Self-Notes on [LDA, CCA, Factor Analysis, Clustering, K-Meas, Hierarchical, DBSCAN, GMM]

[Unat Tekşen] [504241592]

May 2, 2025

1 LDA – Linear Discriminant Analysis

Linear Discriminant Analysis (LDA) is a technique that projects data into a lower-dimensional space while preserving class separability.

Unlike PCA, which is unsupervised and focuses on maximum variance, LDA focuses on maximum class separability using label information.

- We want to maximize $(m_1 - m_2)^2$ and minimize $s_1^2 + s_2^2$
- So, maximize $(m_1 - m_2)^2$ and maximize $\frac{1}{s_1^2+s_2^2}$
- $J(w) + \underset{w}{\operatorname{argmax}} \frac{(m_1-m_2)^2}{s_1^2+s_2^2}$
- $m_1 = w^T m_1$ (left side is italic)
- $m_2 = w^T m_2$ (left side is italic)

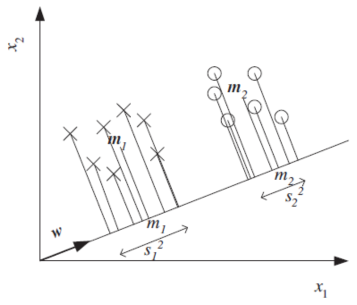

Figure 1: Two-dimensional, two-class data projected on w. Retrieved from [1], page 129.

Variances after projection (left side S1 and S2 are italic):

$$\begin{aligned} s_1^2 &= \sum (w^T x^{(i)} - w^T m_1)^2 \\ s_2^2 &= \sum (w^T x^{(i)} - w^T m_2)^2 \end{aligned} \tag{1}$$

LDA's loss term (argmax):

$$= \frac{(w^T m_1 - w^T m_2)}{\sum (w^T x^{(i)} - w^T m_1)^2 + \sum (w^T x^{(i)} - w^T m_2)^2} \tag{2}$$

LDA's loss term (argmax):

$$= \frac{(w^T m_1 - w^T m_2)(w^T m_1 - w^T m_2)}{\sum (w^T x^{(i)} - w^T m_1)(w^T x^{(i)} - w^T m_1) + \sum (w^T x^{(i)} - w^T m_2)(w^T x^{(i)} - w^T m_2)} \tag{3}$$

Rewrite:

$$\begin{aligned} (w^T m_1 - w^T m_2) &= (m_1^T w - m_2^T w) \\ w^T x^{(i)} - w^T m_1 &= (x^{(i)})^T w - m_1^T w \\ w^T x^{(i)} - w^T m_2 &= (x^{(i)})^T w - m_2^T w \end{aligned} \tag{4}$$

Finally:

$$= \frac{w^T (m_1 - m_2)(m_1 - m_2)^T w}{w^T \sum (x_i - m_1)(x_i - m_1)^T w + w^T \sum (x_i - m_2)(x_i - m_2)^T w} \tag{5}$$

Between scatter matrix:

$$S_B = (m_1 - m_2)(m_1 - m_2)^T \tag{6}$$

Within scatter matrix:

$$S_W = s_1^2 + s_2^2 \tag{7}$$

S_1, S_2 are variances before projection. We can rewrite the whole equation:

$$J(w) = \frac{w^T S_B w}{w^T S_W w} \tag{8}$$

Note:

$$\begin{aligned} \frac{\partial}{\partial \Theta} \Theta^T A \Theta &= (A + A^T)\Theta \\ &= 2A\Theta \quad (A = A^T) \end{aligned} \tag{9}$$

Solve for:

$$\frac{\partial J(w)}{\partial w} = 0 \tag{10}$$

$$\frac{(2S_B w)(w^T S_W w) - (2S_W w)(w^T S_B w)}{(w^T S_W w)^2} = 0 \tag{11}$$

Scalars: $(w^T S_W w)$, $(w^T S_B w)$

$$\frac{w^T S_W w (S_B w)}{w^T S_W w} - \frac{w^T S_B w}{w^T S_W w}(S_W w) = 0 \tag{12}$$

$$\frac{w^T S_B w}{w^T S_W w} = \lambda \quad \text{because it is a scalar} \tag{13}$$

$$S_B w = \lambda S_W w \tag{14}$$

$$S_W^{-1} S_B w = \lambda S_W^{-1} S_W w \tag{15}$$

$$S_W^{-1} S_B w = A \tag{16}$$

$$Aw = \lambda w \tag{17}$$

Extra analysis:

$$S_B x = \lambda x \tag{18}$$

$$(m_1 - m_2)(m_1 - m_2)^T x = \lambda x \tag{19}$$

It will be in the same direction with $(m_1 - m_2) \implies \propto (m_1 - m_2)$

$$S_W^{-1} S_B w = \lambda w \tag{20}$$

$$S_W^{-1} \alpha (m_1 - m_2) = \lambda w \tag{21}$$

$$w = \frac{\lambda}{\alpha} S_W^{-1} (m_1 - m_2) \tag{22}$$

Summarized working procedure:

- **Step 1: Compute the mean vectors for each class**
  For each class i, compute the class mean vector μ_i:

  $$\mu_i = \frac{1}{N_i} \sum_{x \in C_i} x \tag{23}$$

  where N_i is the number of samples in class i, and C_i is the set of samples belonging to that class.

- **Step 2: Compute the within-class scatter matrix S_W**
  This measures how samples scatter around their class mean:

  $$S_W = \sum_{i=1}^{k} \sum_{x \in C_i} (x - \mu_i)(x - \mu_i)^T \tag{24}$$

- **Step 3: Compute the between-class scatter matrix S_B**
  First, compute the overall mean μ of the dataset. Then calculate:

$$S_B = \sum_{i=1}^{k} N_i(\mu_i - \mu)(\mu_i - \mu)^T \tag{25}$$

- **Step 4: Solve the generalized eigenvalue problem**
  Solve for eigenvectors and eigenvalues:

$$S_W^{-1} S_B v = \lambda v \tag{26}$$

  where v are the eigenvectors (linear discriminants) and λ are the corresponding eigenvalues.

- **Step 5: Select top eigenvectors**
  Sort the eigenvectors by descending eigenvalues and select the top d eigenvectors (maximum $d = k - 1$, where k is the number of classes). Form a projection matrix W from the selected eigenvectors.

- **Step 6: Project the data onto the new subspace**
  Transform the original dataset X using:

$$X_{\text{new}} = XW \tag{27}$$

  where X_{new} is the LDA-transformed data with reduced dimensionality and improved class separability.

2 CCA – Canonical Correlation Analysis

Canonical Correlation Analysis (CCA) is a statistical method used to understand the relationship between two sets of variables. CCA finds pairs of linear combinations (called canonical variables) — one from each variable set — such that the correlation between the two combinations is maximized.

In speech recognition, let's say we have acoustic features and images. We want to reduce dimension.

- $X = \{x_i, y_i\}_{i=1}^{n}$

different dimensions

- $x \in \mathbb{R}^p$, $y \in \mathbb{R}^q$

we are interested in lower dimensional space $u \in \mathbb{R}^d, v \in \mathbb{R}^d$ project into lower dimensional space such that we have maximum correlation.

$$\rho = \frac{\sigma_{uv}}{\sqrt{\sigma_u^2}\sqrt{\sigma_v^2}}$$

Variances:

$$S_{xx} = \text{Cov}(x) = E[(x - E[x])^2] = E[(x - \mu_x)^2] \tag{28}$$

Cov(x) matrix is $p \times p$

$$S_{yy} = \text{Cov}(y) = E[(y - E[y])^2] = E[(y - \mu_y)^2] \tag{29}$$

Cov(y) matrix is $q \times q$

$$S_{xy} = \text{Cov}(x, y) = E[(x - E[x])(y - E[y])^T], \quad d \times e \tag{30}$$

$$S_{yx} = \text{Cov}(y, x) = E[(y - E[y])(x - E[x])^T], \quad e \times d \tag{31}$$

$$\rho = \frac{\text{cov}(w^T x, v^T y)}{\sqrt{\text{var}(w^T x)}\sqrt{\text{var}(v^T y)}} \tag{32}$$

$$= \frac{w^T \text{cov}(x, y) v}{\sqrt{w^T \text{var}(x) w}\sqrt{v^T \text{var}(y) v}} \tag{33}$$

We can rewrite the loss term:

$$\underset{w,v}{\text{argmax}} \frac{w^T S_{xy} v}{\sqrt{w^T S_{xx} w}\sqrt{v^T S_{yy} v}} \tag{34}$$

Constraint: if we find w, v s.t. $w^T S_{xx} w = 1$, $v^T S_{yy} v = 1$ We will use Lagrange multipliers for this optimization. We don't know Lagrange multiplier, but we will find.

$$L(w, v, \lambda_x, \lambda_y) = w^T S_{xy} v - \frac{\lambda_x}{2}(w^T S_{xx} w - 1) - \frac{\lambda_y}{2}(v^T S_{yy} v - 1) \tag{35}$$

$$\frac{\partial L}{\partial w} = S_{xy} v - \lambda_x S_{xx} w = 0 \tag{36}$$

$$\frac{\partial L}{\partial v} = S_{yx} w - \lambda_y S_{yy} v = 0 \tag{37}$$

Assume $w^T S_{xy} v = a^T v \quad (a = S_{yx} w)$:

$$S_{xy} v = \lambda_x S_{xx} w \tag{38}$$

$$\lambda_x w = S_{xx}^{-1} S_{xy} v \tag{39}$$

$$S_{yx} w = \lambda_y S_{yy} v \tag{40}$$

$$\lambda_y v = S_{yy}^{-1} S_{yx} w \tag{41}$$

Substitute w:

$$\lambda_y v = \frac{1}{\lambda_x} S_{yy}^{-1} S_{yx} S_{xx}^{-1} S_{xy} v \tag{42}$$

$$\lambda_x \lambda_y v = S_{yy}^{-1} S_{yx} S_{xx}^{-1} S_{xy} v \tag{43}$$

$$\alpha = \lambda_x \lambda_y \tag{44}$$

$$A = S_{yy}^{-1} S_{yx} S_{xx}^{-1} S_{xy} \tag{45}$$

$$\alpha v = A v \tag{46}$$

$$v = \frac{1}{\lambda_y} S_{yy}^{-1} S_{yx} w \quad \text{(Substitute to first equation)} \tag{47}$$

$$S_{xy} \frac{1}{\lambda_y} S_{yy}^{-1} S_{yx} w = \lambda_x S_{xx} w \tag{48}$$

$$S_{xx}^{-1} S_{xy} S_{yy}^{-1} S_{yx} w = \lambda_x \lambda_y w \tag{49}$$

$$A w = \alpha w \tag{50}$$

3 Factor Analysis

Factor Analysis is a generative model that tries to explain observed data (high-dimensional) in terms of a smaller number of unobserved (latent) variables. It assumes that your data is generated by:

- A low-dimensional latent variable z
- A linear transformation of z
- Gaussian noise added on top
- **Latent Variable Definition**

  We consider that our observed data $x \in \mathbb{R}^D$ is generated from a lower-dimensional latent variable $z \in \mathbb{R}^d$ via a linear transformation. In projection-based methods like PCA, we write:

  $$z = W^\top x \tag{51}$$

  However, in **Factor Analysis**, we flip this: we generate x from z.

- **Latent Variable Distribution**

  We assume the latent variable z comes from a standard Gaussian distribution:

$$z \sim \mathcal{N}(0, I) \tag{52}$$

- **Linear Mapping from Latent to Observed Space**

  We use a matrix $V \in \mathbb{R}^{D \times d}$ to project z into the observed space:

$$V = \begin{bmatrix} 2 \\ 1 \end{bmatrix}, \quad \Psi = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$$

  Here, Ψ represents the diagonal covariance matrix of the noise (specific variances for each dimension).

- **Data Generation with Noise and Mean**

  The observed data $x^{(i)}$ is generated by linearly transforming $z^{(i)}$, adding a mean μ, and Gaussian noise ϵ:

$$x^{(i)} = Vz^{(i)} + \mu + \epsilon \tag{53}$$

- **Noise Distribution**

  The noise ϵ is drawn from a multivariate normal distribution:

$$\epsilon \sim \mathcal{N}(0, \Psi) \tag{54}$$

- **Interpretation**

  The observed variables x are noisy linear combinations of a few underlying latent variables z. Our goal in Factor Analysis is to infer this hidden structure. We only observe x, not z, but we believe the data is generated from this process.

Related terms:

- $\mu = 0$ (data is centered, for simplicity)
- $E[x] = \mu$
- $\text{Cov}(x) = \Sigma$
- $E[z_j] = 0, \text{Var}(z_j) = 1$ (Gaussian distribution of lower dimensional space)
- $\text{Cov}(z_i, z_j) = 0 \quad (i \neq j)$
- $\text{Var}(\epsilon_i) = \psi_i$
- $\text{Cov}(\epsilon_i, z_j) = 0 \quad \forall i, j \quad \leftarrow$ Constraints for simplicity

If data is given, we can compute:

$$\Sigma_x = E[(x - E[x])(x - E[x])^T] \tag{55}$$

$$E[x] = 0 \implies E[xx^T] \tag{56}$$

$$\Sigma_x = E[xx^T] \tag{57}$$

$$= E[(Vz + \epsilon)(Vz + \epsilon)^T] \tag{58}$$

$$= E[Vzz^TV^T + Vz\epsilon^T + \epsilon z^TV^T + \epsilon\epsilon^T] \tag{59}$$

V is not random, randomness is about ϵ, z :

$$= VE[zz^T]V^T + 2VE[z\epsilon^T] + E[\epsilon\epsilon^T] \tag{60}$$

Note that:

$$E[\epsilon\epsilon^T] = E[(\epsilon - E[\epsilon])(\epsilon - E[\epsilon])^T] = \Sigma_\epsilon = \Psi \tag{61}$$

$$E[zz^T] = E[(z - E[z])(z - E[z])^T] = I \tag{62}$$

Therefore, the main equation:

$$= VIV^T + z \cdot 0 + \Sigma_\epsilon \tag{63}$$

$$\Sigma_x = VV^T + \Sigma_\epsilon \tag{64}$$

How to obtain V (and also z): Ignore noise:

$$V = U\Sigma A^T \tag{65}$$

$$\Sigma_x = (U\Sigma A^T)(U\Sigma A^T)^T \tag{66}$$

$$= U\Sigma A^TA\Sigma^TU^T \tag{67}$$

$$\Sigma_x = U\Sigma^2U^T \implies \Sigma_x = V\Sigma^2V^T \tag{68}$$

Spectral decomposition:

$$BX = X\Lambda \tag{69}$$

Λ: diagonal matrix with eigenvalues

$$Bx_1 = \lambda_1 x_1 \tag{70}$$

$$Bx_2 = \lambda_2 x_2 \tag{71}$$

$$Bx_n = \lambda_n x_n \tag{72}$$

$$B = X\Lambda X^{-1} \quad (X^T = X^{-1}) \tag{73}$$

$$= X\Lambda X^T \tag{74}$$

How to obtain z:

$$z = Wx \tag{75}$$

(Compute expectations):

$$zx^T = Wxx^T \tag{76}$$

$$E[zx^T] = E[Wxx^T] \tag{77}$$

Substitute x^T to Vz + ϵ = x - μ:

$$E[z(Vz+\epsilon)^T] = E[zz^TV^T + z\epsilon^T] \tag{78}$$

$$= E[zz^T]V^T + 0 \tag{79}$$

$$= IV^T \tag{80}$$

$$E[zx^T] = V^T \tag{81}$$

Then:

$$E[wxx^T] = wE[xx^T] \tag{82}$$

$$= w\Sigma_x \tag{83}$$

So:

$$V^T = w\Sigma_x \tag{84}$$

$$w = V^T\Sigma_x^{-1} \tag{85}$$

w, Σ_x are known so:

$$z = wx \tag{86}$$

$$z = V^T\Sigma_x^{-1}x \tag{87}$$

4 Clustering

4.1 K-Means Clustering

K-Means Clustering is an unsupervised learning algorithm used to divide a dataset into K distinct groups (clusters) based on feature similarity. The goal is to minimize the variance within each cluster and maximize the difference between clusters.

Drawbacks:

- **Random Initialization of Cluster Centers-Stucking in local minima:** The initial placement of cluster centers is random. This can lead to different clustering results depending on the initial configuration, and some results might be suboptimal.
- **It does not work on non-spherical clusters**
- **Evaluation of Clustering Outcome:** Evaluating the quality of the resulting clusters can be challenging. The desired outcome is high similarity within clusters and low similarity between clusters. However, objectively measuring and determining if this has been achieved can be subjective.

K-Means clustering aims to partition N input data vectors $\{x_i\}_{i=1}^N$ into K clusters. The process can be summarized step by step as follows:

1. **Initialization (Implicit):** The algorithm typically starts by initializing K cluster centers, often denoted as $\{m_j\}_{j=1}^{K}$. These initial centers can be chosen randomly from the data points or using other heuristics.

2. **Assignment Step:** Each data vector x^t is assigned to the cluster whose center m_j is the closest, based on the squared Euclidean distance $\|x^t - m_j\|^2$. This assignment is mutually exclusive, meaning each data point belongs to only one cluster. The indicator variable b_i^t represents this assignment:
$$b_i^t = \begin{cases} 1 & \text{if } \|x^t - m_i\| = \min_j \|x^t - m_j\| \\ 0 & \text{otherwise} \end{cases}$$
Here, $b_i^t = 1$ if data point x^t is assigned to cluster i, and 0 otherwise.

3. **Update Step:** Once all data points are assigned to clusters, the center for each cluster m_i is re-estimated as the mean of all data points assigned to that cluster. This is done by minimizing the "total reconstruction error" objective function:
$$E(\{m_i\}_{i=1}^{K}|X) = \sum_t \sum_i b_i^t \|x^t - m_i\|^2$$
To minimize this error for each cluster i, the new cluster center m_i is calculated as:
$$m_i = \frac{\sum_t b_i^t x^t}{\sum_t b_i^t}$$
This is the average of all data points assigned to cluster i.

4. **Iteration:** Steps 2 and 3 are repeated iteratively. In each iteration, data points are reassigned to the nearest cluster center, and then the cluster centers are updated based on the new assignments. This process continues until a stopping criterion is met, such as the cluster assignments no longer changing significantly or a maximum number of iterations is reached.

Definition of Terms:

- N**:** The total number of input data vectors.
- x_i **or** x^t**:** The i-th or t-th input data vector. These are the data points to be clustered.
- K**:** The desired number of clusters. This is a parameter that needs to be specified beforehand.
- m_j **or** m_i**:** The center (centroid) of the j-th or i-th cluster. It represents the mean of the data points assigned to that cluster.
- b_i^t**:** A binary indicator variable that is 1 if the t-th data point x^t is assigned to the i-th cluster, and 0 otherwise.

- **$\|x^t - m_j\|^2$:** The squared Euclidean distance between the data point x^t and the cluster center m_j. This is used as a measure of similarity or proximity.
- **$E(\{m_i\}_{i=1}^K|X)$:** The full objective function, also referred to as the total reconstruction error or within-cluster sum of squares (WCSS). It measures the sum of the squared distances between each data point and its assigned cluster center. The goal of K-Means is to minimize this objective function.

Summarized Working Procedure:

- **Step 1: Choose the Number of Clusters**
  Select the number of clusters K. This can be set manually or determined using methods like the Elbow method.
- **Step 2: Initialize Centroids**
  Randomly choose K data points from the dataset as the initial centroids.
- **Step 3: Assign Points to the Nearest Centroid**
  For each data point x_i, assign it to the cluster with the closest centroid μ_k, using Euclidean distance:
  $$\text{Cluster}(x_i) = \arg\min_k \|x_i - \mu_k\|^2 \tag{88}$$
- **Step 4: Update Centroids**
  For each cluster, compute the new centroid by averaging all the points in that cluster:
  $$\mu_k = \frac{1}{|C_k|} \sum_{x_i \in C_k} x_i \tag{89}$$
- **Step 5: Repeat**
  Repeat steps 3 and 4 until the cluster assignments no longer change or a maximum number of iterations is reached.
- **Step 6: Output the Final Clusters**
  The algorithm returns the K clusters and their centroids.

4.2 Hierarchical Clustering

Hierarchical Clustering is an unsupervised learning method that builds a hierarchy of clusters. It does not require you to specify the number of clusters beforehand (unlike K-Means).

There are two main types:

1. **Agglomerative (bottom-up)** – Each data point starts as its own cluster. Pairs of clusters are merged as you move up the hierarchy.
2. **Divisive (top-down)** – All data points start in one cluster, and splits are performed recursively.

The result is often shown in a dendrogram, a tree-like diagram showing the merge/split operations. Use Hierarchical Clustering when:

- We do not know the number of clusters ahead of time.
- We want to understand the nested (hierarchical) structure in the data.
- We want to visualize how data clusters at different levels of similarity using a dendrogram.

4.2.1 Agglomerative Clustering

Agglomerative clustering is a type of hierarchical clustering that builds the hierarchy bottom-up. It starts with each data point as its own cluster and then merges the closest pairs of clusters step by step until all points are grouped into a single cluster (or a desired number of clusters). Agglomerative Clustering builds a hierarchical tree (dendrogram) where:

- Leaves = individual data points
- Root = final cluster (all data)
- You can choose where to cut the tree to get clusters at different levels of granularity.

Working Procedure:

1. **Initialization:** Start with N clusters, where each data point is its own cluster:
$$G_1 = \{x_1\}, \quad G_2 = \{x_2\}, \quad \dots, \quad G_N = \{x_N\}$$

2. **Compute Distance Between Clusters:** Use a linkage criterion to determine the distance between clusters. Common linkage methods include:

   - **Single Link (Minimum Distance):**
   $$d(G_i, G_j) = \min_{x^r \in G_i,\, x^s \in G_j} d(x^r, x^s) \tag{90}$$
   This method considers the closest pair of points between the two clusters.
   - **Complete Link (Maximum Distance):**
   $$d(G_i, G_j) = \max_{x^r \in G_i,\, x^s \in G_j} d(x^r, x^s) \tag{91}$$
   This method considers the most dissimilar members of the clusters. It is sensitive to outliers.

- **Average Link (Group-Average):**

$$d(G_i, G_j) = \frac{1}{|G_i||G_j|} \sum_{x^r \in G_i} \sum_{x^s \in G_j} d(x^r, x^s) \tag{92}$$

  This method computes the average of all pairwise distances between cluster members.

3. **Merge Closest Clusters:** Find the pair of clusters G_i and G_j with the smallest distance $d(G_i, G_j)$, and merge them:

$$G_{\text{new}} = G_i \cup G_j \tag{93}$$

4. **Update Distance Matrix:** After merging, update the distance matrix to reflect the distances between the new cluster and the remaining clusters using the selected linkage method.

5. **Repeat:** Repeat Steps 2–4 until:
   - All data points are merged into a single cluster.
   - Or, the desired number of clusters is reached by cutting the dendrogram at a chosen level.

4.2.2 Divisive Clustering

Divisive Clustering (Top-Down Hierarchical Clustering) is the opposite of Agglomerative (bottom-up) clustering. Instead of starting with individual points as clusters and merging them, divisive clustering begins with one single cluster containing all data points, and recursively splits it into smaller clusters.

- Top-down approach to hierarchical clustering.
- Start with all data points in one cluster.
- At each step, split the cluster that is least homogeneous (most dissimilar).
- Continue until each data point is in its own cluster, or until a stopping criterion is met.
- Often implemented using techniques like bisecting k-means (split clusters using k-means with $k = 2$).

Working Procedure:

1. **Start:** Begin with all data points in one cluster:

$$G = \{x_1, x_2, \ldots, x_n\}$$

2. **Split:** Identify the cluster to split. This is usually the cluster with the highest heterogeneity.

3. **Partition:** Split the selected cluster into two subclusters using a clustering algorithm like k-means (with $k = 2$).
4. **Update Clusters:** Replace the selected cluster with the two new subclusters.
5. **Repeat:** Continue steps 2–4 until:
    - Each data point is in its own cluster, or
    - A desired number of clusters is reached, or
    - A threshold of dissimilarity is satisfied.

5 DBSCAN – Density-Based Spatial Clustering of Applications with Noise

DBSCAN stands for Density-Based Spatial Clustering of Applications with Noise. It is a clustering algorithm that groups together points that are close to each other based on density, and marks points in low-density regions as outliers (noise). It is **efficient for large datasets** and can find clusters of **arbitrary shape**, unlike K-Means which assumes spherical clusters.
Our Aim with DBSCAN:

- Group data points into dense regions (clusters).
- Identify outliers that don't belong to any cluster.
- Avoid assuming the number of clusters beforehand (unlike K-Means).

When is DBSCAN used?

- When the data has clusters of arbitrary shape (not just spherical like in K-Means).
- When the number of clusters is not known in advance.
- In datasets with noise or outliers.

Parameters

- ϵ **(Epsilon):** The maximum distance between two data points for them to be considered within the same neighborhood.
- **MinPts (Minimum Points):** The minimum number of data points required within the ϵ-neighborhood of a point to consider that point a core point, thus forming a dense region.

Key Definitions:

- Define the **radius** r (also denoted as ε) of a neighborhood with respect to some point.

- A point p is a **core point** if at least `MinPts` points are within distance r of it (including p).
- A point q is **directly reachable** from a core point p if q lies within distance r of p.
- A point q is **reachable** from p if there exists a sequence of points $p_1, p_2, \dots, p_n$ such that:

$$p_1 = p, \quad p_n = q, \quad \text{and } p_{i+1} \text{ is directly reachable from } p_i \text{ for all } i.$$

- If p is a core point, then it forms a cluster together with all points (core or non-core) that are reachable from it.
- Points that are not reachable from any core point are classified as **outliers** or **noise points**.

Working Procedure:

1. **Set Parameters:** Choose values for radius r (or ε) and `MinPts`, the minimum number of points needed to form a dense region.
2. **Visit Points:** Start with an arbitrary unvisited data point p.
3. **Neighborhood Query:** Retrieve all points within distance r of p (i.e., compute the ε-neighborhood of p).
4. **Core Point Check:**
   - If p has at least `MinPts` points in its neighborhood, mark p as a **core point** and start forming a new cluster.
   - Otherwise, mark p as **noise** for now (it may later be included in a cluster if found to be in a core point's neighborhood).
5. **Expand Cluster:**
   - For each point q in p's neighborhood:
     - If q is unvisited, mark it as visited.
     - If q has enough neighbors to be a core point, add its neighbors to p's neighborhood.
     - If q is not yet assigned to a cluster, add it to the current cluster.
6. **Repeat:** Repeat the process for all unvisited points until every point is marked as part of a cluster or as noise.

6 GMM – Gaussian Mixture Model

A Gaussian Mixture Model is a probabilistic model that assumes data is generated from a mixture of several Gaussian (normal) distributions, each with its own mean and variance. Instead of assigning each point to a specific cluster (as in K-Means), GMM assigns probabilities of belonging to each cluster.

When is GMM Used?

- When clusters overlap and a soft assignment (probabilistic) is preferable.
- When data is believed to be generated from multiple Gaussian distributions.
- In applications like speech recognition, image segmentation, density estimation, and anomaly detection.

Working Procedure:

1. **Initialize:**
   - Choose the number of components K.
   - Initialize the parameters for each Gaussian:

   $$\text{Means } \mu_k, \quad \text{Covariances } \Sigma_k, \quad \text{Mixing Coefficients } \pi_k, \quad \text{for } k = 1, \ldots, K.$$

2. **Expectation Step (E-Step):**
   - For each data point x_i, compute the responsibility γ_{ik}, the probability that x_i belongs to cluster k:

   $$\gamma_{ik} = \frac{\pi_k \cdot \mathcal{N}(x_i \mid \mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \cdot \mathcal{N}(x_i \mid \mu_j, \Sigma_j)} \tag{94}$$

3. **Maximization Step (M-Step):**
   - Update parameters using the responsibilities:

   $$N_k = \sum_{i=1}^{N} \gamma_{ik} \tag{95}$$

   $$\mu_k = \frac{1}{N_k} \sum_{i=1}^{N} \gamma_{ik} x_i \tag{96}$$

   $$\Sigma_k = \frac{1}{N_k} \sum_{i=1}^{N} \gamma_{ik} (x_i - \mu_k)(x_i - \mu_k)^T \tag{97}$$

   $$\pi_k = \frac{N_k}{N} \tag{98}$$

4. **Convergence Check:**
    - Repeat E-step and M-step until the change in log-likelihood is below a threshold.
5. **Final Assignment:**
    - Each data point is assigned to the Gaussian with the highest responsibility γ_{ik} or kept as a soft assignment for probabilistic interpretation.

6.1 Choosing Between K-Means and GMM

- Use **K-Means** if your data is clearly separated and roughly spherical.
- Use **GMM** if you want a soft, probabilistic model and your clusters may overlap or have different shapes.

Feature	**K-Means**	**Gaussian Mixture Models (GMM)**
Clustering Type	Hard clustering (each point belongs to one cluster)	Soft clustering (each point has a probability of belonging to each cluster)
Cluster Shape Assumption	Spherical clusters (equal variance in all directions)	Elliptical clusters (different covariance structures)
Distance Metric	Euclidean distance	Probability density from Gaussian distributions
Model Type	Non-probabilistic	Probabilistic (mixture of Gaussians)
Assignment	Assigns each point to the nearest cluster center	Assigns each point to all clusters with probabilities
Flexibility	Less flexible for irregular or overlapping shapes	More flexible, can handle overlapping and anisotropic clusters
Initialization	Centroids (means)	Means, covariances, and mixing coefficients
Algorithm Used	Lloyd's algorithm	Expectation-Maximization (EM) algorithm
Output	Labels per point	Probabilities per point for each cluster

Table 1: Comparison of K-Means and Gaussian Mixture Models (GMMs)

6.2 Gaussian Mixture Density

A Gaussian Mixture Model (GMM) represents a data distribution as a weighted sum of K Gaussian component densities. In this example, the GMM is composed of three Gaussian distributions, $p_1(x)$, $p_2(x)$, and $p_3(x)$, each defined by its mean vector (μ) and covariance matrix (Σ):

- Component 1: $p_1(x) \sim \mathcal{N}\left(\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}\right)$
- Component 2: $p_2(x) \sim \mathcal{N}\left(\begin{bmatrix}6\\6\end{bmatrix}, \begin{bmatrix}4 & 0\\0 & 4\end{bmatrix}\right)$

- Component 3: $p_3(x) \sim \mathcal{N}\left(\begin{bmatrix} 7 \\ -7 \end{bmatrix}, \begin{bmatrix} 6 & 0 \\ 0 & 6 \end{bmatrix}\right)$

Example: Gaussian Mixture Density - Part 2: The Mixture Density

The overall probability density function $p(x)$ of the GMM is a linear combination of these individual Gaussian components, weighted by mixing coefficients (π_k):

$$p(x) = \sum_{k=1}^{K} \pi_k p_k(x) \tag{99}$$

In this specific example, with three components and their respective mixing weights:

$$p(x) = 0.2p_1(x) + 0.3p_2(x) + 0.5p_3(x) \tag{100}$$

The mixing coefficients (0.2, 0.3, 0.5) are non-negative and sum to 1, representing the probability of a data point originating from each component. This formulation allows GMMs to model complex data distributions by combining simpler Gaussian components.

References

[1] Ethem Alpaydın, *Introduction to Machine Learning, 2nd Edition, MIT Press* 2010.