Spectral Clustering Survey

HU, Pili[*]

May 14, 2012[†]

Abstract

The term Spectral Clustering is a collection of algorithms. Many researchers propose their own variations and algorithm specific justifications. We observe that the core operation of Spectral Clustering is eigen value decomposition and an embedding step is implicitly or explicitly performed. This is analogous to Spectral Embedding algorithms and many eigen decomposition based Dimensionality Reduction methods from machine learning community. In this article, we term them all as Spectral Embedding Technique.

In this survey, we first construct a simple-minded sample spectral clustering algorithm. Then we conclude taxonomy of spectral clustering. Next, we propose a general three-stage framework of spectral clustering. Combinatoric, stochastic, and other type of justifications are collected and organized in later sections. After that, we give a brief introduction on several representative dimensionality reduction methods and relate them to our general spectral clustering framework. We give several unifying views of Spectral Embedding Technique: graph framework, kernel framework, trace maximization. We end this article with a discussion on the relationship between Kernel K-Means and Spectral Clustering.

This article aims at providing systematic ways to explore new spectral clustering algorithms. At the same time, we hope to gain some insights through the analysis of a bunch of algorithms. Source code of documents and sample algorithms can be found in the online open source repository[20].

[*] hupili [at] ie [dot] cuhk [dot] edu [dot] hk

[†] Last compile:May 24, 2012

Contents

1 Introduction

Spectral Clustering(SC) has long been used in several disciplines. For example, computer vision[33]. Spectral Embeding(SE) was also widely discussed in the community[10]. Outside spectral community, the machine learning community also developed many linear or non-linear Dimensionality Reduction(DR) methods, like Principal Component Anslysis (PCA), Kernel PCA (KPCA)[32], Locally Linear Embedding (LLE)[29], etc. Other technique like Multi-Dimensional Scaling(MDS) was successfully used in computational psychology for a very long time[9], which can be viewed as both "embedding" or "dimensionality reduction".

According to our survey, although those methods target at different problems and are derived from different assumptions, they do share a lot in common. The most significant sign is that, the core procedure involves Eigen Value Decomposition(EVD) or Singular Value Decomposition(SVD), aka "spectral". They all involve an intermidiate step of embedding high-dimensional / non-Euclidean / non-metric points into a low-dimensional Euclidean space (although some do not embed explicitly). In this case, we categorize all these algorithms as Spectral Embedding Technique(SET).

1.1 A Sample Spectral Clustering Algorithm

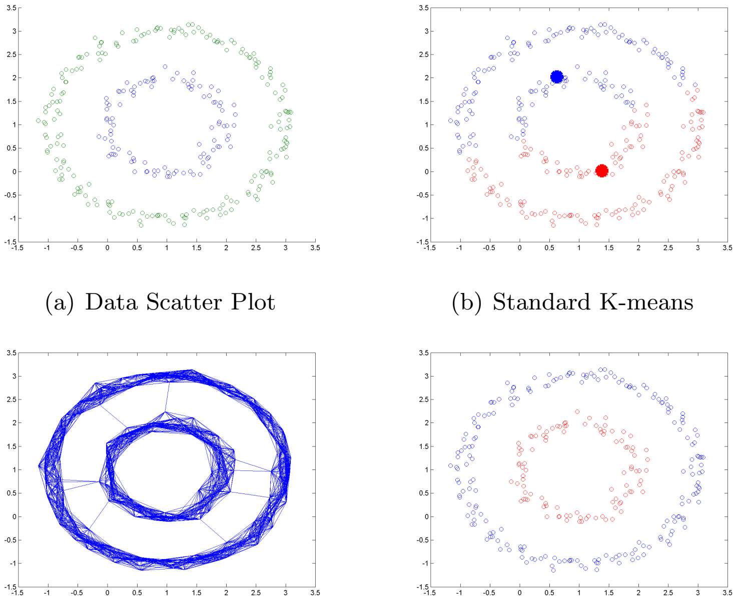
There are many variations of SC. They all work under certain conditions and researchers don't have a rule of thumb so far. Before we analyze their procedure and justification, we present a simple but workable sample algorithm(**Alg 1**).

Algorithm 1 Sample Spectral Clustering

```
Input: Data matrix $X = [x_1, x_2, \dots, x_N]$; Number of Clusters $K$.
Output: Clustering $\{C_i\}$: $C_i \in V$ and $\cap_i C_i = \emptyset$ and $\cup_i C_i = V$.
Form adjacency matrix $A$ within $\epsilon$-ball.
Solve $A = U \Lambda U^{\mathrm{T}}$, indexed according the eigenvalue's magnitude.
$Y \leftarrow$ first $K$ columns of U.
Cluster $Y$'s rows by K-means.
```

In **Alg 1**, the ϵ-ball adjacency graph is constructed as follows. First create one vertex for each data point. If for two points i, j satisfy $||x_i - x_j|| < \epsilon$, connect them with an edge. In this simple demonstration, we consider an unweighted graph, i.e. all entries of A are 0(disconnected) or 1(connected).

Fig 1 shows the result of our sample SC algorithm, compared with standard K-means algorithm. **Fig 1(a)** shows the scatter plot of data. It is composed of one radius 1 circle and another radius 2 circle, both centered at (1,1). **Fig 1(b)** shows the result of standard K-means working on Euclidean distance. **Fig 1(c)** shows the graph representation, where the



(a) Data Scatter Plot

(b) Standard K-means

(c) Adjacency Graph

(d) Sample SC Algorithm

Figure 1: Demonstration of Sample SC Algorithm

adjacency graph is formed by taking a ϵ-ball and $\epsilon = 0.7$ in the example. **Fig 1(d)** shows the output of **Alg 1**. It's obvious that standard K-means algorithm can not correctly cluster the two circles. This is a known major weakness of K-means(in Euclidean): When clusters are not well separated spheres, it has difficulty recovering the underlying clusters. Although K-means works for this case if we transform the points into polar coordinate system(see [20] for code), the solution is not universal. On the contrary, our sample SC algorithm can separate the two clusters in this example. Informally speaking, this is probably because the eigenvectors of adjacency matrix convey adequate information.

A precaution is that **Alg 1** does not always work even in this simple case. Nor have we seen this algorithm in formally published works (so far), let alone justifications. This algorithm is only to show the flavour of spectral clustering and it contains those important steps in other more sophisticated algorithms.

Readers are recommended to learn von Luxburg's tutorial[37] before reading the following sections. Since that paper is very detailed, we'll present overlapping topics concisely (except for when we have different organization and views).

1.2 Spectral Clustering Taxonomy

When the readers start to survey SC related topics, they will soon find that there are two streams of study:

- **Embedding Like**. An example is presented in **Alg 1**. One of the core procedure is an embedding of points into lower dimensional Euclidean space. After that, hard cut algorithm like K-means is invoked to get the final result. This stream has a lot in common with SE and DR. In the current article, we focus on this line of research.
- **Partitioning Like**. One early example is the 2-way cut algorithm presented by Shi and Malik in[33]. Later Kannan et al. analyzed the framework further[23]. The core procedure of this type of algorithm is a 2-way partitioning subroutine. By invoke this subroutine on resultant subgraph repeatedly, we can finally get K clusters. When the subroutine can guarantee certain quality, the global resultant clustering quality can be guaranteed [23]. The attracting aspect of Kannan's framework is that, we can plugin any subroutine as long as they have quality gurantee in one 2-way cut. For example, the eigen vector of left normalized graph Laplacian can induce good 2-way cut in terms of the Normalized Cut criterion[33] (**Section 3.1.3**). This line of research is more close to Spectral Graph Partitioning(SGP), although those algorithms also get the name of Spectral Clustering.

It's worth to note some work of SGP. In Spielman's[34] and Lau's[25] lecture notes, they both presented a graph partitioning algorithm by random walk argument (said to stem from Lovász[26]). Chung[12] made this framework clear: First define a function $f(v)$ on all vertices; Then set threshold T to cut vertices into two groups, $\{v|f(v) \leq T\}$ and $\{v|f(v) > T\}$. The question now becomes to find a good heuristic function f. Note that the original bi-partition problem is of complexity $O(2^N)$ and the complexity of heuristic based bi-partition problem is only $O(N)$(plus the time to get f). If we define f as the second eigen vector of left normalized graph Laplacian, it coincides with the algorithm of Shi and Malik[33]. Besides, there can be multiple f_i and the best cut can still be searched in polynomial time (given f_i). For example, in [25] Lau used random walk probability $P_1, P_2, \ldots, P_t$ of each step as f. Some recent research of the heuristic function are Personalized Pagerank[2], Evolving Sets[3], etc.

In the rest of this article, we refer Spectral Clustering to the first type, i.e. the embedding like algorithms.

2 Spectral Clustering Framework

We propose the following framework to cover currently surveyed variations of SC(and other SET):

1. **Metric Formulation**. This step forms a pairwise metric, upon which an adjacency matrix of (weighted) graph can be constructed. There are several kinds of input: high-dimensional data (usual case); pairwise proximity input (like MDS, see **Section 4.1**); (weighted) graph (like the input of SGP). If the input of SC is already a graph, this step is omitted. For proximity measures, especially dissmilarity, it is usually first converted to approximte pairwise inner product in Euclidean space. The pairwise inner product is a positive related quantity with similarity(e.g. Jaccard's coefficient for 0/1 vector[38]), and thus suits the notion of weights of graph edges. For high-dimensional data, there are more freedom in the metric formulation stage, like similarity graph[37], geodesic distance[35], etc.

2. **Sepctral Embedding**. With the adjacency matrix built in last stage, this stage embeds all vertices into a lower dimensional Euclidean space. For SC community, this embedding makes clustering structure more clear, so that simple algorithms working in Euclidean space can detect the clusters. For DR community, this embedding reveals the shape of manifolds in their parametric space. The two goals are essentially correlated. The core procedure is to do EVD and differences lie before and after EVD:

   - **Matrix Type**. Some authors use graph Laplacian [37, 4, 33] ; others use [28, 10, 23] adjacency matrix.
   - **Normalization**. Both Laplacian and adjacency matrix can be unnormalized, symmetrically normalized, or left(row) normalized[37]. They corresponds to different interpretation and will be explored later.
   - **Scaling**. As is shown in **Alg 1**, we can directly embed vertices using the corresponding row of Y (like [28]). Other alternatives are to scale by square root eigenvalue (like [10]) and scale by eigenvalue (like PCA[8]).
   - **Projection**. For many algorithms, the Y (after scaling) provides an Euclidean space embedding. There are others which further project the rows of Y onto a unit sphere, like [28] and [10].

3. **Clustering**. Based on the embedding result, simple algorithms can be invoked to perform a hard cut on the points. Traditional methods from data mining community are K-means and hierarchical clustering like single/complete linkage[22]. Among those techniques, K-means are the most widely used. A variation of K-means will be proposed later in this article, in order to better fit some angle preserving SET. Simpler hard cut techniques are also possible, e.g. looking at the largest entry of the embedding coordinates[23].

Note that not all of the combinations are justified in published works. We organize them in this way to reveal possibilities from a practitioners perspective. If some combinations yield good results in practice, we can seek for justifications using tools from spectral graph theory or machine learning.

2.1 Metric Formulation

Although metric formulation is not the main body of SET, we think of it highly important. There are always many ways of metric formulation given a practical problem. With poorly constructed metric matrix, even the best embedding technique helps little. Besides, after all the disucssion in this article, we will find that a large portion of differences between algorithms are absorbed in this stage.

2.1.1 High Dimensional Data

For high dimensional data, the following ways can be applied to obtain an adjacency graph:

- ϵ-ball[37]. If $||x_i - x_j|| < \epsilon$, we connect vertices i, j with an edge.
- k-Nearest-Neighbour(kNN)[37]. For each vertex, we connect it with its k nearest neighbours based on Euclidean distance.
- Mutual kNN(m-kNN)[37]. Note the set of kNN is not symmetric. Multual kNN connects those points who are kNN to each other.
- Complete graph[37]. All vertices are connected with each other. This construction is ususally used with Gaussian kernel weighting below.

The adjacency graph only concerns how vertices are connected. After the construction of adjacency graph, edges can be weighted in several ways:

- Unweighted[4]. The adjacency matrix is mere 1(connected) or 0(disconnected).
- Gaussian kernel[37], also called heat kernel[4]. Each edge is weighted by $A_{ij} = \exp\{-||x_i - x_j||^2/t\}$, where t is a super parameter controling the decaying rate of similarity.

Heuristics on selecting ϵ, k, t are proposed by many authors, e.g. ch8 of [37], but there is no real rule of thumb. Since the focus of SET study is on the embedding part, most work do not try hard to tweak the construction of similarity graph. We propose other possibilities, which may be helpful to target different practical problems:

- Mahalanobis distance[39]. The connection condition $||x_i - x_j|| < \epsilon$ is substituted by $(x_i - x_j)^{\mathrm{T}}\Sigma^{-1}(x_i - x_j) < \epsilon^2$. Accodingly, when Gaussian kernel is used, the edge weight is given by $A_{ij} = \exp\{-0.5(x_i - x_j)^{\mathrm{T}}\Sigma^{-1}(x_i - x_j)\}$.
- Jaccard's coefficient[38]. It computes the ratio of the intersection size to the union size of two sets. It is useful when the high dimensional input coordinates can be interpreted as sets.
- Cosine similarity[40]. It computes the angle between two vectors and is widely used in text mining context.

2.1.2 Proximity

Many real problems has proximity as input. Proximity can be described by similarity or dissimilarity. With pairwise similarity input, we can directly fit the data into following SE procedure. A more interesting problem is to transform dissimilarity into similarity, or at least an equivalent quantity. Decomposing the transformed matrix should be able to yield reasonable embedding(under certain justification).

Denote data matrix by $X = [x_1, x_2, \ldots, x_N]$. Every column x_j corresponds to an n dimensional point. We calculate the pairwise squared distance by $d_{ij}^2 = ||x_i - x_j||^2 = x_i^{\mathrm{T}} x_i + x_j^{\mathrm{T}} x_j - 2x_i^{\mathrm{T}} x_j$. Grouping the N^2 entries into matrix form, we have:

$$D^{(2)} = c\vec{1}^{\mathrm{T}} + \vec{1}c^{\mathrm{T}} - 2X^{\mathrm{T}}X \tag{1}$$

where c is a column vector with $x_i^{\mathrm{T}} x_i$ being the entries. We'll see later (**Section 4.1**) that once a matrix B can be written in the form $B = X^{\mathrm{T}}X$, we have standard ways to decompose it and recover the low dimensional embedding. That is, given dissimilarity measures, we can construct the corresponding inner product form. A standard approach is double centering: [9]

$$\begin{aligned} J &= I - \frac{1}{n}\vec{1}\vec{1}^{\mathrm{T}} && (2) \\ X^{\mathrm{T}}X &= -\frac{1}{2}JD^{(2)}J && (3) \end{aligned}$$

There are yet two problems left:

- First, not all dissimilarities are distance. One reason is that the real world data is with noise. In such case, random noise will be reduced by SET. Another reason, maybe more frequently observed, is data inconsistency. This is a usual case in computational psychology when people try to rate the degree of dissimilarity of objects[9]. Those empirical data may break distance laws, like triangle inequality.

- Second, in order to make double centering work, the low dimensional data are required to have zero mean, namely $\sum_i x_i = 0$. Actually, we can choose any points as the origin(rather than sample mean), and corresponding forms can be derived. Since our input in this case is pairwise distance(D or $D^{(2)}$), there is no explicit definition of origin. Or in another word, the effect of embedding is invariant under translation. We can safely locate the embedded sample mean at the origin.

In total, given dissimilarity matrix D, we take the element wise square $D^{(2)}$ and then pass the double centered version $-\frac{1}{2}JD^{(2)}J$ to next stage.

2.1.3 Enhancements

The above sections discussed the basic formulation of an adjacency matrix A. They can be fit direcly into SE procedures. At the same time, people propose some enhancement techniques based on certain assumptions.

Geodesic distance, as is in isomap[35], computes all pair shortest path (geodesic distance) of the adjacency graph first. Then the pairwise geodesic distance is treated as normal distance, D. The standard MDS (**Section 4.1**) can construct an embedding for D. For details, please see **Section 4.2**.

Although we have not found other widely used enhancements in literature, the discussion here do reveal other possibilities. For example, what will be the result if we plug in effective resistance as distance and fit in later SE stage? The effective resistance is closely related with commute time[25]. If it yield good results in practice, justification will not be hard.

2.2 Spectral Embeding

This stage takes a weighted graph adjacency matrix as input. The matrix can be derived from several starting points as described in the last section. Besides, enhancements may have already been performed. Regardless of their origins, we treat them as the adjacency matrix of similarity graph.

The output of SE stage is usually an $N \times d$ matrix Y. The columns of Y are d eigen vectors(or scaled version). The N rows of Y provides a d-dimensional Euclidean embedding.

2.2.1 Diagonal Shifting

Although off-diagonals of A are defined to be affinity / similarity / inner product, the definition of diagonal varies in different contexts.

In spectral community, the diagonals are interpreted as self-loops, and they play little role in objective definition. For example, in Normalized Cut (**Section 3.1.3**), self-loops does not influence the cut and their influence on the volume of each cluster is mighty, which could be absorbed into more general framwork (**Section 3.1.4**). So a natural operation is to zero out those self-loops as is in most SC literature.

In machine learning community, A is more probably expected to be an inner product matrix. In other words, it is Positive SemiDefinite(PSD). In the language of kernels, it is a valid Gram matrix. For example, in **Section 2.1.1**, we use the Gaussian kernel $A_{ij} = \exp\{-||x_i - x_j||^2/t\}$ to weight edges. If we stick to this equation, $A_{ii} = 1$. That means a vertex has highest similarity with itself, which is naturally right. Most importantly, A may now be PSD and fits some DR techniques like MDS (**Section 4.1**), kernel PCA (**Section 4.4**), etc. If we zero out the diagonals, those algorithms can also be invoked and probably still output good results. However, the justification will be different.

Despite this operational difference, we notice that they yield essentially the same result following a diagonal shifting procedure [14]:

$$A' = \sigma I + A \tag{4}$$

where $\sigma = 1$ in our Gaussian kernel example. With large enough σ, any A can be transformed to an equivalent PSD version by linear algebraic argument. The only difference between A and A' is that they have different eigenvalues. Their eigenvectors are essentially the same.

2.2.2 Adjacency or Laplacian

We define the degree matrix R to be a diagonal matrix with $R_{ii} = \sum_j A_{ij}$. The Laplacian is defined as:

$$L = R - A \tag{5}$$

The use of adjacency or Laplacian is closely related with the justification. For adjacency matrix, the normalized version has analogy to a transition matrix of a Markov process. Besides, the adjacency matrix series SET often has angle preserving justification. For Laplacian, its variations may appear in cut or conductance like objectives.

Moreover, the Laplacian is PSD:

$$\begin{aligned} x^{\mathrm{T}} L x &= \sum_i (\sum_j A_{ij}) x_i^2 - \sum_{ij} A_{ij} x_i x_j \\ &= \frac{1}{2} \sum_i \sum_j A_{ij} (x_i^2 + x_j^2) - \sum_{ij} A_{ij} x_i x_j \\ &= \frac{1}{2} \sum_i \sum_j A_{ij} (x_i - x_j)^2 \geq 0 \end{aligned} \tag{6}$$

and Laplacian has $\vec{1}$ as an eigen vector with smallest eigen value 0:

$$L\vec{1} = 0\vec{1} \tag{7}$$

In practice, the first eigen vector of Laplacian (or variations) is ignored, and successive smallest eigen vectors are used. For adjacency matrix, the first largest eigen vectors are used. This is natural due to the negation in **Eq 5**.

We leave further discussion of the use of adjacency or Laplacian to **Section 3**, where the context is more clear.

2.2.3 Normalization

The intuition of normalization can be explained as:

- In spectral graph theory justifications, like Ratio Cut (**Section 3.1.2**), Normalized Cut (**Section 3.1.3**), Weighted Cut (**Section 3.1.4**), the normalization can be interpreted as assigning different importance to nodes. For example, all 1's in Rcut and degree in Ncut. The general normalization can be described by a weighting matrix W with corresponding weights on the diagonal.
- In random walk series justifications, the walk matrix is required to be row-stochastic. Thus $W = R$ and left normalization is the standard operation.

We follow similar notations as [37] and collect possible normalization operations in **Tbl 2.2.3**.

Table 1: Normalization

Normalization	Adjacency	Laplacian
Unnormalized	A	L
Symmetric normalized	$A_{\text{sym}} = W^{-\frac{1}{2}} A W^{-\frac{1}{2}}$	$L_{\text{sym}} = W^{-\frac{1}{2}} L W^{-\frac{1}{2}}$
Left normalized	$A_{\text{rw}} = W^{-1} A$	$L_{\text{rw}} = W^{-1} A$ [1]

Assume $\lambda, \lambda_{\text{sym}}, \lambda_{\text{rw}}$ are eigenvalues of $A, A_{\text{sym}}, A_{\text{rw}}$ with corresponding eigen vectors $v, v_{\text{sym}}, v_{\text{rw}}$, their relationship can be shown through the following array of equations:

$$\begin{aligned}
Av &= \lambda v && (8)\\
W^{-1}Av_{\text{rw}} = A_{\text{rw}}v_{\text{rw}} &= \lambda_{\text{rw}}v_{\text{rw}} && (9)\\
Av_{\text{rw}} &= \lambda_{\text{rw}}Wv_{\text{rw}} && (10)\\
W^{-\frac{1}{2}}W^{-\frac{1}{2}}AW^{-\frac{1}{2}}W^{\frac{1}{2}}v_{\text{rw}} = A_{\text{rw}}v_{\text{rw}} &= \lambda_{\text{rw}}v_{\text{rw}} && (11)\\
A_{\text{sym}}(W^{\frac{1}{2}}v_{\text{rw}}) &= \lambda_{\text{rw}}(W^{\frac{1}{2}}v_{\text{rw}}) && (12)\\
A_{\text{sym}}v_{\text{sym}} &= \lambda_{\text{sym}}v_{\text{sym}} && (13)
\end{aligned}$$

Comparing **Eq 8** and **Eq 10**, we can see that left normalized version corresponds to solving a generalized eigen problem (A, W). It's obvious that

[1] The "rw" stands for "random walk"[37].

normalization makes a big difference in practice. On the other hand, there is only slight difference between two types of normalization. Comparing **Eq 12** and **Eq 13**, we know that they have the same eigen values and their eigen vectors differ by a scaling of $W^{\frac{1}{2}}$.

Similar relationship holds for $L, L_{\text{sym}}, L_{\text{rw}}$.

In the literatures, e.g. [4] [33], L_{sym} often appears as one intermediate step due to variable substitution and eigen vector of L_{rw} is usually the final result.

It's worth to note that which normalization to use is coupled with stages before and after and is also related to justification angle. It's very hard to reach a general conclusion. For example, Ng[28] actually uses a symmetric normalization and Shi[33] uses a left normalization. In [28], Ng claimed superior performance. On the other hand, Luxburg recommend left normalization in general([37] ch8). Some comparisons may be ill-posed because involved algorithms may use different matrix type and do different post-processing on eigenvectors. All the details contribute to performance differences on varied application scenarios.

2.2.4 Eigen Value Decomposition

Eigen Value Decomposition (EVD) and Singular Value Decomposition (SVD) are two important subroutines in SET. They are algebraically related [41]:

$$X = U\Sigma V^{\mathrm{T}} \tag{14}$$

$$XX^{\mathrm{T}} = U\Sigma V^{\mathrm{T}} V\Sigma U^{\mathrm{T}} \tag{15}$$

$$(XX^{\mathrm{T}})U = U\Sigma^2 \tag{16}$$

Similarly,

$$(X^{\mathrm{T}}X)V = V\Sigma^2 \tag{17}$$

Now suppose $X = [x_1, x_2, \ldots, x_N]$ is our data matrix. Some authors view PCA(**Section 4.3**) as an SVD on X ([9] ch24). Others view PCA (**Section 4.3**) EVD on $X^{\mathrm{T}}X$. **Eq 17** shows that they yield the same embedding V.

In our framework, the current SE step take adjacency matrix (or Laplacian) as input. It is essentially a matrix containing pairwise information. So EVD is performed in this stage. The comparison of EVD and SVD in this section is to show that under some SET's settings, the explicit construction of an equivalent similarity matrix (metric formulation in our framework) can be omitted.

2.2.5 Scaling and Projection

The scaling and projection described earlier are both regarded as post-processing of eigen vectors. For example, MDS(**Section 4.1**) and Brand's algorithm [10] scale the eigenvectors by square root eigenvalues; Ng's algorithm [28] projects the coordinates given by eigenvectors to unit sphere. A per case justification and discussion is better and we leave it to **Section 3**.

Despite of their own rationale, we provide one possible interpretation of scaling. Consider 0/1 version adjacency matrix A. If we raise it to the power A^k, the entry $(A^k)_{ij}$ just counts the number of k-hop paths from i to j. We call the graph corresponding to A^k as the "k-th order connectness graph". The original matrix power is only defined for integer k. We notationally generalize it in the following way: (for symmetric A)

$$A^k = (U\Lambda U^{\mathrm{T}})^k = U\Lambda^k U^{\mathrm{T}} \tag{18}$$

$$A^x = U\Lambda^x U^{\mathrm{T}} \tag{19}$$

where x is a real value and $\Lambda^x = \mathrm{diag}(\lambda_1^x, \lambda_2^x, \ldots, \lambda_N^x)$. With this generalization, we can define eigen value scaling as a standard procedure of post-processing. For non-scaled algorithms, just let $x = 0$. This can be interpreted as working on the x-th power of adjacency matrix. It has the same effect if we raise the matrix to the x-th power in the enhancement stage(**Section 2.1.3**).

The justification for x-th power is: x controls the degree of graph distance we want to capture. For example, when $x \to \infty$, by the argument of power method, only the eigenvector with largest eigenvalue (principal eigenvector) is "kept". So the principal eigenvector provides the embedding which captures very "distant" relationships. The successive eigenvectors provides embedding which captures nearer and nearer relationships.

This justification has an analogy. Katz[24] provided an index to measure similarities of vertices in graphs: ([1] ch2.2)

$$\mathrm{Katz}(i,j) = \sum_{k=1}^{\infty} \beta^k (A^k)_{ij} \tag{20}$$

The Katz Index takes all "k-th order connectness graph" into consideration and the decaying factor β imposes a preference between near and distant connectness relationship.

One may think to scale the eigen vectors by a polynomial or more generally a function of eigen values, i.e.

$$Uf(\Lambda) \tag{21}$$

This operation risk losing good justifications. In our survey, we have only seen three choice of f:

- 0. It corresponds to the non-scaled case.
- $\Lambda^{\frac{1}{2}}$. This operation often roots from an error energy minimization / low-rank approximation view point(**Section 3.3**).

- $(\Lambda^+)^{\frac{1}{2}}$ (the square root of Moore-Penrose Inverse[43]). This operation is corresponding to a commute time justification (**Section 3.2.2**).
- Katz polynomial(**Eq 20**). On one hand, it has good justification by considering all length's connectness. On the other hand, we have a closed form for **Eq 20**:[1]

$$\text{Katz} = (I - \beta A)^{-1} - I \tag{22}$$

Nevertheless, seeking for an application tailored $f(\Lambda)$ with good justification is still open work for practitioners.

2.3 Clustering

For K-means and hierarchical clustering, readers can consult standard data mining texts like [22]. For simpler hard cut algorithms, like using the largest entry of embedded coordinates as the cluster index, we omit the discussion in this section because their operation and justification are closely binded. As an example to stimulate exploring more hard cut techniques, we propose a variation of K-means. This variation should work better with Ng's [28] and Brand's [10] SE procedure.

Algorithm 2 Angular K-means

```
Input: output of SE: $Y = [y_1, y_2, \ldots, y_N]$; Number of Clusters $K$.
Output: Clustering $\{C_i\}$: $C_i \in V$ and $\cap_i C_i = \emptyset$ and $\cup_i C_i = V$.
Random initilize cluster centers $t_1^{(new)}, t_2^{(new)}, \ldots, t_K^{(new)}$.
repeat
    $t_i^{(old)} \leftarrow t_i^{(new)}$
    $l_i \leftarrow \arg\max_j < y_i, t_j^{(old)} >, \forall i = 1, 2, \ldots, N$
    $t_j^{(new)} \leftarrow \sum_{i=1}^{N} [l_i = j] x_i, \forall j = 1, 2, \ldots, K$ {[.] is indicator function}
    Normalize $t_j^{(new)}, \forall j = 1, 2, \ldots, K$
until $(\sum_{j=1}^{K} (1- < t_j^{(new)}, t_j^{(old)} >)) < \epsilon$
$C_j = \{i | l_i = j\}$
```

In Ng's [28] and Brand's [10] work, their low dimensional points are projected onto a unit sphere. It is more reasonable to cluster according to the angles in this case. Our variation of K-means (**Alg 2**) takes the embedding property into consideration. Again the hard cut stage is not the focus of SC or SE study, so little work has been found to explore suitable hard cut algorithms. As long as the SE stage induces well located clusters, Euclidean K-means can easily detect them. As reported in [28], with special initialization of K-means centroids, only one recursion is needed for convergence.

3 Spectral Clustering Justification

In this section, we collect justifications from different authors. We will see that not all of the combinations provided in **Section 2** have corresponding justifications.

3.1 Combinatoric Justification

The traditional and most widely study is combinatoric justification. The idea is centered on the concept of "cut". This section is adapted from [33] [37] [14].

3.1.1 Cut

Suppose C_1 and C_2 are two subsets of V. The cut between C_1 and C_2 is defined as:

$$\mathrm{cut}(C_1, C_2) = \sum_{u \in C_1, v \in C_2, (u,v) \in E} A_{uv} \tag{23}$$

The volume of C_1 is defined as:

$$\mathrm{vol}(C_1) = \mathrm{cut}(C_1, V) = \sum_{u \in C_1, v \in V, (u,v) \in E} A_{uv} = \sum_{v \in C_1} d(v) \tag{24}$$

where $d(v) = \sum_{u \in V} A_{vu}$ is the degree of vertex v.

The most straightforward trial to obtain a good clustering is given by the following optimization problem:

$$\underset{\{C_1, C_2, \dots, C_k\}}{\text{minimize}} \sum_{i=1}^{k} \mathrm{cut}(C_i, V - C_i) \tag{25}$$

Using **Eq 6**, it can be converted to an equivalent form:

$$\underset{\{C_1, C_2, \dots, C_k\}}{\text{minimize}} \sum_{i=1}^{k} \chi_{C_i}^{\mathrm{T}} L \chi_{C_i} \tag{26}$$

where χ_{C_i} is the characteristic vector of C_i, defined as:

$$\chi_{C_i}(v) = \begin{cases} 1 & v \in C_i \\ 0 & \text{Else} \end{cases} \tag{27}$$

A more compact form of the above objective is:

$$\underset{\{C_1, C_2, \dots, C_k\}}{\text{minimize}} \ \mathrm{Tr}\left[\chi^{\mathrm{T}} L \chi\right] \tag{28}$$

where $\chi = [\chi_{C_1}, \chi_{C_2}, \dots, \chi_{C_k}]$. This kind of combinatoric problem is shown to be NP-Hard by previous authors. Using standard spectral argument[37], we have one important observation that if the graph is diconnected and contains k connected components, the first k eigen vectors of Laplacian (count from smallest eigen value) are just the linear combination of the characteristic vectors of those connected components. In other words, in such ideal case, those eigen vectors are piecewise linear. Standard clustering algorithms in Euclidean space can easily give the right clustering. So one heuristic is to relax the problem to permit real values, i.e. substitute χ_{C_i} by v_i:

$$\underset{v_i \in \mathbb{R}^N, v_i^{\mathrm{T}} v_j = 0, V = (v_1, v_2, \ldots, v_N)}{\text{minimize}} \quad \mathrm{Tr}\left[V^{\mathrm{T}} L V\right] \tag{29}$$

The orthogonal condition $v_i^{\mathrm{T}} v_j = 0$ is inherited from the fact that characteristic vectors are orthogonal.

Now we find the problem of **Eq 29** is poorly-defined. Without further constraints, the choice $V = 0$ yield the minimum but it's obviously far from our real objective. Note that the unrelaxed version is well-defined, for the value of χ_{C_i} is constrained to $\{0, 1\}$. Certain "normalization" helps to well define our objective. In the following part of this section, we'll see how different normalization induce different objectives.

3.1.2 Ratio Cut

The first attempt of normalization is to constrain V to unit vectors. In this way, the following optimization problem can absorb proportional scaling of V and also prevent the trivial solution mentioned above:

$$\begin{aligned} \underset{v_i \in \mathbb{R}^N}{\text{minimize}} \quad & \sum_{i=1}^{k} v_i^{\mathrm{T}} L v_i \\ s.t. \quad & v_i^{\mathrm{T}} v_i = 1, \forall i = 1, 2, \ldots, k \\ & v_i^{\mathrm{T}} v_j = 0, \forall i \neq j \text{ and } i, j = 1, 2, \ldots, k \end{aligned} \tag{30}$$

This problem is equivalent to the following one:

$$\begin{aligned} \underset{v_i \in \mathbb{R}^N}{\text{minimize}} \quad & \sum_{i=1}^{k} \frac{v_i^{\mathrm{T}} L v_i}{v_i^{\mathrm{T}} v_i} \\ s.t. \quad & v_i^{\mathrm{T}} v_i = 1, \forall i = 1, 2, \ldots, k \\ & v_i^{\mathrm{T}} v_j = 0, \forall i \neq j \text{ and } i, j = 1, 2, \ldots, k \end{aligned} \tag{31}$$

Suppose we have a solution given by $\{v_i^*\}$. Then $\{t_i v_i^*\}, \forall t_i \in \mathbb{R}^N$ induce the same objective by linear algebraic argument. Then the problem can be tackled in two steps:

1. Solve:

$$\begin{aligned} \underset{v_i \in \mathbb{R}^N}{\text{minimize}} \quad & \sum_{i=1}^{k} \frac{v_i^{\mathrm{T}} L v_i}{v_i^{\mathrm{T}} v_i} \\ s.t. \quad & v_i^{\mathrm{T}} v_j = 0, \forall i \neq j \text{ and } i, j = 1, 2, \ldots, k \end{aligned} \tag{32}$$

2. Scale v_i so that $v_i^{\mathrm{T}} v_i = 1$.

The second step is easy, so we focus on the first step. By applying the result of Rayleigh Quotient repeatedly[25], it can be shown the solution is given by the k smallest eigenvectors of L.

Like the above section, **Eq 32** is a relaxed version of some problem. By reverting the relaxation process, we can find the combinatoric justification it corresponds to. Now we substitue orthogonal $\{v_i\}$ in **Eq 32** with characteristic vector $\{\chi_{C_i}\}$ to see its combinatoric version:

$$\begin{aligned} & \sum_{i=1}^{k} \frac{\chi_{C_i}^{\mathrm{T}} L \chi_{C_i}}{\chi_{C_i}^{\mathrm{T}} \chi_{C_i}} \\ = & \sum_{i=1}^{k} \frac{\sum_{(u,v) \in E} A_{uv} (\chi_{C_i}(v) - \chi_{C_i}(u))^2}{|C_i|} \\ = & \sum_{i=1}^{k} \frac{\sum_{(u,v) \in E, u \in C_i, v \in V - C_i} A_{uv}}{|C_i|} \\ = & \sum_{i=1}^{k} \frac{\mathrm{cut}(C_i, V - C_i)}{|C_i|} \end{aligned} \tag{33}$$

Note that the last line in **Eq 33** is right the definition of Ratio Cut(RCut), given a clustering $\{C_1, C_2, \ldots, C_k\}$. Comparing it to our first objective, Cut(**Eq 25**), we find that RatioCut takes cluster size into consideration. This is more reasonable if we consider the existence of outliers. In the extreme case, one outlier may have no connection with other points. The Cut objective will induce a singleton cluster for this outlier. However, RCut may tend to partition the graph into bigger clusters, which is more close to our goal.

3.1.3 Normalized Cut

Normalized Cut(NCut) is another widely studied combinatoric objective. Following the same approach as last section, we can derive its combinatoric version and relaxed version.

The definition of NCut is:

$$\mathrm{NCut} = \sum_{i=1}^{k} \frac{\mathrm{cut}(C_i, V - C_i)}{\mathrm{vol}(C_i)} \tag{34}$$

Its corresponding relaxed version is:

$$\begin{aligned} \underset{v_i \in \mathbb{R}^N}{\text{minimize}} \quad & \sum_{i=1}^{k} v_i^{\mathrm{T}} L v_i \\ s.t. \quad & v_i^{\mathrm{T}} R v_i = 1, \forall i = 1, 2, \ldots, k \\ & v_i^{\mathrm{T}} v_j = 0, \forall i \neq j \text{ and } i, j = 1, 2, \ldots, k \end{aligned} \tag{35}$$

where R is the degree matrix.

From the optimization perspective, the difference between RCut and NCut is that the normalization constraints are $v_i^{\mathrm{T}} I v_i = 1$ and $v_i^{\mathrm{T}} R v_i = 1$, respectively. From the combinatoric perspective, NCut takes the "importance" of vertices into consideration while RCut does not distinguish them. The "importance" here is expressed using degree of nodes. In **Section 3.1.4**, we'll see a generalzation.

3.1.4 Weighted Cut

The Cut defined in **Eq 25** can capture the linkage between clusters. Minimizing it should result in reasonable clustering. However, in real applications, vertices may be of different importance. If we denote the vertex weight by diagonal entries of W, the normalization constraint $v_i^{\mathrm{T}} W v_i = 1$ has a more general combinatoric correspondance:

$$\text{WCut} = \sum_{i=1}^{k} \frac{\text{cut}(C_i, V - C_i)}{W(C_i)} \tag{36}$$

where $W(C_i) = \sum_{v \in C_i} W_{vv}$.

By letting $W = I$ or $W = D$, it degrades to RCut and NCut, respectively.

3.1.5 Ratio/Normalized/Weighted Association

Recall the Cut series objectives try to minimize the inter cluster linkage. Likewise, one may think to maximize the intra cluster linkage, given by the association:

$$\text{assoc}(C_1) = \text{cut}(C_1, C_1) = \sum_{u,v \in C_1, (u,v) \in E} A_{uv} \tag{37}$$

The Ratio Association(RAssoc) is defined as:

$$\text{RAssoc} = \sum_{i} \frac{\text{assoc}(C_i)}{|C_i|} \tag{38}$$

We can derive its corresponding relaxed optimization:

$$\begin{aligned} \underset{v_i \in \mathbb{R}^N}{\text{minimize}} \quad & \sum_{i=1}^{k} v_i^{\mathrm{T}} A v_i \\ s.t. \quad & v_i^{\mathrm{T}} v_i = 1, \forall i = 1, 2, \ldots, k \\ & v_i^{\mathrm{T}} v_j = 0, \forall i \neq j \text{ and } i, j = 1, 2, \ldots, k \end{aligned} \tag{39}$$

The Normalized Association(NAssoc) is defined as:

$$\text{NAssoc} = \sum_i \frac{\text{assoc}(C_i)}{\text{vol}(C_i)} \tag{40}$$

We can derive its corresponding relaxed optimization:

$$\begin{aligned} \underset{v_i \in \mathbb{R}^N}{\text{minimize}} \quad & \sum_{i=1}^{k} v_i^{\mathrm{T}} A v_i \\ s.t. \quad & v_i^{\mathrm{T}} D v_i = 1, \forall i = 1, 2, \ldots, k \\ & v_i^{\mathrm{T}} v_j = 0, \forall i \neq j \text{ and } i, j = 1, 2, \ldots, k \end{aligned} \tag{41}$$

Of course, the Weighted Association(WAssoc) is defined as:

$$\text{WAssoc} = \sum_i \frac{\text{assoc}(C_i)}{W(C_i)} \tag{42}$$

We can derive its corresponding relaxed optimization:

$$\begin{aligned} \underset{v_i \in \mathbb{R}^N}{\text{minimize}} \quad & \sum_{i=1}^{k} v_i^{\mathrm{T}} A v_i \\ s.t. \quad & v_i^{\mathrm{T}} W v_i = 1, \forall i = 1, 2, \ldots, k \\ & v_i^{\mathrm{T}} v_j = 0, \forall i \neq j \text{ and } i, j = 1, 2, \ldots, k \end{aligned} \tag{43}$$

The difference between association series and cut series is that they use A and L, respectively. This can justify why decomposing adjacency and Laplacian are both reasonable to get good embedding. However, in our survey, the use of Laplacian is obviously dominant. There are other properties making Laplacian superior that can not be justified using the combinatoric framework in this section(see **Section 4.6** for example).

3.1.6 Conductance and Expansion

For the last part of combinatoric justification, we quick note two other criteria: ([25], W2)

$$\text{Expansion} = \sum_i \frac{\text{cut}(C_i, V - C_i)}{\min\{|C_i|, |V - C_i|\}} \tag{44}$$

$$\text{Conductance} = \sum_i \frac{\text{cut}(C_i, V - C_i)}{\min\{\text{vol}(C_i), \text{vol}(V - C_i)\}} \tag{45}$$

The notion of expasion and conductance is studied in many spectral graph theory problems. They have close analogy to RCut and NCut. However, due to the min operator, it becomes more difficult to establish their relaxed version. In our survey so far, there are no sophisticated approach to tackle with the two objectives.

3.2 Stochastic Justification

3.2.1 Random Walk

[27] [37] discussed the relationship between NCut and random walk on graphs. In this section, we generalize their discussion of two-cluster scenario to multi-cluster scenario.

In undirected graph, the stationary distribution is given by:

$$P\{v\} = \frac{d(v)}{\mathrm{vol}(V)} \tag{46}$$

The transition probability between vertices are given by the left normalized adjacency matrix, i.e.:

$$P\{u|v\} = (A_{\mathrm{rw}})_{vu} = \frac{(A)_{vu}}{d(v)} \tag{47}$$

Given two clusters $C_i, C_j \in V$, the probability a random walker starting from C_i and go to C_j is denoted by:

$$P\{C_j|C_i\} \tag{48}$$

Then the joint probability that a random walker start from C_i and escape C_i is given by:

$$\begin{aligned}
P\{V - C_i, C_i\} &= \sum_{v \in C_i, u \notin C_i} P\{u|v\}P\{v\} \\
&= \sum_{v \in C_i, u \notin C_i} \frac{(A)_{vu}}{d(v)} \frac{d(v)}{\mathrm{vol}(V)} \\
&= \frac{1}{\mathrm{vol}(V)} \sum_{v \in C_i, u \notin C_i} (A)_{vu} \\
&= \frac{1}{\mathrm{vol}(V)} \mathrm{cut}(C_i, V - C_i)
\end{aligned} \tag{49}$$

The probability that a random walker escape C_i conditioned on the event it starts from C_i is:

$$\begin{aligned} P\{V - C_i | C_i\} &= \frac{P\{V - C_i, C_i\}}{P\{C_i\}} \\ &= \frac{\frac{1}{\mathrm{vol}(V)}\mathrm{cut}(C_i, V - C_i)}{\frac{\mathrm{vol}(C_i)}{\mathrm{vol}(V)}} \\ &= \frac{\mathrm{cut}(C_i, V - C_i)}{\mathrm{vol}(C_i)} \end{aligned} \tag{50}$$

Summing over all clusters:

$$\sum_{i=1}^{k} P\{V - C_i | C_i\} = \sum_{i=1}^{k} \frac{\mathrm{cut}(C_i, V - C_i)}{\mathrm{vol}(C_i)} = \mathrm{NCut} \tag{51}$$

The minimization of NCut can also be interpreted as minimizing the conditional probability that a random walker starts from a certain cluster and esapces it.

3.2.2 Commute Time

The relationship betwen pseudo inverse of graph Laplacian and commute time was observed by many authors. For detailed discussion, readers can also refer to Luxburg's tutorial[37]. In this section, we organize a short discussion from the electric network's perspective and relate it to MDS(**Section 4.1**).

An electric network satisfies Ohm's Law and Kirchhoff's Law (KCL, KVL [42]). Using standard electric network arguments, we have:

$$L\phi = i_{\mathrm{ext}} \tag{52}$$

where L is the Laplacian of conductance matrix A, ϕ is the voltage vector, and i_{ext} is external current we input to each vertex. Note that $\vec{1}^{\mathrm{T}} i_{\mathrm{ext}} = 0$, or the electric system has no solution. The effective resistance between s, t is the voltage difference between the two vertices when one unit of current is injected to s and extracted from t. Using characteristic vectors, we can express the following quantities:

$$\begin{aligned} L\phi &= \chi_s - \chi_t & (53) \\ \mathrm{Reff}(s,t) &= \phi(s) - \phi(t) \\ &= (\chi_s - \chi_t)^{\mathrm{T}}\phi \\ &= (\chi_s - \chi_t)^{\mathrm{T}} L^{+} (\chi_s - \chi_t) & (54) \end{aligned}$$

where L^{+} is Moore-Penrose Inverse[43] of L. It can be shown ([25], W12) that:

$$\mathrm{Comm}(s,t) = 2M\mathrm{Reff}(s,t) = 2M(\chi_s - \chi_t)^{\mathrm{T}} L^{+} (\chi_s - \chi_t) \tag{55}$$

One important observation is that the commute time encoded in L^+ can be viewed as an Euclidean distance. Thus it is intuitive to perform a distance preserving embedding using L^+ as pairwise distance. Suppose $L^+ = U\Lambda^+U^{\mathrm{T}}$, the embedding is given by first several columns of $U(\Lambda^+)^{\frac{1}{2}}$.

Luxburg relate the commute time embedding with unnormalized Laplacian case ([37] ch6). That justification is not very strong. Instead, we loot at the embedding of $U(\Lambda^+)^{\frac{1}{2}}$ from the following two angles:

- In our discussion of enhancements (**Section 2.1.3**), we proposed to pass effective resistance matrix to the next SE stage. Operation wise, it's different from the embedding discussed in this section. As to the outcome, they are the same.
- In **Section 2.2.5**, we proposed the general scaling of eigen vectors in post-processing stage (**Eq 21**). Plugging in the specialized version $f(\Lambda) = (\Lambda^+)^{\frac{1}{2}}$, we see that the commute time embedding also fits into our framework (**Section 2**). The discussion in this section provides a justification of the choice of f. The reason for square root of eigen value is the same as that of MDS (**Section 4.1**).

3.3 Low Rank Approximation

In the work [10], Brand associates a kernel view with adjacency matrix (affinity matrix). Note that in our framework (**Section 2**), after metric formulation, we look on all resulting matrix as adjacency matrix regardless of their origin. Note that the justification in this section only works for those adjacency matrices that can be interpreted as kernels (at least "close" to PSD).

Let $X = [x_1, x_2, \ldots, x_N]$ be data points and $\Phi = [\phi(x_1), \phi(x_2), \ldots, \phi(x_N)]$ be their image in feature space. The mapping $\phi(.)$ can be anything, e.g. $\phi(x) = x$ as identity. If the graph adjacency matrix is a kernel, it should have this structure:

$$A = \Phi^{\mathrm{T}}\Phi \tag{56}$$

Thus an EVD can recover the coordinates of data points in feature space, i.e. $\phi(x_i)$.

Now we have two problems:

- Suppose the feature space is d-dimensional. The rank of A is then $\min\{d, N\}$, namely we have at most $\min\{d, N\}$ non-zero eigen values. If $d > N$, we can not recover all the coordinates.
- Even if we can recover all coordinates, it may not always be what we want. For example, many high-dimensional data is poisoned. A large portion of those dimensions may be pure noise. Recovering them brings no benifit for later clustering stage.

The two problems motivate us to seek for approximate recovery rather than exact recovery, which leads to the following optimization:

$$\underset{Y \in \mathbb{R}^{N \times \hat{d}}}{\text{minimize}} \qquad ||A - YY^{\mathrm{T}}||_F^2 \tag{57}$$

where $||.||_F$ denotes the Frobenius norm[44], and $\hat{d}$ is the dimensionality we want to approximate or inferred from prior knowledge. This is the standard low rank approximation. If our adjacency matrix can be decomposed as $A = U\Lambda U^{\mathrm{T}}$, the solution is given by:

$$Y = U_{\hat{d}}(\Lambda_{\hat{d}})^{\frac{1}{2}} \tag{58}$$

where $U_{\hat{d}}$ and $\Lambda_{\hat{d}}$ denote first $\hat{d}$ columns of corresponding matrix.

In our framework, the algorithm is described as: use adjacency matrix; scale eigen vectors by square root eigen value. With the assumption that A is a kernel matrix, this algorithm should give a reasonable embedding.

Note that this algorithm simply do SE, and can be justified to yield good layout of points in lower-dimensional Euclidean space. However, it does not directly show any clue that there will be easy-to-separate clusters after this embedding. Our explanation is that, although SE is a subprocedure of SC, the objective of SE is stronger at most time. If the layout given by a certain SE is reasonable but the embedded version is still hard to cluster, we should argue that very probably there are no good natural clustering.

3.4 Density Estimation View

In the survey, we found one interesting interpretation of SC given by Chen([11], ch2.4). They first view clustering, embedding, and dimensionality reduction all as density estimation problem. For example, clustering algorithms aim at finding K centers to represent data points. Those centroids can be regarded as very dense blobs if we adopt the density estimation view.

They choose Gaussian kernel as Kernel Density Estimator (KDE). It can be derived that, with proper weight assignments, the NCut algorithm can output the partition with which the Bayes error is minimized. We refer interested readers to their work and omit detailed discussion here.

3.5 Matrix Perturbation

I'm not familiar with matrix perturbation theory. This section here is to make completeness of our discussion. Interested readers can refer to Luxburg's tutorial [37]. In Ng's work [28], there is a more tentative discussion. Careful perturbation bound is derived from 4 assumptions of graph properties.

The main idea is that given a small perturb matrix T, the eigen vector of A and $(A + T)$ does not differ too much. When the graph exhbits ideal clustering, eigen vectors of both (normalized) adjacency[28] matrix and (normalized) Laplacian[37] matrix appear to be characteristic vectors (subject to orthogonal transformation). When the graph is not very far from the ideal case, using the matrix perturbation argument, those vectors are not very far from characteristic vectors. Thus standard Euclidean clustering algorithm like K-means should be able to detect the clustering correctly.

3.6 Polarization

In terms of operation, Ng's algorithm [28] and Brand's algorithm [10] are similar. The former operates on symmetric normalized adjacency matrix and the latter operates on adjacency matrix. What's more, they both engage a projection onto unit sphere in the post-processing stage(**Section 2.2.5** in our framework).

The justification of Brand comes from the polarization theorem. We present the intuition here and refer interested readers to [10] for more information. Suppose now the affinity matrix is a kernel matrix, **Eq 58** gives an embedding justified by low-rank approximation. With the decrease of $\hat{d}$, the angles between points having high affinity decrease, and the angles between points having low affinity increase. Thus the clustering structure should be more and more clear.

4 Other Spectral Embedding Technique

The origin of this article is spectral clustering, especially the spectral graph theory type of work. In this section, we briefly present some relevant spectral embedding techniques from machine learning community and in **Section 5** we give several unifying views those all of those algorithms.

4.1 MDS

The term Multi-Dimensional Scaling(MDS) [13] is actually for a set of algorithms. Readers can refer to [45] for quick information on taxonomy and [9] for detailed texts.

The most relevant one and the simplest form is classical MDS ([9] ch12). Given a pairwise distance matrix, MDS recovers an embedding in low dimensional Euclidean space, which preserves the given pairwise distance. Involved techniques are already discussed separately in **Section 2.1.2** and **Section 3.3**. In this section, we assemble them to give the algorithm(**Alg 3**).

4.2 isomap

Bearing in mind that MDS performs distance preserving embedding, the description of isomap [35] is rather simple: isomap preserves geodesic dis-

Algorithm 3 Multi-Dimensional Scaling

```
Input: Pairwise distance matrix $D_{N \times N}$;
    Dimensionality of embedding space $\hat{d}$
Output: Embedding $Y_{N \times \hat{d}}$
$D^{(2)} \leftarrow$ element wise square of $D$
$J = I - \frac{1}{n}\vec{1}\vec{1}^{\mathrm{T}}$
$A = -\frac{1}{2}JD^{(2)}J$
EVD $A = U\Lambda U^{\mathrm{T}}$ {Descending Eigenvlue}
$Y = U_{\hat{d}}(\Lambda_{\hat{d}})^{\frac{1}{2}}$
```

tance. Suppose we have a manifold in high dimensional space. The geodesic distance between two nodes is the shortest path distance between the nodes along the manifold. See **Alg 4** for the algorithm.

Algorithm 4 isomap

```
Input: High dimensional data $X = [x_1, x_2, \ldots, x_N]$;
    Dimensionality of embedding space $\hat{d}$
Output: Embedding $Y_{N \times \hat{d}}$
Construct adjacency graph $G =< V, E >$. {kNN, $\epsilon$-ball, etc. Section 2.1.1 }
$D_{ij} = \begin{cases} ||x_i - x_j|| & (i,j) \in E \\ \infty & (i,j) \notin E \end{cases}$
$D_G = \text{Floyd}(D)$
$Y = \text{MDS}(D_G, \hat{d})$
```

In **Alg 4**, ”Floyd” denotes the Floyd-Warshall algorithm [46] for shortest path. Tenenbaum also suggests other algorithms for all-pair shortest path which can utilize the sparse structure of graph G[35].

The justification for the notion of geodesic distance preserving comes from two assumptions:[15]

- Global isometry. That means the geodesic distance in observation space X is equal to the Euclidean distance in parametric space Y.
- Convexity. The set of possible configurations in parametric space is convex.

4.3 PCA

Principal Component Analysis(PCA)[8] has a very long history in DR. There are many interpretations of PCA, including subspace learning, maximum variance preserving, minimum projection error and probabilistic formulation [36]. In this section, we present the minimum projection error view of PCA and show one useful equivalent form.

Suppose we have a set of zero-mean data points $X = [x_1, x_2, \dots, x_N]$. We want to project them into a $\hat{d}$-dimensional subspace spanned by an orthonormal set $\{u_1, u_2, \dots, u_{\hat{d}}\}$. We want to find $U = (u_1, u_2, \dots, u_{\hat{d}})$ that results in minimum distortion:

$$\begin{aligned} \underset{U \in \mathbb{R}^{n \times \hat{d}}}{\text{minimize}} \quad & J(U) = \sum_{i=1}^{N} ||UU^{\mathrm{T}}x_i - x_i||^2 \\ s.t. \quad & U^{\mathrm{T}}U = I \end{aligned} \tag{59}$$

The objective can be transformed in the following way:

$$\begin{aligned} J(U) &= \sum_{i=1}^{N} \mathrm{Tr}\left[(UU^{\mathrm{T}}x_i - x_i)^{\mathrm{T}}(UU^{\mathrm{T}}x_i - x_i)\right] && (60) \\ &= \sum_{i=1}^{N} \mathrm{Tr}\left[x_i^{\mathrm{T}}(UU^{\mathrm{T}} - I)^{\mathrm{T}}(UU^{\mathrm{T}} - I)x_i\right] && (61) \\ &= \sum_{i=1}^{N} \mathrm{Tr}\left[x_i x_i^{\mathrm{T}}(UU^{\mathrm{T}} - I)^{\mathrm{T}}(UU^{\mathrm{T}} - I)\right] && (62) \\ &= \mathrm{Tr}\left[\sum_{i=1}^{N}(x_i x_i^{\mathrm{T}})(UU^{\mathrm{T}} - I)^{\mathrm{T}}(UU^{\mathrm{T}} - I)\right] && (63) \\ &= \mathrm{Tr}\left[XX^{\mathrm{T}}(UU^{\mathrm{T}} - I)^{\mathrm{T}}(UU^{\mathrm{T}} - I)\right] && (64) \\ &= \mathrm{Tr}\left[XX^{\mathrm{T}}(UU^{\mathrm{T}}UU^{\mathrm{T}} - 2UU^{\mathrm{T}} + I)\right] && (65) \\ &= -\mathrm{Tr}\left[U^{\mathrm{T}}XX^{\mathrm{T}}U\right] + \mathrm{Tr}\left[XX^{\mathrm{T}}\right] && (66) \end{aligned}$$

The second term is constant to U, so the optimization problem can be transformed to:

$$\begin{aligned} \underset{U \in \mathbb{R}^{n \times \hat{d}}}{\text{maximize}} \quad & \mathrm{Tr}\left[U^{\mathrm{T}}(XX^{\mathrm{T}})U\right] \\ s.t. \quad & U^{\mathrm{T}}U = I \end{aligned} \tag{67}$$

We have seen this form for many times in **Section 3.1**. The solution is given by the first (descending order of eigen values) $\hat{d}$ eigen vectors of XX^{T}.

Note that U is only the subspace basis. We compute the embedding (coordinates in U's system) like:

$$Y = (U^{\mathrm{T}}X)^{\mathrm{T}} = X^{\mathrm{T}}U \tag{68}$$

where we stick to the convention that embedding coordinates are given by rows of Y. From the analysis of PCA, we reach the EVD of $(XX^{\mathrm{T}})U = U\Lambda$ and obtain the embedding by one more projection. On the contrary, in most SE algorithms, we decompose $X^{\mathrm{T}}X$. The relationship between the two matrices are shown below:

$$\begin{aligned} X^{\mathrm{T}}XY &= X^{\mathrm{T}}XX^{\mathrm{T}}U \\ &= X^{\mathrm{T}}U\Lambda \\ &= Y\Lambda \end{aligned} \tag{69}$$

That is to say, columns of Y are eigen vectors of $X^{\mathrm{T}}X$ with corresponding eigen values Λ. We conclude the two operationally different versions of PCA in **Alg 5** and **Alg 6**.

Algorithm 5 Principal Component Analysis (Covariance Version)

```
Input: High dimensional data $X = [x_1, x_2, \ldots, x_N]$;
    Dimensionality of embedding space $\hat{d}$
Output: Embedding $Y_{N\times\hat{d}}$
1: EVD of covariance matrix: $XX^{\mathrm{T}} = U\Lambda U^{\mathrm{T}}$ {Descending Eigenvlue}
2: $Y = X^{\mathrm{T}}U_{\hat{d}}$
```

Algorithm 6 Principal Component Analysis (Inner Product Version)

```
Input: High dimensional data $X = [x_1, x_2, \ldots, x_N]$;
    Dimensionality of embedding space $\hat{d}$
Output: Embedding $Y_{N\times\hat{d}}$
1: EVD of covariance matrix: $X^{\mathrm{T}}X = U\Lambda U^{\mathrm{T}}$ {Descending Eigenvlue}
2: $Y = U_{\hat{d}}$
```

There are two points to note:

- $X^{\mathrm{T}}X$ and XX^{T} are of sizes $N \times N$ and $n \times n$, respectively. Since the EVD is a computationally intensive stage, we can choose the version which results in smaller matrices.

- The same observation can lead to kernel PCA. For probably infinite dimensional feature space, computing XX^{T} is impossible but $X^{\mathrm{T}}X$ is tractable. $X^{\mathrm{T}}X$ can be formed through inner product, or by specifying an explicit kernel function because $(X^{\mathrm{T}}X)_{ij} = k(x_i, x_j)$.

4.4 Kernel PCA

In [32], Schölkopf proposed Kernel PCA(KPCA) and we will see in **Section 5.2** the kernel framework is so general that it can cover almost all the SET.

Let $X_{n\times N} = [x_1, x_2, \ldots, x_N]$ be data points; $\Phi_{d\times N} = [\phi(x_1), \phi(x_2), \ldots, \phi(x_N)]$ be their image in the feature space. Performing covariance version algorithm(**Alg 5**) in feature space may be intractable due to probably very large d. Thus we want to avoid the explicit construction of covariance matrix ($\Phi\Phi^{\mathrm{T}}$). The kernel(Gram matrix) is defind as:

$$K = \Phi^{\mathrm{T}}\Phi \tag{70}$$

Using the argument in **Section 4.3**, we can invoke the inner product version PCA on K(**Alg 6**). When the kernel is specified as kernel function, this leads to the KPCA algorithm, where inner product is not performed explicitly. We show the KPCA algorithm in **Alg 7**.

Algorithm 7 Kernel PCA

```
Input: High dimensional data $X = [x_1, x_2, \ldots, x_N]$;
    Kernel function: $k(x_i, x_j)$
    Dimensionality of embedding space $\hat{d}$
Output: Embedding $Y_{N\times\hat{d}}$
1: Construct Gram matrix: $(G)_{ij} = k(x_i, x_j)$
2: EVD of covariance matrix: $G = U\Lambda U^{\mathrm{T}}$ {Descending Eigenvlue}
3: $Y = U_{\hat{d}}$
```

In this article, we concern more about in-sample embedding, so our way to present KPCA looks quite simple if readers have seen related articles before. For out-of-sample embedding, readers can refer to [32] and [7].

4.5 LLE

Locally Linear Embedding[29] is a neighbourhood preserving embedding. It first computes the neighbourhood (originally kNN, but other techniques like ϵ-ball should also be available) of each vertex. Then, reconstruction weights from the neighbourhoods are computed. Last, lower-dimensional embedding tries to preserve the reconstruction weights.

The first step is easy, and we have discussed a lot about the construction of adjacency graph in **Section 2.1**. Denote the neighbourhood of vertex i by $N(i)$, the reconstruction weights of i, $W_{(i,:)}$, can be obtained through the following quadratic programming:

$$\underset{W_{i,j}, j\in N(i)}{\text{minimize}} \quad ||x_i - \sum_{j\in N(i)} W_{ij}x_j||^2 \tag{71}$$

$$s.t. \quad \sum_{j\in N(i)} W_{ij} = 1 \tag{72}$$

and $W_{ij} = 0$ if $j \notin N(i)$. The objective can again be transformed into a trace minimization form and the constraint can be tackled by standard Lagrangian multiplier. It yields the following closed form solution:

$$W_{N(i)} = \frac{\vec{1}^{\mathrm{T}}G_i^{-1}}{\vec{1}^{\mathrm{T}}G_i^{-1}\vec{1}} \tag{73}$$

where $W_{N(i)}$ is a row vector denoting the $j \in N(i)$ elements of W_{ij} and G_i is called a local Gram matrix defined as:

$$\begin{aligned} N(i) &= \{j_1, j_2, \dots j_{|N(i)|}\} \tag{74} \\ X_{N(i)} &= [(x_i - x_{j_1}), (x_i - x_{j_2}), \dots, (x_i - x_{j_{|N(i)|}})] \tag{75} \\ G_i &= X_{N(i)}^{\mathrm{T}} X_{N(i)} \tag{76} \end{aligned}$$

For the last step, the embedding $Y_{N \times \hat{d}}$ can be obtained through an EVD problem:

$$\begin{aligned} J(Y) &= \sum_i ||Y_{(i,:)}^{\mathrm{T}} - W_{(i,:)} Y_{(i,:)}^{\mathrm{T}}||^2 \tag{77} \\ &= \mathrm{Tr}\left[Y^{\mathrm{T}}(I - W^{\mathrm{T}})(I - W)Y\right] \tag{78} \end{aligned}$$

This is again our familiar form. To absorb scaling and rotational invariance, the embedding is further constrained to have uint covariance, i.e. $Y^{\mathrm{T}}Y = I$. The final result is obtained through EVD of $(I - W^{\mathrm{T}})(I - W)$.

We present the operation in **Alg 8**. Interested readers can consult [31] for more details.

Algorithm 8 Locally Linear Embedding

```
Input: High dimensional data $X = [x_1, x_2, \dots, x_N]$;
    Dimensionality of embedding space $\hat{d}$
Output: Embedding $Y_{N \times \hat{d}}$
Construct adjacency graph $G =< V, E >$. {kNN, $\epsilon$-ball, etc. Section 2.1.1 }
for all $i = 1, 2, \dots N$ do
    $N(i) = \{j|(i,j) \in E\} = \{j_1, j_2, \dots j_{|N(i)|}\}$
    $X_{N(i)} = [(x_i - x_{j_1}), (x_i - x_{j_2}), \dots, (x_i - x_{j_{|N(i)|}})]$
    $G_i = X_{N(i)}^{\mathrm{T}} X_{N(i)}$
    $W_{N(i)} = \frac{\vec{1}^{\mathrm{T}} G_i^{-1}}{\vec{1}^{\mathrm{T}} G_i^{-1} \vec{1}}$
    $W_{ij} = 0, \forall j \notin N(i)$
end for
EVD: $(I - W^{\mathrm{T}})(I - W) = U \Lambda U^{\mathrm{T}}$ {Ascending Eigenvlue}
$Y = U_{\hat{d}}$
```

4.6 Laplacian Eigenmap

By first glance, one may find that Laplacian eigenmap(LEmap)[4] is the same as the widely recognized spectral clustering (very close to the three algorithms in [37]). However, this algorithm is derived from a different objective.

The SE stage in SC is to make the points more separable in a lower-dimensional Euclidean distance. As to general purpose embedding algorithms, or DR methods, the objective is to find a configuration that best describe the data. This informal objective can be realized in many ways. The one LEmap adopted is the weighted distortion:

$$\text{Distortion} = \sum_{(i,j)\in E} ||Y_{(i,:)} - Y_{(j,:)}||^2 A_{ij} \tag{79}$$

where $Y_{(i,:)}$ denotes the i-th row of embedding matrix. The intuition is, when two vertices are connected, we want their embedding in lower dimensional space lie close. The higher their similarity(A_{ij}) is, the more penalty we get if we put them far apart. This objective is equivalent to

$$\begin{aligned}
\text{Distortion} &= \sum_{(i,j)\in E} \sum_{l=1}^{\hat{d}} (Y_{(i,l)} - Y_{(j,l)})^2 A_{ij} && (80)\\
&= \sum_{l=1} \sum_{(i,j)\in E} (Y_{(i,l)} - Y_{(j,l)})^2 A_{ij} && (81)\\
&= \sum_{l=1} Y^{\mathrm{T}}_{(:,l)} L Y_{(:,l)} && (82)\\
&= \mathrm{Tr}\left[Y^{\mathrm{T}} L Y\right] && (83)
\end{aligned}$$

To this end, the objective becomes the same as **Eq 29**. We know that this objective is poorly defined. Since the whole **Section 3.1** is devoted to discussing the normalization constraints, we use the results directly. **Section 3.1** showed we can impose a general weight matrix W as normalization constraints. In Belkin's work[4], $W = R$(degree matrix) is selected. We conclude the algorithm in **Alg 9**.

Algorithm 9 Laplacian Eigenmap

```
Input: High dimensional data $X = [x_1, x_2, \dots, x_N]$;
    Dimensionality of embedding space $\hat{d}$
Output: Embedding $Y_{N\times\hat{d}}$
  1: Construct (weighted) similarity graph $G =< V, E >$. {Section 2.1.1 }
  2: Obtain left normalized graph Laplacian $L_{\mathrm{rw}}$.
  3: EVD: $L_{\mathrm{rw}} U = U\Lambda$ {Ascending Eigenvlue}
  4: $Y = U_{\hat{d}}$
```

4.7 Hessian Eigenmap

Hessian Eigenmap, also called Hessian Locally Linear Embedding [15], is shown to have theoretical advantages in embedding. The two assumptions of isomap (**Section 4.2**) are relaxed to:

- Local isometry. In isomap, global isometry is assumed and geodesic distance is in favour. In Donoho's study, many image examples are shown to exihbit isometry[15]. They keep the isometry assumption but relax it to a local version.
- Connectedness. In isomap, the assumption that parametric set is convex does not always hold. Non-convexity can arise naturally for compound shapes. Missing samples in observation space is also a problem.

5 Unifying Views

In this section, we provide unifying views of SC / SE / DR from several angles. Due to their close relationship, we already term all by SET in this artile.

5.1 Graph Framework

Graph framework is obvious and discussed throughout this article. In **Section 2**, we provided a detailed discussion of SE. We briefly review the three stages:

1. Metric Formulation. This stage converts raw input to a (weighted) graph adjacency matrix.
2. Spectral Embedding. A certain transformed version of adjacency matrix is eigen decomposed. Embedding coordinates are obtained from (possibly eigen value scaled) eigen vectors.
3. Clustering. Hard cut algorithms are invoked to partition points in the embedded space into clusters.

For those algorithms presented in **Section 4**, they all have the EVD procedure, which can be regarded as the Spectral Embedding stage in our graph framework. Processes before EVD (no matter how complex or simple they are) are regarded as forming the (weighted) adjacency matrix. The weighting method is not specified and can thus cover all those methods above.

5.2 Kernel Framework

Through the discussion of KPCA(**Section 4.4**), we find that the kernel function is as general as "graph" discussed in **Section 5.1**. Then it's possible to create output-equivalent KPCA for other SET.

One thing to note is that in KPCA the kernel is Positive SemiDefinite(PSD) while the adjacency matrix of graphs do not necessarily be PSD.

So using KPCA framework to perform other SET may be simply a notational generalization. This observation has been made by several authors, for example [6][5][17]. Allowing an arbitrary kernel function $k(x_i, x_j)$, we can specify many algorithms above in terms of kernel. **Tbl 2** shows some examples.

Table 2: Kernel Example for Several Methods

Method	Kernel $k(x_i, x_j)$
MDS[9]	$-\frac{1}{2}(I - \frac{1}{n}\vec{1}\vec{1}^{\mathrm{T}})D^{(2)}(I - \frac{1}{n}\vec{1}\vec{1}^{\mathrm{T}})$
isomap[35]	$-\frac{1}{2}(I - \frac{1}{n}\vec{1}\vec{1}^{\mathrm{T}})D_G^{(2)}(I - \frac{1}{n}\vec{1}\vec{1}^{\mathrm{T}})$
Ng's SC[28]	A_{sym}
LEmap[4]	$\sigma I - L_{\mathrm{sym}}$ (not L_{rw} !)
LLE[29]	$\sigma I - (I - W^{\mathrm{T}})(I - W)$

There are some points worth to note:

- σI in **Tbl 2** has a similar form with diagonal shifting discussed in **Section 2.2.1**. The reason here is slightly different and much simpler: the KPCA framework solves for principal eigenvectors, so for those algorithms who are interested in smallest eigenvalues we negate the matrix to achieve the goal.

- The discussion of diagonal shifting in **Section 2.2.1** reveals the fact that any symmetric matrix can be converted to a valid kernel without influence the eigen vectors. So even if we stick to the notion that a kernel should be PSD, it's also possible to conclude those algorithms in terms of an output-equivalent KPCA.

- LEmap in **Tbl 2** uses L_{sym} rather than L_{rw}, which differs from our discussion. The reason is that L_{sym} is symmetric. With the relationship developed in **Section 2.2.2**, we can obtain L_{rw}'s embedding from the output of KPCA using L_{sym} as a kernel.

- As is shown in [5], using kernels in **Tbl 2** is not enough to minic those algorithms thoroughly. For example, after solving KPCA with the MDS kernel, we need to scale the eigen vectors by square root eigen values. In this case, KPCA framework is weaker than our framework(**Section 2**). In our framework, post-processing is formalized in the Spectral Embedding stage(**Section 2.2**).

5.3 Trace Maximization

The graph framework and kernel framework provide operational views of SET. In this section, we discuss their fundamental causes.

In **Section 2**, **Section 3** and **Section 4**, we are frequently encountered with an optimization problem:

$$\begin{aligned} \underset{U}{\text{maximize}} \quad & \text{Tr}\left[U^{\mathrm{T}}MU\right] \\ s.t. \quad & U^{\mathrm{T}}WU = I \end{aligned} \tag{84}$$

This is known as trace maximization problem in many literatures. The EVD based solution roots from this type of optimization. The orgins are different:

- Relaxed combinatoric problem. **Section 3.1** presents several combinatoric objectives. Relaxing those objectives result in the trace maximization problem. This batch of algorithms tend to preserve topology. The matrix to be eigen decomposed is usually a sparse one (the neighbourhood graph is usually sparse).
- Distortion. Most algorithms in **Section 4** define certain kind of distortion measure using $||.||^2$. The summation over squared Euclidean distance can be casted into the trace form algebraically. This batch of algorithms tends to preserve distance and induce dense matrix (e.g. all pair geodesic distance in isomap).

This is not a rigid categorization of algorithms. For example, in LLE, the second step computing neighbourhood reconstruction weights tries to preserve topology and results in a sparse matrix. The third step minimizes reconstruction distortion and decomposes $(I - W^{\mathrm{T}})(I - W)$, which is not sparse in general. Thus LLE exihbits both fashions.

The observation in this section is that, if other non clustering / non embedding / non DR problem shows a core procedure of trace maximization, we can try to leverage the sophisticated SETs to solve it. Or vice versa, solve clustering / embedding / DR by solving the new problem. Next section is right an example.

5.4 Kernel Clustering

This section stems from the work of Dhillon[14], who had an interesting observation of Kernel K-Means(KKM) and spectral clustering. However, the logic in the original work has some pitfalls. We will derive the relationship in a coherent way with this article, point out the pitfalls and argue when this kind of correspondance holds.

The motivation comes in two folds:

- We have three stages in SC framework. The first stage can be abstracted by finding a kernel(**Section 5.2**). In the last hard cut stage, traditional choice is K-Means(KM). Will it be possible to directly invoke KKM?

- The computation of EVD is time consuming. For a full decomposition, it has $O(N^3)$ complexity. For partial decomposition, it has $O(\hat{d}N^2)$ complexity. However, one iteration of KM or KKM is only $O(NK)$. KKM will have computational advantage when number of samples N is very large. (of course, on the condition that they yield the approximately the same result)

In **Section 3.1**, we see the Cut minimization objective is equivalent to the following discrete trace minimization objective:

$$\underset{\{C_1,C_2,\ldots,C_k\}}{\text{minimize}} \ \text{Tr}\left[\chi^{\mathrm{T}} L \chi\right] \tag{85}$$

where $\chi = [\chi_{C_1}, \chi_{C_2}, \ldots, \chi_{C_k}]$, denoting the characteristic vectors. The association maximization is equivalent to the following discrete trace maximization objective:

$$\underset{\{C_1,C_2,\ldots,C_k\}}{\text{maximize}} \ \text{Tr}\left[\chi^{\mathrm{T}} A \chi\right] \tag{86}$$

The term "discrete" means that χ can only take a set of discrete values(e.g. $0, 1$-valued). It becomes trace minimization problem after we relax χ to take real value.

The objective of clustering is defined as:

$$\begin{aligned} \underset{\{C_1,C_2,\ldots,C_k\}}{\text{minimize}} \quad & J(\{C_i\}) = \sum_{i=1}^{k} \sum_{j \in C_i} ||x_j - m_i||^2 && (87) \\ \text{where} \quad & m_i = \arg\min_{x} \sum_{j \in C_i} ||x_j - x||^2 && (88) \end{aligned}$$

where m_i is the cluster centroid and it can be shown that $m_i = \frac{1}{|C_i|}\sum_{j \in C_i} x_j$. Now we want to perform a clustering in a feature space, $\phi(x_j)$. Following similar approach, the objective can be written as:

$$\begin{aligned} \underset{\{C_1,C_2,\ldots,C_k\}}{\text{minimize}} \quad & J(\{C_i\}) = \sum_{i=1}^{k} \sum_{j \in C_i} ||\phi(x_j) - m_i||^2 && (89) \\ \text{where} \quad m_i \quad & = \arg\min_{\phi(x)} \sum_{j \in C_i} ||\phi(x_j) - \phi(x)||^2 \\ & = \frac{1}{|C_i|} \sum_{j \in C_i} \phi(x_j) && (90) \end{aligned}$$

Denote $\Phi = [\phi(x_1), \phi(x_2), \ldots, \phi(x_N)]$ and $D^{(2)}$ be the squared distance matrix in feature space. From the analysis in **Section 2.1.2**, we know:

$$D^{(2)} = c\vec{1}^{\mathrm{T}} + \vec{1}c^{\mathrm{T}} - 2\Phi^{\mathrm{T}}\Phi \tag{91}$$

where c is a column vector with $\phi(x_i)^{\mathrm{T}}\phi(x_i)$ being the entries. Using the equality $\sum_{j \in C_i} ||\phi(x_j) - m_i||^2 = 0.5 \sum_{j,l \in C_i} ||\phi(x_j) - \phi(x_l)||^2$, we can transform the objective as:

$$\begin{aligned} J(\{C_i\}) &= \sum_{i=1}^{k} 0.5 \sum_{j,l \in C_i} ||\phi(x_j) - \phi(x_l)||^2 && (92) \\ &= 0.5 \sum_{i=1}^{k} \chi_{C_i}^{\mathrm{T}} D^{(2)} \chi_{C_i} && (93) \\ &= 0.5 \mathrm{Tr}\left[\chi^{\mathrm{T}} D^{(2)} \chi\right] && (94) \end{aligned}$$

Comparing with **Eq 85** and **Eq 86**, we obtained a similar form. However, elements of $D^{(2)}$ represents squared distance and are thus all non-negative but L in **Eq 85** has non-positive off-diagonals. We can not construct the a clustering problem corresponding to **Eq 85**. Now consider the association maximization(**Eq 86**), but our current objective of uses minimization. We need to transform $J(\{C_i\})$ again to obtain a maximization version:

$$\begin{aligned} J(\{C_i\}) &= 0.5 \mathrm{Tr}\left[\chi^{\mathrm{T}}(c\vec{1}^{\mathrm{T}} + \vec{1}c^{\mathrm{T}} - 2\Phi^{\mathrm{T}}\Phi)\chi\right] && (95) \\ &= \mathrm{Tr}\left[\chi^{\mathrm{T}} c\vec{1}^{\mathrm{T}} \chi\right] - \mathrm{Tr}\left[\chi^{\mathrm{T}}\Phi^{\mathrm{T}}\Phi\chi\right] && (96) \end{aligned}$$

This form looks interesting. If the first term is (nearly) constant to χ, we can focus on maximizing the second term. What's more, $\Phi^{\mathrm{T}}\Phi$ is a Gram matrix. If A is also a Gram matrix, the correspondance is clear.

The problem is now to determine under what condition $\mathrm{Tr}\left[\chi^{\mathrm{T}} c\vec{1}^{\mathrm{T}} \chi\right]$ can be treated as constant. In [14], they assign

$$\chi_{C_i}(v) = \begin{cases} \frac{1}{\sqrt{|C_i|}} & v \in C_i \\ 0 & \text{else} \end{cases} \quad (97)$$

and it can be shown:

$$\mathrm{Tr}\left[\chi^{\mathrm{T}} c\vec{1}^{\mathrm{T}} \chi\right] = \sum_{i=1}^{k} \sum_{j \in C_i} \frac{c_j |C_i|}{\sqrt{|C_i|}\sqrt{|C_i|}} = \sum_{i=1}^{k} \sum_{j \in C_i} c_j = \sum_{j=1}^{N} c_j \quad (98)$$

namely, a constant! However, the second term takes $\{0,1\}$-value (or it will not be the clustering objective any more) instead of **Eq 97**. That means, we transform to the problem

$$\underset{\{C_1, C_2, \ldots, C_k\}}{\text{maximize}} \ \mathrm{Tr}\left[\chi^{\mathrm{T}}\Phi^{\mathrm{T}}\Phi\chi\right] \quad (99)$$

with the restriction of **Eq 97** but solve it with another restriction. Nevertheless, observing the analogy between KKM and SC is insightful.

By examing Tr $\left[\chi^{\mathrm{T}} c \vec{1}^{\mathrm{T}} \chi\right]$, we propose some conditions to make it nearly constant: $c_j = 1, \forall j$. That is, c_j is constant. We use 1 for simplicity. It is shown by:

$$\mathrm{Tr}\left[\chi^{\mathrm{T}} c \vec{1}^{\mathrm{T}} \chi\right] = \sum_{i=1}^{k} \sum_{j \in C_i} c_j |C_i| = \sum_{i=1}^{k} |C_i|^2 \tag{100}$$

This condition is often satisfied, for $c_j = \phi(x_i)^{\mathrm{T}} \phi(x_i) = k(x_i, x_i)$. Some data independent kernels have constant diagonals, e.g. $k(x_i, x_j) = \exp\{-\frac{||x_i - x_j||^2}{t}\}$. Now assume we have an ideal algorithm to solve the clustering algorithm, called CL. If some condition can drive CL to make Tr $\left[\chi^{\mathrm{T}} c \vec{1}^{\mathrm{T}} \chi\right]$ a constant, the correspondance can be established (though it is also possible that solving KKM results in local maxima). When $\sum_{i=1}^{k} |C_i|^2$ is a dominant term in the objective subject to the constraint $\sum_{i=1}^{k} |C_i| = N$, the optimal configuration of cluster cardinality is equally distributed. If our data points scatter into similar sized clusters naturally, the ideal CL algorithm should be able to detect it. Then we propose the following conditions when KKM and SC are able to work out nearly equivalent outcomes:

- There are good clustering in feature space.
- Data scatter into similar sized clusters naturally.
- $k(x_i, x_i) = \sigma$ is constant and the larger the better. This is because Tr $\left[\chi^{\mathrm{T}} c \vec{1}^{\mathrm{T}} \chi\right] = \sigma \sum_{i=1}^{k} |C_i|^2$. Larger σ can make the first term dominant and CL will be forced to equal size.

6 Conclusion

Our contribution in this article comes from the following aspects:

- Spectral Clustering is a seemingly abused term. We provide a taxonomy based on our survey, categorizing those algorithms into Spectral Embedding type and Spectral Graph Partitioning type. (**Section 1.2**)
- There are too many variations of SC and they all have algorithm specific justifications. We proposed a general three-stage framework which can cover all the variations we encounter in the survey. The framework is analyzed from a practitioner's perspective without justification. The purpose is to show all kinds of possibilities. Based on our framework, researchers can examine what is done and what is left. If they find some combinations working well on certain problem, seeking for justification will be an interesting research work. (**Section 2**)

- We also collect many justifications for SC and analyze combinatoric justifications and stochastic justifications in depth. For the cut based justification, we start from a mere cut case and obtain a poorly-defined optimization problem. Normalization constraints are proposed to well define the problem from optimization perspective. Different normalization constraints map back to different combinatoric objectives. In this way, we provide a viewpoint different from most literatures. (**Section 3**)
- Dimensionality Reduction methods from machine learning community share a lot in common with SC/SE. In this article, we term them all by Spectral Embedding Technique, because eigen value decomposition is involved and embedding into lower-dimensional Euclidean space is a core procedure. We briefly derive some representative spectral dimensionality reduction methods and map them to our framework. (**Section 4**)
- We provide unifying views from Graph Framework, Kernel Framework, and mathematical foundation(trace minimization). Those views reveal the analogy among SET algorithms. (**Section 5**)
- The Graph Framework and Kernel Framework are too general to give us any hints on how to utilize the similarities/dissimilarities between those algorithms. To compensate for this, we adapt a kernel clustering method and propose the conditions under what Kernel K-Means and Spectral Clustering are nearly output equivalent. This gives a chance for the two kinds of algorithms to work interchangeably. The testing of our hypotheses is left for future work due to time limit. (**Section 5.4**)

The main disadvantage of this survey is too theory oriented. Our framework looks promising to help people construct new algorithms, but I need to work out at least one sample to convice people. This is left to future work and the current document will be maintained in my tutorial repository[21].

Acknowledgements

I would like to thank course staffs of CUHK/CSCI5180, Spectral Algorithms. The lecture notes are very tentatively prepared. I also thank course staffs of CUHK/CSCI5030, Machine Learning Theory. Without those training, I would not be able to come up with a survey with such scope.

References

[1] C.C. Aggarwal. *Social network data analytics.* Springer-Verlag New York Inc, 2011.

[2] R. Andersen and F. Chung. Detecting sharp drops in pagerank and a simplified local partitioning algorithm. *Theory and Applications of Models of Computation*, ¡¡missing¿¿:1–12, 2007.

[3] R. Andersen and Y. Peres. Finding sparse cuts locally using evolving sets. In *Proceedings of the 41st annual ACM symposium on Theory of computing*, pages 235–244. ACM, 2009.

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

[5] Y. Bengio, O. Delalleau, N. Le Roux, J.F. Paiement, P. Vincent, and M. Ouimet. Spectral dimensionality reduction. *Feature Extraction*, ¡¡missing¿¿:519–550, 2006.

[6] Y. Bengio, O. Delalleau, N.L. Roux, J.F. Paiement, P. Vincent, and M. Ouimet. Learning eigenfunctions links spectral embedding and kernel pca. *Neural Computation*, 16(10):2197–2219, 2004.

[7] Y. Bengio, J.F. Paiement, P. Vincent, O. Delalleau, N. Le Roux, and M. Ouimet. Out-of-sample extensions for lle, isomap, mds, eigenmaps, and spectral clustering. *Advances in neural information processing systems*, 16:177–184, 2004.

[8] C.M Bishop. *Pattern recognition and machine learning*, volume 4. springer New York, 2006.

[9] I. Borg and P.J.F. Groenen. *Modern multidimensional scaling: Theory and applications*. Springer Verlag, 2005.

[10] M. Brand and K. Huang. A unifying theorem for spectral embedding and clustering. In *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics*, 2003.

[11] M. Chen, M. Liu, J. Liu, and X. Tang. Isoperimetric cut on a directed graph. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2109–2116. IEEE, 2010.

[12] F. Chung. Random walks and local cuts in graphs. *Linear Algebra and its applications*, 423(1):22–32, 2007.

[13] M.A.A. Cox and T.F. Cox. Multidimensional scaling. *Handbook of data visualization*, ¡¡missing¿¿:315–347, 2008.

[14] I. Dhillon, Y. Guan, and B. Kulis. A unified view of kernel k-means, spectral clustering and graph cuts. Technical report, Univ. of Texas at Austin, 2004.

[15] D.L. Donoho and C. Grimes. Hessian eigenmaps: Locally linear embedding techniques for high-dimensional data. *Proceedings of the National Academy of Sciences of the United States of America*, 100(10):5591, 2003.

[16] S.W. Hadley, B.L. Mark, and A. Vannelli. An efficient eigenvector approach for finding netlist partitions. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 11(7):885–892, 1992.

[17] J. Ham, D.D. Lee, S. Mika, and B. Schölkopf. A kernel view of the dimensionality reduction of manifolds. In *Proceedings of the twenty-first international conference on Machine learning*, page 47. ACM, 2004.

[18] B. Hendrickson and R. Leland. Multidimensional spectral load balancing. *Report SAND93-0074, Sandia National Laboratories, Albuquerque, NM*, ¡¡missing¿¿:¡¡missing¿¿, 1993.

[19] Pili Hu. Matrix calculus. GitHub, https://github.com/hupili/tutorial/tree/master/matrix-calculus, 3 2012. HU, Pili's tutorial collection.

[20] Pili Hu. Spectral techniques for community detection on 2-hop topology. GitHub, https://github.com/hupili/Spectral-2Hop, 4 2012. course project of CUHK/CSCI5160.

[21] Pili Hu. Tutorial collection. GitHub, https://github.com/hupili/tutorial, 3 2012. HU, Pili's tutorial collection.

[22] H. Jiawei and M. Kamber. *Data mining: concepts and techniques*, volume 5. San Francisco, CA, itd: Morgan Kaufmann, 2001.

[23] R. Kannan, S. Vempala, and A. Vetta. On clusterings: Good, bad and spectral. *Journal of the ACM (JACM)*, 51(3):497–515, 2004.

[24] L. Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, 1953.

[25] Lap Chi Lau. Spectral algorithm lecture notes. http://www.cse.cuhk.edu.hk/ chi/csc5160/index.html, 2012.

[26] L. Lovász and M. Simonovits. The mixing rate of markov chains, an isoperimetric inequality, and computing the volume. In *Foundations of Computer Science, 1990. Proceedings., 31st Annual Symposium on*, pages 346–354. IEEE, 1990.

[27] M. Maila and J. Shi. A random walks view of spectral segmentation. *AI and STATISTICS (AISTATS) 2001*, ¡¡misssing¿¿:¡¡misssing¿¿, 2001.

[28] A.Y. Ng, M.I. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems*, 2:849–856, 2002.

[29] S.T. Roweis and L.K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.

[30] L.K. Saul and S.T. Roweis. An introduction to locally linear embedding. Technical report, NYU, 2000.

[31] L.K. Saul and S.T. Roweis. Think globally, fit locally: unsupervised learning of low dimensional manifolds. *The Journal of Machine Learning Research*, 4:119–155, 2003.

[32] B. Schölkopf, A. Smola, and K.R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural computation*, 10(5):1299–1319, 1998.

[33] J. Shi and J. Malik. Normalized cuts and image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(8):888–905, 2000.

[34] DA Spielman. Spectral graph theory lecture notes, 2009. Available at `http://www.cs.yale.edu/homes/spielman/561/`.

[35] J.B. Tenenbaum, V. De Silva, and J.C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

[36] M.E. Tipping and C.M. Bishop. Probabilistic principal component analysis. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 61(3):611–622, 1999.

[37] U. Von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007.

[38] Wikipedia, Jaccard Coefficient, `http://en.wikipedia.org/wiki/Jaccard_index`

[39] Wikipedia, Mahalanobis Distance, `http://en.wikipedia.org/wiki/Mahalanobis_distance`

[40] Wikipedia, Cosine Similarity, `http://en.wikipedia.org/wiki/Cosine_similarity`

[41] Wikipedia, Singular Value Decomposition `http://en.wikipedia.org/wiki/Singular_value_decomposition`

[42] Wikipedia, Kirchhoff's Circuit Laws, http://en.wikipedia.org/wiki/Kirchhoff%27s_circuit_laws

[43] Wikipedia, Moore-Penrose Inverse, http://en.wikipedia.org/wiki/Moore%E2%80%93Penrose_pseudoinverse

[44] Wikipedia, Matrix Norm, http://en.wikipedia.org/wiki/Matrix_norm

[45] Wikipedia, Multi-Dimensional Scaling http://en.wikipedia.org/wiki/Multidimensional_scaling

[46] Wikipedia, Floyd-Warshall Algorithm, http://en.wikipedia.org/wiki/Floyd%E2%80%93Warshall_algorithm