

06_gaussian_mixture_models

September 29, 2021

1 Gaussian Mixture Models

Gaussian mixture models (GMM) are a generative model that assumes the data has been generated by a mix of various multivariate normal distributions. The algorithm aims to estimate the mean & covariance matrices of these distributions.

It generalizes the k-Means algorithm: it adds covariance among features so that clusters can be ellipsoids rather than spheres, while the centroids are represented by the means of each distribution. The GMM algorithm performs soft assignments because each point has a probability to be a member of any cluster.

1.1 The Expectation-Maximization Algorithm

Expectation-Maximization Algorithm

GMM uses the expectation-maximization algorithm to identify the components of the mixture of Gaussian distributions. The goal is to learn the probability distribution parameters from unlabeled data.

The algorithm proceeds iteratively as follows: 1. Initialization: Assume random centroids (e.g. from K-Means) 2. Repeat until convergence (changes in assignments drop below threshold): 1. Expectation Step: Soft assignment - compute probabilities for each point from each distribution 2. Maximization Step: Adjust normal-distribution parameters to make data points most likely

1.2 Imports & Settings

```
[1]: %matplotlib inline
import warnings
import matplotlib.pyplot as plt
from matplotlib import cm
import pandas as pd
import numpy as np
from numpy import atleast_2d
from sklearn.decomposition import PCA
from sklearn.mixture import GaussianMixture
from sklearn.metrics import adjusted_mutual_info_score
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_iris
from mpl_toolkits.mplot3d import Axes3D
```

```
[2]: cmap = cm.get_cmap('viridis')
pd.options.display.float_format = '{:,.2f}'.format
warnings.filterwarnings('ignore')
```

1.3 Load Iris Data

```
[3]: iris = load_iris()
iris.keys()
```

```
[3]: dict_keys(['data', 'target', 'target_names', 'DESCR', 'feature_names',
'filename'])
```

1.4 Create DataFrame

```
[4]: features = iris.feature_names
data = pd.DataFrame(data=np.column_stack([iris.data, iris.target]),
                    columns=features + ['label'])
data.label = data.label.astype(int)
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
sepal length (cm)    150 non-null float64
sepal width (cm)     150 non-null float64
petal length (cm)    150 non-null float64
petal width (cm)     150 non-null float64
label                150 non-null int64
dtypes: float64(4), int64(1)
memory usage: 5.9 KB
```

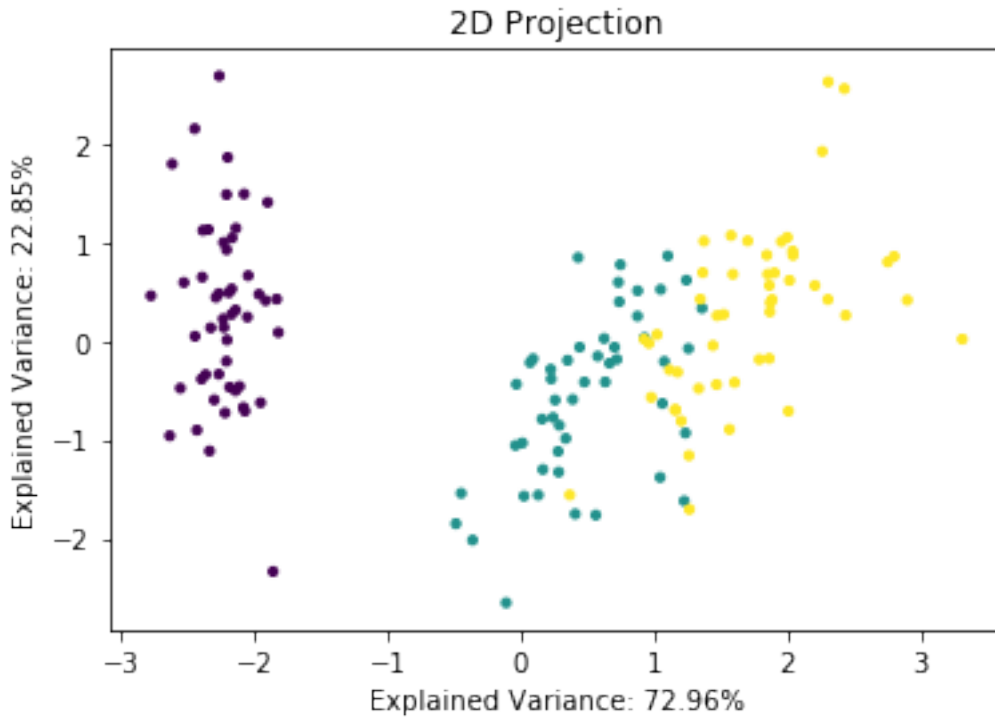
1.4.1 Standardize Data

```
[5]: scaler = StandardScaler()
features_standardized = scaler.fit_transform(data[features])
n = len(data)
```

1.4.2 Reduce Dimensionality to visualize clusters

```
[6]: pca = PCA(n_components=2)
features_2D = pca.fit_transform(features_standardized)
```

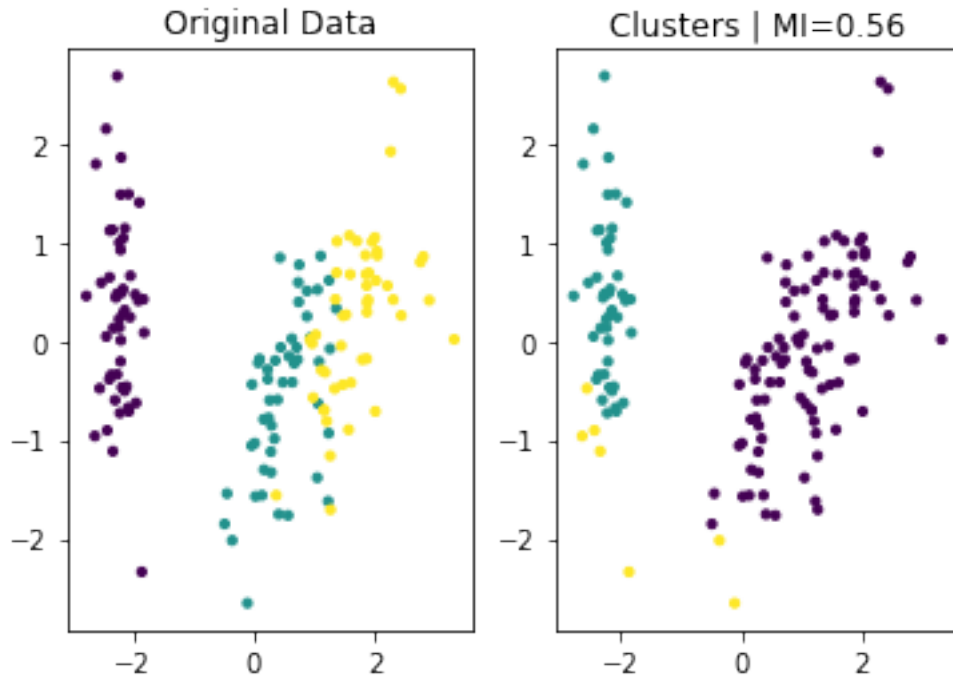
```
[7]: ev1, ev2 = pca.explained_variance_ratio_
ax = plt.figure().gca(title='2D Projection',
                      xlabel='Explained Variance: {:.2%}'.format(ev1),
                      ylabel='Explained Variance: {:.2%}'.format(ev2))
ax.scatter(*features_2D.T, c=data.label, s=10);
```



1.5 Perform GMM clustering

```
[8]: n_components = 3
gmm = GaussianMixture(n_components=n_components)
gmm.fit(features_standardized)
data['clusters'] = gmm.predict(features_standardized)
fig, axes = plt.subplots(ncols=2)
labels, clusters = data.label, data.clusters
mi = adjusted_mutual_info_score(labels, clusters)
axes[0].scatter(*features_2D.T, c=data.label, s=10)
axes[0].set_title('Original Data')
axes[1].scatter(*features_2D.T, c=data.clusters, s=10)
axes[1].set_title('Clusters | MI={:.2f}'.format(mi))
```

```
[8]: Text(0.5,1,'Clusters | MI=0.56')
```



1.5.1 Visualize Gaussian Distributions

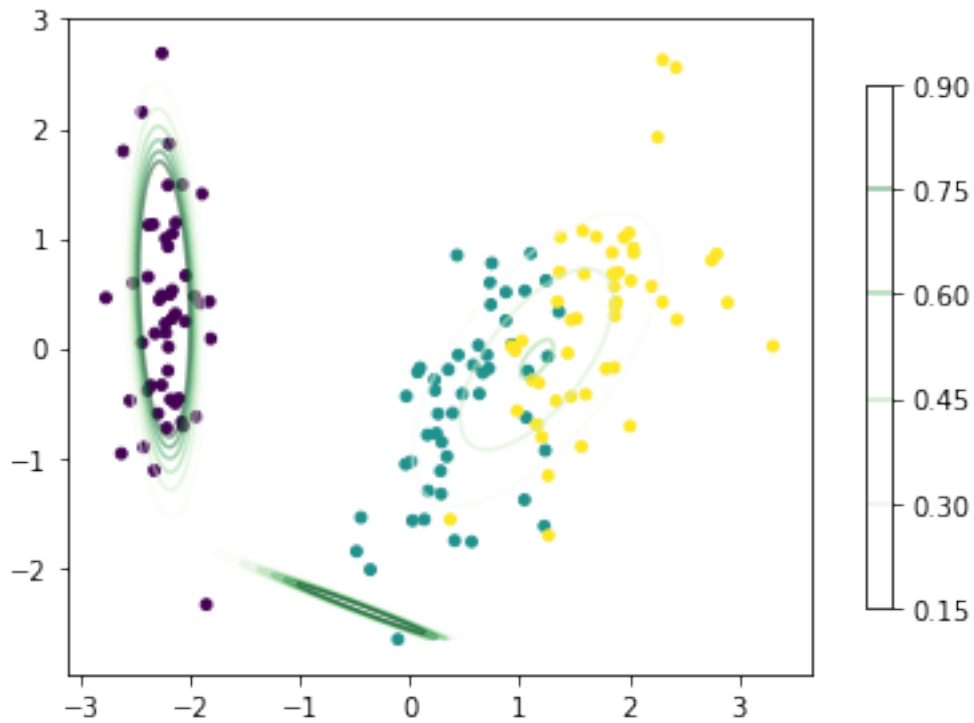
The following figures show the GMM cluster membership probabilities for the iris dataset as contour lines:

```
[9]: xmin, ymin = features_2D.min(axis=0)
      xmax, ymax = features_2D.max(axis=0)

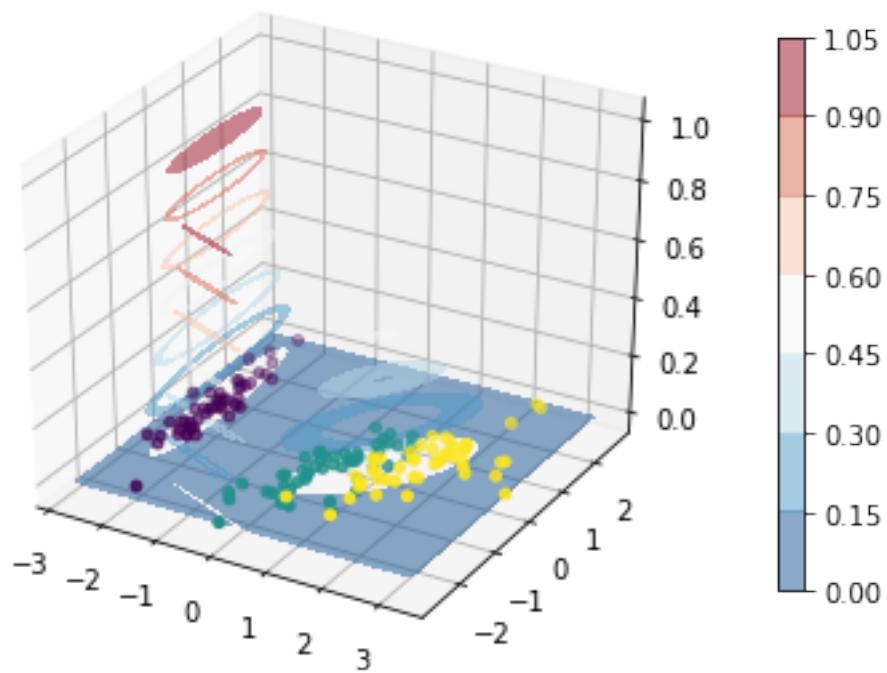
      x = np.linspace(xmin, xmax, 500)
      y = np.linspace(ymin, ymax, 500)
      X, Y = np.meshgrid(x, y)

      simulated_2D = np.column_stack([np.ravel(X), np.ravel(Y)])
      simulated_4D = pca.inverse_transform(simulated_2D)
      Z = atleast_2d(np.clip(np.exp(gmm.score_samples(simulated_4D)), a_min=0,
      ↪a_max=1)).reshape(X.shape)
```

```
[10]: fig, ax = plt.subplots()
      ax.set_aspect('equal')
      CS = ax.contour(X, Y, Z, cmap='Greens', alpha=.5)
      CB = plt.colorbar(CS, shrink=0.8)
      ax.scatter(*features_2D.T, c=data.label, s=15)
      fig.tight_layout();
```



```
[11]: fig = plt.figure()
ax = fig.gca(projection='3d')
ax.set_aspect('equal')
CS = ax.contourf3D(X, Y, Z, cmap='RdBu_r', alpha=.5)
CB = plt.colorbar(CS, shrink=0.8)
ax.scatter(*features_2D.T, c=data.label, s=15)
fig.tight_layout()
fig.savefig('gaussian3d', dpi=300)
```



1.5.2 Bayesian Information Criterion

```
[12]: bic = {}
      for n_components in range(2, 8):
          gmm = GaussianMixture(n_components=n_components)
          gmm.fit(features_standardized)
          bic[n_components] = gmm.bic(features_standardized)
      pd.Series(bic)
```

```
[12]: 2      794.71
      3      801.55
      4      853.82
      5      845.19
      6      962.57
      7     1,001.98
      dtype: float64
```