EE 381V: Large Scale Learning **Spring 2013**

Lecture 3 — January 22

Lecturer: Caramanis & Sanghavi *Scribe: Subhashini Krishnasamy*

3.1 Clustering

In the last lecture, we saw Locality Sensitive Hashing (LSH) which uses hash functions to map high dimensional data to points in lower dimensional space. We will now study a closely related learning problem called *Clustering* that also involves dimensionality reduction. Examples of large dimensional data applications include recommendation engines (grouping of consumers/consumer products), search engines (grouping of files/sites), and the Sloan Sky Survey Project that catalogues millions of sky objects with the aim of constructing a 3-D map of the sky.

Learning algorithms can be classified based on the data available. While *Supervised learning* uses labelled data (sample input and output points) to learn the mapping function, *Unsupervised learning* methods like clustering try to find structure in unlabelled data. More specifically, the problem of clustering is to partition a set n data points $x_1, x_2, \ldots, x_n \in \mathcal{X}$ into k groups based on some notion of similarity. We will study clustering algorithms for two kind of models -

1. Probabilistic Model: In this kind of models, it is assumed that each of the data points was independently generated from a mixture of k distributions. The conditional densities are modelled with cluster dependent parameters, and the algorithms then try to solve the maximum likelihood problem.

$$X_i \sim p(x|c) = \frac{1}{k} \sum_{a=1}^{k} q(x|c_a),$$

   where c_a are the cluster dependent parameters. The goal is to maximize $\prod_{i=1}^{n} p(x_i|c)$.

2. Optimization Model: Some algorithms consider the problem in an optimization framework where the goal is to maximize agreements/similarities within clusters. Given some pseudo-distance metric $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, partition the points such that the sum of distances to the cluster centers is minimized. The center of a group $\{x_i\}_{i \in S}$ is defined as follows -

$$c(S) := \arg\min_{z} \{\sum_{i \in S} d(x_i, z) : z \in \mathcal{X}\}.$$

The problem is to find $S = S_1 \cup S_2 \cup \cdots \cup S_K$ that minimizes $\mathcal{C}(S) = \sum_{a=1}^{k} \sum_{i \in S_a} d(x_i, c(S_a))$. This optimization problem is NP-hard even for points on a 2-D plane, and so we consider approximate algorithms.

3.2 k-means Algorithm

k-means is an approximate algorithm that aims to solve the optimization problem above. The algorithm is given below.

```
Data: $x_1, x_2, \ldots, x_n$
Result: $S = S_1 \cup S_2 \cup \cdots \cup S_K$
Initialize partition $S = S_1 \cup S_2 \cup \cdots \cup S_K$.
while $\mathcal{C}(S)$ improves do
    1. Centers update: Compute the cluster centers of the partition,
    $c(S_a) := \arg\min_z \{\sum_{i \in S_a} d(x_i, z) : z \in \mathcal{X}\}$.
    2. Cluster Assignment: Compute the partition induced by the centers,
    $S_a = \{i \in [n] : a = \arg\min_{b \in [k]} d(x_i, c_b)\}$.
    Compute the objective $\mathcal{C}(S)$.
end
```

Algorithm 1: k-means

Computing the partitions is straight forward; it is the update of the cluster centers that drives the computational cost. The k-means algorithm is popular in applications with distance metric that allows easily computable cluster centers. Some examples are given below.

1. Euclidean distance: If $\mathcal{X} = \mathbb{R}^m$ and $d(x, y) = ||x - y||_2^2 = \sum_{i=1}^{m} (x_i - y_i)^2$, then

$$c(S) = \frac{1}{|S|} \sum_{i \in S} x_i.$$

2. Spherical distance: If $\mathcal{X} = \mathbb{S}^{m-1} = \{x \in \mathbb{R}^m : ||z||_2 = 1\}$ (all points lie on a unit sphere) and $d(x, y) = 1 - < x, y >$, then

$$\begin{aligned} c(S) &= \arg\min_z \{\sum_{i \in S} d(x_i, z) : ||z||_2 = 1\} \\ &= \arg\max_z \{< \sum_{i \in S} x_i, z >: ||z||_2 = 1\} \\ &= \frac{\tilde{c}(S)}{||\tilde{c}(S)||_2}, \end{aligned}$$

where $\tilde{c}(S) = \sum_{i \in S} x_i$.

3. KL divergence: If $\mathcal{X} = \mathbb{P}^{m-1} = \{x \in \mathbb{R}^m : x > 0, \quad ||x||_1 = 1\}$ (all points lie on the probability simplex) and $d(x, y) = D(x||y) =< x, \log(x/y) >$, then
$$c(S) = \frac{1}{|S|} \sum_{i \in S} x_i.$$

For all the three distance metrics, the cluster center is easy to compute as the cost is only that of vector addition. But it should be noted that the cluster centers need not be one of the data points and might not be meaningful in the context of the application. Also, Euclidean distance (which is a convex function) gives large weights to larger distances, and therefore k-means with this distance measure is not very robust to outliers. An alternative algorithm is the k-medoids algorithm which forces the cluster centers to be one of the data points. Here again, the most computationally intensive step is the determination of cluster centers given a partition of the data points.

Theorem 3.1. *k-means terminates after at most k^n iterations.*

Proof: The cost $\mathcal{C}(S)$ decreases monotonically at each iteration, and stabilizes once successive partitions coincide. This is true because the algorithm alternately minimizes the cost over all partitions and over all cluster centers. Since there are at most k^n possible partitions, and the algorithm does not get trapped in cycles, it terminates within k^n iterations. □

The main drawback of the k-means algorithm is that the solution may not be the global optimum, and it may take a long time to converge. It is known to be NP-hard in $\mathbb{R}^d$ for a general d and $k = 2$, and for 2-D plane and a general k. For a fixed d and k, polynomial time algorithms exist. A variant of k-means is the Incremental k-means algorithm. A drawback of this algorithm is that it may not converge, or it may converge to a local optimum.

```
Data: $x_1, x_2, \ldots, x_n$
Result: $S = S_1 \cup S_2 \cup \cdots \cup S_K$
Initialize partition $S = S_1 \cup S_2 \cup \cdots \cup S_K$.
while $\mathcal{C}(S)$ improves do
    For each $a, b \in [k]$, each $i \in S_a$, shift $i$ to $S_b$ if the new partition improves cost
end
```

Algorithm 2: Incremental k-means

Another issue is the choice of the number of clusters, k. One strategy to choose k is to look at the cost versus number of clusters curve and identify the k above which the curve flattens out (see Fig 3.1). That is, increasing the number of clusters beyond this value provides only a marginal improvement in the total cost, and therefore there is no incentive to choose a higher value of k. This technique is called the *Elbow Method.*

The following theorem gives a lower bound on the cost that an optimal algorithm can achieve in the Euclidean distance case.

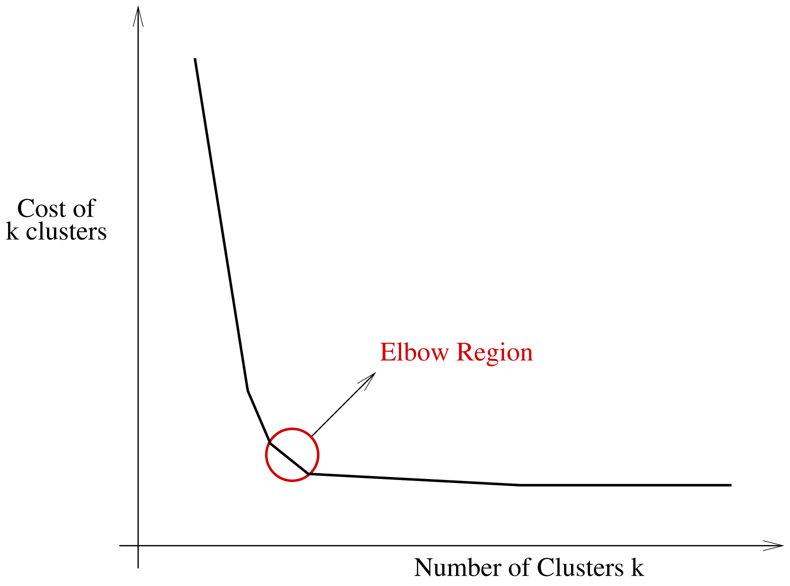

Figure 3.1. Elbow Method

Theorem 3.2.

$$\min_S \mathcal{C}(S) \geq \sum_{l=k+1}^{\min(m,n)} \sigma_l(X)^2,$$

where $X = \begin{bmatrix} \text{——} & x_1 & \text{——} \\ \text{——} & x_2 & \text{——} \\ & \vdots & \\ \text{——} & x_n & \text{——} \end{bmatrix}$,

$x_1, x_2, \ldots, x_n \in \mathbb{R}^m$ *are the data points, and* $\sigma_l(X)$ *is the* l^{th} *largest singular value of* X.

Remark: Here is a simple example where the bound is tight. Suppose that the data points $x_1, x_2, \ldots, x_n \in \{z_1, z_2, \ldots, z_k\}$ where $z_i \perp z_j$ for all $i, j \in [k]$. Then XX^T is a block diagonal matrix with k diagonal blocks, and the rank of each block is 1. Thus the rank of XX^T is k, and the LHS and the RHS in the above inequality are both equal to zero.

Proof:

$$\begin{aligned} \mathcal{C}(S) &= \sum_{a=1}^{k} \sum_{i \in S_a} ||x_i - c_a||^2 \\ &\geq \sum_{a=1}^{k} \sum_{i \in S_a} ||x_i||^2 - ||c_a||^2 \\ &= \sum_{i=1}^{n} ||x_i||^2 - \sum_{a=1}^{k} \frac{1}{n_a} || \sum_{j \in S_a} x_j ||^2. \end{aligned}$$

Let $u_{S_a} = \sum_{i \in S_a} e_i$ and $Y = \begin{bmatrix} \text{——} & \frac{1}{\sqrt{n_1}} u_{S_1} & \text{——} \\ \text{——} & \frac{1}{\sqrt{n_2}} u_{S_2} & \text{——} \\ & \vdots & \\ \text{——} & \frac{1}{\sqrt{n_k}} u_{S_k} & \text{——} \end{bmatrix}$. Then

$$\begin{aligned} \mathcal{F}(S) &= \sum_{a=1}^{k} \|\frac{1}{\sqrt{n_a}} \sum_{j \in S_a} x_j\|^2 \\ &= \sum_{a=1}^{k} \|\frac{1}{\sqrt{n_a}} X^T u_{S_a}\|^2 \\ &= \sum_{a=1}^{k} \frac{1}{\sqrt{n_a}} u_{S_a}^T X X^T u_{S_a} \frac{1}{\sqrt{n_a}} \\ &= Tr(YXX^TY^T). \end{aligned}$$

Note that since $S_a \cap S_b = \emptyset$, and $u_{S_a} \perp u_{S_b}$, we have $YY^T = I_{k \times k}$. Now

$$\begin{aligned} \max_S \mathcal{F}(S) &= \max \quad Tr(YXX^TY^T) \\ & \quad\ \text{s.t. } Y = \begin{bmatrix} \text{——} & \frac{1}{\sqrt{n_1}} u_{S_1} & \text{——} \\ \text{——} & \frac{1}{\sqrt{n_2}} u_{S_2} & \text{——} \\ & \vdots & \\ \text{——} & \frac{1}{\sqrt{n_k}} u_{S_k} & \text{——} \end{bmatrix}, \\ & \quad\ YY^T = I_{k \times k}, \quad Y \in \mathbb{R}^{k \times n} \\ &\leq \max \quad Tr(YXX^TY^T) \\ & \quad\ \text{s.t. } YY^T = I_{k \times k}, \quad Y \in \mathbb{R}^{k \times n} \\ &= \sum_{l=1}^{k} \sigma_l(X)^2 \text{ from Lemma 3.3.} \end{aligned}$$

Therefore,

$$\begin{aligned} \min_S \mathcal{C}(S) &\geq \sum_{i=1}^{n} \|x_i\|^2 - \max_S \mathcal{F}(S) \\ &\geq \sum_{l=k+1}^{\min(m,n)} \sigma_l(X)^2. \end{aligned}$$

□

Lemma 3.3.

$$\begin{aligned} \max \quad & Tr(YXX^TY^T) = \sigma_1(X)^2 + \cdots + \sigma_k(X)^2. \\ s.t. \quad & YY^T = I, \quad Y \in \mathbb{R}^{k \times n} \end{aligned}$$

Proof:

$$\begin{aligned}
Tr(YXX^TY^T) &= \sum_{i=1}^{k} \lambda_i(YXX^TY^T) \\
&\leq \sum_{i=1}^{k} \lambda_i(XX^T) \text{ (since } YY^T = I_{k\times k} \implies \lambda_i(YXX^TY^T) \leq \lambda_i(XX^T) \quad \forall 1 \leq i \leq k).
\end{aligned}$$

In the above inequality, equality is achieved for Y that has the top k eigenvectors of XX^T as its rows. This completes the proof.

□

3.3 Next time: Probabilistic Model

As already mentioned, the k-means algorithm views clustering in an optimization framework. Alternatively, one could view the problem in a probabilistic setting where the data set is modelled by a mixture of distributions with each cluster represented by a parametric distribution. Very often, it is assumed that each data point is equally likely to be from any of the cluster. Thus, the data points $x_1, x_2, \dots x_n$ are assumed to be generated according to $p(\cdot|c) = \frac{1}{k}\sum_{a=1}^{k} q(\cdot|c_a)$, and are used to find an ML estimate of the parameters.

A widely used model is the *Gaussian mixture model* with fixed variances and means (interpreted as the cluster centers) as the parameters of the component Gaussian distributions. Another interesting model is one in which the means are fixed and the variance is cluster dependent; but we will not consider that model here. Thus, given that $q(x|c) = \frac{1}{(2\pi\sigma^2)^{m/2}}\exp(-\frac{||x-c||^2}{2\sigma^2})$, we would like to minimize the negative log-likelihood given by

$$\begin{aligned}
l(c|x_1, \dots x_n) &= -\sum_{i=1}^{n} \log(\sum_{a=1}^{k} \exp(-\frac{||x_i - c_a||^2}{2\sigma^2})) \\
&\xrightarrow{\sigma\to 0} -\sum_{i=1}^{n} \log(\max_a\{\exp(-\frac{||x_i - c_a||^2}{2\sigma^2})\}) \\
&= \sum_{i=1}^{n} \min_a\{\frac{||x_i - c_a||^2}{2\sigma^2}\}.
\end{aligned}$$

This establishes a connection between the optimization and the probabilistic models. For very small σ, $l(c|x) = \min_S \mathcal{C}(S, c)$. The k-means algorithm tries to minimize $\min_S \mathcal{C}(S, c)$ while the EM algorithm (which we will see in the next class) tries to minimize $l(c|x)$.