4. Clustering
Motivation

- **Clustering**, as a kind of unsupervised learning, aims at grouping data points into clusters

**Intuition**: Data points within
- the **same cluster** should be **close** to each other
- **different clusters** should be **far apart** from each other

**Applications**:
- segmentation of customers (e.g., for marketing campaigns)
- organization/exploration of data (e.g., search results)
- detection of outlier data points
Motivation

Machine Learning / Chapter 4: Clustering
Notation and Objective

- Consider a set of data points
  \[ \mathcal{D} = \{x_1, \ldots, x_n\} \quad x_i \in \mathbb{R}^m \]

- **Objective**: Determine clustering (also: grouping, partitioning)
  \[ \mathcal{C} = \{C_1, \ldots, C_k\} \quad \text{with} \quad C_i \subseteq \mathcal{D} \]

  such that
  - clusters are disjoint \( \forall i \neq j : C_i \cap C_j = \emptyset \)
  - each data point is assigned to a cluster
    \[ \bigcup_{C_i \in \mathcal{C}} C_i = \mathcal{D} \]
Euclidean Distance

- Unless otherwise specified, we’ll assume that the distance between data points is measured using **Euclidean distance**

$$d(x_i, x_j) = \sqrt{\sum_{l=1}^{m} (x_i(l) - x_j(l))^2}$$

- The methods that we’ll consider a generic, though, and any other **distance metric** can be used instead.
Agenda

- 4.1 k-Means
- 4.2 Evaluating Clusterings
- 4.3 Hierarchical Clustering
- 4.4 Density-based Clustering
4.1 k-Means

- k-Means is a representative-based clustering method that is simple yet widely used in practice.

- Each cluster $C_i$ is represented by a (virtual) data point $\mu_i$, which is the centroid of all data points in the cluster.

$$\mu_i = \frac{1}{|C_i|} \sum_{x_j \in C_i} x_j$$
k-Means

- k-Means aims to determine a clustering that minimizes the sum of squared errors within clusters

\[ \mathcal{L}(C) = \sum_{C_i \in C} \sum_{x_j \in C_i} d(\mu_i, x_j)^2 \]

- Finding an optimal clustering, i.e. solving

\[ C^* = \arg \min_C \mathcal{L}(C) \]

is NP-hard and k-Means only determines an approximate clustering.
k-Means

- k-Means is a **randomized, iterative, greedy algorithm**

- **Initialization:**
  - Determine $k$ **random data points** as centroids $\mu_i$

- **Repeat** (until no data point changes its cluster):
  - **assign** each data point $x_j$ **greedily** to the cluster whose centroid $\mu_i$ is closest
  - **recompute centroids** $\mu_i$ based on data points assigned to the cluster

- To get closer to an optimal clustering, we can **run k-Means repeatedly** and keep the best seen clustering
k-Means

```java
// Random initialization of centroids
int t = 0;
for (int i=1; i <= k; i++) {
    µ_t^i = random(D);
}

do {
    t++;

    // Reset clusters
    for (int i=1; i <= k; i++) {
        C_i = \emptyset;
    }

    // Assign data points to clusters
    for (x_j \in D) {
        i^* = \arg\min_j d(\mu_i^{(t-1)}, x_j);
        C_{i^*} = C_{i^*} \cup \{x_j\};
    }

    // Recompute centroids
    for (int i=1; i <= k; i++) {
        \mu_t^i = \frac{1}{|C_i|} \sum_{x_j \in C_i} x_j;
    }
}
while \left( \sum_{i=1}^k d(\mu_i^{(t-1)} - \mu_t^i) \geq \epsilon \right)
```
k-Means Example

(a) Initial dataset

(b) Iteration: $t = 1$

(c) Iteration: $t = 2$

(d) Iteration: $t = 3$

(e) Iteration: $t = 4$

(f) Iteration: $t = 5$ (converged)

Source: Zaki and Meira [3]
Clustering Cars based on Power and Weight

```python
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# load data
cars = pd.read_csv("/path-to/data/auto-mpg.data", sep="\s+", header=None)

# extract power and weight as data matrix X
X = cars.iloc[:, [3, 4]].values

# extract origin as target value y
y = cars.iloc[:, 7].values

# normalize data
min_max_scaler = MinMaxScaler()
min_max_scaler.fit(X)  # determine min and max
X_normalized = min_max_scaler.transform(X)

# apply k-Means
km = KMeans(n_clusters=3, random_state=0).fit(X_normalized)
```
Clustering Cars based on Power and Weight

```python
# plot cars
# U.S. : o / Europe: x / Japan : +
m = ['o' if o==1 else 'x' if o==2 else '+' for o in y]
# Cluster 1 : red / Cluster 2 : blue / Cluster 3 : green
c = ['red' if l==0 else 'blue' if l==1 else 'green' for l in km.labels_]
for i in range(0, len(X)):
    plt.scatter(X[i, 0], X[i, 1], color=c[i], marker=m[i])
plt.xlabel('Power [hp]')
plt.ylabel('Weight [lb]')
plt.show()
```
Clustering Cars based on Power and Weight
**k-Means++**

- **k-Means++** is an improved variant of k-Means that chooses initial cluster centroids more wisely
  - Pick centroid $\mu_1$ **randomly** as one of the data points
  - For $i = 2 \ldots k$:
    - Compute **minimal distance** for any data point to any of the centroids $\mu_1 \ldots \mu_{i-1}$
    - Pick centroid $\mu_i$ as one of the data point randomly with **probability**
      $$
      \frac{\text{md}(x_j)}{\sum_l \text{md}(x_l)}
      $$
    - so that data points **far** from already-picked centroids are preferred
How to Choose the Value of k?

- **Choosing the value** of $k$ can be difficult, if we have no further information or intuition about the data.

- One heuristic for choosing $k$ is the so-called **elbow method**
  - consider **decreasing values** of $k$ with their respective sum of squared distances
  - pick the **smallest value** of $k$, so that there is a **steep increase** when going to $k-1$
How to Choose the Value of $k$?

Elbow
4.2 Evaluating Clusterings

- How can we evaluate the quality of a clustering computed?

  - **External measures** assume that **ideal clustering** is known (e.g., class labels assigned to data points)
    \[ \mathcal{I} = \{ I_1, \ldots, I_{|\mathcal{I}|} \} \text{ with } I_i \subseteq \mathcal{D} \]

  - **Internal measures** assume no knowledge of ideal clustering (i.e., we only know the data points and the clustering)
Purity

- Purity of a cluster is the **fraction of data points** therein that belongs to the **dominant cluster** from the **ideal clustering**

\[
purity(C_i) = \frac{1}{|C_i|} \max_{I_j \in \mathcal{I}} |C_i \cap I_j|
\]

- Purity of a clustering is then the **weighted average** of the **purity values** of its clusters

\[
purity(C) = \sum_{C_i \in C} \frac{|C_i|}{n} \ purity(C_i)
\]
Purity

\[ \text{purity}(C_1) = \frac{2}{3} \]
\[ \text{purity}(C_2) = \frac{2}{4} \]
\[ \text{purity}(C_3) = \frac{1}{2} \]

\[ \text{purity}(C) = \frac{3}{9} \cdot \frac{2}{3} + \frac{4}{9} \cdot \frac{2}{4} + \frac{2}{9} \cdot \frac{1}{2} \approx 0.56 \]
BetaCV

- BetaCV, as an internal measure, considers the ratio of average distances between pairs of points within the same or different clusters

\[
\text{BetaCV}(C) = \frac{W_{in}/N_{in}}{W_{out}/N_{out}}
\]

with \( N_{in} \) and \( N_{out} \) as pairs of data points within the same or within different clusters

\[
N_{in} = \frac{1}{2} \sum_{C_i \in C} |C_i| (|C_i| - 1) \quad N_{out} = \frac{1}{2} \sum_{C_i, C_j \in C, C_i \neq C_j} |C_i| |C_j|
\]
BetaCV

- **BetaCV**, as an internal measure, considers the **ratio of average distances** between pairs of points within the same or different clusters.

\[
\text{BetaCV}(C) = \frac{W_{in}/N_{in}}{W_{out}/N_{out}}
\]

and \(W_{in}\) and \(W_{out}\) as the total distance of pairs of data points within the same or within different clusters.

\[
W_{in} = \frac{1}{2} \sum_{C_i \in C} \sum_{x,y \in C_i} d(x, y) \quad W_{out} = \frac{1}{2} \sum_{C_i, C_j \in C} \sum_{x \in C_i} \sum_{y \in C_j} d(x, y)
\]
Dunn Index

- **Dunn Index**, as another internal measure, compares the **minimal distance** between any pair of data points from **different clusters** against the **maximal distance** between any pair of data points from the **same cluster**

\[
\text{DunnIndex}(C) = \frac{\min_{x \in C_i, y \in C_j, C_i \neq C_j} d(x, y)}{\max_{x \in C_i, y \in C_i} d(x, y)}
\]
4.3 Hierarchical Clustering

- k-Means determines a **flat clustering** of data points; there is **no relationship** between the clusters.

- Hierarchical clustering determines a **sequence of increasingly fine-grained clusterings**

  \[ C_1, \ldots, C_n \]

- \( C_1 = \{ D \} \) contains **all data points** in a **single cluster**
- \( C_n = \{ \{ x_i \} : x_i \in D \} \) contains **one cluster per data point**
- Clustering \( C_i \) is **contained in** clustering \( C_{i-1} \)

  \[ \forall C_j \in C_i : \exists C_l \in C_{i-1} : C_j \subseteq C_l \]
Dendrogram

- Sequence of clusterings can be visualized in a **dendrogram**
Hierarchical Agglomerative vs. Divisive Clustering

- Hierarchical Agglomerative Clustering (HAC)
  - starts with the most fine-grained clustering \( C_n \)
  - proceeds bottom-up and merges the two closest clusters in \( C_i \) to obtain the more coarse-grained clustering \( C_{i-1} \)

- Hierarchical Divisive Clustering (HDC)
  - starts with the most coarse-grained clustering \( C_1 \)
  - proceeds top-down and splits one of the clusters in \( C_{i-1} \) to obtain the more fine-grained clustering \( C_i \)
Hierarchical Agglomerative Clustering

- Hierarchical Agglomerative Clustering (HAC)
  - starts with the most fine-grained clustering $C_n$
  - proceeds bottom-up and merges the two closest clusters in $C_i$ to obtain the more coarse-grained clustering $C_{i-1}$

- So far, we can only measure distance between data points, but we need a measure of distance between clusters
Linkage Criteria

- **Linkage criteria** measure distance between two clusters based on the distance between data points therein

  - **Single-Link**
    \[
    \delta(C_i, C_j) = \min\{d(x, y) \mid x \in C_i, y \in C_j\}
    \]

  - **Complete-Link**
    \[
    \delta(C_i, C_j) = \max\{d(x, y) \mid x \in C_i, y \in C_j\}
    \]

  - **Average-Link**
    \[
    \delta(C_i, C_j) = \frac{1}{|C_i||C_j|} \sum_{x \in C_i} \sum_{y \in C_j} d(x, y)
    \]
Hierarchical Agglomerative Clustering

1. Start with each data point in a separate cluster
2. \[ C_n = \{ \{ x_i \} : x_i \in D \} \];
3. 
4. for(int t = n; t > 1; t --) {
5.     // Determine the two clusters closest to each other
6.     \[ C_i^*, C_j^* = \arg\min_{C_i, C_j \in C_t : C_i \neq C_j} \delta(C_i, C_j) \];
7. 
8.     // Merge the two clusters
9.     \[ C_{t-1} = (C_t \setminus \{ C_i^*, C_j^* \}) \cup \{ C_i^* \cup C_j^* \} \];
10. }
HAC Example

- Consider the following data points in $\mathbb{R}^2$

\[
\begin{align*}
\mathbf{x}_1 &= (1, 0) \\
\mathbf{x}_2 &= (2, 1) \\
\mathbf{x}_3 &= (8, 0) \\
\mathbf{x}_4 &= (12, 1) \\
\mathbf{x}_5 &= (15, 1)
\end{align*}
\]

with distance matrix $d$

\[
\begin{bmatrix}
0.00 & 1.41 & 7.00 & 11.05 & 14.04 \\
0.00 & 6.08 & 10.00 & 13.04 \\
0.00 & 4.12 & 7.07 \\
0.00 & 3.00 \\
0.00 \\
\end{bmatrix}
\]
HAC with Single-Link Example

- HAC with single-link based on **distance matrix** $d$

\[
C_1 = \{\{x_1, x_2, x_3, x_4, x_5\}\}
\]

\[
C_2 = \{\{x_1, x_2\}, \{x_3, x_4, x_5\}\}
\]

\[
C_3 = \{\{x_1, x_2\}, \{x_3\}, \{x_4, x_5\}\}
\]

\[
C_4 = \{\{x_1, x_2\}, \{x_3\}, \{x_4\}, \{x_5\}\}
\]

\[
C_5 = \{\{x_1\}, \{x_2\}, \{x_3\}, \{x_4\}, \{x_5\}\}
\]

\[
d = \begin{bmatrix}
0.00 & 1.41 & 7.00 & 11.05 & 14.04 \\
0.00 & 6.08 & 10.00 & 13.04 \\
0.00 & 4.12 & 7.07 \\
0.00 & 3.00 \\
0.00 & 
\end{bmatrix}
\]
HAC with Complete-Link Example

- HAC with complete-link based on **distance matrix** $d$

\[
C_1 = \{ \{x_1, x_2, x_3, x_4, x_5\} \} \\
C_2 = \{ \{x_1, x_2, x_3\}, \{x_4, x_5\} \} \\
C_3 = \{ \{x_1, x_2\}, \{x_3\}, \{x_4, x_5\} \} \\
C_4 = \{ \{x_1, x_2\}, \{x_3\}, \{x_4\}, \{x_5\} \} \\
C_5 = \{ \{x_1\}, \{x_2\}, \{x_3\}, \{x_4\}, \{x_5\} \}
\]

\[
d = \begin{bmatrix}
0.00 & 1.41 & 7.00 & 11.05 & 14.04 \\
0.00 & 6.08 & 10.00 & 13.04 \\
0.00 & 4.12 & 7.07 \\
0.00 & 3.00 \\
0.00 
\end{bmatrix}
\]
Clustering Cars based on Power and Weight

```python
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from scipy.cluster.hierarchy import linkage
from scipy.cluster.hierarchy import dendrogram
import matplotlib.pyplot as plt

# load data
cars = pd.read_csv("/path-to/auto-mpg.data", sep="\s+", header=None)

# keep a sample of 50 cars
cars = cars.sample(50, random_state=0)

# extract labels
labels = cars.iloc[:,8].values

# extract power and weight as data matrix X
X = cars.iloc[:,[3,4]].values

# normalize data
min_max_scaler = MinMaxScaler()
min_max_scaler.fit(X)  # determine min and max
X_normalized = min_max_scaler.transform(X)
```
Clustering Cars based on Power and Weight

```python
# perform hierarchical agglomerative clustering using complete linkage
clusters = linkage(X_normalized, method='complete', metric='euclidean')

# plot dendrogram
dendrogram = dendrogram(clusters, labels=labels)
plt.tight_layout()
plt.ylabel('Euclidean distance')
plt.show()
```
Clustering Cars based on Power and Weight

Machine Learning / Chapter 4: Clustering
Summary

- **k-Means** as a simple-yet-popular clustering method that produces a flat clustering of data points

- **Elbow method** to determine a suitable number of clusters (or tune other parameters of a clustering method)

- **External** (e.g., purity) and **internal** (e.g., Dunn Index) measures to determine quality of clustering

- **Hierarchical clustering** determines a **sequence of clusterings** that can be visualized in a **dendrogram**