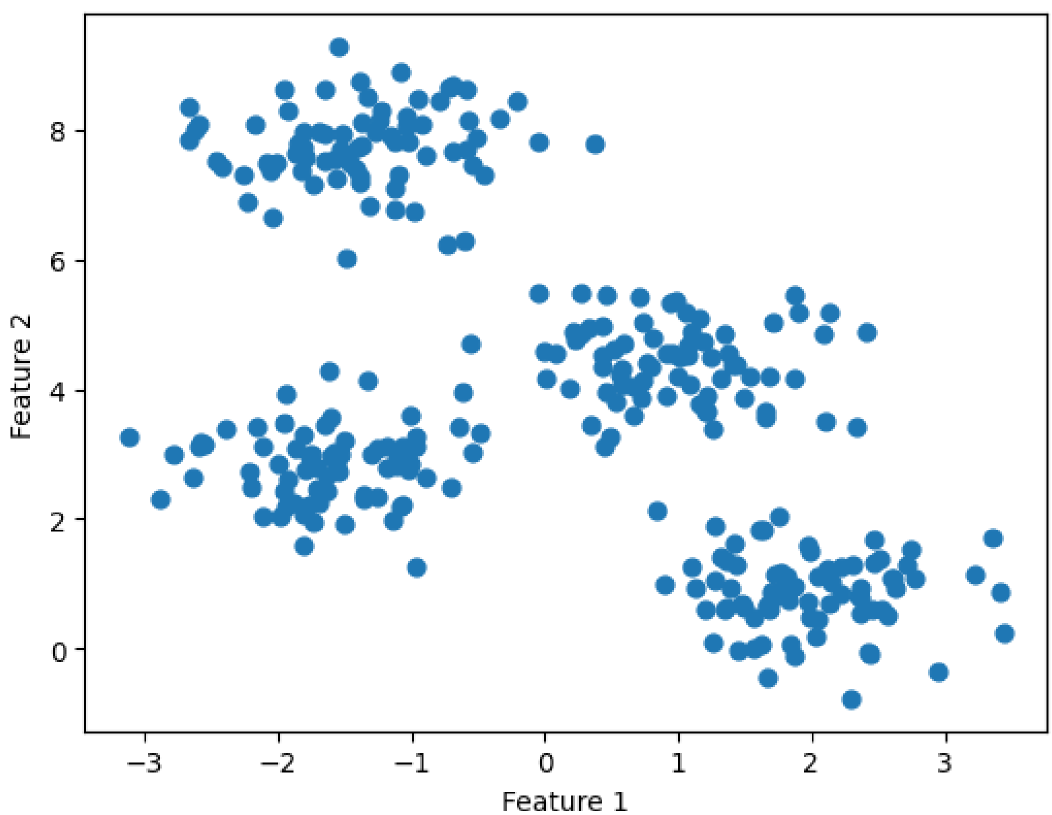
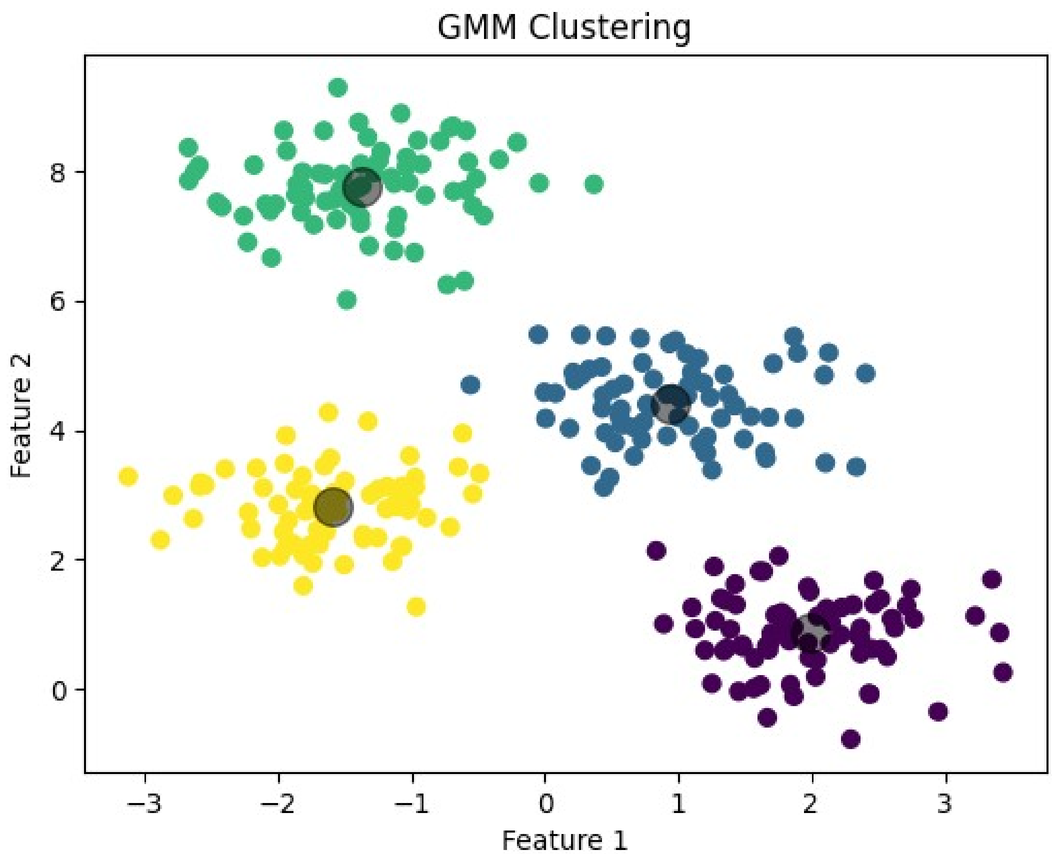
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.mixture import GaussianMixture

# Generate a toy 2D dataset
X, y_true = make_blobs(n_samples=300, centers=4, cluster_std=0.60,
random_state=0)

plt.scatter(X[:, 0], X[:, 1], s=40, cmap='viridis')

plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```

```
<ipython-input-8-88ffe239c992>:1: UserWarning: No data for
colormapping provided via 'c'. Parameters 'cmap' will be ignored
  plt.scatter(X[:, 0], X[:, 1], s=40, cmap='viridis')
```




```python
# Apply Gaussian Mixture Model clustering
gmm = GaussianMixture(n_components=4, random_state=0).fit(X)
labels = gmm.predict(X)
```

```
# Plotting the clusters
plt.scatter(X[:, 0], X[:, 1], c=labels, s=40, cmap='viridis')

# Optionally, plot the center of each cluster
centers = gmm.means_
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5)

plt.title("GMM Clustering")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```




```
gmm.converged_
```

```
True
```

```
gmm.n_iter_
```

```
2
```

```
gmm.covariances_
```

```
array([[[ 0.33998651, -0.02620931],
        [-0.02620931,  0.34588507]],

       [[ 0.3808649 , -0.02231979],
        [-0.02231979,  0.34881473]],

       [[ 0.41216925,  0.02884065],
        [ 0.02884065,  0.37959193]],

       [[ 0.32360185,  0.01027908],
        [ 0.01027908,  0.30862196]]])
```

```
gmm.means_
```

```
array([[ 1.98299679,  0.86735608],
       [ 0.93842466,  4.41564635],
       [-1.37355181,  7.75436785],
       [-1.58913964,  2.82465347]])
```

```
gmm.weights_
```

```
array([0.24991431, 0.25170956, 0.24988383, 0.2484923 ])
```

```
X.shape
```

```
(100, 100)
```

```
gmm.predict(X)
# hard clustering
```

```
array([0, 2, 1, 2, 0, 0, 3, 1, 2, 2, 3, 2, 1, 2, 0, 1, 1, 0, 3, 3, 0,
0,
       1, 3, 3, 1, 0, 1, 3, 1, 2, 2, 1, 2, 2, 2, 2, 2, 3, 0, 1, 3, 1,
1,
       3, 3, 2, 3, 2, 0, 3, 0, 2, 0, 0, 3, 2, 3, 2, 0, 2, 1, 2, 3, 3,
3,
       2, 0, 2, 3, 1, 3, 2, 3, 3, 2, 3, 1, 0, 2, 0, 1, 0, 0, 2, 1, 0,
1,
       2, 2, 1, 0, 2, 3, 3, 1, 0, 0, 1, 3, 2, 0, 2, 0, 1, 0, 0, 1, 2,
1,
       3, 3, 0, 2, 0, 1, 2, 0, 0, 1, 3, 0, 3, 0, 0, 0, 0, 3, 0, 3, 2,
3,
       3, 0, 2, 3, 3, 2, 1, 2, 2, 3, 1, 3, 1, 3, 2, 1, 2, 2, 2, 1, 2,
1,
       0, 3, 2, 3, 0, 1, 2, 1, 1, 0, 1, 3, 3, 1, 0, 1, 1, 2, 0, 1, 3,
2,
       0, 0, 1, 3, 0, 1, 3, 3, 1, 1, 1, 1, 0, 2, 1, 3, 1, 1, 3, 3, 3,
1,
       3, 2, 1, 3, 0, 3, 1, 2, 3, 2, 1, 2, 1, 3, 1, 1, 2, 3, 3, 0, 0,
1,
       2, 0, 0, 3, 0, 3, 1, 2, 2, 1, 1, 2, 1, 0, 3, 1, 0, 3, 2, 3, 0,
```

```
1,
       0, 2, 2, 2, 2, 3, 3, 2, 1, 3, 0, 1, 3, 3, 3, 0, 0, 2, 1, 1, 3,
0,
       2, 3, 1, 2, 1, 0, 0, 3, 3, 1, 0, 0, 0, 1, 2, 2, 0, 0, 1, 0, 0,
0,
       2, 3, 2, 1, 0, 0, 2, 2, 2, 0, 0, 1, 2, 3])
```

```
gmm.predict_proba(X)
# soft clustering
```

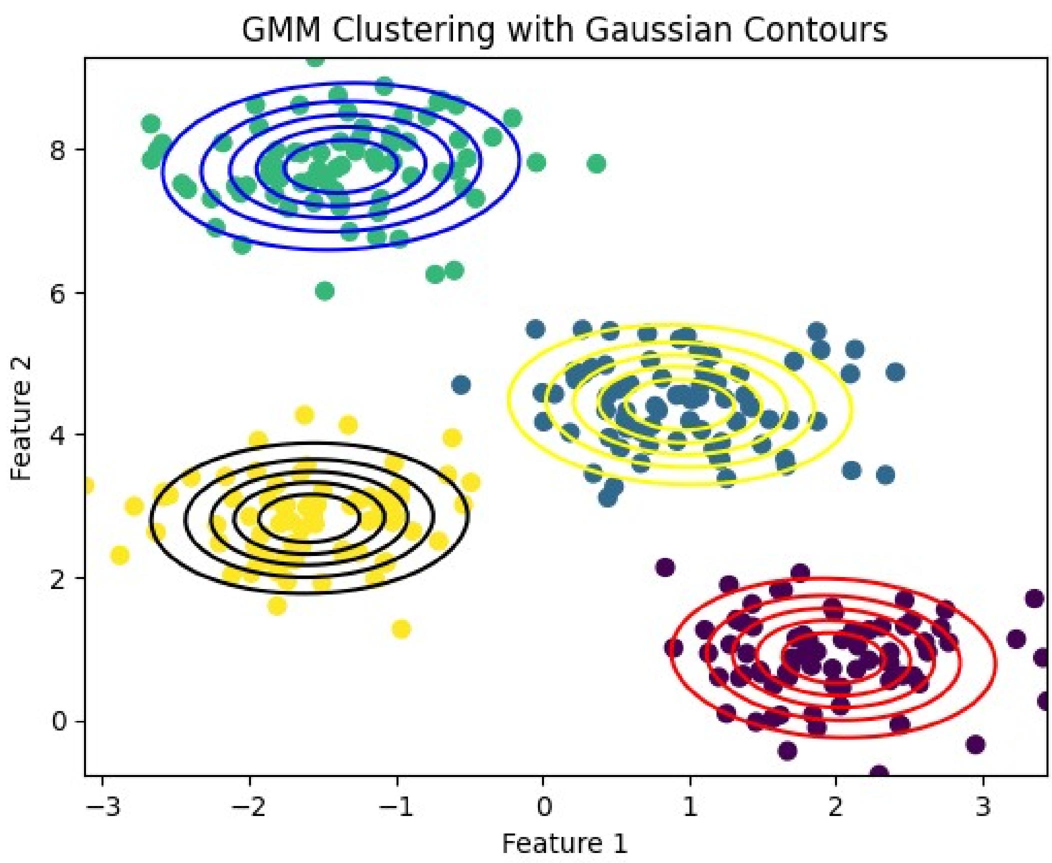
```
array([[9.71688955e-01, 2.59296402e-02, 8.04672295e-21, 2.38140471e-
03],
       [7.13116691e-33, 7.15846263e-09, 9.99999993e-01, 2.34837535e-
15],
       [8.78022033e-12, 9.99999970e-01, 2.04767765e-08, 9.13480329e-
09],
       ...,
       [4.92339895e-10, 9.99965893e-01, 8.75462809e-09, 3.40976171e-
05],
       [6.44915198e-30, 3.01241053e-06, 9.99996988e-01, 1.52780115e-
18],
       [1.99754828e-11, 4.39280339e-07, 4.05180546e-15, 9.99999561e-
01]])
```

```
gmm.sample(10)
# density estimation
# generative model -> ChatGPT
```

```
(array([[ 3.46104002,  0.33956771],
        [ 3.15066329,  1.22412061],
        [ 1.25803247,  1.2513661 ],
        [-0.19339363,  5.7019298 ],
        [ 0.8220839 ,  4.85341196],
        [ 1.41352833,  5.09249057],
        [-1.24527745,  8.02115905],
        [-0.94451894,  8.26765175],
        [-1.77618454,  3.15008423],
        [-0.64849308,  2.9529137 ]]),
 array([0, 0, 0, 1, 1, 1, 2, 2, 3, 3]))
```

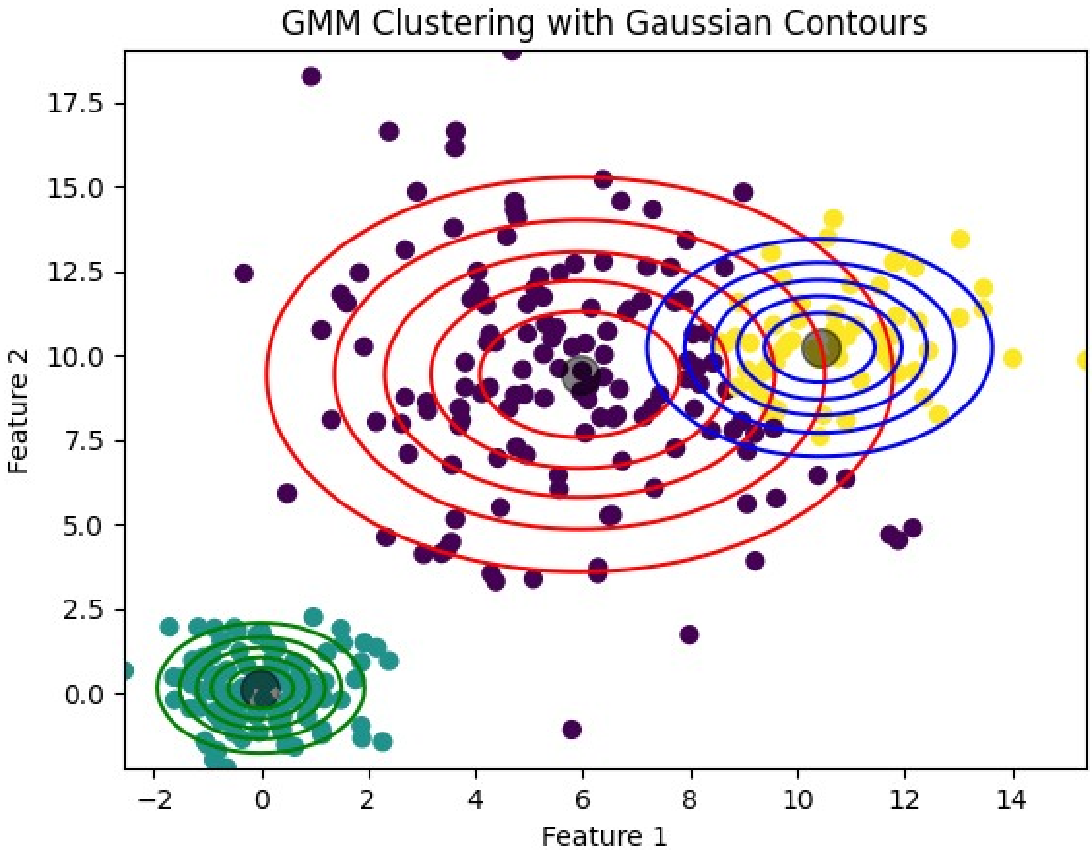
```
from scipy.stats import multivariate_normal

# Plotting the clusters
plt.scatter(X[:, 0], X[:, 1], c=labels, s=40, cmap='viridis')


# Define the grid for plotting
x = np.linspace(np.min(X[:, 0]), np.max(X[:, 0]), 100)
y = np.linspace(np.min(X[:, 1]), np.max(X[:, 1]), 100)
X, Y = np.meshgrid(x, y)
XX = np.array([X.ravel(), Y.ravel()]).T
```

```python
# Plotting the contour plot of each Gaussian component
colors = ['red', 'yellow', 'blue', 'black']
for i, (mean, covar, color) in enumerate(zip(gmm.means_,
gmm.covariances_, colors)):
    rv = multivariate_normal(mean, covar)
    Z = rv.pdf(XX)
    Z = Z.reshape(X.shape)
    plt.contour(X, Y, Z, levels=np.linspace(Z.min(), Z.max(), 7),
colors=color)

plt.title("GMM Clustering with Gaussian Contours")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```

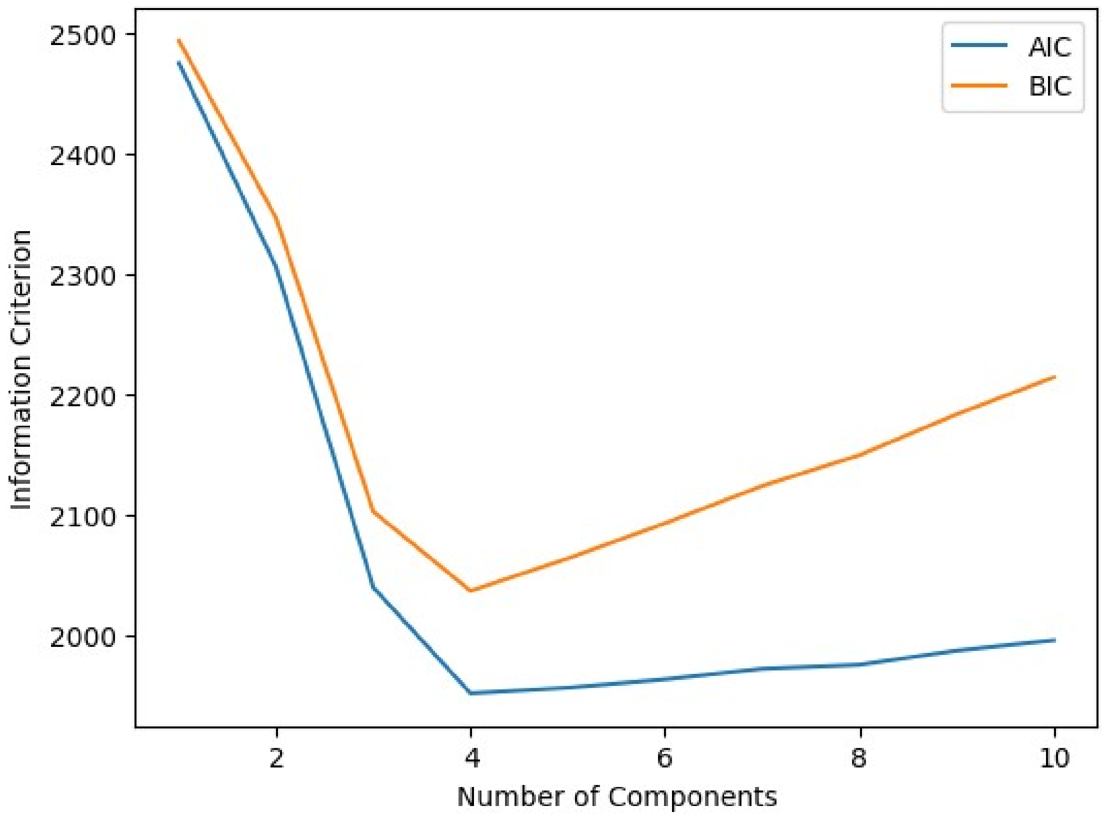



```python
# Generating differently shaped blobs
np.random.seed(0)
n_samples = 300

# First blob: circular
X1 = np.random.normal(0, 1, (n_samples // 3, 2))
```

```
# Second blob: elongated
X2 = np.random.normal(5, 2, (n_samples // 3, 2))
X2[:, 1] *= 2

# Third blob: larger circular
X3 = np.random.normal(10, 2, (n_samples // 3, 2))

# Combine the blobs to form the dataset
X = np.vstack([X1, X2, X3])

# Apply Gaussian Mixture Model clustering
gmm = GaussianMixture(n_components=3, random_state=0,
covariance_type='spherical').fit(X)
labels = gmm.predict(X)

# Plotting the clusters
plt.scatter(X[:, 0], X[:, 1], c=labels, s=40, cmap='viridis')

# Plot the center of each cluster
centers = gmm.means_
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5)

# Define the grid for plotting
x = np.linspace(np.min(X[:, 0]), np.max(X[:, 0]), 100)
y = np.linspace(np.min(X[:, 1]), np.max(X[:, 1]), 100)
X_grid, Y_grid = np.meshgrid(x, y)
XX = np.array([X_grid.ravel(), Y_grid.ravel()]).T

# Plotting the contour plot of each Gaussian component
colors = ['red', 'green', 'blue']
for i, (mean, covar, color) in enumerate(zip(gmm.means_,
gmm.covariances_, colors)):
    rv = multivariate_normal(mean, covar)
    Z = rv.pdf(XX)
    Z = Z.reshape(X_grid.shape)
    plt.contour(X_grid, Y_grid, Z, levels=np.linspace(Z.min(),
Z.max(), 7), colors=color)

plt.title("GMM Clustering with Gaussian Contours")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```



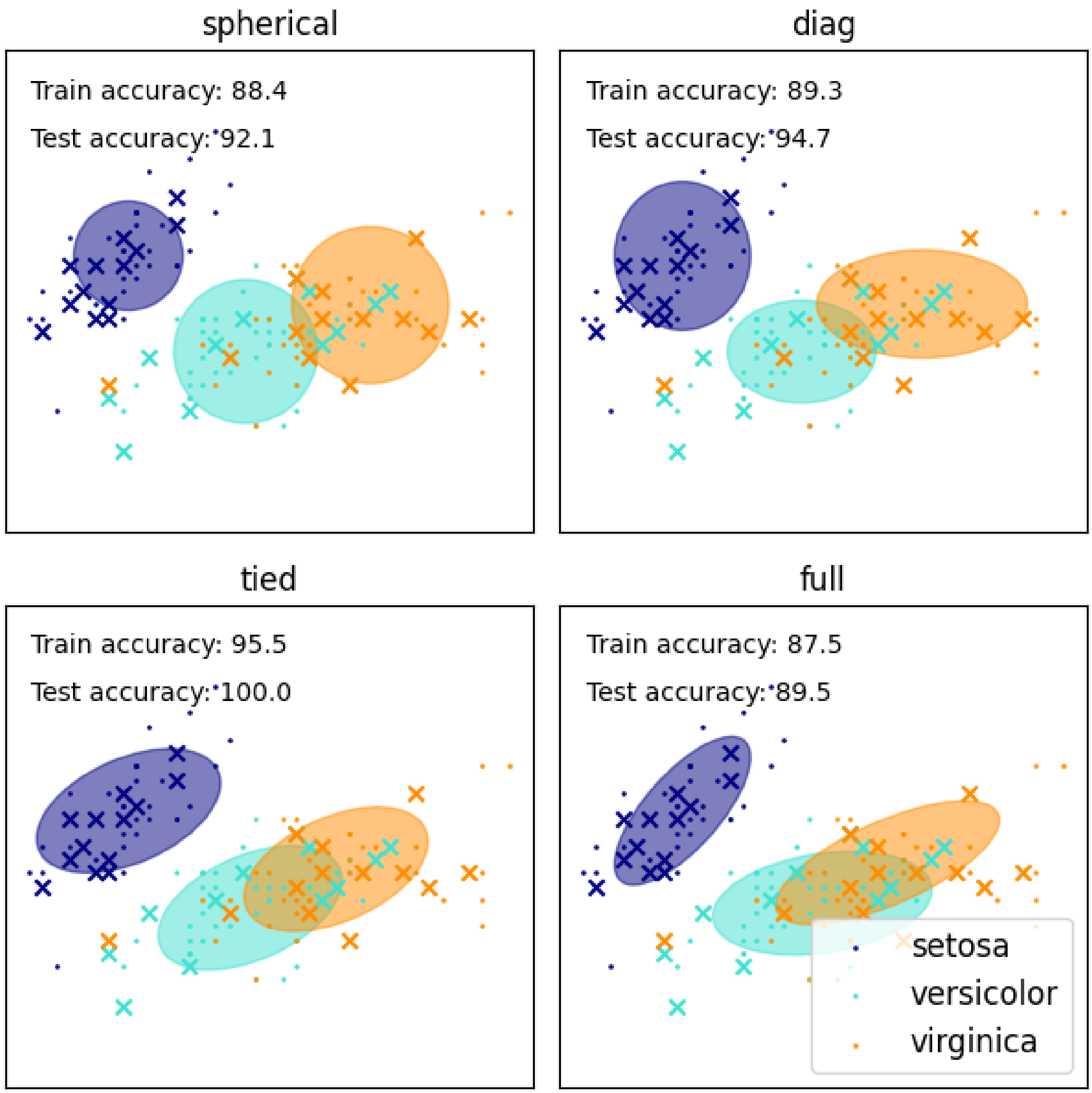

```
# Try GMMs with 1 to N components
N = 10
aics = []
bics = []
for i in range(1, N+1):
    gmm = GaussianMixture(n_components=i).fit(X)
    aics.append(gmm.aic(X))
    bics.append(gmm.bic(X))

# Plotting the AIC and BIC
plt.plot(range(1, N+1), aics, label='AIC')
plt.plot(range(1, N+1), bics, label='BIC')
plt.legend()
plt.xlabel('Number of Components')
plt.ylabel('Information Criterion')
plt.show()
```




```
# Author: Ron Weiss <ronweiss@gmail.com>, Gael Varoquaux
# Modified by Thierry Guillemot <thierry.guillemot.work@gmail.com>
# License: BSD 3 clause

import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np

from sklearn import datasets
from sklearn.mixture import GaussianMixture
from sklearn.model_selection import StratifiedKFold

colors = ["navy", "turquoise", "darkorange"]


def make_ellipses(gmm, ax):
    for n, color in enumerate(colors):
        if gmm.covariance_type == "full":
            covariances = gmm.covariances_[n][:2, :2]
        elif gmm.covariance_type == "tied":
            covariances = gmm.covariances_[:2, :2]
        elif gmm.covariance_type == "diag":
            covariances = np.diag(gmm.covariances_[n][:2])
        elif gmm.covariance_type == "spherical":
            covariances = np.eye(gmm.means_.shape[1]) *
```

```
gmm.covariances_[n]
        v, w = np.linalg.eigh(covariances)
        u = w[0] / np.linalg.norm(w[0])
        angle = np.arctan2(u[1], u[0])
        angle = 180 * angle / np.pi  # convert to degrees
        v = 2.0 * np.sqrt(2.0) * np.sqrt(v)
        ell = mpl.patches.Ellipse(
            gmm.means_[n, :2], v[0], v[1], angle=180 + angle,
color=color
        )
        ell.set_clip_box(ax.bbox)
        ell.set_alpha(0.5)
        ax.add_artist(ell)
        ax.set_aspect("equal", "datalim")


iris = datasets.load_iris()

# Break up the dataset into non-overlapping training (75%) and testing
# (25%) sets.
skf = StratifiedKFold(n_splits=4)
# Only take the first fold.
train_index, test_index = next(iter(skf.split(iris.data,
iris.target)))


X_train = iris.data[train_index]
y_train = iris.target[train_index]
X_test = iris.data[test_index]
y_test = iris.target[test_index]

n_classes = len(np.unique(y_train))

# Try GMMs using different types of covariances.
estimators = {
    cov_type: GaussianMixture(
        n_components=n_classes, covariance_type=cov_type, max_iter=20,
random_state=0
    )
    for cov_type in ["spherical", "diag", "tied", "full"]
}

n_estimators = len(estimators)

plt.figure(figsize=(3 * n_estimators // 2, 6))
plt.subplots_adjust(
    bottom=0.01, top=0.95, hspace=0.15, wspace=0.05, left=0.01,
right=0.99
)
```

```python
for index, (name, estimator) in enumerate(estimators.items()):
    # Since we have class labels for the training data, we can
    # initialize the GMM parameters in a supervised manner.
    estimator.means_init = np.array(
        [X_train[y_train == i].mean(axis=0) for i in range(n_classes)]
    )

    # Train the other parameters using the EM algorithm.
    estimator.fit(X_train)

    h = plt.subplot(2, n_estimators // 2, index + 1)
    make_ellipses(estimator, h)

    for n, color in enumerate(colors):
        data = iris.data[iris.target == n]
        plt.scatter(
            data[:, 0], data[:, 1], s=0.8, color=color,
label=iris.target_names[n]
        )
    # Plot the test data with crosses
    for n, color in enumerate(colors):
        data = X_test[y_test == n]
        plt.scatter(data[:, 0], data[:, 1], marker="x", color=color)

    y_train_pred = estimator.predict(X_train)
    train_accuracy = np.mean(y_train_pred.ravel() == y_train.ravel())
* 100
    plt.text(0.05, 0.9, "Train accuracy: %.1f" % train_accuracy,
transform=h.transAxes)

    y_test_pred = estimator.predict(X_test)
    test_accuracy = np.mean(y_test_pred.ravel() == y_test.ravel()) *
100
    plt.text(0.05, 0.8, "Test accuracy: %.1f" % test_accuracy,
transform=h.transAxes)

    plt.xticks(())
    plt.yticks(())
    plt.title(name)

plt.legend(scatterpoints=1, loc="lower right", prop=dict(size=12))


plt.show()
```


spherical
Train accuracy: 88.4
Test accuracy: 92.1
diag
Train accuracy: 89.3
Test accuracy: 94.7
tied
Train accuracy: 95.5
Test accuracy: 100.0
full
Train accuracy: 87.5
Test accuracy: 89.5
setosa
versicolor
virginica