm on the partition, the monotonicity of the iterative algorithm guarantees us that we can only improve the quality of the clustering. This two-layer approach – first running spectral clustering to get an initial partitioning and then refining the partitioning by running kernel k-means on the partitioning – typically results in a robust partitioning of the data. We will show experimental results indicating that spectral methods provide excellent initializations to iterative clustering algorithms.

4.5 Kernel k-means using Eigenvectors

The reformulation of the kernel k-means objective function allows us to solve a relaxed problem using the eigenvectors of the matrix $W^{1/2}KW^{1/2}$. This yields a spectral approach to minimizing the objective function: we first compute the top k eigenvectors of the matrix $W^{1/2}KW^{1/2}$. This maps the original points to a lower-dimensional space.

We require that some postprocessing be done in order to obtain a discrete clustering of the data points from the eigenvectors. This may be accomplished by clustering the points in this new space, or another method. In [26], for example, the cluster assignment was computed by using a pivoted QR decomposition.

In [1], Bach and Jordan show that, for spectral clustering based on normalized cuts, a good way to obtain a discrete clustering from the matrix of eigenvectors is to run a weighted k-means algorithm. They show that if the rows of the eigenvector matrix are treated as k-dimensional points,

then weighted k-means can be run with w_i weights equal to the sum of the rows of the original affinity matrix, and secondary v_i weights equal to $w_i^{-1/2}$. It is shown that using these weights minimizes a natural cost function between the eigenvector solution and a discrete solution.

We can extend their analysis to our general framework. In this way, we will provide a strong method for postprocessing to achieve a discrete partitioning of the data, thus leading to a complete view of spectral clustering. We see that to minimize the cost function of Bach and Jordan, weighted k-means can be run with w_i weights given before taking eigenvectors, and for every i we multiply row i of the eigenvector matrix by $w_i^{-1/2}$ before running weighted k-means (this corresponds to the v weights in the Bach and Jordan algorithm). Their analysis follows easily for this generalized case. See [1] for a more extensive treatment of postprocessing.

5 Scalability Issues

In this section, we discuss methods for scaling the kernel k-means algorithm to very large data sets.

The paper [27] by Zhang and Rudnicky considers the problem of scalability for kernel k-means. We will combine the ideas presented in that paper with the ideas of [9, 6] to discuss a scalable approach to running kernel k-means.

The key idea in [27] is that, while the affinity matrix may be too large to store in main memory, we can instead read in the affinity matrix block by block. We need only store an n by k matrix C, whose (i, j) entry tells us the current distance from point i to cluster j. We computed the distance from $\phi(\mathbf{a}_i)$ to $\mathbf{m}_c$ as

$$K(i,i) - \frac{2\sum_{\mathbf{b}_j \in \pi_c} w(\mathbf{b}_j) K(i,j)}{\sum_{\mathbf{b}_j \in \pi_c} w(\mathbf{b}_j)} + \frac{\sum_{\mathbf{b}_j, \mathbf{d}_l \in \pi_c} w(\mathbf{b}_j) w(\mathbf{d}_l) K(j,l)}{(\sum_{\mathbf{b}_j \in \pi_c} w(\mathbf{b}_j))^2} \tag{2}$$

We can therefore compute the denominators of this expression for each cluster before reading the blocks, and we simply compute C as we read in each block, which is possible since the distance is a sum of kernel matrix values.

We note that we can improve marginally on the procedure given in [27]. Instead of computing in a completely serial fashion – namely, reading in a block, computing on that block, reading in the next block, and so on – we can break our block into two smaller blocks B_1 and B_2. While we are computing with B_1, we read in a new B_2. Then, when we are finished with B_1, we can immediately begin computing with B_2. Then, while we compute with B_2, we read in a new B_1. This pipelining avoids waiting for each block to be read in.

While the above procedure helps us compute kernel k-means on very large data sets, we still have the problem that kernel k-means is slower than standard k-means. To speed up the computation, we can adapt the pruning procedure used in [9, 6]. The idea behind the acceleration scheme is that we can use the triangle inequality to avoid unnecessary computation.

The acceleration is based on the following observation: using the triangle inequality, we have that, for a point i, $d(i, m') \geq d(i, m) - d(m, m')$, where m is an old center and m' is a new center.

We see that we can compute the distances between new and old centers and store the information into a k by k matrix D. Similarly, we keep a k by n matrix L that contains *lower bound* information for the distance from each point to each center. Suppose after a single iteration, all distances are computed between each point and each center. In the next iteration, we can obtain the lower bounds from the points to the new centers by using the $d(m, m')$ calculations and the distances from the previous iteration. In this way, we calculate the lower bounds as $d(i, m) - d(m, m')$ for the matrix L.

Furthermore, these lower bounds allow us to eliminate explicitly computing many distances. Once we have computed lower bounds and begin to compute exact distances, the lower bound allows us

to determine whether or not to compute remaining distances exactly.

We should also briefly mention scalability of the Lanczos algorithm. The bottleneck for computing eigenvectors of a matrix M using the Lanczos method is the computation of M times a vector $\mathbf{x}$. There are methods for performing this multiplication even if the matrix M is too large to be stored in main memory. Such an approach would help to scale the Lanczos computation to large data sets.

6 Experimental Results

We now provide experimental results. In this section, we first illustrate the kernel k-means algorithm by showing how it correctly identifies non-linearly separable clusters in two artificial data sets: the two circles of the TwoCircle data set and points of the XOR data set; see Section 6.1. Then we apply the kernel clustering idea to a bioinformatics application. We show that degree-2 polynomial kernel k-means generates "diametric clustering" in the two gene expression datasets: human fibroblast gene expression data [13] and the yeast data set of Rosetta Inpharmatics [12]. Further, with the handwriting recognition data set, Pendigits, we show that using eigenvectors to initialize kernel k-means gives better initial and final objective function values and better clustering results. Thus the theoretical connection between spectral clustering and kernel k-means helps in obtaining higher quality results. We then show that our distance estimation techniques save a considerable amount of computation time. Finally, we perform normalized cut image segmentation using our iterative normalized cut algorithm.

6.1 Data sets

We first use two artificial datasets, XOR and TwoCircles. XOR consists of 100 points in two-dimensional space. Assuming points on the diagonals form clusters, we will see that the degree-2 polynomial kernel k-means algorithm as well as the associated spectral algorithm can correctly identify the clusters. TwoCircles has 150 points which form two circles in two-dimensional space. An exponential kernel k-means clustering is applied on it to recover the two circular clusters.

The two gene expression datasets we analyze are: human fibroblast gene expression data and the yeast dataset of Rosetta Inpharmatics. The human fibroblast gene expression dataset records the response of human fibroblasts after addition of serum to the growth media. This dataset contains expression levels for 517 human genes whose expressions change substantially following serum stimulation. The data (12 time points and an unsynchronized sample) is preprocessed by dividing each entry by the unsynchronized sample expression level, taking the log of the result, then normalizing each 12-element expression vector to have unit L_2 norm. The yeast dataset of Rosetta Inpharmatics consists of 300 experiments measuring the expression of 6048 yeast genes, in which transcript levels of a mutant or compound-treated culture were compared to those of a wild-type or mock-treated culture. After removing genes with missing expression measurements, we are left with 5246 genes. Then we normalize each gene vector to have unit L_2 norm after shifting it by its mean.

Another real-life dataset we use is Pendigits downloaded from UCI machine learning repository (ftp://ftp.ics.uci.edu/ pub/machine-learning-databases/pendigits), which contains (x, y) coordinates of hand-written digits. This dataset contains 7494 training digits and 3498 testing digits. Each digit is represented as a vector in 16-dimensional space. The coordinate values are inside the interval $[0, 100]$. In our experiments, the values are normalized to $[0, 1]$.

We also use a sample image in our experiments to show that our weighted kernel k-means algorithm is able to successfully segment the image into components. Image segmentation is often done using a spectral algorithm for normalized cuts, and hence our algorithm may be useful in cases where eigenvector computation is prohibitive (for example, if the image is very large).