7 Clustering

7.1 Introduction

Clustering refers to partitioning a set of objects into subsets according to some desired criterion. Often it is an important step in making sense of large amounts of data. Clustering comes up in many contexts. One might want to partition a set of news articles into clusters based on the topics of the articles. Given a set of pictures of people, one might want to group them into clusters based on who is in the image. Or one might want to cluster a set of protein sequences according to the protein function. A related problem is not finding a full partitioning but rather just identifying natural clusters that exist. For example, given a collection of friendship relations among people, one might want to identify any tight-knit groups that exist. In some cases we have a well-defined correct answer, e.g., in clustering photographs of individuals by who is in them, but in other cases the notion of a good clustering may be more subjective.

Before running a clustering algorithm, one first needs to choose an appropriate representation for the data. One common representation is as vectors in R^d. This corresponds to identifying d real-valued features that are then computed for each data object. For example, to represent documents one might use a "bag of words" representation, where each feature corresponds to a word in the English language and the value of the feature is how many times that word appears in the document. Another common representation is as vertices in a graph, with edges weighted by some measure of how similar or dissimilar the two endpoints are. For example, given a set of protein sequences, one might weight edges based on an edit-distance measure that essentially computes the cost of transforming one sequence into the other. This measure is typically symmetric and satisfies the triangle inequality, and so can be thought of as a finite metric. A point worth noting up front is that often the "correct" clustering of a given set of data depends on your goals. For instance, given a set of photographs of individuals, we might want to cluster the images by who is in them, or we might want to cluster them by facial expression. When representing the images as points in space or as nodes in a weighted graph, it is important that the features we use be relevant to the criterion we care about. In any event, the issue of how best to represent data to highlight the relevant information for a given task is generally addressed using knowledge of the specific domain. From our perspective, the job of the clustering algorithm begins after the data has been represented in some appropriate way.

In this chapter, our goals are to discuss (a) some commonly used clustering algorithms and what one can prove about them, and (b) models and assumptions on data under which we can find a clustering close to the correct clustering.

7.1.1 Preliminaries

We will follow the standard notation of using n to denote the number of data points and k to denote the number of desired clusters. We will primarily focus on the case that

k is known up front, but will also discuss algorithms that produce a sequence of solutions, one for each value of k, as well as algorithms that produce a cluster tree that can encode multiple clusterings at each value of k. We will generally use $A = \{\mathbf{a}_1, \ldots, \mathbf{a}_n\}$ to denote the n data points. We also think of A as a matrix with rows $\mathbf{a}_1, \ldots, \mathbf{a}_n$.

7.1.2 Two General Assumptions on the Form of Clusters

Before choosing a clustering algorithm, it is useful to have some general idea of what a good clustering should look like. In general, there are two types of assumptions often made that in turn lead to different classes of clustering algorithms.

Center-based clusters: One assumption commonly made is that clusters are *center-based*. This means that the clustering can be defined by k centers $\mathbf{c}_1, \ldots, \mathbf{c}_k$, with each data point assigned to whichever center is closest to it. Note that this assumption does not yet tell whether one choice of centers is better than another. For this, one needs an objective, or optimization criterion. Three standard criteria often used are k-center, k-median, and k-means clustering, defined as follows.

k-center clustering: Find a partition $\mathcal{C} = \{C_1, \ldots, C_k\}$ of A into k clusters, with corresponding centers $\mathbf{c}_1, \ldots, \mathbf{c}_k$, to minimize the *maximum* distance between any data point and the center of its cluster. That is, we want to minimize

$$\Phi_{kcenter}(\mathcal{C}) = \max_{j=1}^{k} \max_{\mathbf{a}_i \in C_j} d(\mathbf{a}_i, \mathbf{c}_j).$$

k-center clustering makes sense when we believe clusters should be local regions in space. It is also often thought of as the "firehouse location problem" since one can think of it as the problem of locating k fire-stations in a city so as to minimize the maximum distance a fire-truck might need to travel to put out a fire.

k-median clustering: Find a partition $\mathcal{C} = \{C_1, \ldots, C_k\}$ of A into k clusters, with corresponding centers $\mathbf{c}_1, \ldots, \mathbf{c}_k$, to minimize the *sum* of distances between data points and the centers of their clusters. That is, we want to minimize

$$\Phi_{kmedian}(\mathcal{C}) = \sum_{j=1}^{k} \sum_{\mathbf{a}_i \in C_j} d(\mathbf{a}_i, \mathbf{c}_j).$$

k-median clustering is more noise-tolerant than k-center clustering because we are taking a sum rather than a max. A small number of outliers will typically not change the optimal solution by much, unless they are very far away or there are several quite different near-optimal solutions.

k-means clustering: Find a partition $\mathcal{C} = \{C_1, \ldots, C_k\}$ of A into k clusters, with corresponding centers $\mathbf{c}_1, \ldots, \mathbf{c}_k$, to minimize the *sum of squares* of distances between data points and the centers of their clusters. That is, we want to minimize

$$\Phi_{kmeans}(\mathcal{C}) = \sum_{j=1}^{k} \sum_{\mathbf{a}_i \in C_j} d^2(\mathbf{a}_i, \mathbf{c}_j).$$

k-means clustering puts more weight on outliers than k-median clustering, because we are squaring the distances, which magnifies large values. This puts it somewhat in between k-median and k-center clustering in that regard. Using distance squared has some mathematical advantages over using pure distances when data are points in R^d. For example, Corollary 7.2 that asserts that with the distance squared criterion, the optimal center for a given group of data points is its centroid.

The k-means criterion is more often used when data consists of points in R^d, whereas k-median is more commonly used when we have a finite metric, that is, data are nodes in a graph with distances on edges.

When data are points in R^d, there are in general two variations of the clustering problem for each of the criteria. We could require that each cluster center be a data point or allow a cluster center to be any point in space. If we require each center to be a data point, the optimal clustering of n data points into k clusters can be solved in time $\binom{n}{k}$ times a polynomial in the length of the data. First, exhaustively enumerate all sets of k data points as the possible sets of k cluster centers, then associate each point to its nearest center and select the best clustering. No such naive enumeration procedure is available when cluster centers can be any point in space. But, for the k-means problem, Corollary 7.2 shows that once we have identified the data points that belong to a cluster, the best choice of cluster center is the centroid of that cluster, which might not be a data point.

When k is part of the input or may be a function of n, the above optimization problems are all NP-hard.[33] So, guarantees on algorithms will typically involve either some form of approximation or some additional assumptions, or both.

High-density clusters: If we do not believe our desired clusters will be center-based, an alternative assumption often made is that clusters consist of high-density regions surrounded by low-density "moats" between them. For example, in the clustering of Figure 7.1 we have one natural cluster A that looks center-based but the other cluster B consists of a ring around cluster A. As seen in the figure, this assumption does not require clusters to correspond to convex regions and it can allow them to be long and stringy. We describe a non-center-based clustering method in Section 7.7. In Section 7.9 we prove the effectiveness of an algorithm which finds a "moat", cuts up data "inside" the moat and 'outside" into two pieces and recursively applies the same procedure to each piece.

[33]If k is a constant, then as noted above, the version where the centers must be data points can be solved in polynomial time.

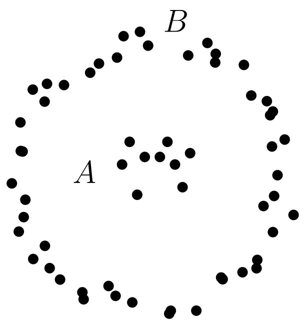


Figure 7.1: Example where the natural clustering is not center-based.

7.1.3 Spectral Clustering

An important part of a clustering toolkit when data lies in R^d is Singular Value Decomposition. Spectral Clustering refers to the following algorithm: Find the space V spanned by the top k right singular vectors of the matrix A whose rows are the data points. Project data points to V and cluster in the projection.

An obvious reason to do this is dimension reduction, clustering in the d dimensional space where data lies is reduced to clustering in a k dimensional space (usually, $k << d$). A more important point is that under certain assumptions one can prove that spectral clustering gives a clustering close to the true clustering. We already saw this in the case when data is from a mixture of spherical Gaussians, Section 3.9.3. The assumption used is "the means separated by a constant number of Standard Deviations". In Section 7.5, we will see that in a much more general setting, which includes common stochastic models, the same assumption, in spirit, yields similar conclusions. Section 7.4, has another setting with a similar result.

7.2 k-Means Clustering

We assume in this section that data points lie in R^d and focus on the k-means criterion.

7.2.1 A Maximum-Likelihood Motivation

We now consider a maximum-likelihood motivation for using the k-means criterion. Suppose that the data was generated according to an equal weight mixture of k spherical well-separated Gaussian densities centered at $\boldsymbol{\mu_1}, \boldsymbol{\mu_2}, \ldots, \boldsymbol{\mu_k}$, each with variance one in every direction. Then the density of the mixture is

$$\text{Prob}(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}} \frac{1}{k} \sum_{i=1}^{k} e^{-|\mathbf{x}-\boldsymbol{\mu_i}|^2}.$$

Denote by $\boldsymbol{\mu}(\mathbf{x})$ the center nearest to $\mathbf{x}$. Since the exponential function falls off fast, assuming x is noticeably closer to its nearest center than to any other center, we can approximate $\sum_{i=1}^{k} e^{-|\mathbf{x}-\boldsymbol{\mu}_i|^2}$ by $e^{-|\mathbf{x}-\boldsymbol{\mu}(\mathbf{x})|^2}$ since the sum is dominated by its largest term. Thus

$$\text{Prob}(\mathbf{x}) \approx \frac{1}{(2\pi)^{d/2}k} e^{-|\mathbf{x}-\boldsymbol{\mu}(\mathbf{x})|^2}.$$

The likelihood of drawing the sample of points $\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_n}$ from the mixture, if the centers were $\boldsymbol{\mu_1}, \boldsymbol{\mu_2}, \ldots, \boldsymbol{\mu_k}$, is approximately

$$\frac{1}{k^n} \frac{1}{(2\pi)^{nd/2}} \prod_{i=1}^{n} e^{-|\mathbf{x^{(i)}}-\boldsymbol{\mu}(\mathbf{x^{(i)}})|^2} = c e^{-\sum_{i=1}^{n} |\mathbf{x^{(i)}}-\boldsymbol{\mu}(\mathbf{x^{(i)}})|^2}.$$

Minimizing the sum of squared distances to cluster centers finds the maximum likelihood $\boldsymbol{\mu_1}, \boldsymbol{\mu_2}, \ldots, \boldsymbol{\mu_k}$. This motivates using the sum of distance squared to the cluster centers.

7.2.2 Structural Properties of the k-Means Objective

Suppose we have already determined the clustering or the partitioning into $C_1, C_2, \ldots, C_k$. What are the best centers for the clusters? The following lemma shows that the answer is the centroids, the coordinate means, of the clusters.

Lemma 7.1 *Let $\{\mathbf{a_1}, \mathbf{a_2}, \ldots, \mathbf{a_n}\}$ be a set of points. The sum of the squared distances of the $\mathbf{a_i}$ to any point $\mathbf{x}$ equals the sum of the squared distances to the centroid of the $\mathbf{a_i}$ plus n times the squared distance from $\mathbf{x}$ to the centroid. That is,*

$$\sum_i |\mathbf{a_i} - \mathbf{x}|^2 = \sum_i |\mathbf{a_i} - \mathbf{c}|^2 + n\,|\mathbf{c} - \mathbf{x}|^2$$

where $\mathbf{c} = \frac{1}{n} \sum\limits_{i=1}^{n} \mathbf{a_i}$ is the centroid of the set of points.

Proof:

$$\begin{aligned}\sum_i |\mathbf{a_i} - \mathbf{x}|^2 &= \sum_i |\mathbf{a_i} - \mathbf{c} + \mathbf{c} - \mathbf{x}|^2 \\ &= \sum_i |\mathbf{a_i} - \mathbf{c}|^2 + 2(\mathbf{c} - \mathbf{x}) \cdot \sum_i (\mathbf{a_i} - \mathbf{c}) + n\,|\mathbf{c} - \mathbf{x}|^2\end{aligned}$$

Since $\mathbf{c}$ is the centroid, $\sum\limits_i (\mathbf{a_i} - \mathbf{c}) = 0$. Thus, $\sum\limits_i |\mathbf{a_i} - \mathbf{x}|^2 = \sum\limits_i |\mathbf{a_i} - \mathbf{c}|^2 + n\,|\mathbf{c} - \mathbf{x}|^2$ ∎

A corollary of Lemma 7.1 is that the centroid minimizes the sum of squared distances since the first term, $\sum\limits_i |\mathbf{a_i} - \mathbf{c}|^2$, is a constant independent of $\mathbf{x}$ and setting $\mathbf{x} = \mathbf{c}$ sets the second term, $n\,\|\mathbf{c} - \mathbf{x}\|^2$, to zero.

Corollary 7.2 *Let $\{\mathbf{a_1}, \mathbf{a_2}, \ldots, \mathbf{a_n}\}$ be a set of points. The sum of squared distances of the $\mathbf{a_i}$ to a point $\mathbf{x}$ is minimized when $\mathbf{x}$ is the centroid, namely $\mathbf{x} = \frac{1}{n} \sum\limits_i \mathbf{a_i}$.*

7.2.3 Lloyd's Algorithm

Corollary 7.2 suggests the following natural strategy for k-means clustering, known as Lloyd's algorithm. Lloyd's algorithm does not necessarily find a globally optimal solution but will find a locally-optimal one. An important but unspecified step in the algorithm is its initialization: how the starting k centers are chosen. We discuss this after discussing the main algorithm.

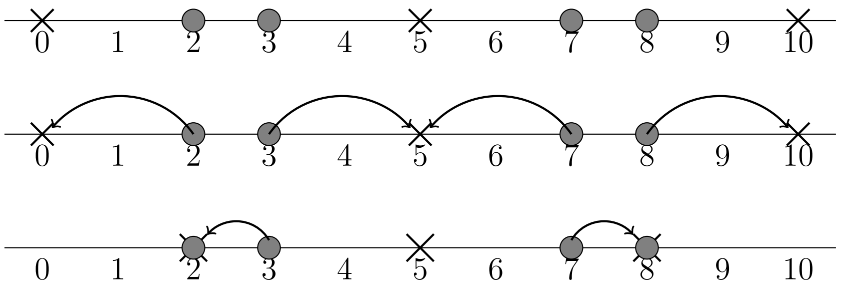
Lloyd's algorithm:

```
Start with $k$ centers.
Cluster each point with the center nearest to it.
Find the centroid of each cluster and replace the set of old centers with the centroids.
Repeat the above two steps until the centers converge according to some criterion, such
as the $k$-means score no longer improving.
```

This algorithm always converges to a local minimum of the objective. To show convergence, we argue that the sum of the squares of the distances of each point to its cluster center always improves. Each iteration consists of two steps. First, consider the step that finds the centroid of each cluster and replaces the old centers with the new centers. By Corollary 7.2, this step improves the sum of internal cluster distances squared. The second step reclusters by assigning each point to its nearest cluster center, which also improves the internal cluster distances.

A problem that arises with some implementations of the k-means clustering algorithm is that one or more of the clusters becomes empty and there is no center from which to measure distance. A simple case where this occurs is illustrated in the following example. You might think how you would modify the code to resolve this issue.

Example: Consider running the k-means clustering algorithm to find three clusters on the following 1-dimension data set: {2,3,7,8} starting with centers {0,5,10}.



The center at 5 ends up with no items and there are only two clusters instead of the desired three. ∎

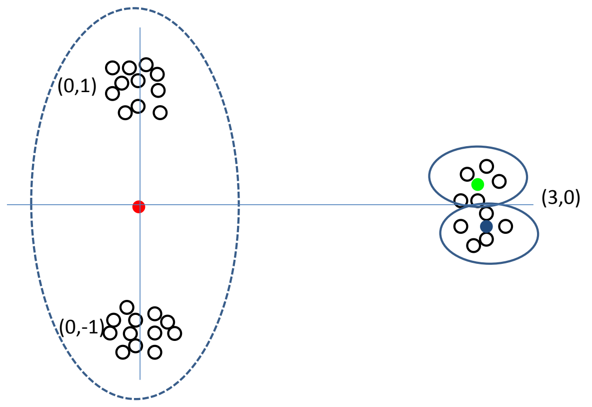


Figure 7.2: A locally-optimal but globally-suboptimal k-means clustering.

As noted above, Lloyd's algorithm only finds a local optimum to the k-means objective that might not be globally optimal. Consider, for example, Figure 7.2. Here data lies in three dense clusters in R^2: one centered at $(0, 1)$, one centered at $(0, -1)$ and one centered at $(3, 0)$. If we initialize with one center at $(0, 1)$ and two centers near $(3, 0)$, then the center at $(0, 1)$ will move to near $(0, 0)$ and capture the points near $(0, 1)$ and $(0, -1)$, whereas the centers near $(3, 0)$ will just stay there, splitting that cluster.

Because the initial centers can substantially influence the quality of the result, there has been significant work on initialization strategies for Lloyd's algorithm. One popular strategy is called "farthest traversal". Here, we begin by choosing one data point as initial center $\mathbf{c}_1$ (say, randomly), then pick the farthest data point from $\mathbf{c}_1$ to use as $\mathbf{c}_2$, then pick the farthest data point from $\{\mathbf{c}_1, \mathbf{c}_2\}$ to use as $\mathbf{c}_3$, and so on. These are then used as the initial centers. Notice that this will produce the correct solution in the example in Figure 7.2.

Farthest traversal can unfortunately get fooled by a small number of outliers. To address this, a smoother, probabilistic variation known as k-means++ instead weights data points based on their distance squared from the previously chosen centers. Then it selects the next center probabilistically according to these weights. This approach has the nice property that a small number of outliers will not overly influence the algorithm so long as they are not too far away, in which case perhaps they should be their own clusters anyway.

Another approach is to run some other approximation algorithm for the k-means problem, and then use its output as the starting point for Lloyd's algorithm. Note that applying Lloyd's algorithm to the output of any other algorithm can only improve its score. An alternative SVD-based method for initialization is described and analyzed in Section 7.5.

7.2.4 Ward's Algorithm

Another popular heuristic for k-means clustering is Ward's algorithm. Ward's algorithm begins with each data point in its own cluster, and then repeatedly merges pairs of clusters until only k clusters remain. Specifically, Ward's algorithm merges the two clusters that minimize the immediate increase in k-means cost. That is, for a cluster C, define $cost(C) = \sum_{\mathbf{a}_i \in C} d^2(\mathbf{a}_i, \mathbf{c})$, where $\mathbf{c}$ is the centroid of C. Then Ward's algorithm merges the pair (C, C') minimizing $cost(C \cup C') - cost(C) - cost(C')$. Thus, Ward's algorithm can be viewed as a greedy k-means algorithm.

7.2.5 k-Means Clustering on the Line

One case where the optimal k-means clustering can be found in polynomial time is when points lie in R^1, i.e., on the line. This can be done using dynamic programming, as follows.

First, assume without loss of generality that the data points $a_1, \ldots, a_n$ have been sorted, so $a_1 \leq a_2 \leq \ldots \leq a_n$. Now, suppose that for some $i \geq 1$ we have already computed the optimal k'-means clustering for points $a_1, \ldots, a_i$ for all $k' \leq k$; note that this is trivial to do for the base case of $i = 1$. Our goal is to extend this solution to points $a_1, \ldots, a_{i+1}$. To do so, observe that each cluster will contain a consecutive sequence of data points. So, given k', for each $j \leq i+1$, compute the cost of using a single center for points $a_j, \ldots, a_{i+1}$, which is the sum of distances of each of these points to their mean value. Then add to that the cost of the optimal $k'-1$ clustering of points $a_1, \ldots, a_{j-1}$ which we computed earlier. Store the minimum of these sums, over choices of j, as our optimal k'-means clustering of points $a_1, \ldots, a_{i+1}$. This has running time of $O(kn)$ for a given value of i. So overall our running time is $O(kn^2)$.

7.3 k-Center Clustering

In this section, instead of using the k-means clustering criterion, we use the k-center criterion. Recall that the k-center criterion partitions the points into k clusters so as to minimize the maximum distance of any point to its cluster center. Call the maximum distance of any point to its cluster center the *radius* of the clustering. There is a k-clustering of radius r if and only if there are k spheres, each of radius r, which together cover all the points. Below, we give a simple algorithm to find k spheres covering a set of points. The following lemma shows that this algorithm only needs to use a radius that is at most twice that of the optimal k-center solution. Note that this algorithm is equivalent to the farthest traversal strategy for initializing Lloyd's algorithm.

The Farthest Traversal k-clustering Algorithm

Pick any data point to be the first cluster center. At time t, for $t = 2, 3, \ldots, k$, pick the farthest data point from any existing cluster center; make it the t^{th} cluster center.

Theorem 7.3 *If there is a k-clustering of radius $\frac{r}{2}$, then the above algorithm finds a k-clustering with radius at most r.*

Proof: Suppose for contradiction that there is some data point $\mathbf{x}$ that is distance greater than r from all centers chosen. This means that each new center chosen was distance greater than r from all previous centers, because we could always have chosen $\mathbf{x}$. This implies that we have $k+1$ data points, namely the centers chosen plus $\mathbf{x}$, that are pairwise more than distance r apart. Clearly, no two such points can belong to the same cluster in any k-clustering of radius $\frac{r}{2}$, contradicting the hypothesis. ∎

7.4 Finding Low-Error Clusterings

In the previous sections we saw algorithms for finding a local optimum to the k-means clustering objective, for finding a global optimum to the k-means objective on the line, and for finding a factor 2 approximation to the k-center objective. But what about finding a clustering that is close to the correct answer, such as the true clustering of proteins by function or a correct clustering of news articles by topic? For this we need some assumption about the data and what the correct answer looks like. The next few sections consider algorithms based on different such assumptions.

7.5 Spectral Clustering

Let A be a $n \times d$ data matrix with each row a data point and suppose we want to partition the data points into k clusters. *Spectral Clustering* refers to a class of clustering algorithms which share the following outline:

- Find the space V spanned by the top k (right) singular vectors of A.
- Project data points into V.
- Cluster the projected points.

7.5.1 Why Project?

The reader may want to read Section 3.9.3, which shows the efficacy of spectral clustering for data stochastically generated from a mixture of spherical Gaussians. Here, we look at general data which may not have a stochastic generation model.

We will later describe the last step in more detail. First, lets understand the central advantage of doing the projection to V. It is simply that for any reasonable (unknown) clustering of data points, the projection brings data points closer to their cluster centers. This statement sounds mysterious and likely false, since the assertion is for ANY reasonable unknown clustering. We quantify it in Theorem 7.4. First some notation: We represent a k-clustering by a $n \times d$ matrix C (same dimensions as A), where row i of C is

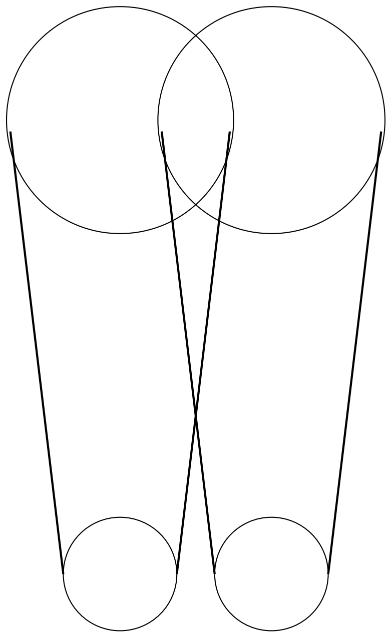


Figure 7.3: Clusters in the full space and their projections

the center of the cluster to which data point i belongs. So, there are only k distinct rows of C and each other row is a copy of one of these rows. The k-means objective function, namely, the sum of squares of the distances of data points to their cluster centers is

$$\sum_{i=1}^{n} |\mathbf{a_i} - \mathbf{c_i}|^2 = ||A - C||_F^2.$$

The projection reduces the sum of distance squares to cluster centers from $||A - C||_F^2$ to at most $8k||A - C||_2^2$ in the projection. Recall that $||A - C||_2$ is the spectral norm, which is the top singular value of $A - C$. Now, $||A - C||_F^2 = \sum_t \sigma_t^2(A)$ and often, $||A - C||_F >> \sqrt{k}||A - C||_2$ and so the projection substantially reduces the sum of squared distances to cluster centers.

We will see later that in many clustering problems, including models like mixtures of Gaussians and Stochastic Block Models of communities, there is a desired clustering C where the regions overlap in the whole space, but are separated in the projection. Figure 7.3 is a schematic illustration. Now we state the theorem and give its surprisingly simple proof.

Theorem 7.4 *Let A be an $n \times d$ matrix with A_k the projection of the rows of A to the subspace of the first k right singular vectors of A. For any matrix C of rank less than or equal to k*

$$||A_k - C||_F^2 \leq 8k||A - C||_2^2.$$

A_k is a matrix that is close to every C, in the sense $||A_k - C||_F^2 \leq 8k||A - C||_2^2$. While this seems contradictory, another way to state this is that for C far away from A_k in Frobenius norm, $||A - C||_2$ will also be high.

Proof: Since the rank of $(A_k - C)$ is less than or equal to $2k$,

$$||A_k - C||_F^2 \leq 2k||A_k - C||_2^2 \quad \text{and}$$

$$||A_k - C||_2 \leq ||A_k - A||_2 + ||A - C||_2 \leq 2||A - C||_2.$$

The last inequality follows since A_k is the best rank k approximation in spectral norm (Theorem 3.9) and C has rank at most k. The theorem follows. ∎

Suppose now in the clustering C we would like to find, the cluster centers that are pairwise at least $\Omega(\sqrt{k}||A - C||_2)$ apart. This holds for many clustering problems including data generated by stochastic models. Then, it will be easy to see that in the projection, most data points are a constant factor farther from centers of other clusters than their own cluster center and this makes it very easy for the following algorithm to find the clustering C modulo a small fraction of errors.

7.5.2 The Algorithm

Denote $||A - C||_2/\sqrt{n}$ by $\sigma(C)$. In the next section, we give an interpretation of $||A - C||_2$ indicating that $\sigma(C)$ is akin to the standard deviation of clustering C and hence the notation $\sigma(C)$. We assume for now that $\sigma(C)$ is known to us for the desired clustering C. This assumption can be removed by essentially doing a binary search.

Spectral Clustering - The Algorithm

1. Find the top k right singular vectors of data matrix A and project rows of A to the space spanned by them to get A_k.(cf. Section 3.5).
2. Select a random row from A_k and form a cluster with all rows of A_k at distance less than $6k\sigma(C)/\varepsilon$ from it.
3. Repeat Step 2 k times.

Theorem 7.5 *If in a k-clustering C, every pair of centers is separated by at least $15k\sigma(C)/\varepsilon$ and every cluster has at least εn points in it, then with probability at least $1 - \varepsilon$, Spectral Clustering finds a clustering C' that differs from C on at most $\varepsilon^2 n$ points.*

Proof: Let $\mathbf{v_i}$ denote row i of A_k. We first show that for most data points, the projection of data point is within distance $3k\sigma(C)/\varepsilon$ of its cluster center. I.e., we show that $|M|$ is small, where,

$$M = \{i : |\mathbf{v_i} - \mathbf{c_i}| \geq 3k\sigma(C)/\varepsilon\}.$$

Now, $||A_k - C||_F^2 = \sum_i |\mathbf{v_i} - \mathbf{c_i}|^2 \geq \sum_{i \in M} |\mathbf{v_i} - \mathbf{c_i}|^2 \geq |M|\frac{9k^2\sigma^2(C)}{\varepsilon^2}$. So, using Theorem 7.4, we get:

$$|M|\frac{9k^2\sigma^2(C)}{\varepsilon^2} \leq ||A_k - C||_F^2 \leq 8kn\sigma^2(C) \quad \Longrightarrow \quad |M| \leq \frac{8\varepsilon^2 n}{9k}. \tag{7.1}$$

Call a data point i "good" if $i \notin M$. For any two good data points i and j belonging to the same cluster, since, their projections are within $3k\sigma(C)/\varepsilon$ of the center of the cluster, projections of the two data points are within $6k\sigma(C)/\varepsilon$ of each other. On the other hand, if two good data points i and k are in different clusters, since, the centers of the two clusters are at least $15k\sigma(C)/\varepsilon$ apart, their projections must be greater than $15k\sigma(C)/\varepsilon - 6k\sigma(C)/\varepsilon = 9k\sigma(C)/\varepsilon$ apart. So, if we picked a good data point (say point i) in Step 2, the set of good points we put in its cluster is exactly the set of good points in the same cluster as i. Thus, if in each of the k executions of Step 2, we picked a good point, all good points are correctly clustered and since $|M| \leq \varepsilon^2 n$, the theorem would hold.

To complete the proof, we must argue that the probability of any pick in Step 2 being bad is small. The probability that the first pick in Step 2 is bad is at most $|M|/n \leq \varepsilon^2/k$. For each subsequent execution of Step 2, all the good points in at least one cluster are remaining candidates. So there are at least $(\varepsilon - \varepsilon^2)n$ good points left and so the probability that we pick a bad point is at most $|M|/(\varepsilon - \varepsilon^2)n$ which is at most ε/k. The union bound over the k executions yields the desired result.

■

7.5.3 Means Separated by $\Omega(1)$ Standard Deviations

For probability distribution on the real line, the mnemonic "means separated by six standard deviations" suffices to distinguish different distributions. Spectral Clustering enables us to do the same thing in higher dimensions provided $k \in O(1)$ and six is replaced by some constant. First we define standard deviation for general not necessarily stochastically generated data: it is just the maximum over all unit vectors $\mathbf{v}$ of the square root of the mean squared distance of data points from their cluster centers in the direction $\mathbf{v}$, namely, the standard deviation $\sigma(C)$ of clustering C is defined as:

$$\sigma(C)^2 = \frac{1}{n}\mathrm{Max}_{\mathbf{v}:|\mathbf{v}|=1} \sum_{i=1}^{n} [(\mathbf{a_i} - \mathbf{c_i}) \cdot \mathbf{v}]^2 = \frac{1}{n}\mathrm{Max}_{\mathbf{v}:|\mathbf{v}|=1} |(A - C)\mathbf{v}|^2 = \frac{1}{n}||A - C||_2^2.$$

This coincides with the definition of $\sigma(C)$ we made earlier. Assuming $k \in O(1)$, it is easy to see that the Theorem 7.5 can be restated as

> If cluster centers in C are separated by $\Omega(\sigma(C))$, then the spectral clustering algorithm finds C' which differs from C only in a small fraction of data points.

It can be seen that the "means separated by $\Omega(1)$ standard deviations" condition holds for many stochastic models. We illustrate with two examples here. First, suppose we have a mixture of $k \in O(1)$ spherical Gaussians, each with standard deviation one. The data is generated according to this mixture. If the means of the Gaussians are $\Omega(1)$ apart, then the condition - means separated by $\Omega(1)$ standard deviations- is satisfied and so if we project to the SVD subspace and cluster, we will get (nearly) the correct clustering. This was already discussed in detail in Chapter **??**.

We discuss a second example. *Stochastic Block Models* are models of communities. Suppose there are k communities $C_1, C_2, \ldots, C_k$ among a population of n people. Suppose the probability of two people in the same community knowing each other is p and if they are in different communities, the probability is q, where, $q < p$.[34] We assume the events that person i knows person j are *independent* across all i and j.

Specifically, we are given an $n \times n$ data matrix A, where $a_{ij} = 1$ if and only if i and j know each other. We assume the a_{ij} are independent random variables, and use $\mathbf{a_i}$ to denote the i^{th} row of A. It is useful to think of A as the adjacency matrix of a graph, such as the friendship network in Facebook. We will also think of the rows $\mathbf{a_i}$ as data points. The clustering problem is to classify the data points into the communities they belong to. In practice, the graph is fairly sparse, i.e., p and q are small, namely, $O(1/n)$ or $O(\ln n/n)$.

Consider the simple case of two communities with $n/2$ people in each and with

$$p = \frac{\alpha}{n} \qquad q = \frac{\beta}{n} \qquad \text{where } \alpha, \beta \in O(\ln n).$$

Let $\mathbf{u}$ and $\mathbf{v}$ be the centroids of the data points in community one and community two respectively; so $u_i \approx p$ for $i \in C_1$ and $u_j \approx q$ for $j \in C_2$ and $v_i \approx q$ for $i \in C_1$ and $v_j \approx p$ for $j \in C_2$. We have

$$|\mathbf{u} - \mathbf{v}|^2 = \sum_{j=1}^{n} (u_j - v_j)^2 \approx \frac{(\alpha - \beta)^2}{n^2} n = \frac{(\alpha - \beta)^2}{n}.$$

$$\text{Inter-centroid distance } \approx \frac{\alpha - \beta}{\sqrt{n}}. \tag{7.2}$$

We need to upper bound $||A - C||_2$. This is non-trivial since we have to prove a uniform upper bound on $|(A - C)\mathbf{v}|$ for all unit vectors $\mathbf{v}$. Fortunately, the subject to Random Matrix Theory (RMT) already does this for us. RMT tells that

$$||A - C||_2 \leq O^*(\sqrt{np}) = O^*(\sqrt{\alpha}),$$

where, the O^* hides logarithmic factors. So as long as $\alpha - \beta \in \Omega^*(\sqrt{\alpha})$, we have the means separated by $\Omega(1)$ standard deviations and spectral clustering works.

[34]More generally, for each pair of communities a and b, there could be a probability p_{ab} that a person from community a knows a person from community b. But for the discussion here, we take $p_{aa} = p$ for all a and $p_{ab} = q$, for all $a \neq b$.

One important observation is that in these examples as well as many others, the k-means objective function in the whole space is too high and so the projection is essential before we can cluster.

7.5.4 Laplacians

An important special case of spectral clustering is when $k = 2$ and a spectral algorithm is applied to the Laplacian matrix L of a graph, which is defined as

$$L = D - A$$

where A is the adjacency matrix and D is a diagonal matrix of degrees. Since A has a negative sign, we look at the lowest two singular values and corresponding vectors rather than the highest,

L is a symmetric matrix and is easily seen to be posiitve semi-definite: for any vector $\mathbf{x}$, we have

$$\mathbf{x}^T L\mathbf{x} = \sum_i d_{ii} x_i^2 - \sum_{(i,j)\in E} x_i x_j = \frac{1}{2} \sum_{(i,j)\in E} (x_i - x_j)^2.$$

Also since all row sums of L (and L is symmetric) are zero, its lowest eignvalue is 0 with the eigenvector $\mathbf{1}$ of all 1's. This is also the lowest singular vector of L. The projection of all data points (rows) to this vector is just the origin and so gives no information. If we take the second lowest singular vector and project to it which is essentially projecting to the space of the bottom two singular vectors, we get the very simple problem of n real numbers which we need to cluster into two clusters.

7.5.5 Local spectral clustering

So far our emphasis has been on partitioning the data into disjoint clusters. However, the structure of many data sets consists of over lapping communities. In this case using k-means with spectral clustering, the overlap of two communities shows up as a community. This is illustrated in Figure 7.4.

An alternative to using k-means with spectral clustering is to find the minimum 1-norm vector in the space spanned by the top singular vectors. Let A be the matrix whose columns are the singular vectors. To find a vector y in the space spanned by the columns of A solve the linear system $Ax = y$. This is a slightly different looking problem then $Ax = c$ where c is a constant vector. To convert $Ax = y$ to the more usual form write it as $[A, -I] \begin{pmatrix} x \\ y \end{pmatrix} = 0$. However, if we want to minimize $||y||_1$ the solution is $x = y = 0$. Thus we add the row $1, 1, \ldots 1, 0, 0, \ldots 0$ to $[A, -I]$ and a 1 to the top of the vector $[x, y]$ to force the coordinates of x to add up to one. Minimizing $||y||_1$ does not appear to be a linear program but we can write $y = y_a - y_b$ and require $y_a \geq 0$ and $y_b \geq 0$. Now finding the minimum one norm vector in the span of the columns of A is the linear program

$$\min\left(\sum_i y_{ai} + \sum_i y_{bi}\right) \text{ subject to } (A, -I, I)\begin{pmatrix} x \\ y_a \\ y_b \end{pmatrix} = 0 \quad y_a \geq 0 \quad y_b \geq 0.$$

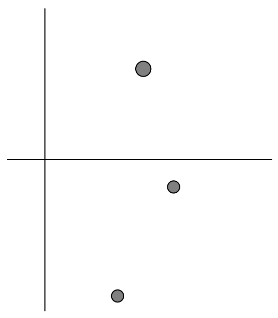
Local communities
In large social networks with a billion vertices, global clustering is likely to result in communities of several hundred million vertices. What you may actually want is a local community containing several individuals with only 50 vertices. To do this if one starts a random walk at a vertex v and computes the frequency of visiting vertices, it will converge to the first singular vector. However, the distribution after a small number of steps will be primarily in the small community containing v and will be proportional to the first singular vector distribution restricted to the vertices in the small community containing v, only will be higher by some constant value. If one wants to determine the local communities containing vertices v_1, v_2, and v_3, start with three probability distributions, one with probability one at v_1, one with probability one at v_2, and one with probability one at v_3 and find early approximation to the first three singular vectors. Then find the minimum 1-norm vector in the space spanned by the early approximations.

Hidden structure
In the previous section we discussed overlapping communities. Another issue is hidden structure. Suppose the vertices of a social network could be partitioned into a number of strongly connected communities. By strongly connected we mean the probability of an edge between two vertices in a community is much higher than the probability of an edge between two vertices in different communities. Suppose the vertices of the graph could be partitioned in another way which was incoherent[35] with the first partitioning and the probability of an edge between two vertices in one of these communities is higher than an edge between two vertices in different communities. If the probability of an edge between two vertices in a community of this second partitioning is less than that in the first, then a clustering algorithm is likely to find the first partitioning rather than the second. However, the second partitioning, which we refer to as hidden structure, may be the structure that we want to find. The way to do this is to use your favorite clustering algorithm to produce the dominant structure and then stochastically weaken the dominant structure by removing some community edges in the graph. Now if you apply the clustering algorithm to the modified graph, it can find the hidden community structure. Having done this, go back to the original graph, weaken the hidden structure and reapply the clustering algorithm. It will now find a better approximation to the dominant structure. This technology can be used to find a number of hidden levels in several types of social networks.

[35]incoherent, give definition

$$\begin{pmatrix} 1&1&1&1&1&1&0&0&0\\ 1&1&1&1&1&1&0&0&0\\ 1&1&1&1&1&1&0&0&0\\ 1&1&1&1&1&1&0&0&0\\ 1&1&1&1&1&1&1&1&1\\ 1&1&1&1&1&1&1&1&1\\ 0&0&0&0&1&1&0&0&0\\ 0&0&0&0&1&1&0&0&0\\ 0&0&0&0&1&1&0&0&0 \end{pmatrix} \qquad \begin{pmatrix} 0.33 & 0.31\\ 0.33 & 0.31\\ 0.33 & 0.31\\ 0.33 & 0.31\\ 0.44 & -0.09\\ 0.44 & -0.09\\ 0.24 & -0.49\\ 0.24 & -0.49\\ 0.24 & -0.49 \end{pmatrix}$$

(a) (b)



(c);

Figure 7.4: (a) illustrates the adjacency matrix of a graph with a six vertex clique that overlaps a five vertex clique in two vertices. (b) illustrates the matrix where columns consist of the top two singular vectors, and (c) illustrates the mapping of rows in the singular vector matrix to three points in two dimensional space. Instead of two cliques we get the non overlapping portion of each of the two clique plus their intersection as communities instead of the two cliques as communities.

Block model

One technique for generating graphs with communities is to use the block model where the vertices are partitioned into blocks and each block is a GNP graph generated with some edge probability. The edges in off diagonal blocks are generated with a lower probability. One can also generate graphs with hidden structure. For example, the vertices in an n vertex graph might be partitioned into two communities, the first community having vertices 1 to $n/2$ and the second community having vertices $n/2+1$ to n. The dominant structure is generated with probability p_1 for edges within communities and probability q_1 for edges between communities. The the vertices are randomly permuted and the hidden structure is generated using the first $n/2$ vertices in the permuted order for one community and the remaining vertices for the second community with probabilities p_2 and q_2 which ar lower than p_1 and q_1..

An interesting question is how to determine the quality of a community found. Many researchers use an existing standard of what the communities are. However, if you want to using clustering techniques to find communities there probably is no external standard or you would just use that instead of clustering. A way to determine if you have found a real community structure is to ask if the graph is more likely generated by a model of the structure found than by a completely random model. Suppose you found a partition of two communities each with $n/2$ vertices. Using the number of edges in each community and the number of inter community edges ask what is the probability of the graph being generated by a bloc model where p and q are the probabilities determined by the edge density within communities and the edge density between communities. One can compare this probability with the probability that the graph was generated by a GNP model with probability $(p+q)/2$.

7.6 Approximation Stability

7.6.1 The Conceptual Idea

We now consider another condition that will allow us to produce accurate clusters from data. To think about this condition, imagine that we are given a few thousand news articles that we want to cluster by topic. These articles could be represented as points in a high-dimensional space (e.g., axes could correspond to different meaningful words, with coordinate i indicating the frequency of that word in a given article). Or, alternatively, it could be that we have developed some text-processing program that given two articles x and y computes some measure of distance $d(x, y)$ between them. We assume there exists some correct clustering $\mathcal{C}_T$ of our news articles into k topics; of course, we do not know what $\mathcal{C}_T$ is, that is what we want our algorithm to find.

If we are clustering with an algorithm that aims to minimize the k-means score of its solution, then implicitly this means we believe that the clustering $\mathcal{C}_{kmeans}^{OPT}$ of minimum k-means score is either equal to, or very similar to, the clustering $\mathcal{C}_T$. Unfortunately, finding the clustering of minimum k-means score is NP-hard. So, let us broaden our belief a bit and assume that any clustering $\mathcal{C}$ whose k-means score is within 10% of the minimum is also very similar to $\mathcal{C}_T$. This should give us a little bit more slack. Unfortunately, finding a clustering of score within 10% of the minimum is also an NP-hard problem. Nonetheless, *we will be able to use this assumption to efficiently find a clustering that is close to $\mathcal{C}_T$*. The trick is that NP-hardness is a worst-case notion, whereas in contrast, this assumption implies structure on our data.In particular, it implies that all clusterings that have score within 10% of the minimum have to be similar to each other. We will then be able to utilize this structure in a natural "ball-growing" clustering algorithm.

7.6.2 Making this Formal

To make this discussion formal, we first specify what we mean when we say that two different ways of clustering some data are "similar" to each other. Let $\mathcal{C} = \{C_1, \ldots, C_k\}$ and $\mathcal{C}' = \{C_1', \ldots, C_k'\}$ be two different k-clusterings of some dataset A. For example, $\mathcal{C}$ could be the clustering that our algorithm produces, and $\mathcal{C}'$ could be the clustering $\mathcal{C}_T$. Let us define the distance between these two clusterings to be the fraction of points that would have to be re-clustered in $\mathcal{C}$ to make $\mathcal{C}$ match $\mathcal{C}'$, where by "match" we mean that there should be a bijection between the clusters of $\mathcal{C}$ and the clusters of $\mathcal{C}'$. We can write this distance mathematically as:

$$dist(\mathcal{C}, \mathcal{C}') = \min_{\sigma} \frac{1}{n} \sum_{i=1}^{k} |C_i \setminus C_{\sigma(i)}'|,$$

where the minimum is over all permutations σ of $\{1, \ldots, k\}$.

For $c > 1$ and $\epsilon > 0$ we say that a data set satisfies (c, ϵ)-*approximation-stability* with respect to a given objective (such as k-means or k-median) if every clustering $\mathcal{C}$

whose cost is within a factor c of the minimum-cost clustering for that objective satisfies $dist(\mathcal{C}, \mathcal{C}_T) < \epsilon$. That is, it is sufficient to be within a factor c of optimal to the our objective in order for the fraction of points clustered incorrectly to be less than ϵ. We will specifically focus in this discussion on the *k-median* objective rather than the k-means objective, since it is a bit easier to work with.

What we will now show is that under this condition, even though it may be NP-hard in general to find a clustering that is within a factor c of optimal, we can nonetheless efficiently find a clustering $\mathcal{C}'$ such that $dist(\mathcal{C}', \mathcal{C}_T) \leq \epsilon$, so long as all clusters in $\mathcal{C}_T$ are reasonably large. To simplify notation, let $\mathcal{C}^*$ denote the clustering of minimum k-median cost, and to keep the discussion simpler, let us also assume that $\mathcal{C}_T = \mathcal{C}^*$; that is, the target clustering is also the clustering with the minimum k-median score.

7.6.3 Algorithm and Analysis

Before presenting an algorithm, we begin with a helpful lemma that will guide our design. For a given data point $\mathbf{a}_i$, define its *weight* $w(\mathbf{a}_i)$ to be its distance to the center of its cluster in $\mathcal{C}^*$. Notice that the k-median cost of $\mathcal{C}^*$ is $OPT = \sum_{i=1}^{n} w(\mathbf{a}_i)$. Define $w_{avg} = OPT/n$ to be the average weight of the points in A. Finally, define $w_2(\mathbf{a}_i)$ to be the distance of $\mathbf{a}_i$ to its second-closest center in $\mathcal{C}^*$.

Lemma 7.6 *Assume dataset A satisfies (c, ϵ) approximation-stability with respect to the k-median objective, each cluster in $\mathcal{C}_T$ has size at least $2\epsilon n$, and $\mathcal{C}_T = \mathcal{C}^*$. Then,*

1. *Fewer than ϵn points $\mathbf{a}_i$ have $w_2(\mathbf{a}_i) - w(\mathbf{a}_i) \leq (c-1)w_{avg}/\epsilon$.*
2. *At most $5\epsilon n/(c-1)$ points $\mathbf{a}_i$ have $w(\mathbf{a}_i) \geq (c-1)w_{avg}/(5\epsilon)$.*

Proof: For part (1), suppose that ϵn points $\mathbf{a}_i$ have $w_2(\mathbf{a}_i) - w(\mathbf{a}_i) \leq (c-1)w_{avg}/\epsilon$. Consider modifying $\mathcal{C}_T$ to a new clustering $\mathcal{C}'$ by moving each of these points $\mathbf{a}_i$ into the cluster containing its second-closest center. By assumption, the k-means cost of the clustering has increased by at most $\epsilon n(c-1)w_{avg}/\epsilon = (c-1) \cdot OPT$. This means that the cost of the new clustering is at most $c\dot{O}PT$. However, $dist(\mathcal{C}', \mathcal{C}_T) = \epsilon$ because (a) we moved ϵn points to different clusters, and (b) each cluster in $\mathcal{C}_T$ has size at least $2\epsilon n$ so the optimal permutation σ in the definition of $dist$ remains the identity. So, this contradicts approximation stability. Part (2) follows from the definition of "average"; if it did not hold then $\sum_{i=1}^{n} w(\mathbf{a}_i) > nw_{avg}$, a contradiction. ∎

A datapoint $\mathbf{a}_i$ is *bad* if it satisfies either item (1) or (2) of Lemma 7.6 and *good* if it satisfies neither one. So, there are at most $b = \epsilon n + \frac{5\epsilon n}{c-1}$ bad points and the rest are good. Define "critical distance" $d_{crit} = \frac{(c-1)w_{avg}}{5\epsilon}$. Lemma 7.6 implies that the good points have distance at most d_{crit} to the center of their own cluster in $\mathcal{C}^*$ and distance at least $5d_{crit}$ to the center of any other cluster in $\mathcal{C}^*$.

This suggests the following algorithm. Suppose we create a graph G with the points $\mathbf{a}_i$ as vertices, and edges between any two points $\mathbf{a}_i$ and $\mathbf{a}_j$ with $d(\mathbf{a}_i, \mathbf{a}_j) < 2d_{crit}$. Notice that by triangle inequality, the good points within the same cluster in $\mathcal{C}^*$ have distance less than $2d_{crit}$ from each other so they will be fully connected and form a clique. Also, again by triangle inequality, any edge that goes between different clusters must be between two bad points. In particular, if $\mathbf{a}_i$ is a good point in one cluster, and it has an edge to some other point $\mathbf{a}_j$, then $\mathbf{a}_j$ must have distance less than $3d_{crit}$ to the center of $\mathbf{a}_i$'s cluster. This means that if $\mathbf{a}_j$ had a different closest center, which obviously would also be at distance less than $3d_{crit}$, then $\mathbf{a}_i$ would have distance less than $2d_{crit} + 3d_{crit} = 5d_{crit}$ to that center, violating its goodness. So, bridges in G between different clusters can only occur between bad points.

Assume now that each cluster in $\mathcal{C}_T$ has size at least $2b+1$; this is the sense in which we are requiring that ϵn be small compared to the smallest cluster in $\mathcal{C}_T$. In this case, create a new graph H by connecting any two points $\mathbf{a}_i$ and $\mathbf{a}_j$ that share at least $b+1$ neighbors in common in G, themselves included. Since every cluster has at least $2b + 1 - b = b + 1$ good points, and these points are fully connected in G, this means that H will contain an edge between every pair of good points in the same cluster. On the other hand, since the only edges in G between different clusters are between bad points, and there are at most b bad points, this means that H will not have any edges between different clusters in $\mathcal{C}_T$. Thus, if we take the k largest connected components in H, these will all correspond to subsets of different clusters in $\mathcal{C}_T$, with at most b points remaining.

At this point we have a correct clustering of all but at most b points in A. Call these clusters $C_1, \ldots, C_k$, where $C_j \subseteq C_j^*$. To cluster the remaining points $\mathbf{a}_i$, we assign them to the cluster C_j that minimizes the median distance between $\mathbf{a}_i$ and points in C_j. Since each C_j has more good points than bad points, and each good point in C_j has distance at most d_{crit} to center $\mathbf{c}_j^*$, by triangle inequality the median of these distances must lie in the range $[d(\mathbf{a}_i,\ \mathbf{c}_i^*) - d_{crit}, d(\mathbf{a}_i, \mathbf{c}_i^*) + d_{crit}]$. This means that this second step will correctly cluster all points $\mathbf{a}_i$ for which $w_2(\mathbf{a}_i) - w(\mathbf{a}_i) > 2d_{crit}$. In particular, we correctly cluster all points except possibly for some of the at most ϵn satisfying item (1) of Lemma 7.6.

The above discussion assumes the value d_{crit} is known to our algorithm; we leave it as an exercise to the reader to modify the algorithm to remove this assumption. Summarizing, we have the following algorithm and theorem.

Algorithm k-Median Stability (given c, ϵ, d_{crit})

1. Create a graph G with a vertex for each datapoint in A, and an edge between vertices i and j if $d(\mathbf{a}_i, \mathbf{a}_j) \leq 2d_{crit}$.

2. Create a graph H with a vertex for each vertex in G and an edge between vertices i and j if i and j share at least $b + 1$ neighbors in common, themselves included, for $b = \epsilon n + \frac{5\epsilon n}{c-1}$. Let $C_1, \ldots, C_k$ denote the k largest connected components in H.

3. Assign each point not in $C_1 \cup \ldots \cup C_k$ to the cluster C_j of smallest median distance.

Theorem 7.7 *Assume A satisfies (c, ϵ) approximation-stability with respect to the k-median objective, that each cluster in $\mathcal{C}_T$ has size at least $\frac{10\epsilon}{c-1}n + 2\epsilon n + 1$, and that $\mathcal{C}_T = \mathcal{C}^*$. Then Algorithm k-*Median Stability *will find a clustering $\mathcal{C}$ such that $dist(\mathcal{C}, \mathcal{C}_T) \leq \epsilon$.*

7.7 High-Density Clusters

We now turn from the assumption that clusters are center-based to the assumption that clusters consist of high-density regions, separated by low-density moats such as in Figure 7.1.

7.7.1 Single Linkage

One natural algorithm for clustering under the high-density assumption is called *single linkage*. This algorithm begins with each point in its own cluster and then repeatedly merges the two "closest" clusters into one, where the distance between two clusters is defined as the *minimum* distance between points in each cluster. That is, $d_{min}(C, C') = \min_{\mathbf{x} \in C, \mathbf{y} \in C'} d(\mathbf{x}, \mathbf{y})$, and the algorithm merges the two clusters C and C' whose d_{min} value is smallest over all pairs of clusters breaking ties arbitrarily. It then continues until there are only k clusters. This is called an *agglomerative* clustering algorithm because it begins with many clusters and then starts merging, or agglomerating them together.[36] Single-linkage is equivalent to running Kruskal's minimum-spanning-tree algorithm, but halting when there are k trees remaining. The following theorem is fairly immediate.

Theorem 7.8 *Suppose the desired clustering $C_1^*, \ldots, C_k^*$ satisfies the property that there exists some distance σ such that*

1. *any two data points in different clusters have distance at least σ, and*
2. *for any cluster C_i^* and any partition of C_i^* into two non-empty sets A and $C_i^* \setminus A$, there exist points on each side of the partition of distance less than σ.*

Then, single-linkage will correctly recover the clustering $C_1^*, \ldots, C_k^*$.

Proof: Consider running the algorithm until all pairs of clusters C and C' have $d_{min}(C, C') \geq \sigma$. At that point, by (2), each target cluster C_i^* will be fully contained within some cluster of the single-linkage algorithm. On the other hand, by (1) and by induction, each cluster C of the single-linkage algorithm will be fully contained within some C_i^* of the target clustering, since any merger of subsets of distinct target clusters would require $d_{min} \geq \sigma$. Therefore, the single-linkage clusters are indeed the target clusters. ∎

[36]Other agglomerative algorithms include *complete linkage* which merges the two clusters whose *maximum* distance between points is smallest, and Ward's algorithm described earlier that merges the two clusters that cause the k-means cost to increase by the least.

7.7.2 Robust Linkage

The single-linkage algorithm is fairly brittle. A few points bridging the gap between two different clusters can cause it to do the wrong thing. As a result, there has been significant work developing more robust versions of the algorithm.

One commonly used robust version of single linkage is Wishart's algorithm. A ball of radius r is created for each point with the point as center. The radius r is gradually increased starting from $r = 0$. The algorithm has a parameter t. When a ball has t or more points the center point becomes active. When two balls with active centers intersect the two center points are connected by an edge. The parameter t prevents a thin string of points between two clusters from causing a spurious merger. Note that Wishart's algorithm with $t = 1$ is the same as single linkage.

In fact, if one slightly modifies the algorithm to define a point to be live if its ball of radius $r/2$ contains at least t points, then it is known [CD10] that a value of $t = O(d \log n)$ is sufficient to recover a nearly correct solution under a natural distributional formulation of the clustering problem. Specifically, suppose data points are drawn from some probability distribution D over R^d, and that the clusters correspond to high-density regions surrounded by lower-density moats. More specifically, the assumption is that

1. for some distance $\sigma > 0$, the σ-interior of each target cluster C_i^* has density at least some quantity λ (the σ-interior is the set of all points at distance at least σ from the boundary of the cluster),
2. the region between target clusters has density less than $\lambda(1 - \epsilon)$ for some $\epsilon > 0$,
3. the clusters should be separated by distance greater than 2σ, and
4. the σ-interior of the clusters contains most of their probability mass.

Then, for sufficiently large n, the algorithm will with high probability find nearly correct clusters. In this formulation, we allow points in low-density regions that are not in any target clusters at all. For details, see [CD10].

Robust Median Neighborhood Linkage robustifies single linkage in a different way. This algorithm guarantees that if it is possible to delete a small fraction of the data such that for all remaining points x, most of their $|C^*(x)|$ nearest neighbors indeed belong to their own cluster $C^*(x)$, then the hierarchy on clusters produced by the algorithm will include a close approximation to the true clustering. We refer the reader to [BLG14] for the algorithm and proof.

7.8 Kernel Methods

Kernel methods combine aspects of both center-based and density-based clustering. In center-based approaches like k-means or k-center, once the cluster centers are fixed, the

Voronoi diagram of the cluster centers determines which cluster each data point belongs to. This implies that clusters are pairwise linearly separable.

If we believe that the true desired clusters may not be linearly separable, and yet we wish to use a center-based method, then one approach, as in the chapter on learning, is to use a kernel. Recall that a kernel function $K(\mathbf{x}, \mathbf{y})$ can be viewed as performing an implicit mapping ϕ of the data into a possibly much higher dimensional space, and then taking a dot-product in that space. That is, $K(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x}) \cdot \phi(\mathbf{y})$. This is then viewed as the affinity between points $\mathbf{x}$ and $\mathbf{y}$. We can extract distances in this new space using the equation $|\mathbf{z_1} - \mathbf{z_2}|^2 = \mathbf{z_1} \cdot \mathbf{z_1} + \mathbf{z_2} \cdot \mathbf{z_2} - 2\mathbf{z_1} \cdot \mathbf{z_2}$, so in particular we have $|\phi(\mathbf{x}) - \phi(\mathbf{y})|^2 = K(\mathbf{x}, \mathbf{x}) + K(\mathbf{y}, \mathbf{y}) - 2K(\mathbf{x}, \mathbf{y})$. We can then run a center-based clustering algorithm on these new distances.

One popular kernel function to use is the Gaussian kernel. The Gaussian kernel uses an affinity measure that emphasizes closeness of points and drops off exponentially as the points get farther apart. Specifically, we define the affinity between points $\mathbf{x}$ and $\mathbf{y}$ by

$$K(\mathbf{x}, \mathbf{y}) = e^{-\frac{1}{2\sigma^2}\|\mathbf{x}-\mathbf{y}\|^2}.$$

Another way to use affinities is to put them in an affinity matrix, or weighted graph. This graph can then be separated into clusters using a graph partitioning procedure such as the one in following section.

7.9 Recursive Clustering based on Sparse Cuts

We now consider the case that data are nodes in an undirected connected graph $G(V, E)$ where an edge indicates that the end point vertices are similar. Recursive clustering starts with all vertices in one cluster and recursively splits a cluster into two parts whenever there is a small number of edges from one part to the other part of the cluster. Formally, for two disjoint sets S and T of vertices, define

$$\Phi(S, T) = \frac{\text{Number of edges from } S \text{ to } T}{\text{Total number of edges incident to } S \text{ in } G}.$$

$\Phi(S, T)$ measures the relative strength of similarities between S and T. Let $d(i)$ be the degree of vertex i and for a subset S of vertices, let $d(S) = \sum_{i \in S} d(i)$. Let m be the total number of edges in the graph. The following algorithm aims to cut only a small fraction of the edges and to produce clusters that are internally consistent in that no subset of the cluster has low similarity to the rest of the cluster.

Recursive Clustering: Select an appropriate value for ϵ. If a current cluster W has a subset S with $d(S) \leq \frac{1}{2}d(W)$ and $\Phi(S, S \subseteq W) \leq \varepsilon$, then split W into two clusters S and $W \setminus S$. Repeat until no such split is possible.

Theorem 7.9 *At termination of Recursive Clustering, the total number of edges between vertices in different clusters is at most $O(\varepsilon m \ln n)$.*

Proof: Each edge between two different clusters at the end was deleted at some stage by the algorithm. We will "charge" edge deletes to vertices and bound the total charge. When the algorithm partitions a cluster W into S and $W \setminus S$ with $d(S) \leq (1/2)d(W)$, each $k \in S$ is charged $\frac{d(k)}{d(W)}$ times the number of edges being deleted. Since $\Phi(S, W \setminus S) \leq \varepsilon$, the charge added to each $k \in W$ is a most $\varepsilon d(k)$. A vertex is charged only when it is in the smaller part, $d(S) \leq d(W)/2$, of the cut. So between any two times it is charged, $d(W)$ is reduced by a factor of at least two and so a vertex can be charged at most $\log_2 m \leq O(\ln n)$ times, proving the theorem. ■

Implementing the algorithm requires computing $\text{Min}_{S \subseteq W} \Phi(S, W \setminus S)$ which is an NP-hard problem. So the theorem cannot be implemented right away. Luckily, eigenvalues and eigenvectors, which can be computed fast, give an approximate answer. The connection between eigenvalues and sparsity, known as Cheeger's inequality, is deep with applications to Markov chains among others. We do not discuss this here.

7.10 Dense Submatrices and Communities

Represent n data points in d-space by the rows of an $n \times d$ matrix A. Assume that A has all nonnegative entries. Examples to keep in mind for this section are the document-term matrix and the customer-product matrix. We address the question of how to define and find efficiently a coherent large subset of rows. To this end, the matrix A can be represented by a bipartite graph Figure 7.5. One side has a vertex for each row and the other side a vertex for each column. Between the vertex for row i and the vertex for column j, there is an edge with weight a_{ij}.

We want a subset S of row vertices and a subset T of column vertices so that

$$A(S, T) = \sum_{i \in S, j \in T} a_{ij}$$

is high. This simple definition is not good since $A(S, T)$ will be maximized by taking all rows and columns. We need a balancing criterion that ensures that $A(S, T)$ is high relative to the sizes of S and T. One possibility is to maximize $\frac{A(S,T)}{|S||T|}$. This is not a good measure either, since it is maximized by the single edge of highest weight. The definition we use is the following. Let A be a matrix with nonnegative entries. For a subset S of rows and a subset T of columns, the *density* $d(S, T)$ of S and T is $d(S, T) = \frac{A(S,T)}{\sqrt{|S||T|}}$. The *density* $d(A)$ of A is defined as the maximum value of $d(S, T)$ over all subsets of rows and columns. This definition applies to bipartite as well as non bipartite graphs.

One important case is when A's rows and columns both represent the same set and a_{ij} is the similarity between object i and object j. Here $d(S, S) = \frac{A(S,S)}{|S|}$. If A is an $n \times n$ 0-1 matrix, it can be thought of as the adjacency matrix of an undirected graph, and $d(S, S)$ is the average degree of a vertex in S. The subgraph of maximum average degree in a graph can be found exactly by network flow techniques, as we will show in the next

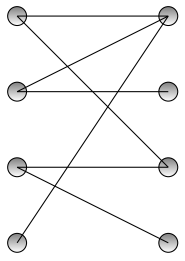


Figure 7.5: Example of a bipartite graph.

section. We do not know an efficient (polynomial-time) algorithm for finding $d(A)$ exactly in general. However, we show that $d(A)$ is within a $O(\log^2 n)$ factor of the top singular value of A assuming $|a_{ij}| \le 1$ for all i and j. This is a theoretical result. The gap may be much less than $O(\log^2 n)$ for many problems, making singular values and singular vectors quite useful. Also, S and T with $d(S,T) \ge \Omega(d(A)/\log^2 n)$ can be found algorithmically.

Theorem 7.10 *Let A be an $n \times d$ matrix with entries between 0 and 1. Then*

$$\sigma_1(A) \ge d(A) \ge \frac{\sigma_1(A)}{4 \log n \log d}.$$

Furthermore, subsets S and T satisfying $d(S,T) \ge \frac{\sigma_1(A)}{4 \log n \log d}$ may be found from the top singular vector of A.

Proof: Let S and T be the subsets of rows and columns that achieve $d(A) = d(S,T)$. Consider an n-vector $\mathbf{u}$ that is $\frac{1}{\sqrt{|S|}}$ on S and 0 elsewhere and a d-vector $\mathbf{v}$ that is $\frac{1}{\sqrt{|T|}}$ on T and 0 elsewhere. Then,

$$\sigma_1(A) \ge \mathbf{u}^T A \mathbf{v} = \sum_{ij} u_i v_j a_{ij} = d(S,T) = d(A)$$

establishing the first inequality.

To prove the second inequality, express $\sigma_1(A)$ in terms of the first left and right singular vectors $\mathbf{x}$ and $\mathbf{y}$.

$$\sigma_1(A) = \mathbf{x}^T A \mathbf{y} = \sum_{i,j} x_i a_{ij} y_j, \qquad |\mathbf{x}| = |\mathbf{y}| = 1.$$

Since the entries of A are nonnegative, the components of the first left and right singular vectors must all be nonnegative, that is, $x_i \ge 0$ and $y_j \ge 0$ for all i and j. To bound $\sum_{i,j} x_i a_{ij} y_j$, break the summation into $O(\log n \log d)$ parts. Each part corresponds to a given α and β and consists of all i such that $\alpha \le x_i < 2\alpha$ and all j such that $\beta \le y_i < 2\beta$. The $\log n \log d$ parts are defined by breaking the rows into $\log n$ blocks with α equal to $\frac{1}{2}\frac{1}{\sqrt{n}}$, $\frac{1}{\sqrt{n}}$, $2\frac{1}{\sqrt{n}}$, $4\frac{1}{\sqrt{n}}$, ... , 1 and by breaking the columns into $\log d$ blocks with β equal to $\frac{1}{2}\frac{1}{\sqrt{d}}$, $\frac{1}{\sqrt{d}}$, $\frac{2}{\sqrt{d}}$, $\frac{4}{\sqrt{d}}$, $\ldots$, 1. The i such that $x_i < \frac{1}{2\sqrt{n}}$ and the j such that $y_j < \frac{1}{2\sqrt{d}}$ will be ignored at a loss of at most $\frac{1}{4}\sigma_1(A)$. Exercise 7.27 proves the loss is at most this amount.

Since $\sum\limits_i x_i^2 = 1$, the set $S = \{i|\alpha \le x_i < 2\alpha\}$ has $|S| \le \frac{1}{\alpha^2}$ and similarly, $T = \{j|\beta \le y_j \le 2\beta\}$ has $|T| \le \frac{1}{\beta^2}$. Thus

$$\begin{aligned}\sum_{\substack{i\\ \alpha\le x_i\le 2\alpha}} \sum_{\substack{j\\ \beta\le y_j\le 2\beta}} x_i y_j a_{ij} &\le 4\alpha\beta A(S,T)\\ &\le 4\alpha\beta d(S,T)\sqrt{|S||T|}\\ &\le 4d(S,T)\\ &\le 4d(A).\end{aligned}$$

From this it follows that

$$\sigma_1(A) \le 4d(A)\log n \log d$$

or

$$d(A) \ge \frac{\sigma_1(A)}{4\log n \log d}$$

proving the second inequality.

It is clear that for each of the values of (α, β), we can compute $A(S,T)$ and $d(S,T)$ as above and taking the best of these $d(S,T)$'s gives us an algorithm as claimed in the theorem. ∎

Note that in many cases, the nonzero values of x_i and y_j after zeroing out the low entries will only go from $\frac{1}{2}\frac{1}{\sqrt{n}}$ to $\frac{c}{\sqrt{n}}$ for x_i and $\frac{1}{2}\frac{1}{\sqrt{d}}$ to $\frac{c}{\sqrt{d}}$ for y_j, since the singular vectors are likely to be balanced given that a_{ij} are all between 0 and 1. In this case, there will be $O(1)$ groups only and the log factors disappear.

Another measure of density is based on similarities. Recall that the similarity between objects represented by vectors (rows of A) is defined by their dot products. Thus, similarities are entries of the matrix AA^T. Define the average cohesion $f(S)$ of a set S of rows of A to be the sum of all pairwise dot products of rows in S divided by $|S|$. The average cohesion of A is the maximum over all subsets of rows of the average cohesion of the subset.

Since the singular values of AA^T are squares of singular values of A, we expect $f(A)$ to be related to $\sigma_1(A)^2$ and $d(A)^2$. Indeed it is. We state the following without proof.

Lemma 7.11 *$d(A)^2 \le f(A) \le d(A)\log n$. Also, $\sigma_1(A)^2 \ge f(A) \ge \frac{c\sigma_1(A)^2}{\log n}$.*

$f(A)$ can be found exactly using flow techniques as we will see later.

7.11 Community Finding and Graph Partitioning

Assume that data are nodes in a possibly weighted graph where edges represent some notion of affinity between their endpoints. In particular, let $G = (V, E)$ be a weighted graph. Given two sets of nodes S and T, define

$$E(S,T) = \sum_{\substack{i \in S \\ j \in T}} e_{ij}.$$

We then define the *density* of a set S to be

$$d(S,S) = \frac{E(S,S)}{|S|}.$$

If G is an undirected graph, then $d(S,S)$ can be viewed as the average degree in the vertex-induced subgraph over S. The set S of maximum density is therefore the subgraph of maximum average degree. Finding such a set can be viewed as finding a tight-knit community inside some network. In the next section, we describe an algorithm for finding such a set using network flow techniques.

Flow Methods Here we consider dense induced subgraphs of a graph. An induced subgraph of a graph consisting of a subset of the vertices of the graph along with all edges of the graph that connect pairs of vertices in the subset of vertices. We show that finding an induced subgraph with maximum average degree can be done by network flow techniques. This is simply maximizing the density $d(S,S)$ over all subsets S of the graph. First consider the problem of finding a subset of vertices such that the induced subgraph has average degree at least λ for some parameter λ. Then do a binary search on the value of λ until the maximum λ for which there exists a subgraph with average degree at least λ is found.

Given a graph G in which one wants to find a dense subgraph, construct a directed graph H from the given graph and then carry out a flow computation on H. H has a node for each edge of the original graph, a node for each vertex of the original graph, plus two additional nodes s and t. There is a directed edge with capacity one from s to each node corresponding to an edge of the original graph and a directed edge with infinite capacity from each node corresponding to an edge of the original graph to the two nodes corresponding to the vertices the edge connects. Finally, there is a directed edge with capacity λ from each node corresponding to a vertex of the original graph to t.

Notice there are three types of cut sets of the directed graph that have finite capacity, Figure 7.6. The first cuts all arcs from the source. It has capacity e, the number of edges of the original graph. The second cuts all edges into the sink. It has capacity λv, where v is the number of vertices of the original graph. The third cuts some arcs from s and some arcs into t. It partitions the set of vertices and the set of edges of the original graph into two blocks. The first block contains the source node s, a subset of the edges e_s, and a

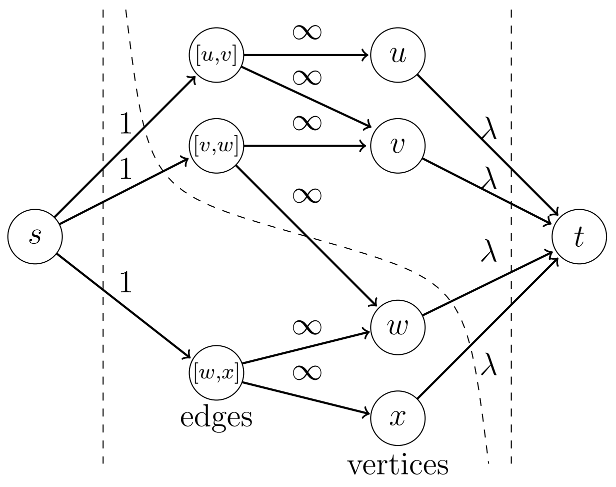


Figure 7.6: The directed graph H used by the flow technique to find a dense subgraph

subset of the vertices v_s defined by the subset of edges. The first block must contain both end points of each edge in e_s; otherwise an infinite arc will be in the cut. The second block contains t and the remaining edges and vertices. The edges in this second block either connect vertices in the second block or have one endpoint in each block. The cut set will cut some infinite arcs from edges not in e_s coming into vertices in v_s. However, these arcs are directed from nodes in the block containing t to nodes in the block containing s. Note that any finite capacity cut that leaves an edge node connected to s must cut the two related vertex nodes from t, Figure 7.6. Thus, there is a cut of capacity $e - e_s + \lambda v_s$ where v_s and e_s are the vertices and edges of a subgraph. For this cut to be the minimal cut, the quantity $e - e_s + \lambda v_s$ must be minimal over all subsets of vertices of the original graph and the capcity must be less than e and also less than λv.

If there is a subgraph with v_s vertices and e_s edges where the ratio $\frac{e_s}{v_s}$ is sufficiently large so that $\frac{e_S}{v_S} > \frac{e}{v}$, then for λ such that $\frac{e_S}{v_S} > \lambda > \frac{e}{v}$, $e_s - \lambda v_s > 0$ and $e - e_s + \lambda v_s < e$. Similarly $e < \lambda v$ and thus $e - e_s + \lambda v_s < \lambda v$. This implies that the cut $e - e_s + \lambda v_s$ is less than either e or λv and the flow algorithm will find a nontrivial cut and hence a proper subset. For different values of λ in the above range there maybe different nontrivial cuts.

Note that for a given density of edges, the number of edges grows as the square of the number of vertices and $\frac{e_s}{v_s}$ is less likely to exceed $\frac{e}{v}$ if v_S is small. Thus, the flow method works well in finding large subsets since it works with $\frac{e_S}{v_S}$. To find small communities one would need to use a method that worked with $\frac{e_S}{v_S^2}$ as the following example illustrates.

Example: Consider finding a dense subgraph of 1,000 vertices and 2,000 internal edges in a graph of 10^6 vertices and $6{\times}10^6$ edges. For concreteness, assume the graph was generated by the following process. First, a 1,000-vertex graph with 2,000 edges was generated as a

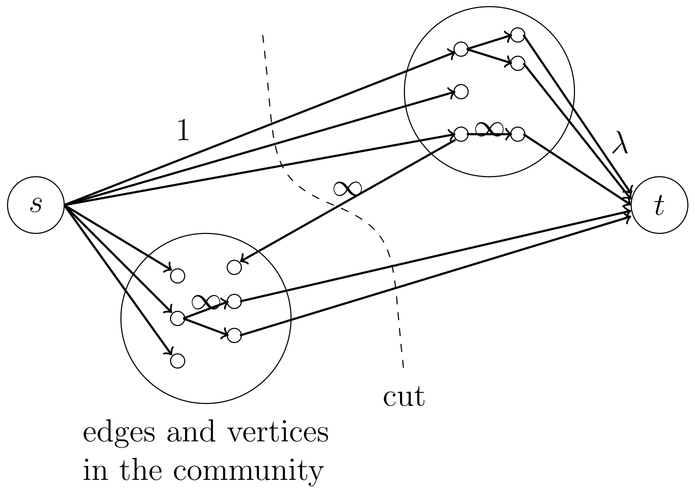


Figure 7.7: Cut in flow graph

random regular degree four graph. The 1,000-vertex graph was then augmented to have 10^6 vertices and edges were added at random until all vertices were of degree 12. Note that each vertex among the first 1,000 has four edges to other vertices among the first 1,000 and eight edges to other vertices. The graph on the 1,000 vertices is much denser than the whole graph in some sense. Although the subgraph induced by the 1,000 vertices has four edges per vertex and the full graph has twelve edges per vertex, the probability of two vertices of the 1,000 being connected by an edge is much higher than for the graph as a whole. The probability is given by the ratio of the actual number of edges connecting vertices among the 1,000 to the number of possible edges if the vertices formed a complete graph.

$$p = \frac{e}{\left(\binom{v}{2}\right)} = \frac{2e}{v(v-1)}$$

For the 1,000 vertices, this number is $p = \frac{2\times 2{,}000}{1{,}000\times 999} \cong 4\times 10^{-3}$. For the entire graph this number is $p = \frac{2\times 6\times 10^6}{10^6\times 10^6} = 12\times 10^{-6}$. This difference in probability of two vertices being connected should allow us to find the dense subgraph. ∎

In our example, the cut of all arcs out of s is of capacity 6×10^6, the total number of edges in the graph, and the cut of all arcs into t is of capacity λ times the number of vertices or $\lambda\times 10^6$. A cut separating the 1,000 vertices and 2,000 edges would have capacity $6\times 10^6 - 2,000 + \lambda\times 1,000$. This cut cannot be the minimum cut for any value of λ since $\frac{e_s}{v_s} = 2$ and $\frac{e}{v} = 6$, hence $\frac{e_s}{v_s} < \frac{e}{v}$. The point is that to find the 1,000 vertices, we have to maximize $A(S,S)/|S|^2$ rather than $A(S,S)/|S|$. Note that $A(S,S)/|S|^2$ penalizes large |S| much more and therefore can find the 1,000 node "dense" subgraph.

$$A = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix} \qquad V = \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{pmatrix}$$

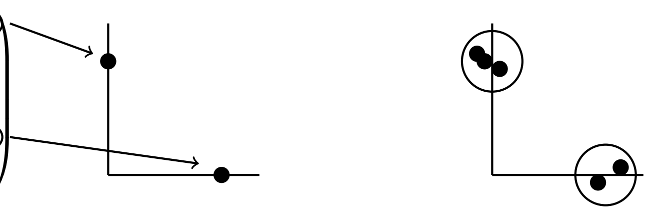


Figure 7.8: Illustration of spectral clustering.

7.12 Spectral clustering applied to social networks

Finding communities in social networks is different from other clustering for several reasons. First we often want to find communities of size say 20 to 50 in networks with 100 million vertices. Second a person is in a number of overlapping communities and thus we are not finding disjoint clusters. Third there often are a number of levels of structure and a set of dominant communities may be hiding a set of weaker communities that are of interest. Spectral clustering is one approach to these issues.

In spectral clustering of the vertices of a graph, one creates a matrix V whose columns correspond to the first k singular vectors of the adjacency matrix. Each row of V is the projection of a row of the adjacency matrix to the space spanned by the k singular vectors. In the example below, the graph has five vertices divided into two cliques, one consisting of the first three vertices and the other the last two vertices. The top two right singular vectors of the adjacency matrix, not normalized to length one, are $(1, 1, 1, 0, 0)^T$ and $(0, 0, 0, 1, 1)^T$. The five rows of the adjacency matrix projected to these vectors form the 5×2 matrix in Figure 7.8. Here, there are two ideal clusters with all edges inside a cluster being present including self-loops and all edges between clusters being absent. The five rows project to just two points, depending on which cluster the rows are in. If the clusters were not so ideal and instead of the graph consisting of two disconnected cliques, the graph consisted of two dense subsets of vertices where the two sets were connected by only a few edges, then the singular vectors would not be indicator vectors for the clusters but close to indicator vectors. The rows would be mapped to two clusters of points instead of two points. A k-means clustering algorithm would find the clusters.

If the clusters were overlapping, then instead of two clusters of points, there would be three clusters of points where the third cluster corresponds to the overlapping vertices of the two clusters. Instead of using k-means clustering, we might instead find the minimum 1-norm vector in the space spanned by the two singular vectors. The minimum 1-norm vector will not be an indicator vector, so we would threshold its values to create an indicator vector for a cluster. Instead of finding the minimum 1-norm vector in the space spanned by the singular vectors in V, we might look for a small 1-norm vector close to the subspace.

$$\min_{\mathbf{x}}(1 - |\mathbf{x}|_1 + \alpha \cos(\theta))$$

Here θ is the cosine of the angle between $\mathbf{x}$ and the space spanned by the two singular vectors. α is a control parameter that determines how close we want the vector to be to the subspace. When α is large, $\mathbf{x}$ must be close to the subspace. When α is zero, $\mathbf{x}$ can be anywhere.

Finding the minimum 1-norm vector in the space spanned by a set of vectors can be formulated as a linear programming problem. To find the minimum 1-norm vector in V, write $V\mathbf{x} = \mathbf{y}$ where we want to solve for both $\mathbf{x}$ and $\mathbf{y}$. Note that the format is different from the usual format for a set of linear equations $A\mathbf{x} = \mathbf{b}$ where $\mathbf{b}$ is a known vector.

Finding the minimum 1-norm vector looks like a nonlinear problem.

$$\min |\mathbf{y}|_1 \text{ subject to } V\mathbf{x} = \mathbf{y}$$

To remove the absolute value sign, write $\mathbf{y} = \mathbf{y_1} - \mathbf{y_2}$ with $\mathbf{y_1} \geq 0$ and $\mathbf{y_2} \geq 0$. Then solve

$$\min \left(\sum_{i=1}^{n} y_{1i} + \sum_{i=1}^{n} y_{2i} \right) \text{ subject to } V\mathbf{x} = \mathbf{y},\ \mathbf{y_1} \geq 0, \text{ and } \mathbf{y}_2 \geq 0.$$

Write $V\mathbf{x} = \mathbf{y_1} - \mathbf{y_2}$ as $V\mathbf{x} - \mathbf{y_1} + \mathbf{y_2} = 0$. then we have the linear equations in a format we are accustomed to.

$$[V, -I, I] \begin{pmatrix} \mathbf{x} \\ \mathbf{y_1} \\ \mathbf{y_2} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

This is a linear programming problem. The solution, however, happens to be $\mathbf{x} = 0$, $\mathbf{y_1} = 0$, and $\mathbf{y_2} = 0$. To resolve this, add the equation $y_{1i} = 1$ to get a community containing the vertex i.

Often we are looking for communities of 50 or 100 vertices in graphs with hundreds of million of vertices. We want a method to find such communities in time proportional to the size of the community and not the size of the entire graph. Here spectral clustering can be used but instead of calculating singular vectors of the entire graph, we do something else. Consider a random walk on a graph. If we walk long enough the probability distribution converges to the first eigenvector. However, if we take only a few steps from a start vertex or small group of vertices that we believe define a cluster, the probability will distribute over the cluster with some of the probability leaking out to the remainder of the graph. To get the early convergence of several vectors that ultimately converge to the first few singular vectors, take a subspace $[\mathbf{x}, A\mathbf{x}, A^2\mathbf{x}, A^3\mathbf{x}]$ and propagate the subspace. At each iteration find an orthonormal basis and then multiply each basis vector by A. Then take the resulting basis vectors after a few steps, say five, and find a minimum 1-norm vector in the subspace.

A third issue that arises is when a dominant structure hides an important weaker structure. One can run their algorithm to find the dominant structure and then weaken the dominant structure by randomly removing edges in the clusters so that the edge density is similar to the remainder of the network. Then reapplying the algorithm often will uncover weaker structure. Real networks often have several levels of structure. The technique can also be used to improve state of the art clustering algorithms. After weakening the dominant structure to find the weaker hidden structure one can go back to the original data and weaken the hidden structure and reapply the algorithm to again find the dominant structure. This improves most state of the art clustering algorithms.