MATH5836: Data and Machine Learning

Week 9: Unsupervised Learning

Sarat Moka *UNSW, Sydney*

Key Topics

Book:

(A) Data Science and Machine Learning: Mathematical and Statistical Methods, by Kroese, Botev, Taimre, and Vaisman. Click here to download a pdf copy.

Recall that, unlike supervised learning, in unsupervised learning, there is no *response* (or output) variable y. The goal is to extract useful information and pattern from the given feature vectors $\boldsymbol{x}$.

In this lecture, we will learn about the curse of dimensionality and how to overcome this problem. We will then learn techniques to reduce the dimensions of datasets. We will also learn about unsupervised learning that will help you to categorise data by identifying the common features.

9.1 Curse of dimensionality

- The curse of dimensionality is a term coined by Richard Bellman in 1957 to describe the exponential increase in volume associated with adding extra dimensions to a mathematical space.

  Example 9.1.1

  Consider a d-dimensional hypercube with side length $2r$. The volume V of this hypercube is given by:
  $$V = (2r)^d.$$
  As d increases, the volume grows exponentially, making it harder to cover the space uniformly with a finite number of data points. For example, if $r = 1$, the volume of the hypercube is 2^d, which becomes impractically large as d increases.

- In the context of machine learning, the phrase 'curse of dimensionality' typically refers to the various challenges that arise when analyzing and modeling data with a high number of features.
- As the dimensionality of the feature space increases, several problems become more pronounced, impacting the performance and feasibility of learning algorithms.

Challenges with High-Dimensional Feature Spaces

1. **Data Sparsity**: As the number of features increases, the data points become more sparse in the feature space. For a fixed number of data points, the density of data points decreases exponentially with the increase in dimensionality, making it harder to capture the underlying patterns.
2. **Overfitting**: High-dimensional spaces provide more flexibility for models to fit the training data. However, this often leads to overfitting, where the model captures noise in the training data rather than the true underlying distribution. This results in poor generalization to new, unseen data.

3. **Increased Computational Complexity**: The computational cost of many machine learning algorithms increases with the number of features. For example, the time complexity of algorithms such as k-nearest neighbors (KNN) and support vector machines (SVMs) grows with the dimensionality, making them impractical for very high-dimensional data.

Hughes Phenomenon

- An important result related to the curse of dimensionality is the Hughes phenomenon, named after G.F. Hughes who described it in 1968.
- The Hughes phenomenon illustrates that, for a fixed number of training samples, the classification accuracy of a machine learning model initially improves as the number of features increases, but after a certain point, it starts to decline.
- This is due to the increased variance in the estimates of the model parameters as the number of features grows, leading to overfitting.

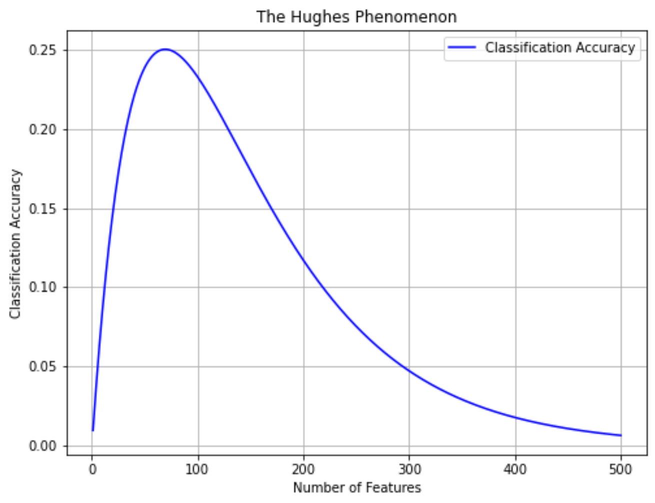


Figure 9.1: The Hughes phenomenon: classification accuracy vs. number of features.

- Mathematically, suppose we have a classification problem with N training samples and d features. If we denote the classification accuracy by $A(d, N)$, then Hughes' result can be summarized as

$$A(d, N) \approx \begin{cases} \text{Increases with } d & \text{for small } d \\ \text{Decreases with } d & \text{for large } d. \end{cases}$$

  This behavior is illustrated in Figure 9.1.

Distance Measures in High Dimensions

- In high-dimensional spaces, the concept of distance becomes less intuitive. For instance, consider two points $\mathbf{x}$ and $\mathbf{x}'$ in $\mathbb{R}^d$.
- The Euclidean distance between these points is

$$\|\mathbf{x} - \mathbf{x}'\|_2 = \sqrt{\sum_{i=1}^{d} (x_i - x_i')^2}.$$

As d increases, the distance between points tends to become more uniform.

Example

To illustrate this, let's assume we have two points $\mathbf{x}$ and $\mathbf{x}'$ in $\mathbb{R}^d$, where each coordinate is drawn independently from a uniform distribution over the interval $[0, 1]$. The Euclidean distance between these points is given by

$$\|\mathbf{x} - \mathbf{x}'\|_2 = \sqrt{\sum_{i=1}^{d} (x_i - x_i')^2}.$$

Due to the law of large numbers, as the dimensionality d increases, the sum of squared differences $\sum_{i=1}^{d} (x_i - x_i')^2$ will tend to be concentrated around its expected value. For uniformly distributed points, the expected value of $(x_i - x_i')^2$ is $\frac{1}{6}$. Thus, for large d, $\sum_{i=1}^{d} (x_i - x_i')^2 \approx \frac{d}{6}$.

So, the Euclidean distance will be approximately $\|\mathbf{x} - \mathbf{x}'\|_2 \approx \sqrt{\frac{d}{6}}$.

This means that the distances between different pairs of points will tend to be around $\sqrt{\frac{d}{6}}$. The variance of these distances decreases relative to the mean distance as d increases, making the distances between all pairs of points more similar, or "uniform".

- This phenomenon is evident when comparing the distances between the nearest and farthest neighbors of a point in high dimensions.
- The ratio of the distances between the nearest and farthest neighbors approaches 1 as d increases, leading to a loss of contrast in distances.
- In the context of nearest neighbor searches, this uniformity of distances implies that the ratio of the distance to the nearest neighbor to the distance to the farthest neighbor approaches 1.
- Mathematically, if dist_{min} and dist_{max} are the minimum and maximum distances from a given point to all other points in the dataset, then

$$\lim_{d \to \infty} \frac{\text{dist}_{\text{min}}}{\text{dist}_{\text{max}}} \to 1$$

- This makes it difficult to distinguish between the closest and farthest neighbors, which can significantly degrade the performance of algorithms that rely on distance measures, such as k-nearest neighbors (KNN) and clustering algorithms like k-means.

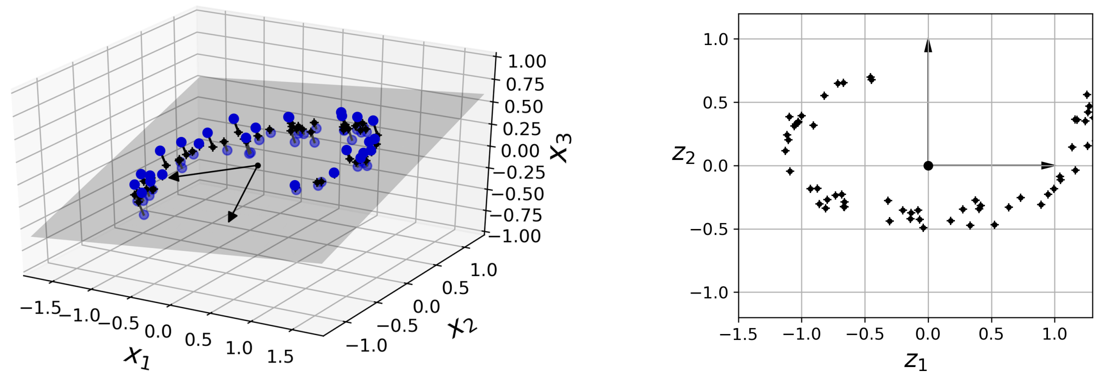
Implications and Mitigation Strategies

The curse of dimensionality implies that adding more features to a dataset can lead to diminishing returns and potentially worse performance if the model overfits the data. To mitigate these effects, several strategies can be employed:

1. **Feature Selection**: Select a subset of relevant features that contribute most to the predictive power of the model. Techniques include mutual information, recursive feature elimination, and regularization methods like Lasso.
2. **Dimensionality Reduction**: Transform the high-dimensional feature space into a lower-dimensional space while preserving the important information. Common techniques include Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA), and t-Distributed Stochastic Neighbor Embedding (t-SNE).
3. **Regularization**: Add a penalty term to the learning algorithm to constrain the model complexity, thereby preventing overfitting. Examples include Ridge regression, Lasso regression, and Elastic Net.
4. **Increase Training Data**: Collect more training samples to provide sufficient coverage of the high-dimensional space. This can be challenging and expensive but is often necessary for high-dimensional data.

9.2 Projection Based Dimensionality Reduction

- Often training instances are not distributed equally across all dimensions, and it's possible to transform them from a high-dimensional space to a low-dimensional space such that the low-dimensional space representation keeps most of the properties available in the original dimensions.
- For example, converting the following 3D data set to a new 2D subspace after projection retains most of the properties in the data set. See Figure 9.2.
- However, the above approach may not prove useful for some data sets. For example, for the Swiss roll toy data set shown in Figure 9.3, its projection onto a plane doesn't retain properties from the original data set properties.



(a) A 3d dataset (b) 2d projection of the 3d dataset

Figure 9.2: The new 2D data set after projection. Adapted from Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow by A. Géron, 2019, Sebastopol; CA: O'Reilly Media.

9.3 Manifold Learning and Dimensionality Reduction

- Manifold learning is a technique in machine learning used for dimensionality reduction. The goal of manifold learning is to discover the low-dimensional structure (manifold) embedded within high-dimensional data.
- Unlike traditional linear dimensionality reduction methods like Principal Component Analysis (PCA) (which is covered later), manifold learning techniques are capable of capturing the nonlinear structure of the data.
- Manifold learning techniques are powerful tools for uncovering the underlying structure in high-dimensional datasets, making them useful in various applications such as visualization, clustering, and classification.

Concept of Manifold

- A manifold is a topological space that locally resembles Euclidean space.
- For example, the surface of a sphere is a 2-dimensional manifold embedded in 3-dimensional space.
- In the context of high-dimensional data, we assume that the data points lie on or near a low-dimensional manifold within the high-dimensional space; see Figure 9.3.

Common Manifold Learning Techniques

There are several techniques for manifold learning, including:

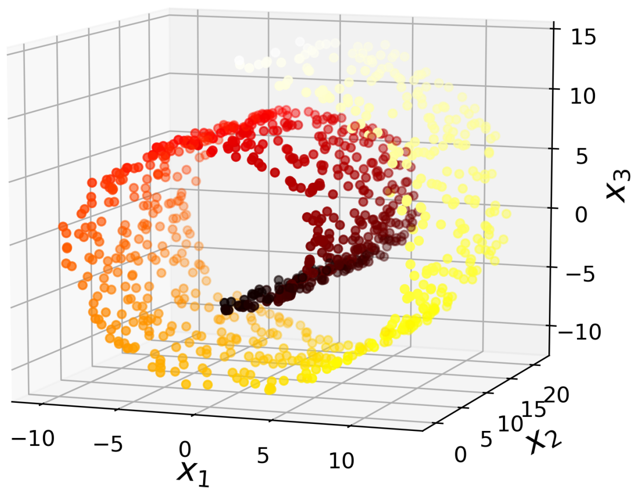


Figure 9.3: Swiss roll data set. Adapted from Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow by A. Géron, 2019, Sebastopol; CA: O'Reilly Media.

- **Isomap**: This method seeks to preserve the geodesic distances between all pairs of data points. It uses the shortest path algorithm to approximate the geodesic distance on the manifold.
- **Locally Linear Embedding (LLE)**: LLE preserves local relationships by attempting to keep each point's local neighborhood similar to that in the high-dimensional space.
- **t-Distributed Stochastic Neighbor Embedding (t-SNE)**: t-SNE converts similarities between data points to joint probabilities and minimizes the Kullback-Leibler divergence between the joint probabilities of the low-dimensional embedding and the high-dimensional data.
- **Laplacian Eigenmaps**: This method uses the graph Laplacian to preserve locality by constructing a weighted graph of the data points and then finding the low-dimensional representation that best preserves the graph structure.

Mathematical Formulation: Given a high-dimensional dataset $\mathbf{X} = \{x_1, x_2, \ldots, x_N\}$ with $x_i \in \mathbb{R}^D$, the aim is to find a low-dimensional representation $\mathbf{Y} = \{y_1, y_2, \ldots, y_N\}$ with $y_i \in \mathbb{R}^d$ where $d \ll D$.

- **Isomap**:

$$\text{Minimize} \quad \sum_{i<j} (\delta_{ij}^{(G)} - \|y_i - y_j\|)^2$$

  where $\delta_{ij}^{(G)}$ is the geodesic distance between points x_i and x_j in the high-dimensional space.

- **LLE**:

$$\text{Minimize} \quad \sum_i \|y_i - \sum_{j \in \mathcal{N}(i)} w_{ij} y_j\|^2$$

subject to $\sum_{j\in\mathcal{N}(i)} w_{ij} = 1$, where $\mathcal{N}(i)$ denotes the set of neighbors of x_i.

- **t-SNE**:
$$\text{Minimize} \quad KL(P\|Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}$$
where p_{ij} and q_{ij} are the probabilities that points x_i and x_j are neighbors in high-dimensional and low-dimensional spaces, respectively.

- **Laplacian Eigenmaps**:
$$\text{Minimize} \quad \sum_{i,j} \|y_i - y_j\|^2 w_{ij}$$
where w_{ij} is the weight of the edge connecting x_i and x_j in the graph representation.

Remark

Manifold learning often operates under the implicit assumption that the task at hand (e.g., classification or regression) will be simpler when expressed in the lower-dimensional space of the manifold. For instance, in the top row of Figure 9.4, the Swiss roll is divided into two classes: in the 3D space (left), the decision boundary is quite complex, but in the 2D unrolled manifold space (right), the decision boundary is a straight line.

However, this assumption does not always hold true. For example, in the bottom row of Figure 9.4, the decision boundary is at $x_1 = 5$. This boundary is very simple in the original 3D space (a vertical plane), but it becomes more complex in the unrolled manifold (a collection of four independent line segments).

9.4 Principal Component Analysis (PCA)

- The main goal of PCA is to reduce the dimensionality of a given data set with many features. This is why PCA is also referred to as a *feature reduction* or *feature extraction* method.
- In PCA, we start with a d-dimensional data set consisting of vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathbb{R}^d$.
- Our aim is to represent this d-dimensional input data using n feature vectors of dimension $k < d$. This is achieved through the following steps:
- First, write the input data as a matrix:

$$\mathbf{X} = \begin{bmatrix} \boldsymbol{x}_1^\top \\ \boldsymbol{x}_2^\top \\ \vdots \\ \boldsymbol{x}_n^\top \end{bmatrix} = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,d} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,d} \\ \vdots & \vdots & \vdots & \vdots \\ x_{n,1} & x_{n,2} & \cdots & x_{n,d} \end{bmatrix}.$$

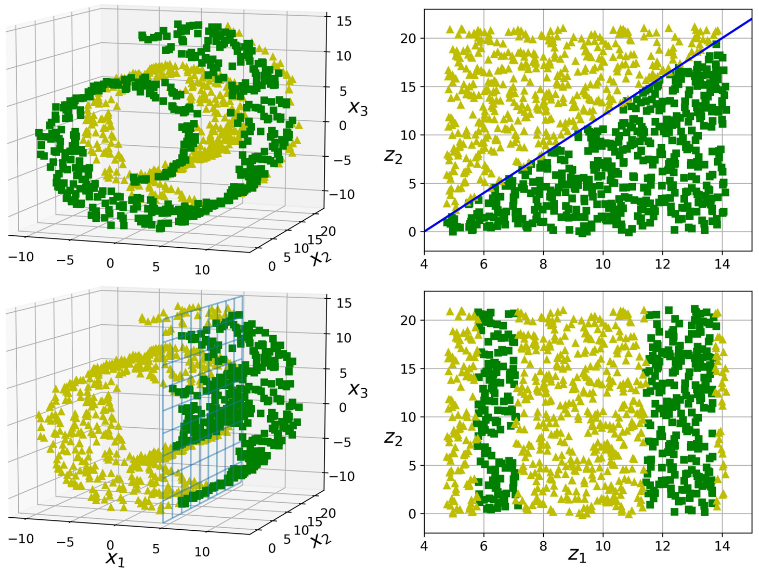


Figure 9.4: The decision boundary may not always be simpler with lower dimensions. Adapted from Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow by A. Géron, 2019, Sebastopol; CA: O'Reilly Media

- Mean-centering the data:
  To ensure that the data has zero mean, we subtract the column-wise sample mean from each data point:
  $$x'_{i,j} = x_{i,j} - \frac{1}{n}\sum_{r=1}^{n} x_{r,j} \quad \text{for all } i = 1,\ldots,n,\ j = 1,\ldots,d.$$
  Let the mean-centered data matrix be denoted by $\mathbf{X}'$. Each row of $\mathbf{X}'$ is the vector $\boldsymbol{x}'_i = \boldsymbol{x}_i - \bar{\boldsymbol{x}}$, where $\bar{\boldsymbol{x}}$ is the sample mean vector.

- Let $\boldsymbol{\Sigma}$ denote the covariance matrix of the underlying distribution. It is defined as:
  $$\boldsymbol{\Sigma} = \mathbb{E}\left[(\boldsymbol{x} - \mathbb{E}[\boldsymbol{x}])(\boldsymbol{x} - \mathbb{E}[\boldsymbol{x}])^\top\right].$$
  and is estimated from the centered data as:
  $$\widehat{\boldsymbol{\Sigma}} = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{x}'_i \boldsymbol{x}'^\top_i.$$

- Since $\widehat{\boldsymbol{\Sigma}}$ is a symmetric matrix, it has the spectral decomposition (i.e., singular value decomposition):
  $$\widehat{\boldsymbol{\Sigma}} = \mathbf{U}\mathbf{D}\mathbf{U}^\top,$$
  where $\mathbf{U}$ is an orthogonal matrix of eigenvectors, and $\mathbf{D}$ is a diagonal matrix of eigenvalues.

- **Principal Components:** The k columns of $\mathbf{U}$ corresponding to the k largest diagonal elements of $\mathbf{D}$ are called the k principal components.

- Let $\mathbf{U}_k$ be the matrix whose columns are the top k principal components.
- The transformation
$$\mathbf{z}' := \mathbf{U}_k^\top \boldsymbol{x}'$$
projects the centered vector $\boldsymbol{x}'$ onto the k-dimensional subspace spanned by the principal components, and expresses it in the principal component basis.
- That is, the vectors $\mathbf{z}'_1, \ldots, \mathbf{z}'_n \in \mathbb{R}^k$ are the k-dimensional representations of the original data vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$.
- This procedure is called PCA.
- Let $\mathbf{D} = \mathrm{diag}(d_{1,1}, d_{2,2}, \ldots, d_{d,d})$. Then, $d_{l,l}$ represents the variance of the data in the l-th principal component direction. Hence, the total variance in the data can be measured by:
$$\nu = \sum_{l=1}^{d} d_{l,l}.$$

Exercise 9.4.1

Why to select k principal components?

Ans: It minimises the total squared distance error between the projected points and the original points, i.e., keeps as much variance of the given data as possible. In other words, for any $k \le d$, the eigenvectors $\mathbf{u}_1, \ldots, \mathbf{u}_k$ corresponds to the k principal components are the solution of the following optimization:

$$\min_{\mathbf{v}_1,\ldots,\mathbf{v}_k} \sum_{i=1}^{n} \|\boldsymbol{x}_i - \mathbf{V}_k \mathbf{V}_k^\top \boldsymbol{x}_i\|^2, \tag{9.1}$$

where the minimization is taken over all k unit length orthogonal vectors $\{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$ in $\mathbb{R}^d$ and $\mathbf{V}_k = [\mathbf{v}_1, \ldots, \mathbf{v}_k]$.

9.5 K-means Clustering

- In *clustering*, the goal is to divide the given unlabelled feature vectors $D = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ into a set of K groups (or clusters), so that the samples belong to the same group are more *similar* to each other than the samples belong to different groups.

Q: What is difference between classification and clustering?

- K-means clustering is a simple heuristic method which ignores the distributional properties of the data.

- In K-means clustering, the entire feature vector space is divided into K regions using a distance function $\text{Dist}(\boldsymbol{x}, \boldsymbol{x}')$. Some well-known choice of the distance functions are

$$\text{Dist}(\boldsymbol{x}, \boldsymbol{x}') = \begin{cases} \|\boldsymbol{x} - \boldsymbol{x}'\| = \sqrt{\sum_{i=1}^d (x_i - x_i')^2}, & \text{(Euclidean)} \\ \sum_{i=1}^d |x_i - x_i'|, & \text{(Manhattan)} \\ \max_{i=1,\dots,d} |x_i - x_i'|, & \text{(Maximum)} \\ \sum_{i=1}^d \mathbb{I}(x_i \neq x_i'), & \text{(Hamming distance for binary features)} \end{cases}$$

- Once we fix the distance function, we can choose K points $\boldsymbol{c}_1, \dots, \boldsymbol{c}_K$ as *cluster centers*. Further, we partition the whole feature space into regions $\mathcal{R}_1, \dots, \mathcal{R}_K$, where

$$\mathcal{R}_k = \{\boldsymbol{x} : \text{Dist}(\boldsymbol{x}, \boldsymbol{c}_k) \leq \text{Dist}(\boldsymbol{x}, \boldsymbol{c}_j), \quad \forall j \neq k\}.$$

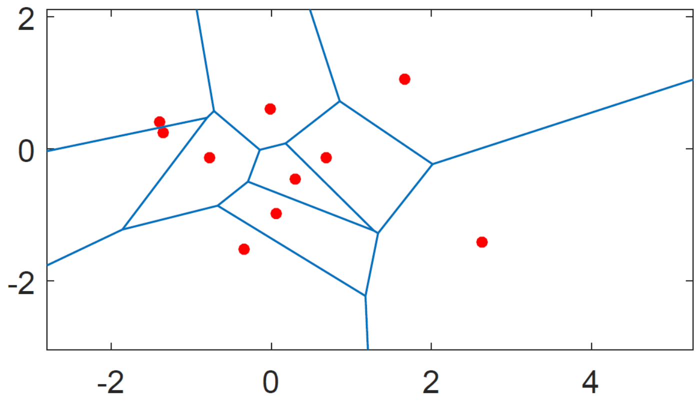


Figure 9.5: Voronoi diagram

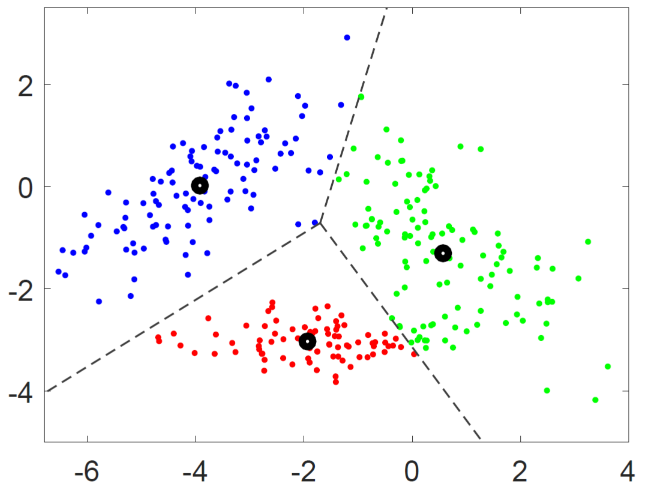
Q: How to find the centers $\boldsymbol{c}_1, \dots, \boldsymbol{c}_K$?

- Starting from an initial guess for the centers, it iteratively finds the new centers as the centroids of the points in each Voronoi cell formed by the current centers.
- Let's see the algorithm:

Algorithm 1: *K*-Means Algorithm

```
Input: The training data, number of clusters $K$, initial centers $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_K$.
Output: Cluster centers and cells
while a stopping criteria is not met do
    $\mathcal{R}_1, \ldots, \mathcal{R}_K \leftarrow \varnothing$
    for $i = 1$ to $n$ do
        $\mathbf{d} \leftarrow (\mathsf{Dist}(\mathsf{x_i}, \mathsf{c_1}), \ldots, \mathsf{Dist}(\mathsf{x_i}, \mathsf{c_K}))$
        $k \leftarrow \arg\min_j d_j$
        $\mathcal{R}_k \leftarrow \mathcal{R}_k \cup \{\boldsymbol{x}_i\}$
    end
    for $k = 1$ to $K$ do
        $\boldsymbol{c}_k \leftarrow \frac{\sum_{\boldsymbol{x} \in \mathcal{R}_k} \boldsymbol{x}}{|\mathcal{R}_k|}$
    end
end
Return: $\{\mathcal{R}_k\}$ and $\{\boldsymbol{c}_k\}$ ;
```

- An example of K-means clustering can be seen in the following figure. The thick dark circles are the final centroids.



9.6 Hierarchical clustering

Hierarchical clustering is a method of cluster analysis which seeks to build a hierarchy of clusters. It is particularly useful in fields such as evolutionary biology for constructing phylogenetic trees. There are two primary types of hierarchical clustering:

1. **Agglomerative Clustering (Bottom-Up Approach)**: This method starts with each data point as its own cluster and iteratively merges the closest pairs of clusters until only one cluster

remains or a certain number of clusters is achieved.

2. **Divisive Clustering (Top-Down Approach)**: This method starts with all data points in a single cluster and iteratively splits the clusters into smaller clusters.

Agglomerative Clustering

Steps in Agglomerative Clustering:

1. **Initialization**: Each data point is considered as a single cluster.
2. **Distance Calculation**: Compute the distance between each pair of clusters.
3. **Merge Clusters**: Merge the pair of clusters with the smallest distance.
4. **Update Distances**: Recompute the distances between the new cluster and the remaining clusters.
5. **Repeat**: Continue the process until a single cluster remains or the desired number of clusters is reached.

Example

Consider a dataset with points labeled from 1 to 8. Initially, each point is its own cluster. As the algorithm proceeds, clusters are merged step-by-step. For example, points 1 and 2 might merge to form a new cluster, labeled 9. Points 3 and 4 merge to form cluster 10. The process continues until a hierarchy is formed, which can be represented using a **dendrogram**.

Dendrogram

A dendrogram is a tree-like diagram that records the sequences of merges or splits. It provides a visual representation of the hierarchical clustering process. The vertical axis represents the distance or dissimilarity between clusters, while the horizontal axis represents the individual data points or clusters. See Figure 9.6 for an illustration.

Linkage Criteria

The way distances between clusters are defined can vary, leading to different types of linkage criteria:

- **Single Linkage**: The distance between two clusters is defined as the minimum distance between any single data point in the first cluster and any single data point in the second cluster.

$$d_{\min}(I, J) = \min_{i \in I, j \in J} \operatorname{dist}(x_i, x_j).$$

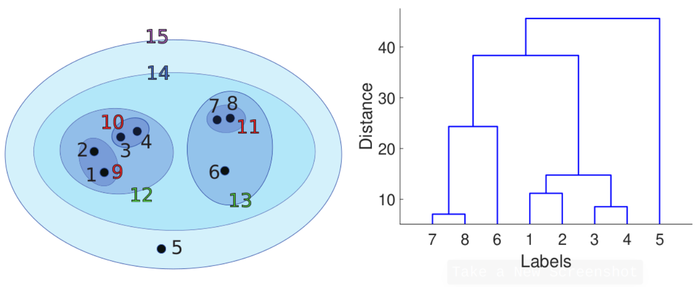


Figure 9.6: Dendrogram Example: Left: a cluster hierarchy of 15 clusters. Right: the corresponding dendrogram.

- **Complete Linkage**: The distance between two clusters is defined as the maximum distance between any single data point in the first cluster and any single data point in the second cluster.

$$d_{\max}(I, J) = \max_{i\in I, j\in J} \operatorname{dist}(x_i, x_j).$$

- **Average Linkage**: The distance between two clusters is defined as the average distance between all pairs of data points from the two clusters.

$$d_{\text{avg}}(I, J) = \frac{1}{|I||J|} \sum_{i\in I} \sum_{j\in J} \operatorname{dist}(x_i, x_j).$$

- **Ward's Minimum Variance Method**: This method aims to minimize the total within-cluster variance. The distance between two clusters is defined as the increase in the total within-cluster variance after merging the two clusters.

$$d_{\text{Ward}}(I, J) = \frac{|I||J|}{|I| + |J|} \|\mathbf{x}_I - \mathbf{x}_J\|^2$$

where $\mathbf{x}_I$ and $\mathbf{x}_J$ are the centroids of clusters I and J respectively.

Algorithm for Agglomerative Clustering

Here is a pseudocode representation of the agglomerative clustering algorithm:

Algorithm 4.7.1: Greedy Agglomerative Clustering

```
1. Initialize the set of cluster identifiers: $I = \{1, \ldots, n\}$.
2. Initialize the corresponding label sets: $L_i = \{i\}, \forall i \in I$.
3. Initialize a distance matrix $D = [d_{ij}]$ with $d_{ij} = d(\{i\}, \{j\})$.
4. For $k = n + 1$ to $2n - K$ do:
    (a) Find $i$ and $j > i$ in $I$ such that $d_{ij}$ is minimal.
```

   (b) Create a new label set $L_k := L_i \cup L_j$.
   (c) Add the new identifier k to I and remove the old identifiers i and j from I.
   (d) Update the distance matrix D with respect to the identifiers i, j, and k.

5. Return $L_i, \forall i = 1, \ldots, 2n - K$.

Lance-Williams Update

- The Lance-Williams update formula is a recursive method for calculating the distance between clusters in hierarchical clustering. It allows for efficient computation of cluster distances when clusters are merged during the agglomerative clustering process.
- Suppose at some stage in the clustering algorithm, clusters I and J, with identifiers i and j respectively, are merged into a new cluster $K = I \cup J$ with identifier k. Let M, with identifier m, be another existing cluster. The Lance-Williams update rule provides a way to update the distance d_{km} between the new cluster K and the cluster M using the distances d_{im}, d_{jm}, and d_{ij}.
- The general form of the Lance-Williams update formula is:
$$d_{km} = \alpha d_{im} + \beta d_{jm} + \gamma d_{ij} + \delta |d_{im} - d_{jm}|,$$
where α, β, γ, and δ are constants that depend on the chosen linkage criterion. These constants are derived based on the specific characteristics of the clusters involved, such as their sizes.
- Table 9.1 lists the constants α, β, γ, and δ for some common linkage criteria.

Table 9.1: Constants for the Lance-Williams update rule for various linkage functions, where n_i, n_j, and n_m denote the number of elements in the corresponding clusters.

Linkage	α	β	γ	δ
Single	$\frac{1}{2}$	$\frac{1}{2}$	0	$-\frac{1}{2}$
Complete	$\frac{1}{2}$	$\frac{1}{2}$	0	$\frac{1}{2}$
Group Average	$\frac{n_i}{n_i+n_j}$	$\frac{n_j}{n_i+n_j}$	0	0
Ward	$\frac{n_i+n_m}{n_i+n_j+n_m}$	$\frac{n_j+n_m}{n_i+n_j+n_m}$	$-\frac{n_m}{n_i+n_j+n_m}$	0

- For example, in the single linkage criterion, the distance between two clusters is defined as the minimum distance between any pair of points from the two clusters. Using the Lance-Williams update, this is expressed as:
$$d_{km} = \min\{d_{im}, d_{jm}\} = \frac{d_{im}}{2} + \frac{d_{jm}}{2} - \frac{|d_{im} - d_{jm}|}{2}.$$
- In the complete linkage criterion, the distance between two clusters is defined as the maximum distance between any pair of points from the two clusters. The update formula for complete linkage is:
$$d_{km} = \max\{d_{im}, d_{jm}\} = \frac{d_{im}}{2} + \frac{d_{jm}}{2} + \frac{|d_{im} - d_{jm}|}{2}.$$

- In practical applications, the Lance-Williams update formula significantly reduces the computational complexity of hierarchical clustering algorithms by enabling efficient recalculation of distances as clusters are merged. This makes it feasible to apply hierarchical clustering to larger datasets.

Divisive Approach

- In contrast to the bottom-up (agglomerative) approach to hierarchical clustering, the divisive approach starts with a single cluster containing all data points. This cluster is then recursively divided into smaller clusters. The goal of each division is to create two clusters that are as "dissimilar" as possible, which can then be further divided in subsequent steps.
- The divisive approach can utilize the same linkage criteria as agglomerative clustering to determine the best way to divide a parent cluster into two child clusters. These linkage criteria include single linkage, complete linkage, average linkage, and Ward's minimum variance linkage. The choice of linkage criterion affects how the distance between clusters is calculated and, consequently, how the clusters are split.
- One challenge with the divisive approach is that its implementation tends to scale poorly with the number of data points, n. This issue is related to the well-known *max-cut problem*, which is defined as follows: given an $n \times n$ matrix of positive costs c_{ij} for $i, j \in \{1, \ldots, n\}$, partition the index set $I = \{1, \ldots, n\}$ into two subsets J and K such that the total cost across the sets, $\sum_{j \in J} \sum_{k \in K} c_{jk}$, is maximal. Solving this problem exactly is computationally expensive, but approximate solutions can be found using heuristic methods such as the cross-entropy algorithm.

9.7 Density-based Spatial Clustering of Applications with Noise (DBSCAN)

- This is a simple data clustering algorithm proposed by Martin Ester, Hans-Peter Kriegel, Jörg Sander and Xiaowei Xu in 1996 (see Reference (B)). It is one of the most cited articles in scientific literature. The algorithm was awarded the test of time award in 2014 at the leading data mining conference, ACM SIGKDD.
- Let $D = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ is the set of given unlabelled d-dimensional data points. The goal is again to divide the data into a set of groups (or clusters) so that the points belong to the same group are more similar to each other than the samples belong to different groups.
- This algorithm takes two parameters: radius parameter r that specifies a neighbourhood around a point, and a parameter n_{min} to determine a point is *core point* or not.
- **Definitions:**
  - *Core point:* A point $\boldsymbol{x} \in D$ is called core point if there are n_{min} in D that are within r distance from $\boldsymbol{x}$ (excluding $\boldsymbol{x}$).

  - *Directly reachable:* If $\boldsymbol{x}$ is a core point, a point $\mathbf{y}$ is said to be directly reachable from $\boldsymbol{x}$ if $\mathbf{y}$ is within a distance r from $\boldsymbol{x}$. Note that we do not say $\boldsymbol{x}$ and $\mathbf{y}$ are directly reachable even if they are within a distance r from each other if none of them is a core point.
  - *Reachable:* A point $\boldsymbol{x}'$ is reachable from a core point $\boldsymbol{x}$, if there is a path of points $\boldsymbol{x}_{(1)}, \dots, \boldsymbol{x}_{(k)}$ for some k such that $\boldsymbol{x}_{(1)} = \boldsymbol{x}$, $\boldsymbol{x}_{(k)} = \boldsymbol{x}'$, and $\boldsymbol{x}_{(i+1)}$ is directly reachable from $\boldsymbol{x}_{(i)}$ for all $i = 1, \dots, k-1$. Note that all the points $\boldsymbol{x}_{(i)}$ for $i = 1, \dots, k-1$ are core points.
  - *Outliers or Noise:* All the points D that are not reachable from any other points are called outliers or noise.

- The number of clusters obtaining by algorithm depends on r and n_{min}.

Algorithm 2: DBSCAN

```
Data: $D$, Dist, $r$, and $n_{min}$
Result: Clustered data
$C \leftarrow 0$ ;
for each $\boldsymbol{x}$ in $D$ do
    if $\boldsymbol{x}$ is unlabelled then
        Find neighbours $N_{\boldsymbol{x}}$ of $\boldsymbol{x}$ that are within $r$ distance;
        if $|N_{\boldsymbol{x}}| < n_{min}$ then
            label($\boldsymbol{x}$) = Noise;
        else
            $C \leftarrow C + 1$;
            label($\boldsymbol{x}$) = $C$;
            for every point $\boldsymbol{x}'$ reachable from $\boldsymbol{x}$ do
                label($\boldsymbol{x}'$)= $C$;
            end
        end
    end
end
return Clustered data ;
```

- The above algorithm sequentially goes through all the points in D.
- If a point $\boldsymbol{x}$ is not a core point, it labels $\boldsymbol{x}$ as a outlier (or noise).
- On the other hand, if $\boldsymbol{x}$ is a core point, it increases the cluster label C by 1 and labels $\boldsymbol{x}$ and every point $\mathbf{y}$ that is reachable from $\boldsymbol{x}$ as points of cluster C. In this process, the points that are already labelled as noise can be relabelled as cluster C points.

Exercise 9.7.1

Use your choice of programming language to implement the above algorithm.