FADML Scribe

(15th April 2025)

Supon Mazumdar

(24CS92R06)

Indian Institute of Technology Kharagpur

Partitioning in Clustering

The most popular class of clustering algorithms that we have is the **iterative relocation algorithms**. These algorithms minimize a given clustering criterion by iteratively relocating data points between clusters until a (locally) optimal partition is attained.

There are many algorithms that come under the partitioning method. Some of the popular ones include:

- **K-Means**
- **PAM (k-Medoid)**
- **CLARA (Clustering Large Applications)**

K-Means Algorithm

K-means clustering is a technique used to organize data into groups based on their similarity.

How K-Means Clustering Works

Here's how the K-Means algorithm works:

1. **Initialization:** Randomly select K points from the dataset. These serve as the initial cluster centroids.
2. **Assignment:** For each data point in the dataset, compute the distance to each of the K centroids. Assign the data point to the cluster with the nearest centroid. This step forms K clusters.
3. **Update Centroids:** Recalculate the centroid of each cluster as the mean of all data points assigned to that cluster.
4. **Repeat:** Repeat steps 2 and 3 until convergence. Convergence occurs when the centroids do not change significantly or when a predefined number of iterations is reached.
5. **Final Result:** Once convergence is reached, the algorithm returns the final cluster centroids and the cluster assignments of all data points.

K-Means Algorithm

1. Initialize cluster centroids $\mu_1, \mu_2, \ldots, \mu_k \in \mathbb{R}^n$ randomly.
2. Repeat until convergence:
   - **Assignment step:** For each data point i, assign it to the closest cluster centroid:
   $$c^{(i)} := \arg\min_j \|x^{(i)} - \mu_j\|^2$$
   - **Update step:** For each cluster j, recompute the centroid as the mean of all points assigned to it:
   $$\mu_j := \frac{\sum_{i=1}^{m} \mathbf{1}\{c^{(i)} = j\} x^{(i)}}{\sum_{i=1}^{m} \mathbf{1}\{c^{(i)} = j\}}$$

Here, $\mathbf{1}\{c^{(i)} = j\}$ is the indicator function, which is 1 if $x^{(i)}$ is assigned to cluster j, and 0 otherwise.

Example of K-Means

K-Means Example with 7 Points and 2 Clusters. We are given the information that we need to make 2 clusters. It means we are given K=2 .We will solve this numerical on k-means clustering.

Point	Coordinates
P1	(2, 10)
P2	(2, 5)
P3	(8, 4)
P4	(5, 8)
P5	(7, 5)
P6	(6, 4)
P7	(1, 2)

Initial Centroids (Iteration 1)

- Centroid C1: P1 (2, 10)
- Centroid C2: P5 (7, 5)

Iteration 1: Assign Points to Nearest Centroid

Point	Dist. to C1	Dist. to C2	Cluster
P1	0.0	7.07	C1
P2	5.0	5.0	C1 (tie)
P3	7.21	1.0	C2
P4	3.61	3.61	C1 (tie)
P5	5.83	0.0	C2
P6	6.08	1.0	C2
P7	8.06	6.08	C2

(For tie cases, we assign to C1 by default.)

New Centroids After Iteration 1

- Cluster 1: P1, P2, P4 $\Rightarrow$ Centroid = $\left(\frac{2+2+5}{3}, \frac{10+5+8}{3}\right) = (3, 7.67)$
- Cluster 2: P3, P5, P6, P7 $\Rightarrow$ Centroid = $\left(\frac{8+7+6+1}{4}, \frac{4+5+4+2}{4}\right) = (5.5, 3.75)$

Iteration 2: Reassign Points

Point	Dist. to C1	Dist. to C2	Cluster
P1	1.20	8.08	C1
P2	2.74	4.03	C1
P3	5.80	2.55	C2
P4	2.24	4.42	C1
P5	5.38	1.80	C2
P6	5.90	0.56	C2
P7	6.80	4.60	C2

Final Clusters (Stable)

- Cluster 1: P1, P2, P4
- Cluster 2: P3, P5, P6, P7

Final Centroids

- C1: (3, 7.67)
- C2: (5.5, 3.75)

Demerits of K-Means Clustering

1. **Requires Predefined Number of Clusters:** K-means requires the number of clusters K to be specified beforehand, which may not be known in advance and can lead to suboptimal clustering.
2. **Assumes Convex and Spherical Clusters:** K-means performs well only when clusters are convex, spherical, and of similar size. It fails to correctly identify clusters with complex shapes or different densities.
3. **Sensitive to Initial Centroid Selection:** The final clustering result can vary depending on the initial positions of centroids. Poor initialization can lead to convergence to a local minimum.
4. **Not Robust to Noise and Outliers:** K-means is highly sensitive to noise and outliers, as they can significantly distort the cluster centroids and degrade the clustering performance.
5. **Poor Performance with Varying Density:** K-means assumes that all clusters have similar density and size, which is not always the case in real-world data. This can result in incorrect clustering.

DBSCAN Clustering Algorithm

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a density-based clustering algorithm that groups data points that are closely packed together and marks outliers as noise based on their density in the feature space. It identifies clusters as dense regions in the data space, separated by areas of lower density.

Advantages over Traditional Clustering Methods

Unlike K-Means or hierarchical clustering, which assume clusters are compact and spherical, DBSCAN excels in handling real-world data irregularities such as:

- **Arbitrary-Shaped Clusters:** Clusters can take any shape, not just circular or convex.
- **Noise and Outliers:** It effectively identifies and handles noise points without assigning them to any cluster.

DBSCAN Categorizes Points Into Three Types

1. **Core Points:** Points that have a sufficient number of neighbors within a specified radius (ε).
2. **Border Points:** Points that are within the neighborhood of a core point but do not themselves have enough neighbors to be core points.
3. **Noise Points:** Points that do not belong to any cluster; they are considered outliers.

The DBSCAN Algorithm

Density-based clustering uses the local density of points to determine the clusters, rather than using only the distance between points.

We define a ball of radius ϵ around a point $x \in \mathbb{R}^d$, called the ϵ-neighborhood of x, as follows:

$$N_\epsilon(x) = B_d(x, \epsilon) = \{y \mid \|x - y\| \leq \epsilon\} \tag{1}$$

Here, $\|x - y\|$ is the Euclidean distance between points x and y. However, other distance metrics can also be used.

For any point $x \in D$, we say that x is a **core point** if there are at least `minpts` points in its ϵ-neighborhood. In other words, x is a core point if

$$|N_\epsilon(x)| \geq \texttt{minpts}$$

where `minpts` is a user-defined local density or frequency threshold.

A **border point** is defined as a point that does not meet the `minpts` threshold, that is, it has $|N_\epsilon(x)| < \texttt{minpts}$, but it belongs to the ϵ-neighborhood of some core point z, that is,

$$x \in N_\epsilon(z)$$

Finally, if a point is neither a core nor a border point, then it is called a **noise point** or an **outlier**.

We say that a point x is *directly density reachable* from another point y if $x \in N_\epsilon(y)$ and y is a core point. We say that x is *density reachable* from y if there exists a chain of points $x_0, x_1, \ldots, x_l$, such that $x = x_0$ and $y = x_l$, and x_i is directly density reachable from x_{i-1} for all $i = 1, \ldots, l$. In other words, there is a set of core points leading from y to x. Note that density reachability is an asymmetric or directed relationship.

We define any two points x and y to be *density connected* if there exists a core point z, such that both x and y are density reachable from z. A *density-based cluster* is defined as a maximal set of density connected points.

Algorithm 1 Density-based Clustering Algorithm

DBSCAN(D, ε, $minpts$):
  $Core \leftarrow \emptyset$
for each $\mathbf{x}_i \in D$ **do**
    Compute $N_\varepsilon(\mathbf{x}_i)$
    $id(\mathbf{x}_i) \leftarrow \emptyset$ {// cluster id for $\mathbf{x}_i$}
  if $|N_\varepsilon(\mathbf{x}_i)| \geq minpts$ **then**
      $Core \leftarrow Core \cup \{\mathbf{x}_i\}$
  end if
end for
  $k \leftarrow 0$ {// cluster id}
for each $\mathbf{x}_i \in Core$ **such that** $id(\mathbf{x}_i) = \emptyset$ **do**
    $k \leftarrow k + 1$
    $id(\mathbf{x}_i) \leftarrow k$ {// assign $\mathbf{x}_i$ to cluster id k}
    DensityConnected($\mathbf{x}_i, k$)
end for
  $C \leftarrow \{C_i\}_{i=1}^{k}$, where $C_i \leftarrow \{\mathbf{x} \in D \mid id(\mathbf{x}) = i\}$
  $Noise \leftarrow \{\mathbf{x} \in D \mid id(\mathbf{x}) = \emptyset\}$
  $Border \leftarrow D \setminus \{Core \cup Noise\}$
  return C, $Core$, $Border$, $Noise$
Procedure DensityConnected($\mathbf{x}, k$):
for each $y \in N_\varepsilon(\mathbf{x})$ **do**
    $id(y) \leftarrow k$ {// assign y to cluster id k}
  if $y \in Core$ **then**
      DensityConnected(y, k)
  end if
end for

Hybrid Clustering

Hybrid clustering is a technique that combines two or more clustering algorithms to leverage the strengths of each while mitigating their individual weaknesses. It is particularly useful in complex or high-dimensional datasets where a single clustering approach may not perform adequately across all dimensions of the data. Each traditional clustering algorithm has specific advantages and limitations:

- **K-Means**: Efficient but assumes spherical clusters and struggles with noise.

- **Hierarchical Clustering**: Captures nested relationships but is computationally expensive.
- **DBSCAN (Density-Based)**: Effective with arbitrary-shaped clusters and noise, but sensitive to parameter selection.

By combining these methods, hybrid clustering seeks to produce more accurate, robust, and interpretable cluster results.

How Hybrid Clustering Works

There are several common strategies in hybrid clustering:

1. Preprocessing + Clustering

One algorithm is used to preprocess the data. For example, DBSCAN can be used to remove noise, followed by K-Means for final clustering on the cleaned data.

2. Dimensionality Reduction + Clustering

A dimensionality reduction method such as PCA or t-SNE is first applied, followed by a clustering algorithm. This is especially useful for high-dimensional datasets.

3. Ensemble Clustering

Multiple clustering algorithms are applied independently, and their results are combined using a consensus function to produce the final cluster labels. This approach is also known as *cluster ensemble* or *clustering aggregation*.

4. Hierarchical-Hybrid Approach

A bottom-up method (like agglomerative clustering) is first used to identify small subgroups, followed by a top-down method (like K-Means) to merge and refine the clusters.

CLARA (Clustering Large Applications)

CLARA (Clustering Large Applications) is a clustering algorithm designed to handle large datasets by improving the efficiency of PAM (Partitioning Around Medoids).

How It Works

- Randomly selects small samples from the dataset.
- Applies PAM to each sample to find medoids.
- Assigns all data points to the nearest medoids.
- Repeats the process multiple times with different samples.
- Chooses the best clustering based on minimum average dissimilarity.

Advantages

- Scalable to large datasets.
- More robust to noise than K-Means.

Limitations

- Relies on the quality of samples.
- May miss optimal clusters not present in samples.

Clustering Evaluation Metrics

Silhouette Score

Assume the data have been clustered via any technique, such as k-medoids or k-means, into k clusters.

For data point $i \in C_I$ (data point i in the cluster C_I), let

$$a(i) = \frac{1}{|C_I| - 1} \sum_{j \in C_I, i \neq j} d(i, j)$$

be the mean distance between i and all other data points in the same cluster, where $|C_I|$ is the number of points belonging to cluster C_I, and $d(i, j)$ is the distance between data points i and j in the cluster C_I (we divide by $|C_I| - 1$ because we do not include the distance $d(i, i)$ in the sum). We can interpret $a(i)$ as a measure of how well i is assigned to its cluster (the smaller the value, the better the assignment).

We then define the mean dissimilarity of point i to some cluster C_J as the mean of the distance from i to all points in C_J (where $C_J \neq C_I$).

For each data point $i \in C_I$, we now define

$$b(i) = \min_{J \neq I} \frac{1}{|C_J|} \sum_{j \in C_J} d(i, j)$$

to be the smallest (hence the min operator in the formula) mean distance of i to all points in any other cluster (i.e., in any cluster of which i is not a member). The cluster with this smallest mean dissimilarity is said to be the "neighboring cluster" of i because it is the next best fit cluster for point i.

We now define a silhouette (value) of one data point i

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}, \text{ if } |C_I| > 1$$

and

$$s(i) = 0, \text{ if } |C_I| = 1$$

Which can be also written as:

$$s(i) = \begin{cases} 1 - a(i)/b(i), & \text{if } a(i) < b(i) \\ 0, & \text{if } a(i) = b(i) \\ b(i)/a(i) - 1, & \text{if } a(i) > b(i) \end{cases}$$

From the above definition it is clear that

$$-1 \leq s(i) \leq 1$$

Note that $a(i)$ is not clearly defined for clusters with size = 1, in which case we set $s(i) = 0$. This choice is arbitrary, but neutral in the sense that it is at the midpoint of the bounds, -1 and 1.[1] For $s(i)$ to be close to 1 we require $a(i) \ll b(i)$. As $a(i)$ is a measure of how dissimilar i is to its own cluster, a small value means it is well matched. Furthermore, a large $b(i)$ implies that i is badly matched to its neighboring cluster. Thus an $s(i)$ close to 1 means that the data is appropriately clustered.

If $s(i)$ is close to -1, then by the same logic we see that i would be more appropriate if it was clustered in its neighboring cluster. An $s(i)$ near zero means that the datum is on the border of two natural clusters.

Semi-supervised learning

Semi-supervised learning (SSL) is halfway between supervised and unsupervised learning. In addition to unlabeled data, the algorithm is provided with some supervision information—but not necessarily for all examples. Often, this information will be the targets associated with some of the examples.

In this case, the dataset

$$X = (x_i)_{i \in [n]}$$

can be divided into two parts: the points

$$X_l := (x_1, \ldots, x_l),$$

for which labels

$$Y_l := (y_1, \ldots, y_l)$$

are provided, and the points

$$X_u := (x_{l+1}, \ldots, x_{l+u}),$$

the labels of which are not known.

0.1 Transductive Support Vector Machines (TSVMs)

In contrast to **inductive learning**, which aims to learn a general prediction rule for any unseen data, **Transductive Learning** (proposed by V. Vapnik) focuses on making predictions only at a fixed set of known test points. This approach allows the learning algorithm to leverage the positions of the test points, making it a specialized form of **semi-supervised learning**.

Transductive Support Vector Machines (TSVMs) extend this idea by incorporating test points into the margin computation, improving classification performance, particularly in tasks like **text classification**.

0.1.1 Key Concepts

1. **Transductive vs. Inductive Learning**
   - **Inductive Learning**: Learns a general model from labeled data to predict unseen examples.
   - **Transductive Learning**: Predicts labels only for a specific set of unlabeled test points, using their distribution to guide learning.
2. **How TSVMs Work**
   TSVMs optimize the margin while considering both labeled and unlabeled (test) data. The goal is to find a decision boundary that:
   - Separates labeled data correctly.
   - Maximizes the margin on both labeled and unlabeled points.
3. **Advantages of TSVMs**
   - Better generalization when test points follow a specific structure.
   - Particularly effective in high-dimensional problems (e.g., text classification).
   - Leverages unlabeled data to improve decision boundaries.

0.1.2 Optimization Challenges

The TSVM optimization problem is **non-convex** and computationally challenging.

1. **Exact Optimization Methods**
   - Typically involve **integer programming** or **branch-and-bound techniques**.
   - Computationally expensive, limiting scalability.
2. **Approximate Optimization Methods**
   - Local search heuristics (e.g., label switching).
   - Continuation methods (gradually introducing unlabeled points).
   - Semi-definite programming relaxations.

0.2 Semi-Supervised Graph-Based Approach

- A graph is constructed where each node represents a data point, including both labeled and unlabeled instances.
- Edges in the graph encode similarity relationships between data points, often computed using distance metrics or kernel functions.
- Known labels from the labeled data points are propagated through the graph to infer labels for the unlabeled nodes.
- Various graph-based semi-supervised learning algorithms can be unified under a common framework based on minimizing a quadratic cost function.
- The solution to this optimization problem results in a linear system of equations of size $n \times n$, where n is the total number of data points.
- This framework naturally extends to the inductive setting, where test samples arrive sequentially.
- An induction formula is derived, allowing efficient label prediction for new test samples.
- The inductive formula can be computed in $\mathcal{O}(n)$ time, which is significantly faster than solving the entire linear system from scratch (which typically takes $\mathcal{O}(kn^2)$ time for sparse graphs, where each data point has k neighbors).
- When similarity measures satisfy a locality property (i.e., are based on local neighborhoods), the method is affected by the curse of dimensionality.
- In such cases, the performance degrades as the dimensionality of the underlying data manifold increases.

0.3 Co-Training: A Semi-Supervised Learning Approach

Co-training, introduced by Blum and Mitchell in 1998, is a semi-supervised learning technique designed to improve classification performance by leveraging multiple distinct "views" of the data. This method is particularly useful when labeled data is scarce but unlabeled data is abundant.

Key Concepts:

- **Multiple Views**: Co-training assumes that each data instance can be represented by two or more independent feature sets (views). For example, a webpage can be described by its content (text on the page) and its inbound links (text from hyperlinks pointing to the page).
  - Mathematically, the input space is $\mathcal{X} = \mathcal{X}^{(1)} \times \mathcal{X}^{(2)}$, where $x = (x^{(1)}, x^{(2)})$ represents an instance with two views.
- **Compatibility Assumption**: The core idea is that the true label of an instance must be consistent across all views. A target concept $\theta = (\theta^{(1)}, \theta^{(2)})$ is compatible with the data distribution if $\theta^{(1)}(x^{(1)}) = \theta^{(2)}(x^{(2)})$ for all x in the support of the distribution.
  - This allows unlabeled data to constrain the hypothesis space, as any valid concept must satisfy $\theta \in \Theta(D_u \cup X_l)$, where D_u is unlabeled data and X_l is labeled data.
- **Bayesian Interpretation**: Co-training can be framed as Bayesian inference with conditional priors enforcing compatibility. The prior $P(\theta|S)$ is zero for concepts incompatible with the support S of the data distribution. The posterior $P(\theta|D_l, D_u)$ is non-zero only for concepts consistent with both labeled data and the compatibility constraint derived from unlabeled data.

How Co-Training Works:

1. **Initial Training**: Two separate classifiers are trained on the labeled data, each using one of the views.
2. **Iterative Labeling**: The classifiers predict labels for unlabeled data. High-confidence predictions from one classifier are used to augment the training set of the other, reinforcing agreement between views.
3. **Convergence**: The process repeats until the classifiers stabilize, effectively shrinking the version space to hypotheses consistent with all data.

Active Learning

A subset of machine learning known as "active learning" allows a learning algorithm to interactively query a user to label data with the desired outputs. The algorithm actively chooses from the pool of unlabeled data the subset of examples to be labelled next in active learning. The basic idea behind the active learner algorithm concept is that if an ML algorithm could select the data it wants to learn from, it might be able to achieve a higher degree of accuracy with fewer training labels.

Step-by-Step Procedure

1. **Initialization**
   - Start with a small labeled dataset L_0 to train an initial machine learning model.
2. **Model Training**
   - Train the first model M_0 using the available labeled data L_0.
3. **Uncertainty Estimation**
   - Apply the trained model M_i to predict labels for unlabeled data U.
   - Calculate the model's prediction uncertainty using metrics such as:
     - *Margin*: Difference between probabilities of top two predictions
     - *Variance*: Variability in predictions (for ensemble methods)
     - *Entropy*: Measure of prediction uncertainty
4. **Query Technique**
   - Select the most informative instances from U where the model is least confident.
   - Common query strategies include:
     - *Uncertainty Sampling*: Select instances with highest uncertainty
     - *Query-by-Committee*: Choose instances with maximal disagreement among ensemble models
     - *Margin Sampling*: Pick instances with smallest prediction margins
5. **Labeling**
   - Send the selected instances to an oracle (human annotator) for labeling.
   - Obtain ground truth labels for these instances.
6. **Model Update**
   - Add the newly labeled instances to the training set: $L_{i+1} = L_i \cup \text{new data}$.
   - Retrain the model M_{i+1} on the expanded labeled dataset.
7. **Repeat**
   - Iterate steps 2-6 until either:
     - A labeling budget is exhausted, or
     - Model performance reaches a satisfactory threshold