GMM

April 13, 2020

Fit Gaussian Mixture Model on Iris Dataset

[1]:
```
# Import Libraries
import numpy as np
import pandas as pd
from sklearn import datasets
import matplotlib.pyplot as plt
```

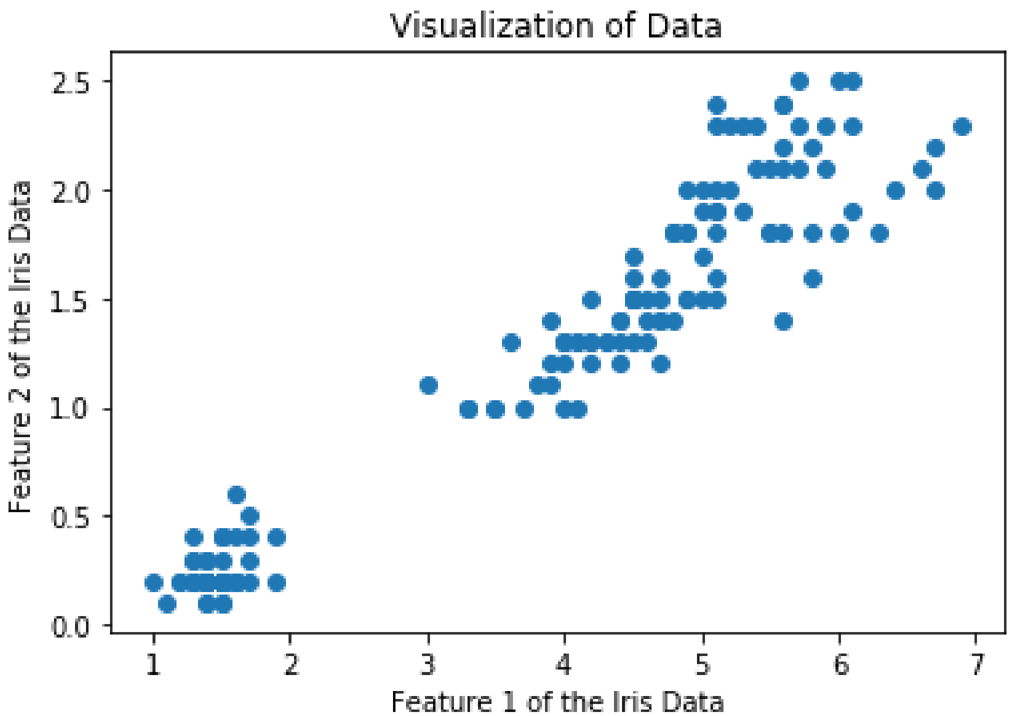
[2]:
```
# Import Iris dataset
iris = datasets.load_iris()
X_train = iris.data[:, 2:4] # We are considering only two features
X_train[:20]
```

[2]:
```
array([[1.4, 0.2],
       [1.4, 0.2],
       [1.3, 0.2],
       [1.5, 0.2],
       [1.4, 0.2],
       [1.7, 0.4],
       [1.4, 0.3],
       [1.5, 0.2],
       [1.4, 0.2],
       [1.5, 0.1],
       [1.5, 0.2],
       [1.6, 0.2],
       [1.4, 0.1],
       [1.1, 0.1],
       [1.2, 0.2],
       [1.5, 0.4],
       [1.3, 0.4],
       [1.4, 0.3],
       [1.7, 0.3],
       [1.5, 0.3]])
```

[3]:
```
# Visualize the data
plt.scatter(X_train[:,0], X_train[:,1])
plt.xlabel('Feature 1 of the Iris Data')
plt.ylabel('Feature 2 of the Iris Data')
```

```
plt.title('Visualization of Data')
```

[3]: Text(0.5, 1.0, 'Visualization of Data')



0.0.1 Standardizing the dataset

[4]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
```

0.1 Using Expectation Maximization (EM) Approach For Fitting GMM

0.2 Step 1 (Initialization step)

This is the initialization step of the GMM. At this point, we must initialise our parameters π_k, μ_k, and Σ_k.

Here, we are going to use the results of KMeans as an initial value for μ_k, set π_k to one over the number of clusters and Σ_k to the identity matrix.

We could also use random numbers for everything, but using a sensible initialisation procedure will help the algorithm achieve better results.

Before initialisation let's implement the Gaussian density function. As beginners we should try (atleast once) to write a mathematical functions rather than using sklearn functions. The function is:

$$p(\mathbf{x}|\mu, \ ) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^T \ ^{-1}(\mathbf{x}-\mu)\right)$$

[5]:
```
def gaussian(X, mu, cov):
    n = X.shape[1]
    diff = (X - mu).T
    return np.diagonal(1 / ((2 * np.pi) ** (n / 2) * np.linalg.det(cov) ** 0.5) * np.exp(-0.5 * np.dot(np.dot(diff.T, np.linalg.inv(cov)), diff))).reshape(-1,1)
```

[6]:
```
from sklearn.cluster import KMeans
def initialize_clusters(X, n_clusters):
    clusters = []
    idx = np.arange(X.shape[0])

    # We use the KMeans centroids to initialise the GMM

    kmeans = KMeans().fit(X)
    mu_k = kmeans.cluster_centers_

    for i in range(n_clusters):
        clusters.append({
            'pi_k': 1.0 / n_clusters,
            'mu_k': mu_k[i],
            'cov_k': np.identity(X.shape[1], dtype=np.float64)
        })

    return clusters
```

0.3 Step 2 (Expectation step)

We should now calculate $\gamma(z_{nk})$. We can achieve this by means of the following expression:

$$\gamma(z_{nk}) = \frac{\pi_k \mathcal{N}(\mathbf{x}_n|\mu_k, \ _k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x}_n|\mu_j, \ _j)}$$

For convenience, we just calculate the denominator as a sum over all terms in the numerator, and then assign it to a variable named totals.

[7]:
```
from scipy.stats import multivariate_normal
def expectation_step(X, clusters):
    totals = np.zeros((X.shape[0], 1), dtype=np.float64)
```

```
    for cluster in clusters:
        pi_k = cluster['pi_k']
        mu_k = cluster['mu_k']
        cov_k = cluster['cov_k']

        gamma_nk = (pi_k * gaussian(X, mu_k, cov_k)).astype(np.float64)

        for i in range(X.shape[0]):
            totals[i] += gamma_nk[i]

        cluster['gamma_nk'] = gamma_nk
        cluster['totals'] = totals


    for cluster in clusters:
        cluster['gamma_nk'] /= cluster['totals']
```

0.4 Step 3 (Maximization step):

Let us now implement the maximization step. Since $\gamma(z_{nk})$ is common to the expressions for π_k, μ_k and Σ_k, we can simply define:

$$N_k = \sum_{n=1}^{N} \gamma(z_{nk})$$

And then we can calculate the revised parameters by using:

$$\pi_k^* = \frac{N_k}{N}$$

$$\mu_k^* = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) \mathbf{x}_n$$

$$\Sigma_k^* = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) (\mathbf{x}_n - \mu_k)(\mathbf{x}_n - \mu_k)^T$$

Note: To calculate the covariance, we define an auxiliary variable diff that contains $(x_n - \mu_k)^T$.

[8]:
```
def maximization_step(X, clusters):
    N = float(X.shape[0])

    for cluster in clusters:
        gamma_nk = cluster['gamma_nk']
        cov_k = np.zeros((X.shape[1], X.shape[1]))

        N_k = np.sum(gamma_nk, axis=0)
```

```
        pi_k = N_k / N
        mu_k = np.sum(gamma_nk * X, axis=0) / N_k

        for j in range(X.shape[0]):
            diff = (X[j] - mu_k).reshape(-1, 1)
            cov_k += gamma_nk[j] * np.dot(diff, diff.T)

        cov_k /= N_k

        cluster['pi_k'] = pi_k
        cluster['mu_k'] = mu_k
        cluster['cov_k'] = cov_k
```

Let us now determine the log-likelihood of the model. It is given by:

$$\ln p(\mathbf{X}) = \sum_{n=1}^{N} \ln \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_n | \mu_k, \Sigma_k)$$

However, the second summation has already been calculated in the expectation_step function and is available in the totals variable. So we just make use of it.

[9]:
```
def get_likelihood(X, clusters):
    likelihood = []
    sample_likelihoods = np.log(np.array([cluster['totals'] for cluster in␣
 ↪clusters]))
    return np.sum(sample_likelihoods), sample_likelihoods
```

Finally, let's put everything together!

First, we are going to initialise the parameters by using the initialise_clusters function, and then perform several expectation-maximization steps. In this case, we set the number of iterations of the training procedure to a fixed n_epochs number.

It been done on purpose to generate graphs of the log-likelihood later.

[10]:
```
def train_gmm(X, n_clusters, n_epochs):
    clusters = initialize_clusters(X, n_clusters)
    likelihoods = np.zeros((n_epochs, ))
    scores = np.zeros((X.shape[0], n_clusters))
    history = []

    for i in range(n_epochs):
        clusters_snapshot = []

        # This is for our later use in the graphs
        for cluster in clusters:
            clusters_snapshot.append({
```

```
                'mu_k': cluster['mu_k'].copy(),
                'cov_k': cluster['cov_k'].copy()
            })

        history.append(clusters_snapshot)

        expectation_step(X, clusters)
        maximization_step(X, clusters)

        likelihood, sample_likelihoods = get_likelihood(X, clusters)
        likelihoods[i] = likelihood

        print('Epoch: ', i + 1, 'Likelihood: ', likelihood)

    for i, cluster in enumerate(clusters):
        scores[:, i] = np.log(cluster['gamma_nk']).reshape(-1)

    return clusters, likelihoods, scores, sample_likelihoods, history
```

0.5 Let's train our model!

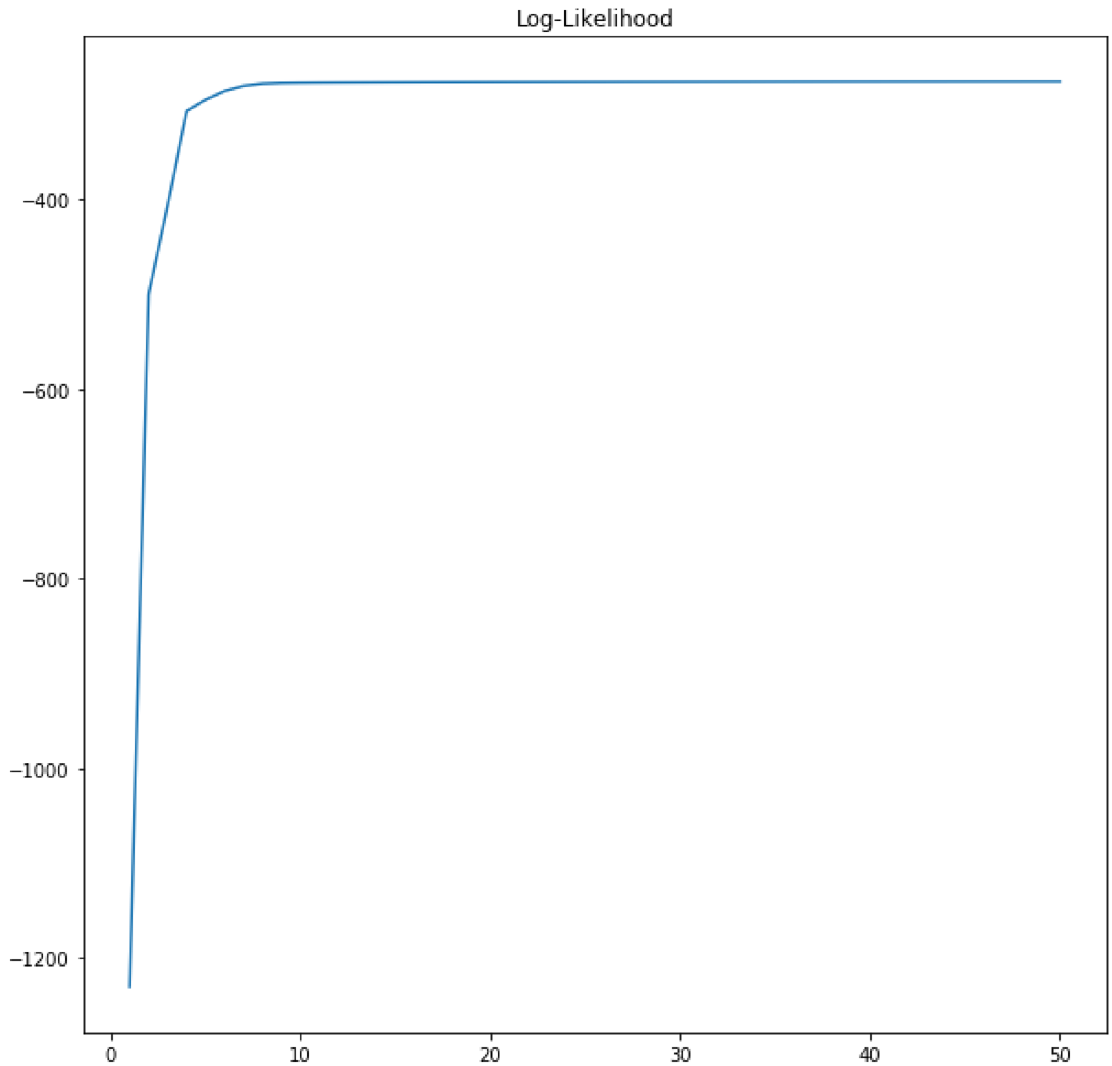
[11]:
```
n_clusters = 3
n_epochs = 50

clusters, likelihoods, scores, sample_likelihoods, history = train_gmm(X_train,␣
↪n_clusters, n_epochs)
```

```
Epoch:  1 Likelihood:  -1230.574371293787
Epoch:  2 Likelihood:  -499.897941232015
Epoch:  3 Likelihood:  -405.93922624851484
Epoch:  4 Likelihood:  -306.6394026082001
Epoch:  5 Likelihood:  -294.6828403674749
Epoch:  6 Likelihood:  -285.5145149214625
Epoch:  7 Likelihood:  -280.1000359037016
Epoch:  8 Likelihood:  -277.72664894555817
Epoch:  9 Likelihood:  -276.9778954089621
Epoch:  10 Likelihood:  -276.72815814989565
Epoch:  11 Likelihood:  -276.59117735809286
Epoch:  12 Likelihood:  -276.4846843700984
Epoch:  13 Likelihood:  -276.3928771839544
Epoch:  14 Likelihood:  -276.31156486948214
Epoch:  15 Likelihood:  -276.2387313750017
Epoch:  16 Likelihood:  -276.1729751862423
Epoch:  17 Likelihood:  -276.1132024945687
Epoch:  18 Likelihood:  -276.05853275588305
Epoch:  19 Likelihood:  -276.0082497561672
```

```
Epoch:  20 Likelihood:  -275.96176807424933
Epoch:  21 Likelihood:  -275.91860781099456
Epoch:  22 Likelihood:  -275.8783749439176
Epoch:  23 Likelihood:  -275.84074583711566
Epoch:  24 Likelihood:  -275.80545489974696
Epoch:  25 Likelihood:  -275.77228464797736
Epoch:  26 Likelihood:  -275.74105760337557
Epoch:  27 Likelihood:  -275.7116295918982
Epoch:  28 Likelihood:  -275.68388410794284
Epoch:  29 Likelihood:  -275.6577274867741
Epoch:  30 Likelihood:  -275.6330846920744
Epoch:  31 Likelihood:  -275.6098955775991
Epoch:  32 Likelihood:  -275.5881115256701
Epoch:  33 Likelihood:  -275.56769240211173
Epoch:  34 Likelihood:  -275.5486037979185
Epoch:  35 Likelihood:  -275.5308145524692
Epoch:  36 Likelihood:  -275.51429457110294
Epoch:  37 Likelihood:  -275.4990129608642
Epoch:  38 Likelihood:  -275.48493651195406
Epoch:  39 Likelihood:  -275.47202854901786
Epoch:  40 Likelihood:  -275.4602481666742
Epoch:  41 Likelihood:  -275.4495498490986
Epoch:  42 Likelihood:  -275.43988345602304
Epoch:  43 Likelihood:  -275.43119453956723
Epoch:  44 Likelihood:  -275.4234249402154
Epoch:  45 Likelihood:  -275.41651359800517
Epoch:  46 Likelihood:  -275.41039750797773
Epoch:  47 Likelihood:  -275.40501274772413
Epoch:  48 Likelihood:  -275.4002955092051
Epoch:  49 Likelihood:  -275.39618307600983
Epoch:  50 Likelihood:  -275.39261469941187
```

So let's create a graph reflecting that value of the log-likelihood.

[12]:
```
plt.figure(figsize=(10, 10))
plt.title('Log-Likelihood')
plt.plot(np.arange(1, n_epochs + 1), likelihoods)
plt.show()
```



0.6 Let's now test if our calculations are correct.

0.7 In this case, we are using sklearn's GMM implementation to check for the parameters and probabilities.

[13]:
```
from sklearn.mixture import GaussianMixture

gmm = GaussianMixture(n_components=n_clusters, max_iter=20).fit(X_train)
gmm_scores = gmm.score_samples(X_train)

print('Means by sklearn:\n', gmm.means_)
print('Means by our implementation:\n', np.array([cluster['mu_k'].tolist() for␣
 ↪cluster in clusters]))
```

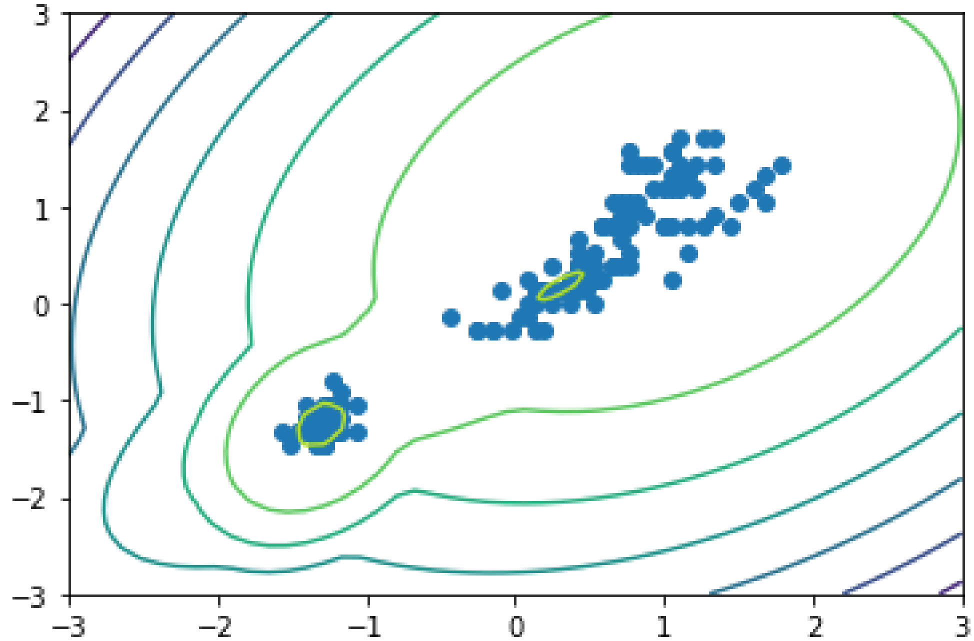
```python
print('Scores by sklearn:\n', gmm_scores[0:20])
print('Scores by our implementation:\n', sample_likelihoods.reshape(-1)[0:20])
```

```
Means by sklearn:
 [[ 1.0174453   1.09482233]
 [-1.30498753 -1.25489382]
 [ 0.29899444  0.17473848]]
Means by our implementation:
 [[ 1.02553951  1.10274582]
 [-1.30498758 -1.25489383]
 [ 0.30538906  0.1852003 ]]
Scores by sklearn:
 [ 1.30975161  1.30975161  0.97958436  1.25981632  1.30975161 -0.10648374
  1.1396797   1.25981632  1.30975161  0.18993696  1.25981632  0.82977848
  0.44758259 -1.06009582  0.26931455  0.30285231 -0.80822106  1.1396797
  0.47269736  1.29745475]
Scores by our implementation:
 [ 1.30983399  1.30983399  0.97963059  1.25988718  1.30983399 -0.10648392
  1.13973655  1.25988718  1.30983399  0.18992263  1.25988718  0.82979016
  0.44761644 -1.06020343  0.26927698  0.30287134 -0.80837329  1.13973655
  0.4726867   1.29753674]
```

Perfect !

0.8 Visualization of Gaussian Density Contours

[14]:
```python
X, Y = np.meshgrid(np.linspace(-3, 3), np.linspace(-3,3))
XX = np.array([X.ravel(), Y.ravel()]).T
Z = gmm.score_samples(XX)
Z = Z.reshape(X.shape)

plt.contour(X, Y, Z)
plt.scatter(X_train[:, 0], X_train[:, 1])

plt.show()
```



Although we are expecting three separate clusters, two separate clusters are clearly visible.