Unknown Title

Ashkan Beheshti : : 3/11/2023

Ashkan Beheshti

Mar 11

·

16 min read

·

Unsupervised Learning: Clustering using Gaussian Mixture Model (GMM)

Clustering is a fundamental task in unsupervised machine learning that involves grouping data points based on their similarity. Gaussian mixture model (GMM) is a popular clustering method that models the data as a mixture of Gaussian distributions. GMM is a probabilistic clustering method that assigns a probability distribution to each cluster, allowing for more flexible and accurate clustering than other methods. GMM can model complex cluster shapes and can handle overlapping clusters. GMM is also useful for density estimation, which involves estimating the probability distribution of a set of data points.

In this tutorial, we will cover the theoretical background of GMM and its implementation in Python using the scikit-learn library. We will also discuss examples and applications of GMM in image segmentation and customer segmentation. Finally, we will examine the limitations and extensions of GMM, including its sensitivity to noise and outliers, and explore alternative clustering methods.

By the end of this tutorial, you should have a good understanding of how GMM works, how to implement it in Python, and how to use it for clustering and density estimation. You should also be familiar with the advantages and limitations of GMM and how to choose the appropriate clustering method for your data and goals. This tutorial is suitable for anyone with a basic understanding of Python and machine learning.

🦊 If you want to learn more about clustering in unsupervised learning, you may be interested in reading my other post "Clustering Methods 101: An Introduction to Unsupervised Learning Techniques." This tutorial provides a comprehensive overview of different clustering methods, including hierarchical clustering, density-based clustering, and model-based clustering, and discusses their advantages and disadvantages. It also covers topics such as cluster evaluation and selection, and provides examples of clustering applications in different fields.

Introduction

Clustering is a widely used unsupervised learning technique that aims to group data points into distinct clusters based on their similarity. Clustering can be used for various applications such as image segmentation, customer segmentation, anomaly detection, and more. One of the popular clustering algorithms is the Gaussian Mixture Model (GMM), which is based on the assumption that each cluster is generated from a mixture of Gaussian distributions.

GMM is a probabilistic model that can capture complex patterns in data and assign each data point to a corresponding cluster with a probability score. GMM can also handle data that is not linearly separable or has overlapping clusters, which makes it a versatile and powerful clustering algorithm.

In this tutorial, we will provide an in-depth overview of clustering with GMM, including the theoretical background, algorithm, implementation in Python, and examples of its applications. We will also explain the math behind GMM, including the probability density function, multivariate Gaussian distribution, mixture models, Expectation-Maximization (EM) algorithm, and Maximum Likelihood Estimation (MLE).

By the end of this tutorial, you should have a solid understanding of how GMM works and how to implement it in Python to cluster your own datasets. So, let's get started!

🔍 Theoretical Background

📌 Probability Density Function (PDF)

In probability theory, a probability density function (PDF) is a function that describes the probability of a continuous random variable taking on a specific value or range of values. The PDF of a random variable X, denoted as f(x), must satisfy the following properties:

1. f(x) is non-negative for all values of x.

2. The integral of $f(x)$ over the entire range of X is equal to 1, i.e., $\int f(x)dx = 1$.

The PDF can be used to compute the probability that a random variable X falls within a certain range (a,b) by integrating $f(x)$ over that range:

$Pr(a \le X \le b) = \int[a,b]\ f(x)dx$

📌 Multivariate Gaussian Distribution

The multivariate Gaussian distribution, also known as the multivariate normal distribution, is a probability distribution over vectors of real-valued variables. The PDF of a multivariate Gaussian distribution is defined as:

$f(x \mid \mu, \Sigma) = (1 / ((2\pi)^\wedge(d/2)\ |\Sigma|^{0}.5)) * \exp(-(x-\mu)^{T}\Sigma^\wedge(-1)(x-\mu) / 2)$

where x is a d-dimensional vector, μ is the mean vector, Σ is the covariance matrix, and $|\Sigma|$ is the determinant of the covariance matrix.

The multivariate Gaussian distribution is characterized by its mean and covariance matrix, which describe the center and spread of the distribution, respectively. The mean vector μ represents the center of the distribution, while the covariance matrix Σ describes the spread of the distribution in each direction.

📌 Mixture Models

A mixture model is a probabilistic model that represents a distribution as a mixture of simpler component distributions. In the context of clustering, a mixture model can be used to represent the distribution of data points as a mixture of Gaussian distributions, with each Gaussian representing a separate cluster.

Formally, a mixture model is defined as follows:

$f(x \mid \theta) = \Sigma_i w_i * f_i(x \mid \theta_i)$

where θ is a set of parameters, $f_i(x \mid \theta_i)$ is a component density function, w_i is a weight parameter for each component, and $\Sigma_i w_i = 1$.

In the case of GMM, each component density function is a multivariate Gaussian distribution, and the weight parameters represent the proportion of data points assigned to each cluster.

📌 Expectation-Maximization (EM) Algorithm

The Expectation-Maximization (EM) algorithm is a iterative method used to estimate the parameters of a statistical model that involves latent variables. In the context of GMM, the latent variables represent the assignment of each data point to a specific cluster.

The EM algorithm for GMM can be summarized in two steps:

1. Expectation step (E-step): Compute the expected value of the latent variables given the current estimate of the model parameters.
2. Maximization step (M-step): Update the model parameters to maximize the likelihood of the observed data, given the expected values of the latent variables.

The EM algorithm is guaranteed to converge to a local maximum of the likelihood function, but may not converge to the global maximum.

📌 Maximum Likelihood Estimation (MLE)

Maximum Likelihood Estimation (MLE) is a method used to estimate the parameters of a statistical model by maximizing the likelihood of the observed data, given the model parameters. In the context of GMM, the likelihood function is given by:

$L(\theta \mid X) = \Pi_i f(x_i \mid \theta)$

where θ represents the model parameters, $X = \{x_1, x_2, \ldots, x_n\}$ is the observed data, and $f(x_i \mid \theta)$ is the probability density function of the GMM.

The log-likelihood function can be written as:

$l(\theta \mid X) = \Sigma_i \log f(x_i \mid \theta)$

The maximum likelihood estimate of the model parameters is obtained by maximizing the log-likelihood function with respect to the model parameters. This can be done using the EM algorithm or other optimization techniques.

In the case of GMM, the model parameters include the mean vectors, covariance matrices, and weight parameters of the component Gaussian distributions.

In the next section, we will explain how the concepts covered in this section are used in the GMM algorithm for clustering.

📌 GMM Algorithm

The GMM algorithm for clustering can be summarized in the following steps:

1. Data preprocessing: Normalize the data to ensure that each feature has zero mean and unit variance.
2. Initialization of parameters: Choose an initial guess for the mean vectors, covariance matrices, and weight parameters of the component Gaussian distributions. One common way to initialize the means is to use the K-means algorithm.
3. EM algorithm for GMM:
4. Expectation step (E-step): Compute the posterior probability of each data point belonging to each cluster, based on the current estimate of the model parameters.
5. Maximization step (M-step): Update the model parameters to maximize the expected complete log-likelihood of the observed data, given the posterior probabilities obtained in the E-step. The updated parameters are obtained using maximum likelihood estimation (MLE).
6. Repeat the E-step and M-step until convergence is achieved, i.e., the log-likelihood of the observed data no longer increases significantly or a maximum number of iterations is reached.
7. Determining the optimal number of clusters: Use a model selection criterion such as the Bayesian Information Criterion (BIC) or the Akaike Information Criterion (AIC) to determine the optimal number of clusters.

📌 Expectation Step (E-step)

In the E-step, we compute the posterior probability of each data point belonging to each cluster, given the current estimate of the model parameters. This is also known as the responsibility of each cluster for each data point.

The posterior probability of data point i belonging to cluster j is given by:

$r_{ij} = \pi_j * N(x_i \mid \mu_j, \Sigma_j) / \Sigma_k \pi_k * N(x_i \mid \mu_k, \Sigma_k)$

where r_{ij} is the posterior probability, π_j is the weight parameter of cluster j, $N(x_i \mid \mu_j, \Sigma_j)$ is the probability density function of the multivariate Gaussian distribution for cluster j evaluated at data point i, and the denominator is the normalization constant that ensures that the sum of the posterior probabilities for each data point adds up to 1.

📌 Maximization Step (M-step)

In the M-step, we update the model parameters to maximize the expected complete log-likelihood of the observed data, given the posterior probabilities obtained in the E-step. The complete log-likelihood is given by:

$L(\theta \mid X, Z) = \Sigma_i \Sigma_j r_{ij} \log(\pi_j * N(x_i \mid \mu_j, \Sigma_j))$

where θ represents the model parameters, X is the observed data, and Z is the latent variables (i.e., the posterior probabilities computed in the E-step).

The updated parameters are obtained using MLE. The MLE estimate of the weight parameter π_j is simply the sum of the posterior probabilities of all data points assigned to cluster j:

$\pi_j = \Sigma_i r_{ij} / n$

where n is the total number of data points.

The MLE estimate of the mean vector μ_j is the weighted average of the data points assigned to cluster j:

$\mu_j = \Sigma_i r_{ij} x_i / \Sigma_i r_{ij}$

The MLE estimate of the covariance matrix Σ_j is the weighted covariance matrix of the data points assigned to cluster j:

$\Sigma_j = \Sigma_i r_{ij} (x_i — \mu_j)(x_i — \mu_j)^T / \Sigma_i r_{ij}$

📌 Convergence and Initialization

The EM algorithm for GMM is guaranteed to converge to a local maximum of the likelihood function, but may not converge to the global maximum. Therefore, the algorithm is often run multiple times with different initializations, and the result with the highest likelihood is chosen.

The initialization of the GMM parameters can have a significant impact on the convergence and quality of the clustering. One common way to initialize the means is to use the K-means algorithm, which clusters the data into K clusters and uses the cluster centers as the initial means for the GMM.

🐍 Implementing GMM in Python

In this section, we will show you how to implement GMM for clustering in Python using the scikit-learn library. We will use the iris flower dataset as an example.

🦊 If you want to learn more about the available datasets in scikit-learn and how to select the right dataset for your machine learning modeling, you can consider reading "[Selecting the Right Dataset for Machine Learning Modeling with scikit-learn: An Overview of the Most Popular Datasets]".

📌 Loading and preparing data

First, we need to load the iris dataset and prepare the data for clustering. We will use the sepal length and width features for simplicity.

```
from sklearn.datasets import load_iris

iris = load_iris()X = iris.data[:, :2]
```

Next, we will normalize the data using Z-score normalization to ensure that each feature has zero mean and unit variance.

```
from sklearn.preprocessing importStandardScaler

   StandardScaler()X_norm = scaler.fit_transform(X)
```

📌 Defining the GMM model and hyperparameters

We will use the scikit-learn implementation of GMM, which can be imported from the `mixture` module. To create a GMM model, we need to specify the number of components (i.e., clusters) and the covariance type.

```
from sklearn.mixture import GaussianMixture

n_components = 3  covariance_type = 'full'  gmm = GaussianMixture(n_components=n_components,
covariance_type=covariance_type)
```

We can also specify other hyperparameters such as the maximum number of iterations and the convergence tolerance. By default, scikit-learn uses the EM algorithm to estimate the parameters of the GMM.

```
max_iter = 100  tol = 1e-3  gmm.set_params(max_iter=max_iter, tol=tol)
```

To fit the GMM model to the data, we simply call the `fit` method on the GMM object.

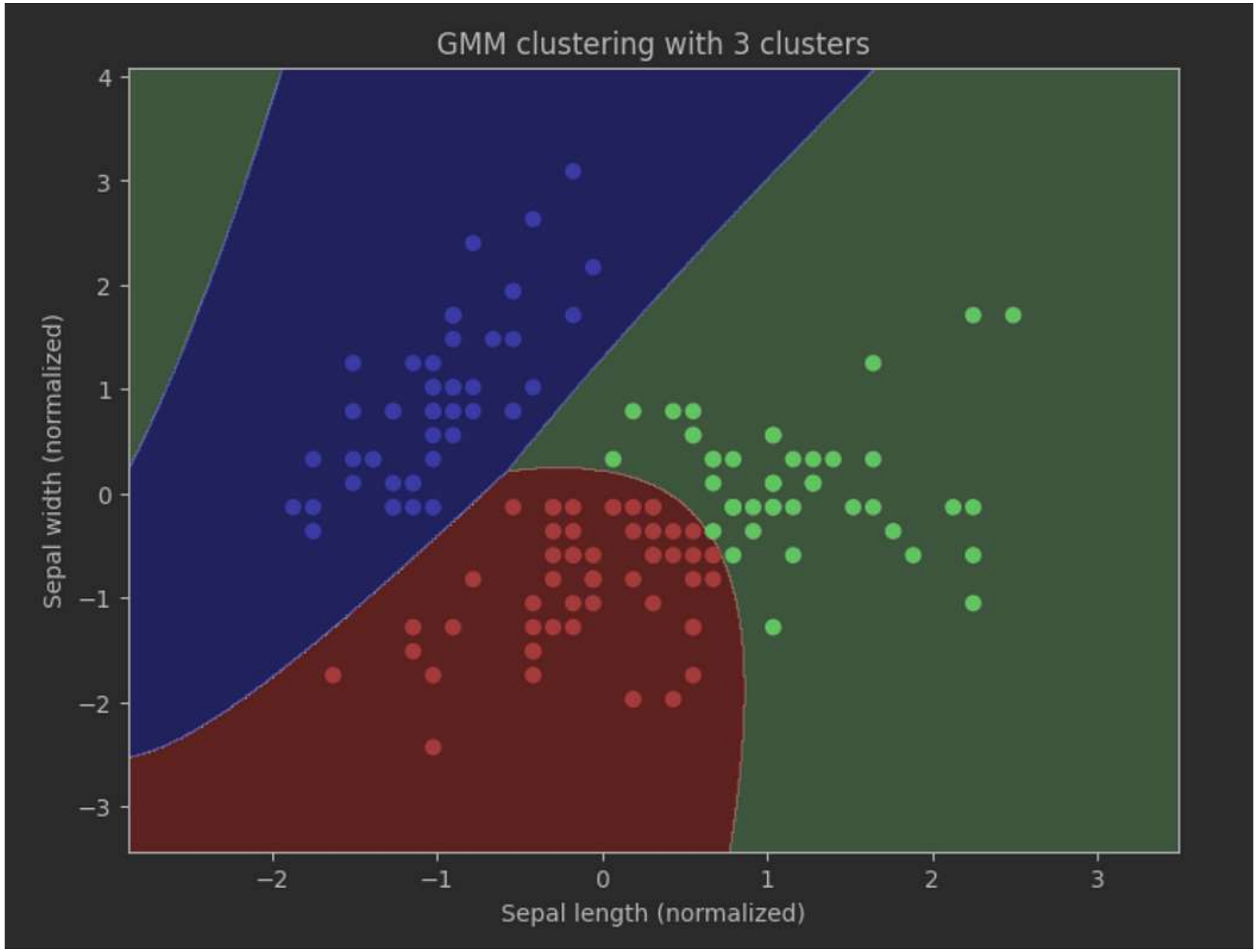
```
gmm(X_norm)
```

This will estimate the parameters of the GMM using the EM algorithm and assign each data point to a cluster based on the highest posterior probability.

📌 Visualizing the clusters

To visualize the clusters, we can plot the data points with different colors representing the assigned clusters. We can also plot the contours of the Gaussian distributions for each cluster.

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

# Define colormap
cmap = ListedColormap(['r', 'g', 'b'])

# Create meshgrid for contour plot
x_min, x_max = X_norm[:, 0].min() - 1, X_norm[:, 0].max() + 1
y_min, y_max = X_norm[:, 1].min() - 1, X_norm[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01), np.arange(y_min, y_max, 0.01))
Z = gmm.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.figure(figsize=(, ))plt.contourf(xx, yy, Z, cmap=cmap, alpha=)plt.scatter(X_norm[:, ], X_norm[:, ],
c=gmm.predict(X_norm), cmap=cmap)plt.xlabel()plt.ylabel()plt.title(.(n_components))plt.show()
```



Gaussian Mixture Model Clustering plot

This will show a scatter plot of the data points with different colors representing the assigned clusters, as well as the contours of the Gaussian distributions for each cluster.

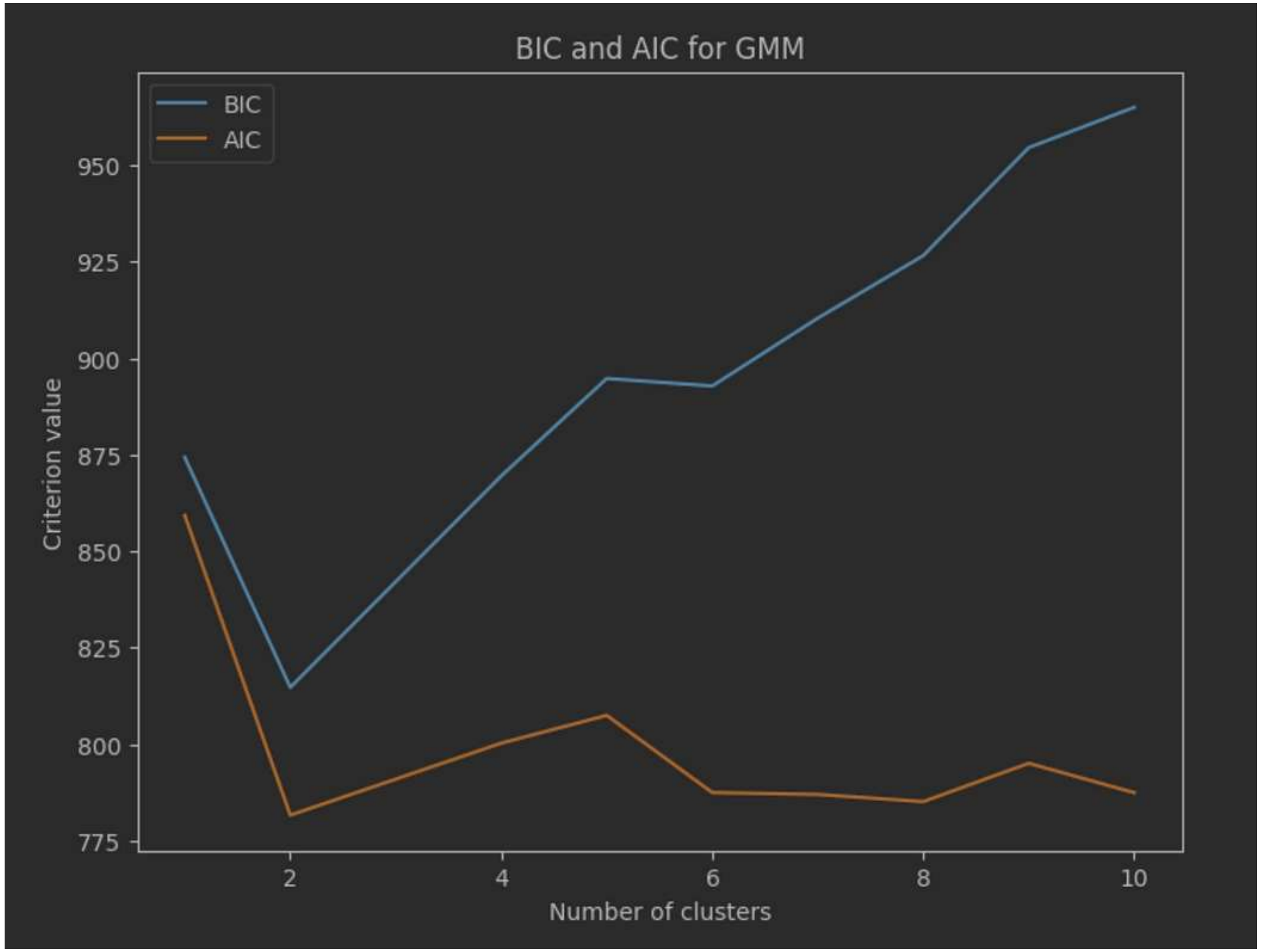
📌 Determining the optimal number of clusters

One of the challenges in clustering is determining the optimal number of clusters. One way to do this is by using the Bayesian Information Criterion (BIC) or the Akaike Information Criterion (AIC), which penalize models with more parameters. The lower the BIC or AIC, the better the model.

```
# Compute BIC and AIC for different number of clusters
n_components_range = range(1, 11)
bic = []
aic = []
for n_components in n_components_range:
    gmm = GaussianMixture(n_components=n_components, covariance_type=covariance_type)
    gmm.fit(X_norm)
    bic.append(gmm.bic(X_norm))
    aic.append(gmm.aic(X_norm))
```

```
plt.figure(figsize=(, ))plt.plot(n_components_range, bic, label=)plt.plot(n_components_range, aic,
label=)plt.legend()plt.xlabel()plt.ylabel()plt.title()plt.show()
```

This will show a plot of the BIC and AIC as a function of the number of clusters. The optimal number of clusters is the one that minimizes the BIC or AIC.



BIC and AIC plot for determining the optimal number of clusters

In the next section, we will show some examples of using GMM for clustering.

Examples and Applications

GMM is a powerful algorithm for clustering that can be applied to a wide range of data types and domains. In this section, we will show some examples and applications of GMM for clustering.

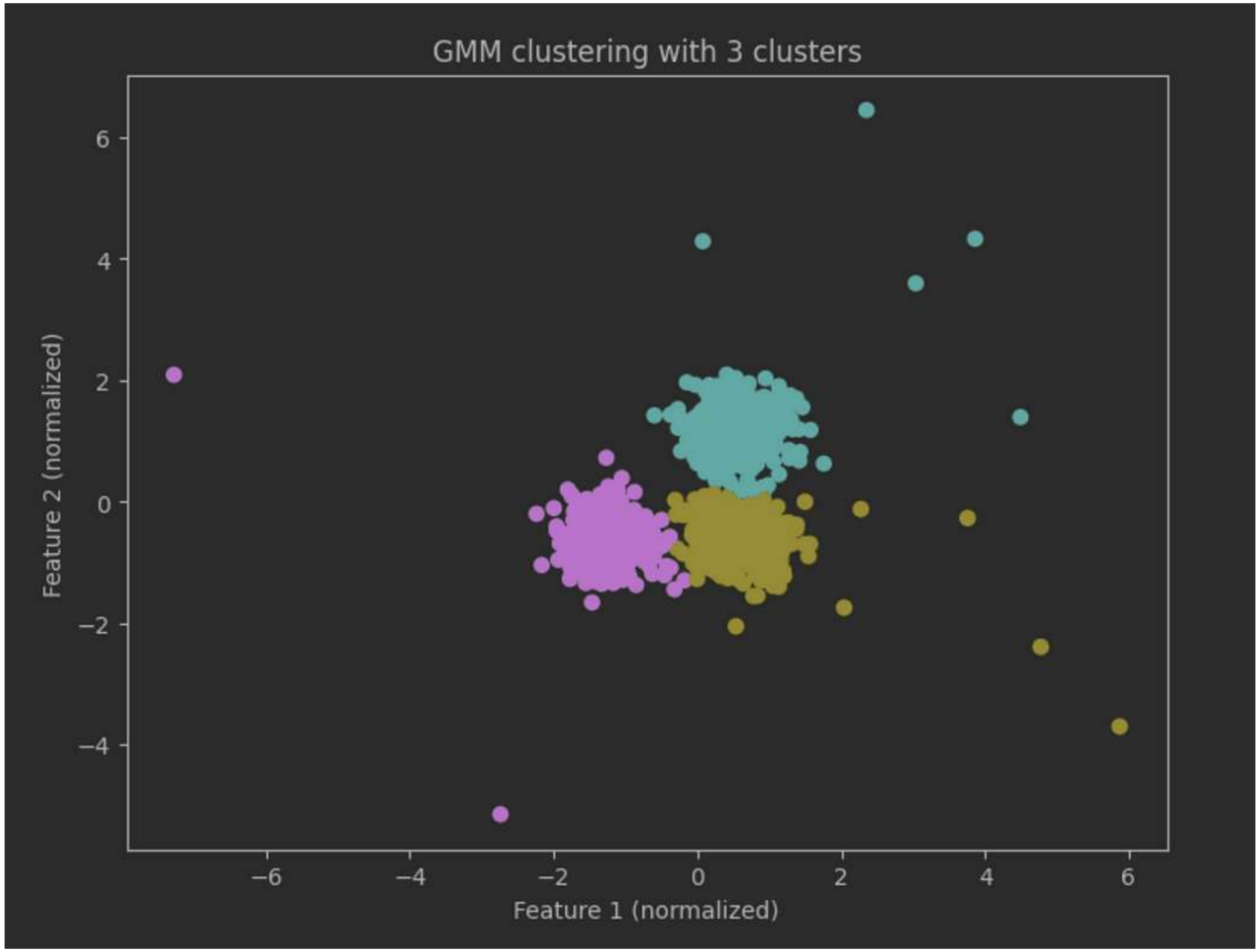
💡 Example 1: Clustering of Synthetic Data

To illustrate the performance of GMM for clustering, we will generate some synthetic data and cluster it using GMM. We will generate data points from three different multivariate Gaussian distributions with different means and covariance matrices, and add some noise.

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

# Generate synthetic data
n_samples = 1000
centers = [[1, 1], [-1, -1], [1, -1]]
X, y_true = make_blobs(n_samples=n_samples, centers=centers, cluster_std=0.4, random_state=0)

# Add some noise
np.random.seed(0)
X = np.vstack((X, 3 * np.random.randn(20, 2)))

# Normalize the data
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_norm = scaler.fit_transform(X)

# Cluster the data using GMM
n_components = 3
covariance_type = 'full'
gmm = GaussianMixture(n_components=n_components, covariance_type=covariance_type)
gmm.fit(X_norm)
```

```
plt.figure(figsize=(, ))plt.scatter(X_norm[:, ], X_norm[:, ], c=gmm.predict(X_norm),
cmap=)plt.xlabel()plt.ylabel()plt.title(.(n_components))plt.show()
```

This will show a scatter plot of the data points with different colors representing the assigned clusters, as well as the contours of the Gaussian distributions for each cluster.



GMM Clustering of Synthetic Data

💡 Example 2: Image Segmentation

GMM can also be used for image segmentation, which involves partitioning an image into multiple regions based on similarities in pixel values. Each region corresponds to a cluster in the GMM.

```
import numpy as np
import matplotlib.pyplot as plt
from skimage import data, segmentation, color, graph
from sklearn.mixture import GaussianMixture

# Load an image
img = data.coffee()

# Convert the image to grayscale
img_gray = color.rgb2gray(img)

# Normalize the data
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
X_norm = scaler.fit_transform(img_gray.reshape(-1, 1))

# Cluster the data using GMM
n_components = 3
covariance_type = 'full'
gmm = GaussianMixture(n_components=n_components, covariance_type=covariance_type)
gmm.fit(X_norm)

# Segment the image based on the clusters
labels = gmm.predict(X_norm).reshape(img_gray.shape)
```

```
# Create a graph and merge small regions
g = graph.RAG(labels)
labels = graph.cut_normalized(labels, g)
plt.figure(figsize=(, ))plt.subplot()plt.imshow(img)plt.axis()plt.title()plt.subplot()plt.imshow(color.label2rgb(labels,
img, kind=))plt.axis()plt.title(.(n_components))plt.show()
```

This will show the original image and the segmented image, where each region corresponds to a cluster in the GMM.

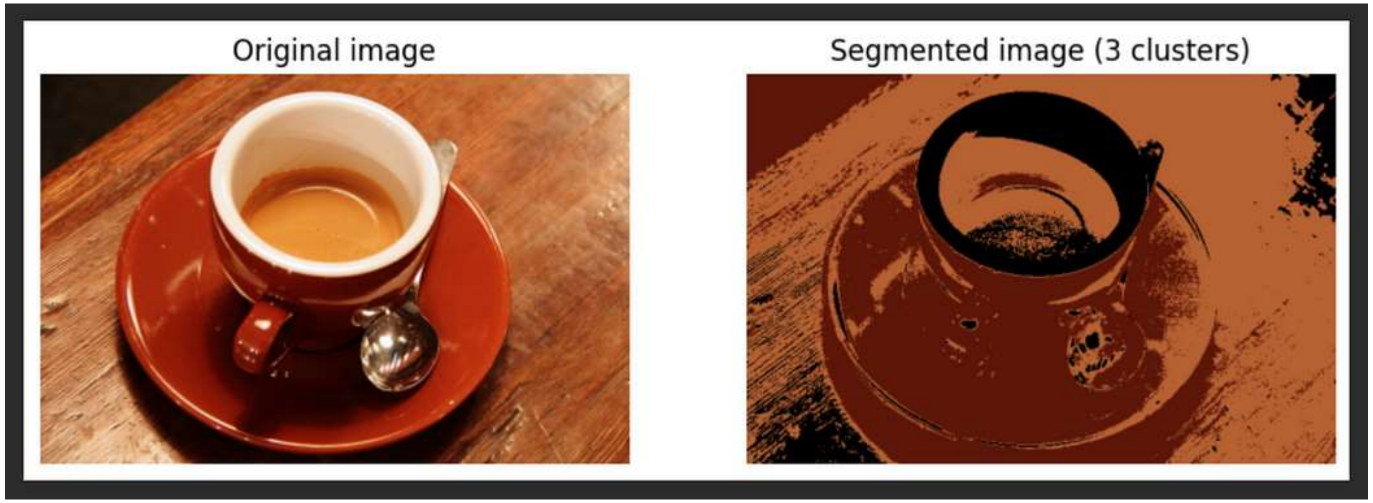


Image Segmentation using GMM Clustering

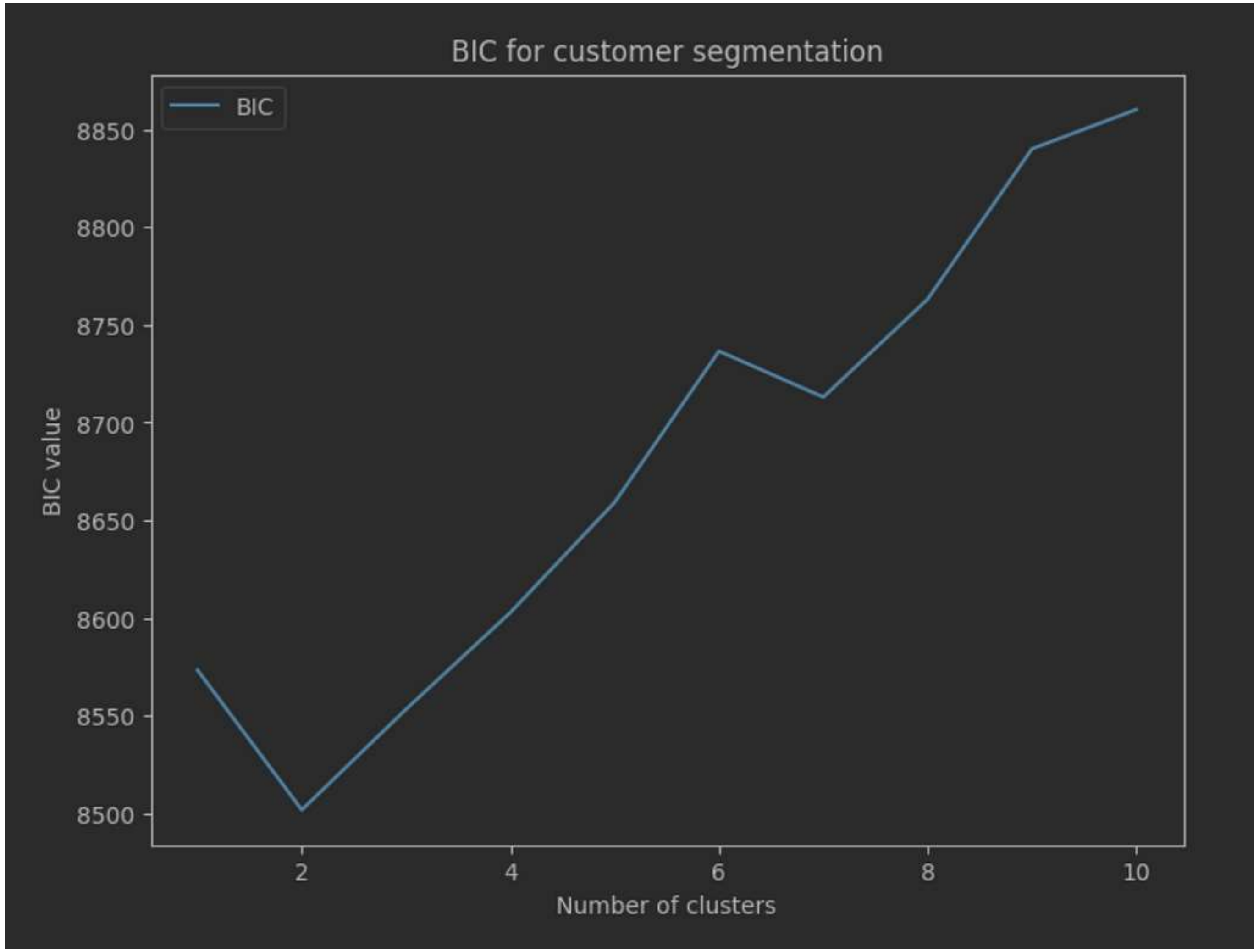
💡 Example 3: Customer Segmentation

GMM can also be used for customer segmentation, which involves clustering customers based on their demographic and behavioral data. This can help businesses to identify different segments of customers with similar needs and preferences, and tailor their marketing strategies accordingly.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.mixture import GaussianMixture
from sklearn.preprocessing import StandardScaler
```

```
# Generate some imaginary customer data
age = np.random.randint(18, 65, size=(1000,))
income = np.random.normal(50000, 10000, size=(1000,))
spending = np.random.normal(1000, 500, size=(1000,))
data = np.column_stack((age, income, spending))

# Normalize the data
scaler = StandardScaler()
X_norm = scaler.fit_transform(data)

# Determine the optimal number of clusters using BIC
n_components_range = range(1, 11)
bic = []
for n_components in n_components_range:
    gmm = GaussianMixture(n_components=n_components, covariance_type='full')
    gmm.fit(X_norm)
    bic.append(gmm.bic(X_norm))
plt.figure(figsize=(, ))plt.plot(n_components_range, bic, label=)plt.legend()plt.xlabel()plt.ylabel()plt.title()plt.show()
```

This will show a plot of the BIC as a function of the number of clusters. The optimal number of clusters is the one that minimizes the BIC.

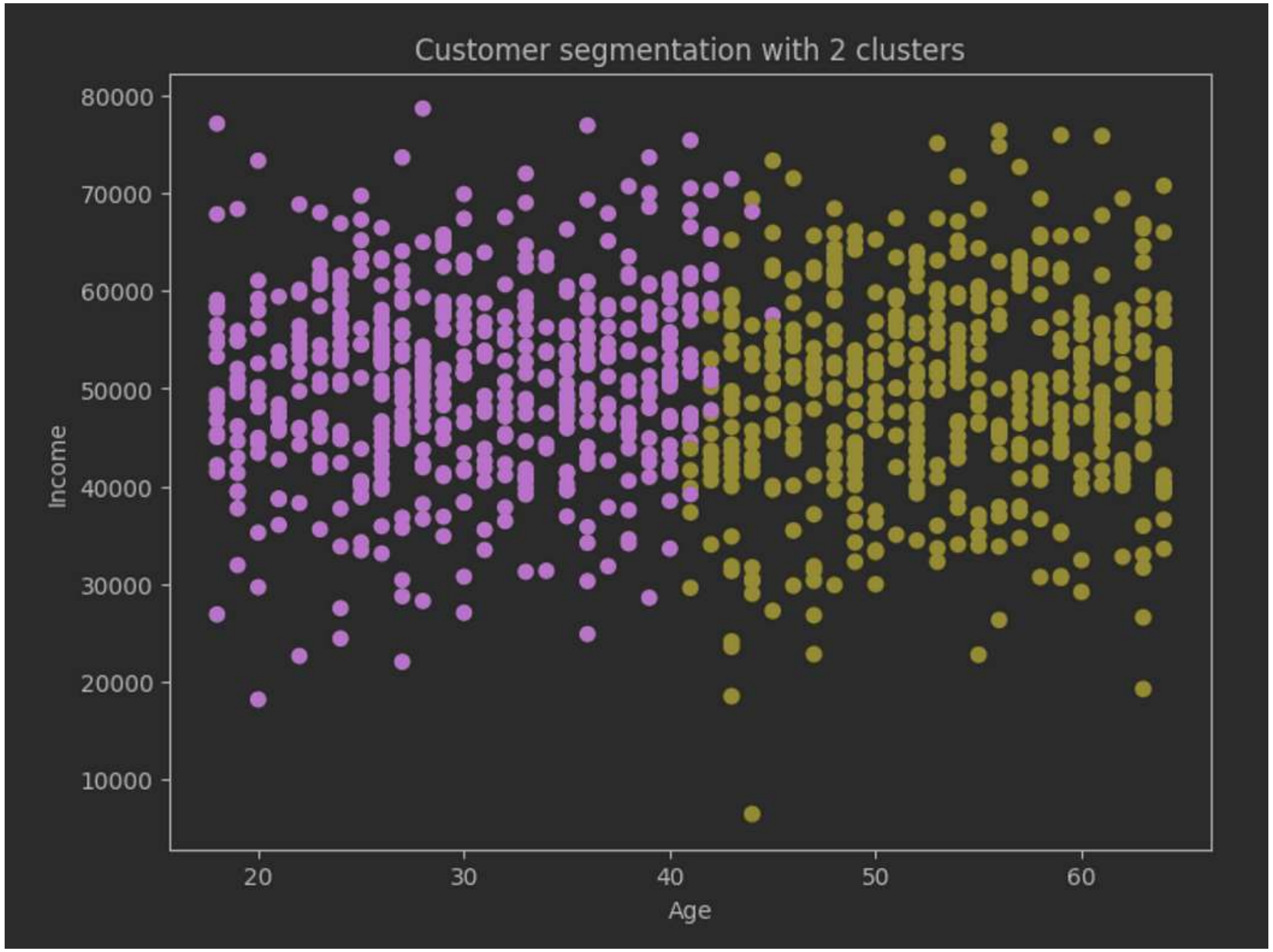


Customer Segmentation using GMM Clustering

```
# Cluster the customers using GMM
n_components = 2
gmm = GaussianMixture(n_components=n_components, covariance_type='full')
```

```
gmm.fit(X_norm)
labels = gmm.predict(X_norm)

# Add the cluster labels to the data
df = {'Age': age, 'Income': income, 'Spending': spending, 'Cluster': labels}
plt.figure(figsize=(, ))plt.scatter(age, income, c=labels, cmap=)plt.xlabel()plt.ylabel()plt.title(.
(n_components))plt.show()
```

This will show a scatter plot of the customers with different colors representing the assigned clusters, based on their age and income. Businesses can use this information to target specific segments of customers with tailored marketing strategies.



Customer Segmentation using GMM Clustering

In this tutorial, we have covered the fundamentals of clustering with Gaussian Mixture Model (GMM). We have shown how GMM can be used to cluster data points into multiple components, each modeled by a multivariate Gaussian distribution. We have also described the EM algorithm for GMM, and discussed the importance of initialization and convergence. Finally, we have demonstrated some examples and applications of GMM for clustering, including clustering of synthetic data, image segmentation, and customer segmentation.

GMM is a powerful algorithm for clustering that can be applied to a wide range of data types and domains. It has several advantages over other clustering algorithms, including the ability to model non-spherical and overlapping clusters, and the ability to estimate the uncertainty of cluster assignments. However, GMM also has some limitations, such as the sensitivity to the number of clusters and the initialization of the parameters.

🤔 Limitations and extensions of GMM

📌 Advantages of GMM

One of the advantages of GMM is that it is a probabilistic clustering method. This means that it provides a measure of uncertainty in the cluster assignments. Unlike other clustering methods, such as K-means, which assigns each point to a single cluster, GMM allows for overlapping clusters. This makes GMM a more flexible and powerful clustering method.

Another advantage of GMM is that it can model complex cluster shapes using a combination of Gaussian distributions. This allows for more accurate clustering of data that has complex or irregular patterns.

📌 Disadvantages and limitations of GMM

Despite its advantages, GMM also has some limitations and disadvantages that must be considered. One of the main limitations of GMM is its sensitivity to the choice of the number of clusters. If the number of clusters is not chosen carefully, GMM can easily overfit or underfit the data. This can lead to poor clustering results and inaccurate interpretations.

GMM is also sensitive to noise and outliers. In particular, the use of a full covariance matrix in GMM can make the method more prone to overfitting in the presence of noise or outliers. This is because the full covariance matrix allows for more flexibility in the shape of the clusters, which can result in spurious clusters that capture noise or outliers.

📌 Extensions and alternatives to GMM

To address some of the limitations of GMM, several extensions and alternatives have been proposed. One extension is to use Bayesian GMM, which places a prior distribution on the number of clusters and allows for automatic determination of the number of clusters. This can avoid the need for manual selection of the number of clusters and improve the robustness of the clustering method.

Another alternative is to use K-means or other clustering methods that are less sensitive to noise and outliers. K-means is a simple and efficient clustering method that assigns each point to the nearest cluster center. It is less sensitive to noise and outliers than GMM, but it cannot model overlapping clusters or complex cluster shapes.

In addition to these extensions and alternatives, it is important to carefully preprocess the data before clustering and to carefully select the appropriate clustering method based on the characteristics of the data and the goals of the analysis.

Further Reading

If you are interested in learning more about clustering with GMM, here are some resources that you might find helpful:

- Bishop, C. M. (2006). Pattern recognition and machine learning. Springer. Chapter 9 covers GMM for clustering and provides a comprehensive mathematical treatment of the algorithm.
- Scikit-learn documentation: The official documentation of the scikit-learn library provides detailed information on how to use GMM for clustering, as well as other clustering algorithms.
- An introduction to clustering and Gaussian mixture models: A tutorial by Jeff Calder, which provides a gentle introduction to clustering and GMM, with Python code examples.
- How to estimate the number of clusters in your data: An article by Jake VanderPlas, which provides a detailed explanation of the BIC and AIC criteria for model selection, and how to use them to estimate the number of clusters in your data.
- Cluster analysis in R: A tutorial by Brian Caffo, which provides a comparison of different clustering algorithms, including GMM, and how to implement them in R.

🦊 If you want to learn more about clustering in unsupervised learning, you may be interested in reading my other post "Clustering Methods 101: An Introduction to Unsupervised Learning Techniques." This tutorial provides a comprehensive overview of different clustering methods, including hierarchical clustering, density-based clustering, and model-based clustering, and discusses their advantages and disadvantages. It also covers topics such as cluster evaluation and selection, and provides examples of clustering applications in different fields.

🦊 I also invite you to explore my tutorials on supervised and unsupervised learning, which can be found under the following topic lists: 'Topics on Supervised Learning', 'Topics on Unsupervised Learning', and 'General Topics on Machine Learning'.