Gaussian Mixture Model

Last Updated : 27 Feb, 2025

Clustering is a key technique in unsupervised learning, used to group similar data points together. While traditional methods like **K-Means** and **Hierarchical Clustering** are widely used, they assume that clusters are well-separated and have rigid shapes. This can be limiting in real-world scenarios where clusters can be more complex.

To overcome these limitations, **Gaussian Mixture Models (GMM)** offer a more flexible approach. Unlike K-Means, which assigns each point to a single cluster, GMM uses a probabilistic approach to cluster the data, allowing clusters to have more varied shapes and soft boundaries. Let's dive into what GMM is and how it works

Gaussian Mixture Model vs K-Means

- A Gaussian mixture model is a **soft clustering technique** used in unsupervised learning to determine the probability that a given data point belongs to a cluster. It's composed of several Gaussians, each identified by $k \in \{1,\ldots, K\}$, where K is the number of clusters in a data set and is comprised of the following parameters.

- K-means is a clustering algorithm that assigns each data point to one cluster based on the closest centroid. It's a **hard clustering** method, meaning each point belongs to only one cluster with no uncertainty.

On the other hand, **Gaussian Mixture Models (GMM)** use **soft clustering**, where data points can belong to multiple clusters with a certain probability. This provides a more flexible and nuanced way to handle clusters, especially when points are close to multiple centroids.

How Gaussian Mixture Models Work?

Now that we understand the key differences between K-Means and GMM, let's dive deeper into how GMM actually works. Unlike K-Means, where the clustering process relies solely on the centroid and assigns each data point to one cluster, GMM uses a probabilistic approach. Here's how GMM performs clustering:

1. **Multiple Gaussians (Clusters):** Each cluster is represented by a Gaussian distribution, and the data points are assigned probabilities of belonging to different clusters based on their distance from each Gaussian.
2. **Parameters of a Gaussian:** The core of GMM is made up of three main parameters for each Gaussian:
   - **Mean (μ):** The center of the Gaussian distribution.
   - **Covariance (Σ):** Describes the spread or shape of the cluster.
   - **Mixing Probability (π):** Determines how dominant or likely each cluster is in the data.

The Gaussian mixture model assigns a probability to each data point xnx_nxn of belonging to a cluster. The probability of data point coming from Gaussian cluster k is expressed as

$P(z_n = k \mid x_n) = \frac{\pi_k \cdot \mathcal{N}(x_n|\mu_k,\Sigma_k)}{\sum_{k=1}^{K} \pi_k \cdot \mathcal{N}(x_n|\mu_k,\Sigma_k)}$

- where,
- z_n=k is a latent variable indicating which Gaussian the point belongs to.
- π_k is the mixing probability of the k-th Gaussian.
- $\mathcal{N}(x_n \mid \mu_k, \Sigma_k)$is the Gaussian distribution with mean μk\mu_kμk and covariance Σk

Next, we need to calculate the overall likelihood of observing a data point xnx_nxn under all Gaussians. This is achieved by summing over all possible clusters (Gaussians) for each point:

$P(x_n) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x_n \mid \mu_k, \Sigma_k)$

- Where:
- $P(x_n)$ is the overall likelihood of observing the data point x_n

Open In App

- The sum accounts for all possible Gaussians k.

The Expectation-Maximization (EM) Algorithm

To fit a Gaussian Mixture Model to the data, we use the **Expectation-Maximization (EM)** algorithm, which is an iterative method that optimizes the parameters of the Gaussian distributions (mean, covariance, and mixing coefficients). It works in two main steps:

1. **Expectation Step (E-step)**:
   - In this step, the algorithm calculates the probability that each data point belongs to each cluster based on the current parameter estimates (mean, covariance, mixing coefficients).

2. **Maximization Step (M-step)**:
   - After estimating the probabilities, the algorithm updates the parameters (mean, covariance, and mixing coefficients) to better fit the data.

These two steps are repeated until the model converges, meaning the parameters no longer change significantly between iterations.

How GMM Works

Here's a simple breakdown of the process:

1. **Initialization**: Start with initial guesses for the means, covariances, and mixing coefficients of each Gaussian distribution.
2. **E-step**: For each data point, calculate the probability of it belonging to each Gaussian distribution (cluster).
3. **M-step**: Update the parameters (means, covariances, mixing coefficients) using the probabilities calculated in the E-step.
4. **Repeat**: Continue alternating between the E-step and M-step until the log-likelihood of the data (a measure of how well the model fits the data) converges.

Formula:

$L(\mu_k, \Sigma_k, \pi_k) = \prod_{n=1}^{N} \sum_{k=1}^{K} \pi_k \mathcal{N}(x_n \mid \mu_k, \Sigma_k)$

The E-step computes the probabilities that each data point belongs to each Gaussian, while the M-step updates the parameters μk, Σk , and πk based on these probabilities.

Implementation of GMM in python

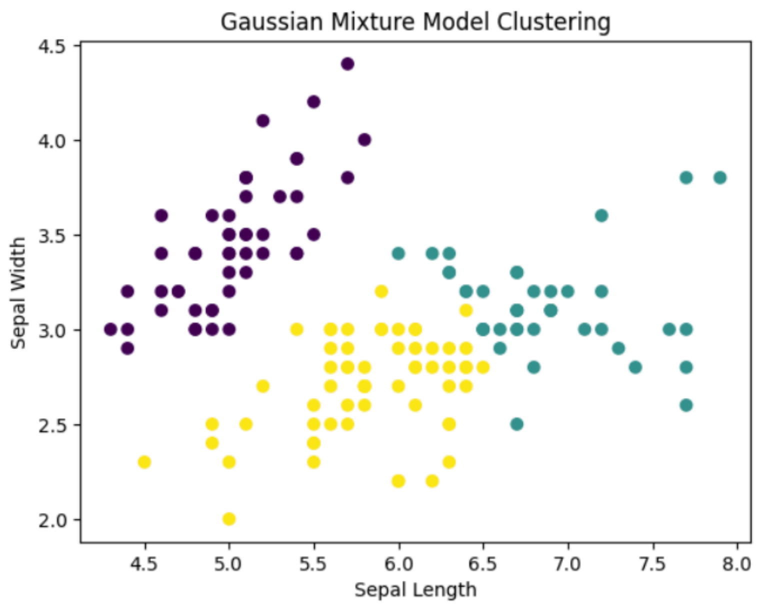
So far, we have discussed about the working of GMM. Let's take a simple example using the **Iris dataset** and fit a Gaussian Mixture Model with 3 clusters (since we know there are 3 species of Iris).

1. **Dataset**: We use only two features: sepal length and sepal width.
2. **Fitting the Model**: We fit the data as a mixture of 3 Gaussians.
3. **Result**: The model assigns each data point to a cluster based on the probabilities. After convergence, the GMM model will have updated the parameters to best fit the data.

```python
from sklearn.mixture import GaussianMixture
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris

# Load iris dataset
iris = load_iris()
X = iris.data[:, :2]  # using only sepal length and sepal width

# Fit GMM with 3 components (clusters)
gmm = GaussianMixture(n_components=3)
gmm.fit(X)

# Predict the labels (cluster assignments)
labels = gmm.predict(X)

# Plot the results
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.title('Gaussian Mixture Model Clustering')
plt.show()
```

Open In App

Output:



GMM Clustering

This scatter plot shows the result of clustering data using a **Gaussian Mixture Model (GMM)**. The data points represent measurements of flower species based on **sepal length** and **sepal width**. The points are divided into three clusters, represented by different colors (yellow, purple, and teal), which indicate that GMM has identified three distinct groups within the data.

Advantages of Gaussian Mixture Models (GMM)

1. **Flexible Cluster Shapes**: Unlike K-Means, which assumes spherical clusters, GMM can model clusters with arbitrary shapes.
2. **Soft Assignment**: GMM assigns a probability for each data point to belong to each cluster, while K-Means assigns each point to exactly one cluster.
3. **Handles Overlapping Data**: GMM performs well when clusters overlap or have varying densities. Since it uses probability distributions, it can assign a point to multiple clusters with different probabilities

Limitations of GMM

Open In App

1. **Computational Complexity**: GMM tends to be computationally expensive, particularly with large datasets, as it requires iterative processes like the Expectation-Maximization (EM) algorithm to estimate the parameters.

- **Choosing the Number of Clusters**: Like other clustering methods, GMM requires you to specify the number of clusters beforehand. However, methods like the **Bayesian Information Criterion (BIC)** and **Akaike Information Criterion (AIC)** can help in selecting the optimal number of clusters based on the data

Conclusion

Gaussian Mixture Models provide a flexible and powerful tool for clustering when data has more complex structures. By treating clustering as a probabilistic problem, GMM allows for clusters with varied shapes and soft boundaries, unlike traditional methods like K-Means. The **Expectation-Maximization** algorithm is used to iteratively refine the model’s parameters, making GMM a great choice for many real-world clustering problems.

When should I use Gaussian mixture model?

> *Gaussian Mixture Models (GMM) are used for clustering and density estimation in unsupervised learning, particularly when data comes from multiple normal distributions. It’s applied in areas like image segmentation, market segmentation, and anomaly detection.*

Why is GMM better than K means ?

> *K-means assigns each data point to one cluster, while GMM assigns points to multiple clusters with probabilities. GMM can handle elliptical shapes, unlike K-means, which only detects spherical clusters. This makes GMM more flexible for complex data*

How does GMM handle overlapping clusters?

ML | Expectation-Maximization Algorithm

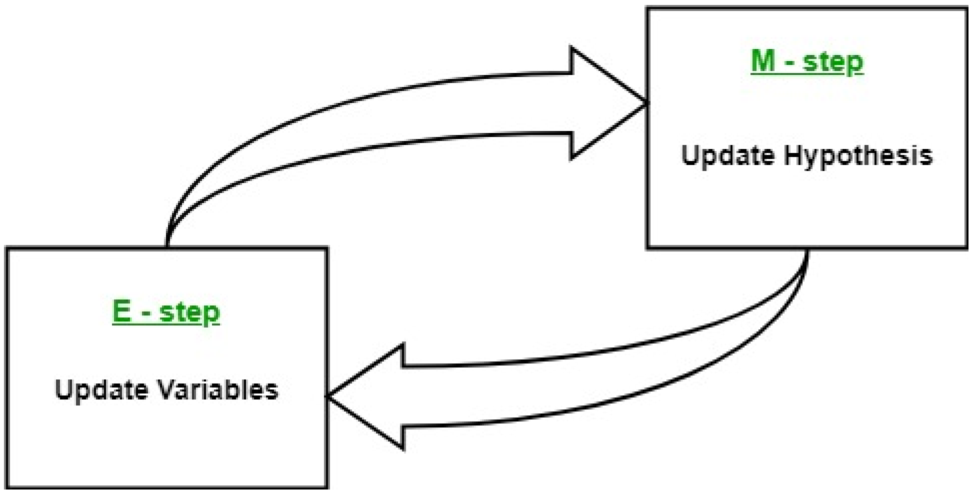
Last Updated : 04 Feb, 2025

The **Expectation-Maximization (EM) algorithm** is an **iterative method** used in unsupervised machine learning to estimate unknown parameters in statistical models. It helps find the best values for unknown parameters, especially when some data is missing or hidden.

It works in two steps:

1. **E-step (Expectation Step):** Estimates missing or hidden values using current parameter estimates.
2. **M-step (Maximization Step):** Updates model parameters to maximize the likelihood based on the estimated values from the E-step.

This process repeats until the model reaches a stable solution, improving accuracy with each iteration. EM is widely used in clustering (e.g., **Gaussian Mixture Models**) and handling missing data



Expectation-Maximization in EM Algorithm

By iteratively repeating these steps, the EM algorithm seeks to maximize the likelihood of the observed data. It is commonly used for clustering, where latent variables a inferred and has applications in various fields, including machine learning, computer vision, and natural language processing.

Key Terms in Expectation-Maximization (EM) Algorithm

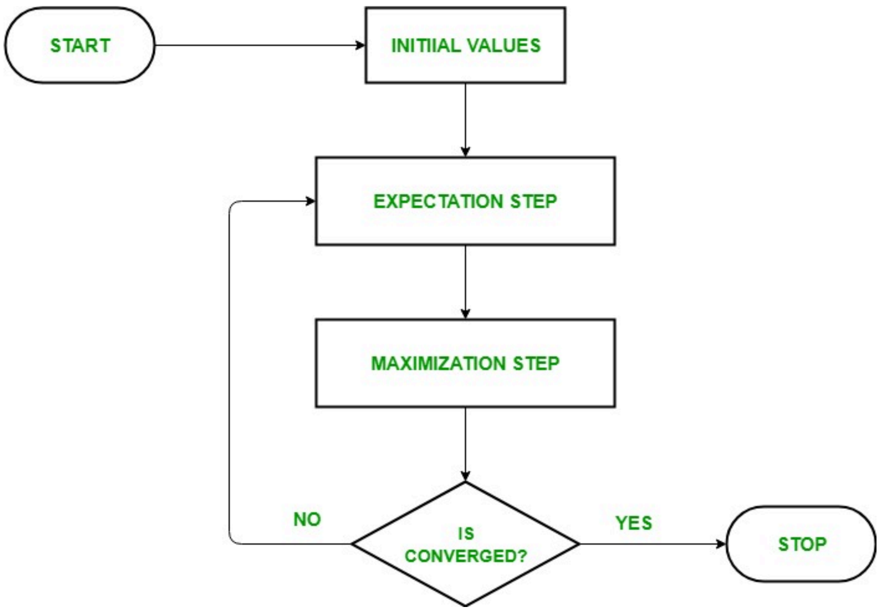
Lets understand about some of the most commonly used key terms in the Expectation-Maximization (EM) Algorithm below:

- **Latent Variables:** These are hidden or unmeasured variables that affect what we can observe in the data. We can't directly see them, but we can make educated guesses about them based on the data we can see.
- **Likelihood:** This refers to the probability of seeing the data we have, based on certain assumptions or parameters. The EM algorithm tries to find the best parameters that make the data most likely.
- **Log-Likelihood:** This is just the natural log of the likelihood function. It's used to make calculations easier and measure how well the model fits the data. The EM algorithm tries to maximize the log-likelihood to improve the model fit.
- **Maximum Likelihood Estimation (MLE):** This is a technique for estimating the parameters of a model. It does this by finding the parameter values that make the observed data most likely (maximizing the likelihood).
- **Posterior Probability:** In Bayesian methods, this is the probability of the parameters, given both prior knowledge and the observed data. In EM, it helps estimate the "best" parameters when there's uncertainty about the data.
- **Expectation (E) Step:** In this step, the algorithm estimates the missing or hidden information (latent variables) based on the observed data and current parameters. It calculates probabilities for the hidden values given what we can see.
- **Maximization (M) Step:** This step updates the parameters by finding the values that maximize the likelihood, based on the estimates from the E-step. It often involves running optimization methods to get the best parameters.

- **Convergence:** Convergence happens when the algorithm has reached a stable point. This is checked by seeing if the changes in the model's parameters or the log-likelihood are small enough to stop the process.

How Expectation-Maximization (EM) Algorithm Works:

So far, we've discussed the key terms in the **EM algorithm**. Now, let's dive into how the **EM algorithm works**. Here's a step-by-step breakdown of the process:



EM Algorithm Flowchart

1. **Initialization:**
   The algorithm starts with initial parameter values and assumes the observed data comes from a specific model.

- **E-Step (Expectation Step):**
  - Estimate the missing or hidden data based on the current parameters.
  - Calculate the posterior probability (responsibility) of each latent variable given the observed data.
  - Compute the log-likelihood of the observed data using the current parameter estimates.
- **M-Step (Maximization Step):**
  - Update the model parameters by maximizing the log-likelihood computed in the E-step.

Open In App

    - This involves solving an optimization problem to find parameter values that improve the model fit.
- **Convergence:**
    - Check if the model parameters are stable (converging).
    - If the changes in log-likelihood or parameters are below a set threshold, stop. If not, repeat the E-step and M-step until convergence is reached

Expectation-Maximization Algorithm Step by Step Implementation

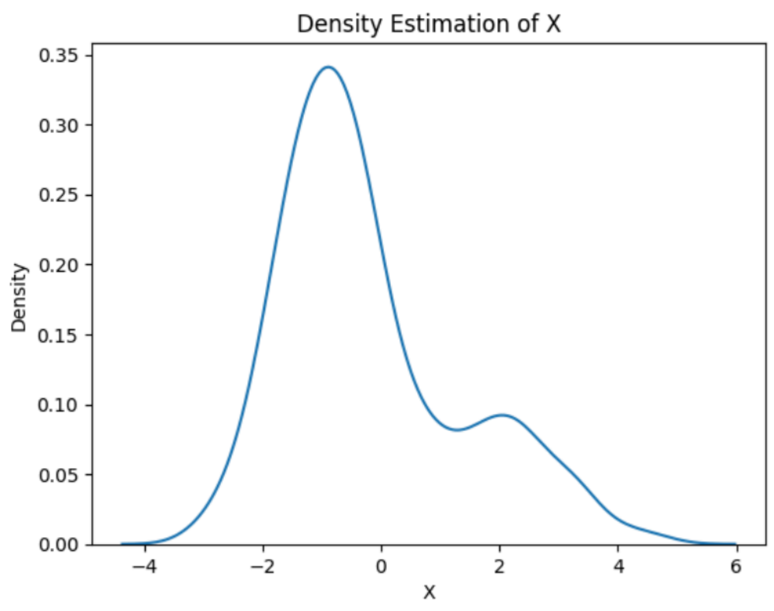
Step 01 : Import the necessary libraries

```
import numpy as np
import seaborn as sns
from scipy.stats import norm
from scipy.stats import gaussian_kde
import matplotlib.pyplot as plt
```

Step 02 : Generate a dataset with two Gaussian components

```
mu1, sigma1 = 2, 1
mu2, sigma2 = -1, 0.8
X1 = np.random.normal(mu1, sigma1, size=200)
X2 = np.random.normal(mu2, sigma2, size=600)
X = np.concatenate([X1, X2])

sns.kdeplot(X)
plt.xlabel('X')
plt.ylabel('Density')
plt.title('Density Estimation of X')
plt.show()
```

Output:



Density Plot

Step 03: Initialize parameters

```
mu1_hat, sigma1_hat = np.mean(X1), np.std(X1)
mu2_hat, sigma2_hat = np.mean(X2), np.std(X2)
pi1_hat, pi2_hat = len(X1) / len(X), len(X2) / len(X)
```

Step 04: Perform EM algorithm

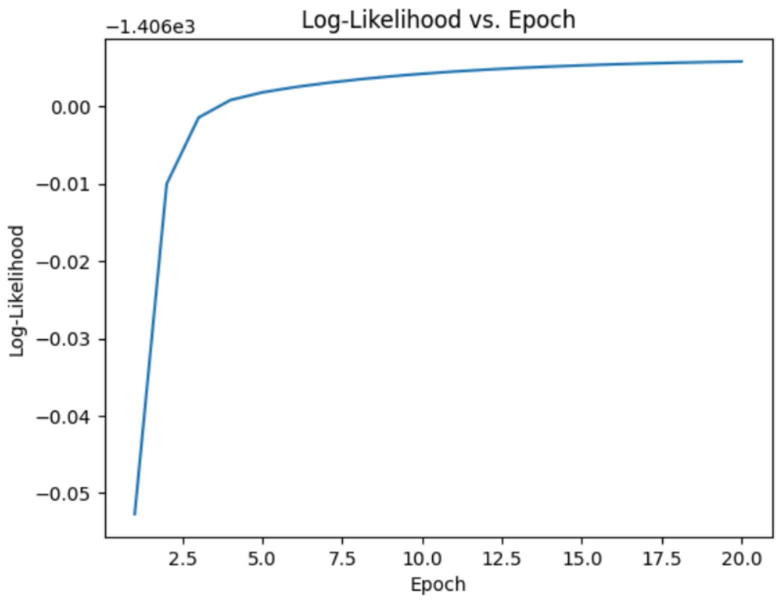
- Iterates for the specified number of epochs (20 in this case).
- In each epoch, the E-step calculates the responsibilities (gamma values) by evaluating the Gaussian probability densities for each component and weighting them by the corresponding proportior
- The M-step updates the parameters by computing the weighted mean and standard deviation for each component

```
num_epochs = 20
log_likelihoods = []

for epoch in range(num_epochs):
    # E-step: Compute responsibilities
    gamma1 = pi1_hat * norm.pdf(X, mu1_hat, sigma1_hat)
    gamma2 = pi2_hat * norm.pdf(X, mu2_hat, sigma2_hat)
    total = gamma1 + gamma2
    gamma1 /= total
    gamma2 /= total

    # M-step: Update parameters
    mu1_hat = np.sum(gamma1
```

Open In App

```
    mu2_hat = np.sum(gamma2 * X) / np.sum(gamma2)
    sigma1_hat = np.sqrt(np.sum(gamma1 * (X - mu1_hat)**2) /
np.sum(gamma1))
    sigma2_hat = np.sqrt(np.sum(gamma2 * (X - mu2_hat)**2) /
np.sum(gamma2))
    pi1_hat = np.mean(gamma1)
    pi2_hat = np.mean(gamma2)

    # Compute log-likelihood
    log_likelihood = np.sum(np.log(pi1_hat * norm.pdf(X, mu1_hat,
sigma1_hat)
                                   + pi2_hat * norm.pdf(X, mu2_hat,
sigma2_hat)))
    log_likelihoods.append(log_likelihood)


plt.plot(range(1, num_epochs+1), log_likelihoods)
plt.xlabel('Epoch')
plt.ylabel('Log-Likelihood')
plt.title('Log-Likelihood vs. Epoch')
plt.show()
```

Output:



Epoch vs Log-likelihood

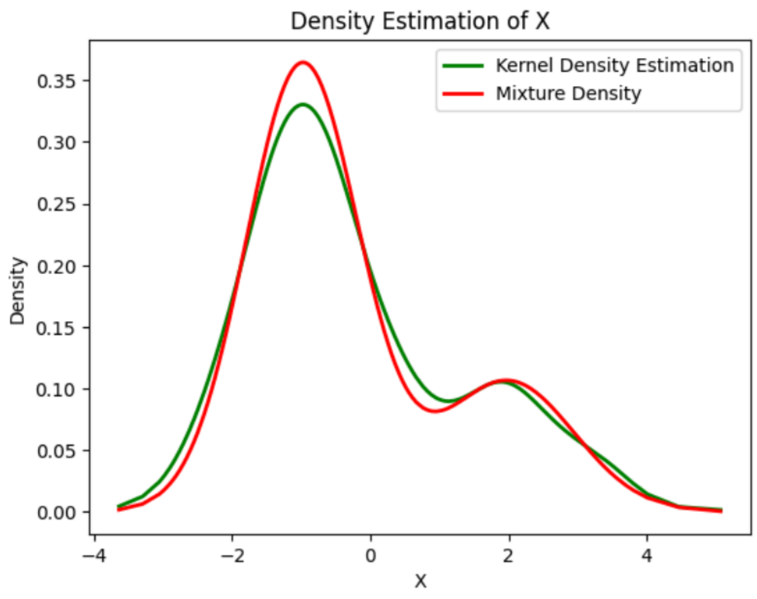
Step 05: Plot the final estimated density

```
X_sorted = np.sort(X)
density_estimation = pi1_hat*norm.pdf(X_sorted,
                                        mu1_hat,
                                        sigma1_hat) + pi2_hat *
norm.pdf(X_sorted,

mu2_hat,

sigma2_hat)
```

Open In App

```
plt.plot(X_sorted, gaussian_kde(X_sorted)(X_sorted), color='green',
linewidth=2)
plt.plot(X_sorted, density_estimation, color='red', linewidth=2)
plt.xlabel('X')
plt.ylabel('Density')
plt.title('Density Estimation of X')
plt.legend(['Kernel Density Estimation','Mixture Density'])
plt.show()
```

Output:



Estimated density

Advantages of EM algorithm

- **Always improves results** – With each step, the algorithm improves the likelihood (chances) of finding a good solution.
- **Simple to implement** – The two steps (E-step and M-step) are often easy to code for many problems.
- **Quick math solutions** – In many cases, the M-step has a direct mathematical solution (closed-form), making it efficient

Disadvantages of EM algorithm

- **Takes time to finish** – It converges slowly, meaning it may take many iterations to reach the best solution.
- **Gets stuck in local best** – Instead of finding the absolute best solution, it might settle for a "good enough" one.
- **Needs extra probabilities** – Unlike some optimization methods that only need forward probability, EM requires both forward and backward probabilities, ma[illegible]ly more complex.

Open In App

Bayesian Estimation in Pattern Recognition

Overview:

Bayesian estimation is a probabilistic approach in pattern recognition that incorporates prior knowledge (prior probabilities) with observed data to make inferences. It helps estimate parameters (like mean and variance) of probability distributions for classification tasks.

Bayesian Parameter Estimation:

1. Basic Concept:

Bayesian estimation updates the prior distribution of a parameter using the observed data to form the posterior distribution:

P(theta | X) = (P(X | theta) * P(theta)) / P(X)

where:

theta: parameter(s) to estimate

X: observed data

P(theta): prior probability

P(X | theta): likelihood

P(theta | X): posterior probability

P(X): evidence or marginal likelihood

2. Bayesian vs. Maximum Likelihood Estimation (MLE):

- MLE estimates parameters by maximizing P(X | theta), ignoring prior.

- Bayesian Estimation incorporates prior knowledge via P(theta).

3. Posterior Mean as Estimator:

If loss is squared-error, the optimal Bayesian estimate is:

theta_Bayes = E[theta | X] = integral theta * P(theta | X) dtheta

4. Example: Gaussian Distribution

Assume data X = {x1, x2, ..., xn} drawn from a normal distribution with unknown mean mu and known variance sigma^2.

If prior mu ~ N(mu0, sigma0^2), then the posterior is also normal:

mu | X ~ N(mun, sigman^2)

where:

sigman^2 = (n/sigma^2 + 1/sigma0^2)^(-1)

mun = sigman^2 * (n * x / sigma^2 + mu0 / sigma0^2)

5. MAP Estimation (Maximum A Posteriori):

Finds the mode of the posterior:

theta_MAP = argmax P(theta | X) = argmax P(X | theta) * P(theta)

- Similar to MLE, but includes prior.

6. Applications in Pattern Recognition:

- Estimating class-conditional densities in classifiers.
- Updating model parameters as more data becomes available.
- Reducing overfitting through regularization via priors.

Advantages:

- Incorporates prior knowledge into the model.
- Provides a full posterior distribution over parameters.
- Helps prevent overfitting (especially in small datasets).
- Naturally accommodates model updating with new data.

Disadvantages:

- Computationally intensive for complex models or large data.
- Requires selection of a prior, which may be subjective.
- May be intractable without approximations or numerical methods.