Machine Learning - CPEN 355

Lecture Notes

Souradeep Dutta
(pronounced as Show-Ro-Deep Duh-tah)

Contents

Chapter 1

Clustering

In the problem of clustering, we are given a dataset comprised only of input features without labels. We wish to assign to each data point a discrete label indicating which "cluster" it belongs to, in such a way that the resulting cluster assignment "fits" the data. We are given flexibility to choose our notion of goodness of fit for cluster assignments.
Clustering is an example of unsupervised learning, where we are not given labels and desire to infer something about the underlying structure of the data. Another example of unsupervised learning is dimensionality reduction, where we desire to learn important features from the data. Clustering is most often used in exploratory data visualization, as it allows us to see the different groups of similar data points within the data. Combined with domain knowledge, these clusters can have a physical interpretation - for example, different clusters can represent different species of plant in the biological setting, or types of consumers in a business setting. If desired, these clusters can be used as pre-processing to make the data more compact. Clustering is also used for outlier detection, as in Figure 1.2: data points that do not seem to belong in their assigned cluster may be flagged as outliers.
In order to create an algorithm for clustering, we first must determine what makes a good clustering assignment. Here are some possible desired proper-

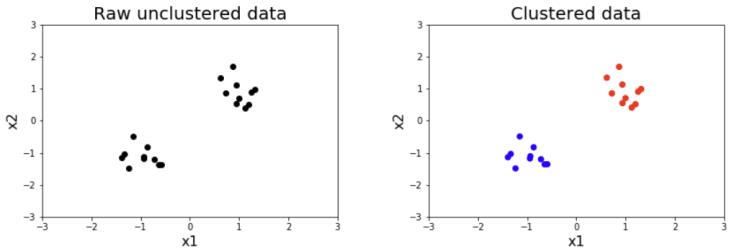

Figure 1.1: Left: unclustered raw data; Right: clustered data

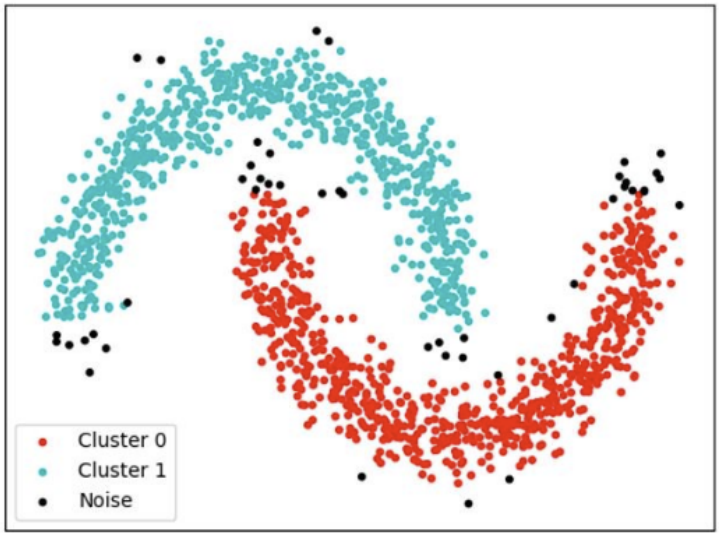

Figure 1.2: A nonspherical clustering assignment. Possible outliers are shown in black. Click here for details

ties:

1. High intra-cluster similarity - points within a given cluster are very similar.

2. Low inter-cluster similarity - points in different clusters are not very similar.

Of course, this depends on our notion of similarity. For now, we will say that points in $\mathbb{R}^d$ are similar if their L_2 distance is small, and dissimilar otherwise. A generalization of this notion is provided in the appendix.

1.1 K-means Clustering

Let X denote the set of N data points $x_i \in \mathbb{R}^d$. A cluster assignment is a partition $C_1, \ldots, C_K \subseteq X$ such that the sets C_k are disjoint and $X = C_1 \cup \cdots \cup C_K$. A data point $x \in X$ is said to belong to cluster k if it is in C_k.
One approach to the clustering problem is to represent each cluster C_k by a single point $c_k \in \mathbb{R}^d$ in the input space - this is called the centroid approach. K-means is an example of centroid-based clustering where we choose centroids and a cluster assignment such that the total distance of each point to its assigned centroid is minimized. In this regard, K-means optimizes for high intra-cluster similarity, but the clusters do not necessarily need to be far apart, so we may also have high inter-cluster similarity.

Formally, K-means solves the following problem:

$$\underset{\{C_k\}_{k=1}^K, \{c_k\}_{k=1}^K : X = C_1 \cup \cdots \cup C_K}{\arg\min} \sum_{k=1}^{K} \sum_{x \in C_k} \|x - c_k\|^2 \tag{1.1}$$

It has been shown that this problem is NP hard, so solving it exactly is intractable. However, we can come up with a simple algorithm to compute a candidate solution. If we knew the cluster assignment $C_1, \ldots, C_K$, then we would only need to determine the centroid locations. Since the choice of centroid location c_i does not affect the distances of points in C_j to c_j for $i \neq j$, we can consider each cluster separately and choose the centroid that minimizes the sum of squared distances to points in that cluster. The centroid we compute, $\hat{c}_k$, is

$$\hat{c}_k = \arg\min_{c_k} \sum_{x \in C_k} \|x - c_k\|^2$$

But this is simply the mean of the data in C_k, that is,

$$\hat{c}_k = \frac{1}{|C_k|} \sum_{x \in C_k} x$$

Similarly, if we knew the centroids c_k, in order to choose the cluster assignment $C_1, \ldots, C_K$ that minimizes the sum of squared distances to the centroids, we simply assign each data point x to the cluster represented by its closest centroid, that is, we assign x to

$$\arg\min_k \|x - c_k\|^2$$

Now we can perform alternating minimization - on each iteration of our algorithm, we update the clusters using the current centroids, and then update the centroids using the new clusters. This algorithm is sometimes called Lloyd's Algorithm.

Algorithm 1: K-means Algorithm

Initialize $c_k, k = 1, \ldots, K$
while K-means objective has not converged do

- Update partition $C_1 \cup \cdots \cup C_K$ given the c_k by assigning each $x \in X$ to the cluster represented by its nearest centroid
- Update centroids c_k given $C_1 \cup \cdots \cup C_K$ as

$$c_k = \frac{1}{|C_k|} \sum_{x \in C_k} x$$

This algorithm will always converge to some value. To show this, note the following facts:

1. There are only finitely many (say, M) possible partition/centroid pairs that can be produced by the algorithm.
2. Each update of the cluster assignment and centroids does not increase the value of the objective.

If the value of the objective has not converged after M iterations, then we have cycled through all the possible partition/centroid pairs attainable by the algorithm. On the next iteration, we would obtain a partition and centroid assignment that we have already seen. In practice, it is common to run the K-means algorithm multiple times with different initialization points, and the cluster corresponding to the minimum objective value is chosen. There are also ways to choose a smarter initialization than a random seed, which can improve the quality of the local optimum found by the algorithm.[1] It should be emphasized that no efficient algorithm for solving the K-means optimization is guaranteed to give a good cluster assignment, as the problem is NP hard and there are local optima. Choosing the number of clusters k is similar to choosing the number of principal components for PCA - we can compute the value of the objective for multiple values of k and find the "elbow" in the curve.

1.2 Practical Use of K-means

K-means is one of the most well-known and widely used clustering algorithms in practice. Despite the fact that the underlying optimization problem is NP-hard and the algorithm only guarantees convergence to a local optimum, K-means remains popular due to its conceptual simplicity, ease of implementation, scalability to large datasets, and strong empirical performance in many real-world applications. It is commonly used in exploratory data analysis, data compression, vector quantization, image segmentation, and as a preprocessing step in larger machine learning pipelines.
Because of its practical importance, K-means has been implemented in essentially all major scientific computing and machine learning libraries. In Python, commonly used implementations include:

- **scikit-learn**: The `sklearn.cluster.KMeans` class provides a highly optimized implementation with support for multiple random initializations, K-means++ initialization, and efficient convergence checks. SciKit
- **SciPy**: The `scipy.cluster.vq.kmeans` and `kmeans2` functions provide classical implementations closely aligned with Lloyd's algorithm. SciPy
- **PyTorch (community implementations)**: While PyTorch does not include K-means in its core library, a number of GPU-accelerated implementations are available and commonly used when clustering large, high-dimensional datasets. Pytorch

[1] For example, K-means++.

- **FAISS**: Facebook AI Similarity Search provides highly optimized CPU and GPU implementations of K-means designed for very large-scale clustering problems. FAISS

These implementations reflect the continued relevance of K-means as a practical tool, even as more expressive probabilistic and nonparametric clustering models have been developed.

Bibliography