Clustering

compiled by Alvin Wan from Professor Benjamin Recht's lecture, Samaneh's discussion

1 Overview

With clustering, we have several key motivations:

- archetypes (factor analysis)
- segmentation
- hierarchy
- faster lookups (quantization)

It's not trivial to choose an objective to minimize. In PCA, the algorithm was fixed, regardless of the objective. With clustering, different objective can result in a different algorithms. There is no preferred way to do clustering, but we will explore several popular methods in this note. Here are three approaches to consider:

- k-means (quantization)
- agglomeration (hierarchy)
- spectral (segmentation)

2 K-Means Clustering

In k-means clustering, we segment our data by describing each data point using a centroid μ_i. In other words, x_i is in cluster j if x_i to closer to cluster j than any other cluster, $\|x_i - \mu_j\| < \|x_i - \mu_{j'}\|$ for $j \neq j'$. Given centroids, this is how we assign points to clusters. The question is now: how do we pick centroids? We have the following optimization problem:

$$\textsc{Minimize}_{\mu_1,\mu_2,\ldots\mu_k} \sum_{i=1}^{n} \textsc{Min}_{1\leq j_i\leq k} \|x_i - \mu_{j_i}\|^2$$

(j_i is an index) This is effectively an SVM, where we're fitting parameters to some loss function. As it turns out, minimizing this cost is NP-hard.

Explore "I want hue"?

2.1 Lloyd's Algorithm

The following is called **alternating minimization**. If we fix the cluster assignments, the problem becomes easy. If the cluster assignment is fixed, the objective is a convex function. Then, if we fix the means, then we can easily cluster. In the following algorithm, we then alternately fix the cluster assignments or the means and minimize over the other.

1. Initialize $\mu_1, \ldots, \mu_k$.
2. Assign each point to the jth cluster if it is closest to j. For $i = 1, \ldots n$, assign x_i to C_j if $\|x_i - \mu_j\|^2 \leq \|x_2 - \mu_{j'}\| \forall j' \neq j$.
3. If not assignments changed, then return.
4. Assign the mean to the mean of the cluster. $\mu_j \leftarrow \frac{1}{|C_j|} \sum_{i \in C_j} x_i$.
5. Go back to 1.

The number of clusters is in fact a hyper-parameter for this algorithm. How do we initialize μ_i? We have a few options:

- Pick $\mu_1, \mu_2, \ldots \mu_k$ at random.
- Initialize using k-means++. (*See stronger results by Schulman, Rabani, Swarmy, Ostrovski.*)
  - Set μ_1 to be randomly-selected x_i. In high-dimensional space, a randomly-selected point may easily be distant from our data.
  - For $c = 1, \ldots k-1$, for $i = 1, \ldots, n$, $d_i = \textsc{Min}_{1 \leq j \leq C} \|x_i - \mu_j\|^2$.
  - $z = \sum d_i$
  - For $i = 1, \ldots, n$, $p_i = \frac{d_i}{z}$. Set $\mu_{c+1} = x_i$ with probability p_i.
  - If the distance is large, we have a high probability of picking that point. We have 0 probability of picking the original point.

As it turns out, if there exists a good clustering, and we know the number of clusters, this algorithm is guaranteed to find that clustering.

3 Hierarchical Clustering

Previously, we had a top-down approach, where we took clusters and then assigned samples. Here, we take a bottom-up approach; we form clusters incrementally. Take clusters of 2, merge the pairs, then the quadruples etc. This inherently gives us a hierarchy. Let us define one possible distance metric, called **average linkage**:

$$d(A, B) = \frac{1}{|A||B|} \sum_{a \in A} \sum_{a \in B} \textsc{Dist}(a, b)$$

We can also define **centroid linkage**, where $\mu_A = \sum_{a \in A} a$.

$$d(A, B) = \textsc{Dist}(\mu_A, \mu_B)$$

We could similarly and arbitrarily apply any valid metric:

$$d(A, B) = \textsc{Max}(\textsc{Dist}(a, b) : a \in A, b \in B)$$

3.1 Greedy Algorithm

1. Initialize with n clsuters, $C_i = \{x_i\}$.
2. Repeat.
3. For all pairs of clusters (A, B), compute $d(A, B)$
4. $C_{new} = A \cup B$, where $d(A, B)$ is minimized.

The **dendogram** represents our steps to union each set of clusters.

We can examine a random greedy algorithm

- Choose A uniformly at random
- $C_{new} = A \cup B$, where B is closest to A.

This reduces runtime from n^3 to n^2 an often produces more stable results.

4 Spectral Clustering

View data as a graph, where our *nodes* are data points $x_1, \ldots x_n$, and *edges* are w_{ij}, which denote similarity of two data points, $\textsc{Sim}(x_i, x_j)$. Here are a few sample similarity functions.

- cosine similarity: $\frac{x^T x_j}{\|x_i\|\|x_j\|}$
- a kernel function $k(x_i, x_j)$
- $\begin{cases} 1 & \|x_i - x_j\| \leq D_0 \\ 0 & \text{otherwise} \end{cases}$

4.1 Cuts

As it turns out, we can convert clustering into a graph partition problem. Let us formalize the problem parameters. Our goal is find *cut* for our graph. Let V be the set of all nodes, then our partitions V_1, V_2 must satisfy the following.

- $V_1 \cup V_2 = V$
- $V_1 \cap V_2 = \emptyset$

The number of cuts is $\textsc{Cut}(V_1, V_2) = \sum_{i \in V_1} \sum_{j \in V_2} w_{ij}$. However, we can find a trivial solution that minimizes the number of cuts, which is to consider $V_1 = V, V_2 = \emptyset$. So, we introduce a penalty term to make a **balanced cut**.

$$\textsc{MinimizeCut}(V_1, V_2)$$

subject to $|V_1| = |V_2| = \frac{n}{2}$. (We ignore the odd case for now.) This problem is also NP-hard. We are now going to transform a discrete problem into a continuous problem.

4.2 Graph Laplacian

We have several types of matrices that describe the structure of a graph.

- adjacency matrix (A): $A_{ij} = 1$ if i, j connected and 0 otherwise
- affinity matrix (W): entries are $s(i, j)$ if i, j connected and 0 otherwise (no self-loops, so diagonal entries are 0)
- degree matrix (D): In the derivation below, D is a diagonal matrix with sums of the
- Laplacian matrix ($L = D - W$): symmetric, PSD, always has $\lambda_i = 0, v_i = 1$

Let us call $\textsc{Mass}(G_1)$ the number of nodes in G_1, or $|V_1|$. We wish to find 2 or more parittions of similar sizes, where we cut edges with low weight. We can see that our problem can be formally expressed as the following.

$$\textsc{Minimize} \frac{\textsc{Cut}(G_1, G_2)}{\textsc{Mass}(G_1)\textsc{Mass}(G_2)}$$

4.3 Minimizing the Cut

Let us first define the cut indicator.

$$v_i = \begin{cases} 1 & i \in V_1 \\ -1 & i \in V_2 \end{cases}$$

We can then define a cut indicator, which tells us if i, j is in the cut.

$$\textsc{Cut}(V_1, V_2) = \frac{1}{4} \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij}(v_i - v_j)^2$$

If the weight is high, we want nodes to be closer together, and if the weight is low, nodes are repelled. As it turns out, we can simplify this expression.

$$\begin{aligned}
\textsc{Cut}(V_1, V_2) &= \sum_{i \in G_1} \sum_{j \in G_2} w_{ij} \\
&= \sum_{(i,j) \in E} \frac{1}{4} w_{ij} (y_i - y_j)^2 \\
&= \frac{1}{4} \sum_{(i,j) \in E} (w_{ij} y_i^2 - 2 w_{ij} y_i y_j + w_{ij} y_j^2) \\
&= \frac{1}{4} \sum_{(i,j) \in E} (-2 w_{ij} y_i y_j) + \frac{1}{4} \sum_{(i,j) \in E} (w_{ij} y_i^2 + w_{ij} y_j^2)
\end{aligned}$$

In the second summation, we sum over all edges in the cut w_{ij}, adding weight for both vertices i, j. This is equivalent to summing over all vertices in the cut, and for each vertex, adding all weights for edges in the cut.

$$\begin{aligned}
\textsc{Cut}(V_1, V_2) &= \frac{1}{4} \sum_{(i,j) \in E} (-2 w_{ij} y_i y_j) + \sum_{i=1}^{n} y_i^2 \sum_{k=1}^{n} w_{ik} \\
&= \frac{1}{4} v^T (D - W) v \\
&= \frac{1}{4} v^T L v
\end{aligned}$$

where $L_{ij} = \begin{cases} -w_{ij} & i \neq j \\ \sum_k w_{ik} & i = j \end{cases}$. L is known as the Graph Laplacian. This, like the adjacency matrix, can uniquely identify a graph. We know a few properties about this matrix L.

- L is symmetric.
- L is positive semidefinite, if $w_{ij} > 0$. Since all terms are squared and non-negative, $v^T L v \geq 0, \forall v$.
- $L1 = 0$, where 1 is the vector of 1s.

We thus have a new objective.

$$\textsc{Minimize} \frac{1}{4} v^T L v$$

such that $v_i \in \{-1, 1\}, \mathbb{1}^T v = 0$. To make this more explicit, note that along the diagonal of L, we have $\sum_j w_{ij}$. Since $w_{ii} = 0$, then we have that this sum is equal to the sum of all other terms in that row. Thus, $L\mathbb{1} = 0$. Since $v \neq 0$, $\lambda = 0$.

We claim only one such λ exists. Note that if $y = \mathbb{1}$, $Ly = 0$ and $\textsc{Cut}(G_1, G_2) = y^T L y = 0$, and all nodes are in G_1 or G_2.

Proof: Assume for contradiction that another $v_2 \neq 1$ so that $\lambda_2 = 0$. So $Lv_2 = 0 = \lambda_2 v_2 \implies \textsc{Cut}(G_1, G_2) = 0$. We know $v_2^T L v_2 = 0$, and thus the graph is still connected. This means $v_2 = \mathbb{1}$. Contradiction.

Note that this minimization problem is the exact same problem as the one proposed earlier. The only difference is that this for continuous-valued numbers. Now, we make an approximation. Instead, we will subject our problem to $\|v\|^2 = n$ and $1^T v = 0$. As it turns out, the solution to this minimization problem is the second-smallest eigenvalue. If $1^T v = 0$ was not added, the solution would be the first eigenvalue.

There are a variety of other related to the Graph Laplacian - the normalized cut, maximum cut etc. All of these are NP-hard.

4.4 Minimizing the Masses

Now, let us consider the denominator. We need to additionally constrian the sizes of the partitions to be similar. How can we ensure that $|V_1| = |V_2| = \frac{n}{2}$. We want thte sum of all entries in v to be 0. So, $\mathbb{1}^T y = 0$. The problem is formally

$$\textsc{Minimize} v^T L v$$

subject to the constraint that $\forall i, y_i = 1$ or $y_i = -1$. Consider a two-dimensional representation, on only y_1, y_2. Plotting all combinations of $\{1, -1\}$, we have the corners of a square. We can loosen this constraint so that y_1, y_2 are anywhere on the circle that passes through

all corners of the square. This is a circle of radius $\sqrt{2} = \sqrt{n}$. Generalizing to n, we can relax this constraint to $\|v\|_2^2 = n$ or identically, $\mathbb{1}^T v = 0$. Without any constraints, note that

$$\textsc{Minimize} \frac{v^T L v}{v^T v} = \lambda_{min}(L) = 0$$

So, v_1 is not a solution. However, we note that $v_1 = \mathbb{1}$. Note that v_2 is orthogonal to v_1, so v_2 satisfies the constraint. Our solution is thus the second-smallest eigenvalue.

Consider now the ellipsoid, $\{x : x^T A x = 1\}$. Our semi-axis length is given by $\frac{1}{\sqrt{\lambda_i}}$. Our principal directions are given by v_i. When $A = L$, we have an eigenvalue of $\lambda_i = 0$, so we have one axis with length infinity. Seen geometrically, this is a cylinder, where the length of the cylinder runs along v_1. Since we want to $v_1^T v = 0$, then we want v to be orthogonal to v_1. This is a hyperplane orthogonal to v_1. Per before, we want $\|v\|_2^2 = n$. The constraint in three-dimensional space is a sphere. Thus, we are looking for the intersection of the hyperplane with the sphere. This is precisely v_2.