Chapter 14

Clustering

So far in this course, we have focused on the task of *supervised learning.* In the supervised setting, we are given a training dataset $\mathcal{D}_{\text{trn}}$ where each example $(\mathbf{x}, y) \in \mathcal{D}_{\text{trn}}$ contains both *input features* $\mathbf{x} \in \mathbb{R}^m$ as well as a *target label* $y \in \mathbb{R}$.

However, in many real-world settings, the target labels are *unobserved.* For example, suppose we have a dataset of user activity profiles on a social network. The features in the dataset might indicate things such as the number of minutes a user spends online on the social network per week, the number of messages they send, how many friends they have, how many posts they make, and so on. Moreover, suppose our goal is to classify all the users into different activity categories, such as "content creators", "sporadic users", "influencers" , etc. Now, the challenge in such a task is that we do not know the labels for these users in advance. There is no training set that provides us with this information. Instead, we need to somehow classify these users without any examples of what the classification should look like.

This problem is a variant of *unsupervised learning*, which we will focus on in this part of the course. The idea in unsupervised learning is that we want to infer patterns from data, without having any training labels. In unsupervised learning, our dataset $\mathcal{D}$ consists only of feature inputs $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$ and no labels y.

Latent variables There are many variants of unsupervised learning, and we will cover several of them in the course. However, many—if not most—variants of unsupervised learning can be captured via the notion of *latent variables.* The idea behind latent variables is as follows: we assume that there is some label (e.g., the target y) that is associated with all our datapoints $\mathbf{x} \in \mathcal{D}$, but we just assume that these labels are unobserved or latent. Our goal is to infer these latent (or hidden) variables using machine learning.

14.1 Clustering

Clustering is perhaps the most quintessential unsupervised learning task. It is the unsupervised analogue of classification. The goal in clustering is to infer discrete labels over all the points in our unlabeled set $\mathcal{D}$. More formally, we assume that all the points in our dataset belong to one of K different clusters, giving us a discrete latent variable z over K categorical values $\mathcal{Z} = \{1, 2, .., K\}$.[1] The goal of clustering is to infer the cluster assignments for the datapoints in $\mathcal{D}$. In other words, we want a function

$$f(\mathbf{x}) : \mathbb{R}^m \to \mathcal{Z} \tag{14.1}$$

that maps each datapoint $\mathbf{x}$ to a specific cluster assignment $z \in \mathcal{Z}$.

14.1.1 A simple approach: agglomerative clustering

The general strategy in most clustering approaches is to cluster together points that are close together. In other words, we want our final clusters to correspond to sets of points that have similar features. One of the simplest approaches that we can use for this is known as bottom-up or agglomerative clustering. The intuition in agglomerative clustering is that we simply form clusters by merging together pairs of points that have similar input features.

The basic steps are as follows:

1. Let $\mathbf{x}_i \in \mathcal{D}$ and $\mathbf{x}_j \in \mathcal{D}$ be the pair of points that are closest to each other in the dataset, i.e., the points that optimize the expression

$$\arg \min_{\mathbf{x}_i, \mathbf{x}_j \in \mathcal{D} : i \neq j} \|\mathbf{x}_i - \mathbf{x}_j\|. \tag{14.2}$$

2. Merge points $\mathbf{x}_i$ and $\mathbf{x}_j$ together into a cluster $\mathcal{C} = \{\mathbf{x}_i, \mathbf{x}_j\}$.

3. Remove $\mathbf{x}_i$ and $\mathbf{x}_j$ from $\mathcal{D}$ and add a new point $\mathbf{x}_C = \frac{\mathbf{x}_i + \mathbf{x}_j}{2}$ to $\mathcal{D}$.

4. Repeat the steps 1-3 until $|\mathcal{D}| < \gamma$ or $d(\mathbf{x}_i, \mathbf{x}_j) > \alpha, \forall \mathbf{x}_i, \mathbf{x}_j \in \mathcal{D} : i \neq j$, where γ and α are hyperparameters that control the stopping criterion of the algorithm.

The idea behind agglomerative clustering is that it iteratively merges together subsets of the data, based on the distance between the points. This process forms a tree or hierarchy of clusters and is illustrated in Figure 14.1.

14.2 K-means

Agglomerative clustering is simple and intuitive, but it does have certain drawbacks. Most prominently, agglomerative clustering does not produce a fixed

[1]We typically use z for latent variables and y for observed target values.

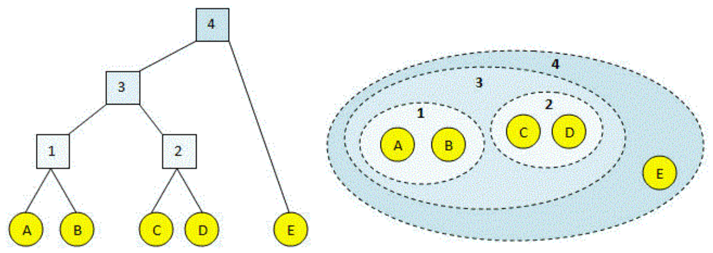

Figure 14.1: Illustration of agglomerative clustering over a simple dataset with 5 points. Each node in the tree corresponds to a possible clustering of the data, with the root node representing the full dataset.

set K of clusters; instead, it produces a hierarchy of cluster assignments. For example, this hierarchical outcome makes it difficult to interpret the results of agglomerative clustering from a latent variable perspective: if we are trying to infer the hidden label y associated with each point, hierarchical clustering is not as useful since there are many possible cluster assignments, depending on how we partition the inferred tree. Thus, hierarchical clustering can be useful in some circumstances, but in many cases, we want to assign our datapoints to a fixed number of clusters and are not interested in obtaining such a hierarchy.

An alternative approach—and one of the most popular techniques to cluster points—is known as K-means. Unlike agglomerative clustering, K-means will always cluster the points into a fixed number of clusters and it has a natural interpretation in terms of latent variables. The basic idea behind K-means are as follows:

1. For each of the $z \in 1, 2, ..., K$ clusters, start with a random initial guess for the *cluster centroid* $\boldsymbol{\mu}_{\mathcal{C}_z}$.

2. For each point $\mathbf{x} \in \mathcal{D}$ in the dataset, assign that point to the cluster with the closest centroid. That is, form clusters sets

$$\mathcal{C}_z = \{\mathbf{x} \in \mathcal{D} : \|\mathbf{x} - \boldsymbol{\mu}_{\mathcal{C}_z}\| < \|\mathbf{x} - \boldsymbol{\mu}_{\mathcal{C}_j}\|, \forall j \in [K], j \neq z\} \tag{14.3}$$

3. Recompute the cluster centroids based on the (new) assignments:

$$\boldsymbol{\mu}_{\mathcal{C}_z} = \frac{1}{|\mathcal{C}_z|} \sum_{\mathbf{x} \in \mathcal{C}_z} \mathbf{x} \tag{14.4}$$

4. Repeat steps 2 and 3 until the cluster assignments stabilize (i.e., until no points are re-assigned in step 2).

Thus, K-means is essentially an iterative refinement approach: we start with an initial guess for the clusters, and then we iteratively update these guesses until the algorithm converges.

14.2.1 Convergence of K-means

One important aspect of the K-means algorithm is that it is guaranteed to converge. In order to show that this is the case, we must first define the sum-squared error of a particular cluster z by measuring the squared distance between points in the cluster and the cluster centroid $\boldsymbol{\mu}_{\mathcal{C}_z}$:

$$S(\mathcal{C}_k, \boldsymbol{\mu}_{\mathcal{C}_z}) = \sum_{\mathbf{x} \in \mathcal{C}_k} \|\mathbf{x} - \boldsymbol{\mu}_{\mathcal{C}_z}\|^2. \tag{14.5}$$

We can then measure the overall squared error S_* of K clusters by summing these scores:

$$S_* = \sum_{k \in 1,2,\ldots,K} S(\mathcal{C}_k, \boldsymbol{\mu}_{\mathcal{C}_z}). \tag{14.6}$$

We can then show that K-means converges by proving that S_* monotonically decreases, except when no nodes are re-assigned, which is the stopping criterion.

To see that S_* monotonically decreases, we must use the following lemma:

Lemma 3. *Let $\mathcal{C} = \{\mathbf{x}_1, ..., \mathbf{x}_{|\mathcal{C}|}\}$ be a set of points $\mathbf{x}_i \in \mathbb{R}^m$ and let*

$$\boldsymbol{\mu}_{\mathcal{C}} = \frac{1}{|\mathcal{C}|} \sum_{\mathbf{x}_i \in \mathcal{C}} \mathbf{x}_i \tag{14.7}$$

denote the mean of the points in this set. Then, we have that $\forall \mathbf{q} \in \mathbb{R}^m$

$$\sum_{\mathbf{x}_i \in \mathcal{C}} \|\mathbf{x}_i - \boldsymbol{\mu}_{\mathcal{C}}\| \leq \sum_{\mathbf{x}_i \in \mathcal{C}} \|\mathbf{x}_i - \mathbf{q}\|. \tag{14.8}$$

In other words, Lemma 3 states that the sum-squared error for a fixed cluster is minimized when the centroid is defined as the mean of the points. The proof of Lemma 3 is left as an exercise.

Using Lemma 3 we can then prove a convergence theorem for K-means:

Theorem 4. *K-means converges in a finite number of iterations.*

Proof. Suppose that the algorithm does not terminate after an iteration t. This means that at least one point was re-assigned to a new cluster. We will show that this also implies that $S_*^{(t+1)} < S_*^{(t)}$, where $S_*^{(t)}$ and $S_*^{(t+1)}$ denote the squared error of the clusters (Equation 14.6) before and after iteration t, respectively. Showing that $S_*^{(t+1)} < S_*^{(t)}$ is sufficient to prove convergence, since (i) this monotonic decrease in the squared error guarantees that no clustering can be revisited and (ii) there are a finite number possible clusterings ($K^{|\mathcal{D}|}$ to be exact).

Now, to show that $S_*^{(t+1)} < S_*^{(t)}$, we must consider the both steps 2 and 3 in the K-means algorithm. We use $\mathcal{C}_k^t$ to denote the cluster at the beginning of iteration t and $\mathcal{C}_z^{t+1}$ to denote the cluster after performing the re-assignment of points. Similarly, we use $\boldsymbol{\mu}_{\mathcal{C}_z^t}$ and $\boldsymbol{\mu}_{\mathcal{C}_z^{t+1}}$ to denote the means/centroids of these clusters.

First, for step 2 of the K-means algorithm, we can see that

$$\sum_{k\in[K]} S(\mathcal{C}_k^{t+1}, \boldsymbol{\mu}_{\mathcal{C}_k^t}) < \sum_{k\in[K]} S(\mathcal{C}_k^{t}, \boldsymbol{\mu}_{\mathcal{C}_k^t}), \tag{14.9}$$

i.e., we know that the re-assignment step must decrease the sum-squared error. Otherwise, we would not make a re-assignment. To complete the proof, we simply combine the above inequality with Lemma 3:

$$\begin{aligned}
S_*^t &= \sum_{k\in[K]} S(\mathcal{C}_k^{t}, \boldsymbol{\mu}_{\mathcal{C}_k^t}) && (14.10)\\
&< \sum_{k\in[K]} S(\mathcal{C}_k^{t+1}, \boldsymbol{\mu}_{\mathcal{C}_k^t}) \qquad \text{by Equation 14.9} && (14.11)\\
&= \sum_{k\in[K]} \sum_{\mathbf{x}\in\mathcal{C}_k^{t+1}} \|\mathbf{x} - \boldsymbol{\mu}_{\mathcal{C}_k^t}\| && (14.12)\\
&\leq \sum_{k\in[K]} \sum_{\mathbf{x}\in\mathcal{C}_k^{t+1}} \|\mathbf{x} - \boldsymbol{\mu}_{\mathcal{C}_k^{t+1}}\| \qquad \text{by Lemma 3} && (14.13)\\
&= S_*^{t+1}. && (14.14)
\end{aligned}$$

□

14.2.2 Soft K-means

One drawback of the K-means approach that we introduced previously is that it only provides *hard* assignments. In other words, every point is assigned to a cluster, but we have no notion of how *strongly* a point belongs to a certain cluster. One way to get around this is to use *soft* K-means. The key difference between (hard) K-means and soft K-means is that in the soft version, we assign a probability to each point belonging to each cluster. The steps in soft K-means are as follows:

1. For each of the $z \in 1, 2, ..., K$ clusters, start with a random initial guess for the *cluster centroid* $\boldsymbol{\mu}_{\mathcal{C}_k}$.

2. For each point $\mathbf{x} \in \mathcal{D}$ in the dataset compute a score $r(\mathbf{x}, z)$, corresponding to how likely it is to belong to each cluster:

$$r(\mathbf{x}, z) = \frac{e^{-\|\mathbf{x}-\boldsymbol{\mu}_{\mathcal{C}_z}\|^2}}{\sum_{j\in[K]} e^{-\|\mathbf{x}-\boldsymbol{\mu}_{\mathcal{C}_j}\|^2}}. \tag{14.15}$$

   Here, we use a softmax normalization approach (see Chapter 6) to ensure that these scores correspond to probabilities.

3. Recompute the cluster centroids based on the (new) soft assignments:

$$\boldsymbol{\mu}_{\mathcal{C}_z} = \frac{\sum_{\mathbf{x}\in\mathcal{D}} \mathbf{x} r(\mathbf{x}, z)}{\sum_{\mathbf{x}\in\mathcal{D}} r(\mathbf{x}, z)} \tag{14.16}$$

4. Repeat steps 2 and 3 until the cluster assignments stabilize (e.g., until the change in assignment scores is less than ϵ for all points).

Thus, in the soft K-means approach, we end up with each point having a probability $r(\mathbf{x}, z)$ of belonging to each cluster z.