

Introduction to Machine Learning: Homework IV

Due on Dec 7th, 2022 at 11:59pm

Professor Ziping Zhao

Bingnan Li
2020533092

1. [Clustering and Mixture Models]

(a) K-means algorithm.

Solution:

- i. Initialize K cluster centers m_i by randomly selecting K input data points.
- ii. Repeat the following procedure until convergence:
 - A. For all $x^{(l)} \in \mathcal{X}$, we obtain the estimated labels

$$b_i^{(l)} = \begin{cases} 1, & \text{if } i = \arg \min_j \|x^{(l)} - m_j\| \\ 0, & \text{elsewhere} \end{cases}$$

- B. For all m_i , we obtain

$$m_i = \frac{\sum_l b_i^{(l)} x^{(l)}}{\sum_l b_i^{(l)}}$$

(b) Cluster the samples into 2 clusters.

Solution:

First, we select $m_1 = (0, 0)$ and $m_2 = (5, 0)$ as initialized cluster center. Then for the first iteration, we have the following result:

$$\begin{aligned} b_1^{(1)} &= 1 & b_2^{(1)} &= 0 \\ b_1^{(2)} &= 1 & b_2^{(2)} &= 0 \\ b_1^{(3)} &= 1 & b_2^{(3)} &= 0 \\ b_1^{(4)} &= 0 & b_2^{(4)} &= 1 \\ b_1^{(5)} &= 0 & b_2^{(5)} &= 1 \\ m_1 &= \frac{(0, 2) + (0, 0) + (1, 0)}{3} = \left(\frac{1}{3}, \frac{2}{3}\right) \\ m_2 &= \frac{(5, 0) + (5, 2)}{2} = (5, 1) \end{aligned}$$

Next, for the second iteration, we find that

$$\begin{aligned} b_1^{(1)} &= 1 & b_2^{(1)} &= 0 \\ b_1^{(2)} &= 1 & b_2^{(2)} &= 0 \\ b_1^{(3)} &= 1 & b_2^{(3)} &= 0 \\ b_1^{(4)} &= 0 & b_2^{(4)} &= 1 \\ b_1^{(5)} &= 0 & b_2^{(5)} &= 1 \\ m_1 &= \frac{(0, 2) + (0, 0) + (1, 0)}{3} = \left(\frac{1}{3}, \frac{2}{3}\right) \\ m_2 &= \frac{(5, 0) + (5, 2)}{2} = (5, 1) \end{aligned}$$

The result converged, so we terminated the algorithm and cluster centers are

$$m_1 = \left(\frac{1}{3}, \frac{2}{3}\right) \quad m_2 = (5, 1)$$

2. [Clustering and Mixture Models]

- (a) Advantages of GMM and Why it can be used for clustering.

Solution:

Advantages: GMM is a kind of "soft-label" method, the projected data do not represent deterministic classification label but the probability of belonging to any classes.

Why it can be used for clustering: K-means is a special case of GMM. In practice, the higher the $h_i^{(l)}$ is, the more likely that $x^{(l)}$ is generated by component \mathcal{G}_i , which can be interpreted as $x^{(l)}$ belongs to cluster i .

- (b) Estimate the parameters of the GMM.

Solution:

Define $\mathcal{Q}(\phi|\phi^t)$ as following

$$\begin{aligned}\mathcal{Q}(\phi|\phi^t) &= \mathbb{E}[\mathcal{L}_C(\phi|\mathcal{X}, \mathcal{Z})|\mathcal{X}, \phi^t] \\ &\text{where} \\ \mathcal{L}_C(\phi) &= \log \prod_l p(x^{(l)}, z^{(l)}|\phi) \\ &= \sum_l \left[\log P(z^{(l)}|\phi) + \log p(x^{(l)}|z^{(l)}, \phi) \right] \\ &= \sum_l \sum_i z_i^{(l)} [\log \pi_i + \log p_i(x^{(l)}|\phi)]\end{aligned}$$

Hence

$$\begin{aligned}\mathcal{Q}(\phi|\phi^t) &= \mathbb{E}[\mathcal{L}_C(\phi|\mathcal{X}, \mathcal{Z})|\mathcal{X}, \phi^t] \\ &= \sum_l \sum_i \mathbb{E}[z_i^{(l)}|\mathcal{X}, \phi^t] [\log \pi_i + \log p_i(x^{(l)}|\phi)]\end{aligned}$$

where

$$\begin{aligned}\mathbb{E}[z_i^{(l)}|\mathcal{X}, \phi^t] &= \mathbb{E}[z_i^{(l)}|\mathbf{x}^{(l)}, \phi] \\ &= P(z_i^{(l)} = 1|\mathbf{x}^{(l)}, \phi^t) \\ &= \frac{p(\mathbf{x}^{(l)}|z_i^{(l)} = 1, \phi^t)P(z_i^{(l)} = 1|\phi^t)}{p(\mathbf{x}^{(l)}|\phi^t)} \\ &= \frac{p_i(\mathbf{x}^{(l)}|\phi^t)\pi_i}{\sum_j p_j(\mathbf{x}^{(l)}|\phi^t)\pi_j} \\ &= \frac{P(x^{(l)}|\mathcal{G}_i, \phi^t)\pi_i}{\sum_j P(x^{(l)}|\mathcal{G}_j, \phi^t)\pi_j} \\ &\equiv h_i^{(l)}\end{aligned}$$

Therefore, we have

$$\begin{aligned}h_i^{(l)} &= \frac{P(x^{(l)}|\mathcal{G}_i, \phi^t)\pi_i}{\sum_j P(x^{(l)}|\mathcal{G}_j, \phi^t)\pi_j} \\ &= \frac{|\Sigma_i|^{-\frac{1}{2}} \exp[-\frac{1}{2}(\mathbf{x}_l - \boldsymbol{\mu}_i)^T (\Sigma_i)^{-1} (\mathbf{x}_l - \boldsymbol{\mu}_i)]\pi_i}{\sum_{j=1}^K |\Sigma_j|^{-\frac{1}{2}} \exp[-\frac{1}{2}(\mathbf{x}_l - \boldsymbol{\mu}_j)^T (\Sigma_j)^{-1} (\mathbf{x}_l - \boldsymbol{\mu}_j)]\pi_j} \\ &= \frac{\mathcal{N}(\mathbf{x}_l|\boldsymbol{\mu}_i, \Sigma_i)\pi_i}{\sum_{j=1}^K \mathcal{N}(\mathbf{x}_l|\boldsymbol{\mu}_j, \Sigma_j)\pi_j}\end{aligned}$$

and

$$\mathcal{Q}(\phi|\phi^t) = \sum_l \sum_i h_i^{(l)} [\log \pi_i + \log \mathcal{N}(\mathbf{x}_l | \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)]$$

Then, maximization of $\mathcal{Q}(\phi|\phi^t)$ is equivalent to

$$\begin{aligned} & \underset{\{\pi_i\}, \{\boldsymbol{\mu}_i\}, \{\boldsymbol{\Sigma}_i\}}{\text{maximize}} & \mathcal{Q}(\phi|\phi^t) &= \sum_l \sum_i h_i^{(l)} \log \pi_i + h_i^{(l)} \log \mathcal{N}(\mathbf{x}_l | \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) \\ & \text{subject to} & \sum_i \pi_i &= 1 \end{aligned}$$

Since the second term does not depend on π_i , the problem for $\{\pi_i\}$ is

$$\begin{aligned} & \underset{\{\pi_i\}}{\text{maximize}} & \sum_l \sum_i h_i^{(l)} \log \pi_i \\ & \text{subject to} & \sum_i \pi_i = 1 \end{aligned}$$

By using Lagrangian, we solve for

$$\frac{\partial}{\partial \pi_i} \left[\sum_l \sum_i h_i^{(l)} \log \pi_i - \lambda \left(\sum_i \pi_i - 1 \right) \right] = 0$$

And we get

$$\pi_i = \frac{\sum_l h_i^{(l)}}{N}$$

Then the first term of \mathcal{Q} does not depend on $\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i$. Hence, the problem for $\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i$ is

$$\underset{\{\boldsymbol{\mu}_i\}, \{\boldsymbol{\Sigma}_i\}}{\text{maximize}} \quad \sum_l \sum_i h_i^{(l)} \log \mathcal{N}(\mathbf{x}_l | \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$$

By solving

$$\frac{\partial}{\partial \boldsymbol{\mu}_i} \left[\sum_l \sum_i h_i^{(l)} \log \mathcal{N}(\mathbf{x}_l | \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) \right] = 0$$

and

$$\frac{\partial}{\partial \boldsymbol{\Sigma}_i} \left[\sum_l \sum_i h_i^{(l)} \log \mathcal{N}(\mathbf{x}_l | \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) \right] = 0$$

we get

$$\boldsymbol{\mu}_i^{t+1} = \frac{\sum_l h_i^{(l)} \mathbf{x}_l}{\sum_l h_i^{(l)}}$$

and

$$\boldsymbol{\Sigma}_i^{t+1} = \frac{\sum_l h_i^{(l)} (\mathbf{x}_l - \boldsymbol{\mu}_i^{t+1})(\mathbf{x}_l - \boldsymbol{\mu}_i^{t+1})^T}{\sum_l h_i^{(l)}}$$

3. [Nonparametric Density Estimation]

(a) Expression of $\hat{p}(x)$.

Proof:

By definition, the histogram estimator is defined as following:

$$\hat{p}(x) = \frac{\#\{x^{(l)} \text{ in the same bin as } x\}}{nh} = \frac{\#\{x^{(l)} \in [\lfloor \frac{x}{h} \rfloor h, \lceil \frac{x}{h} \rceil h)\}}{nh}$$

- (b) Expression of
- $L'(h)$
- based on the histogram estimator
- $\hat{p}(x)$
- .

Proof:First, for the first term of L' , we can split the integral by bins:

$$\begin{aligned}\int_0^1 \hat{p}^2(x) dx &= \sum_{j=0}^{\lfloor \frac{1}{h} \rfloor} \int_{jh}^{(j+1)h} \frac{\#^2 \{x^{(l)} \in [jh, (j+1)h)\}}{n^2 h^2} dx \\ &= \frac{\sum_{j=0}^{\lfloor \frac{1}{h} \rfloor} \#^2 \{x^{(l)} \in [jh, (j+1)h)\}}{n^2 h}\end{aligned}$$

For the second term of L' , we can rewrite it as following:

$$\begin{aligned}\frac{2}{n} \sum_{i=1}^n \hat{p}(x_i) &= \frac{2}{n} \sum_{i=1}^n \frac{\# \{x^{(l)} \in [\lfloor \frac{x}{h} \rfloor h, \lceil \frac{x}{h} \rceil h)\}}{nh} \\ &= \frac{2 \sum_{j=0}^{\lfloor \frac{1}{h} \rfloor} \#^2 \{x^{(l)} \in [jh, (j+1)h)\}}{n^2 h}\end{aligned}$$

Hence, we have

$$L'(h) = \int_0^1 \hat{p}^2(x) dx - \frac{2}{n} \sum_{i=1}^n \hat{p}(x_i) = - \frac{\sum_{j=0}^{\lfloor \frac{1}{h} \rfloor} \#^2 \{x^{(l)} \in [jh, (j+1)h)\}}{n^2 h}$$

- (c)
- h
- that minimizes
- $L'(h)$
- .

Proof:

Since as h goes to 0, as long as there are not coincide sample points, the numerator of $L'(h)$ will go to n . However, the denominator of $L'(h)$ is $n^2 h$, which goes to 0 as $h \rightarrow 0$, thus $\lim_{h \rightarrow 0} L'(h) = -\infty$.

$$h = \arg \min_{h>0} L'(h) = 0$$

4. [Nonparametric Regression]

- (a) Estimated output
- \hat{y}
- and is linear regression a linear smoother?

Solution:Given that the least squares estimate for \mathbf{w} is

$$\mathbf{w}^* = (H^T H)^{-1} H^T Y$$

we have the following estimated output \hat{y}

$$\begin{aligned}\hat{y} &= (\mathbf{w}^*)^T \cdot \mathbf{h}(\mathbf{x}) \\ &= \left[(H^T H)^{-1} H^T Y \right]^T \mathbf{h}(\mathbf{x}) \\ &= Y^T H (H^T H)^{-1} \mathbf{h}(\mathbf{x}) \\ &= \left[H (H^T H)^{-1} \mathbf{h}(\mathbf{x}) \right]^T Y \\ &= \mathbf{h}(\mathbf{x})^T (H^T H)^{-1} H^T Y \\ &\Rightarrow \\ \mathbf{l}(\mathbf{x}) &= H (H^T H)^{-1} \mathbf{h}(\mathbf{x})\end{aligned}$$

Hence, linear regression is a linear smoother.

- (b) In kernel regression, if we use kernel $K(x_i, x) = \exp\left\{-\frac{\|x_i - x\|^2}{2\sigma^2}\right\}$, given an input x , please derive the estimated output \hat{y} . Furthermore, is this kernel regression a linear smoother?

Solution:

By the definition of Kernel regression, we have the estimated output \hat{y} as following:

$$\begin{aligned}\hat{y} &= \frac{\sum_{i=1}^n K(x_i, x) y_i}{\sum_{i=1}^n K(x_i, x)} \\ &= \frac{\sum_{i=1}^n \exp\left\{-\frac{\|x_i - x\|^2}{2\sigma^2}\right\} y_i}{\sum_{i=1}^n \exp\left\{-\frac{\|x_i - x\|^2}{2\sigma^2}\right\}} \\ &= \sum_{i=1}^n \text{softmax}\left(\frac{\|x_i - x\|^2}{2\sigma^2}\right) y_i\end{aligned}$$

Then define that

$$S = \begin{bmatrix} \text{softmax}\left(\frac{\|x_1 - x\|^2}{2\sigma^2}\right) \\ \text{softmax}\left(\frac{\|x_2 - x\|^2}{2\sigma^2}\right) \\ \vdots \\ \text{softmax}\left(\frac{\|x_n - x\|^2}{2\sigma^2}\right) \end{bmatrix}$$

we have

$$\begin{aligned}\hat{y} &= \sum_{i=1}^n \text{softmax}\left(\frac{\|x_i - x\|^2}{2\sigma^2}\right) y_i \\ &= S^T Y \\ &\Rightarrow \\ l(x) = S &= \begin{bmatrix} \text{softmax}\left(\frac{\|x_1 - x\|^2}{2\sigma^2}\right) \\ \text{softmax}\left(\frac{\|x_2 - x\|^2}{2\sigma^2}\right) \\ \vdots \\ \text{softmax}\left(\frac{\|x_n - x\|^2}{2\sigma^2}\right) \end{bmatrix}\end{aligned}$$

Hence, this kernel regression is a linear smoother.

HW4-Coding

December 2, 2022

1 Fit GMM by EM algorithm

Please convert your coding notebook from .ipynb into .pdf and concatenate it with your writing part.

You need to implement the EM algorithm, using closed-forms derived in './HW4-Writing.pdf/2(b)'.

```
[63]: import matplotlib.pyplot as plt
      # Do NOT change this cell.
      import numpy as np
```

```
[64]: # E-step: compute posterior probabilities  $h_{li}$ 
def probabilities(data, weights, means, covariances):
    """
    :param data: size: (N, D)
    :param weights: corresponds to  $\pi$ , size: (K,)
    :param means: corresponds to  $\mu$ , size: (K, D)
    :param covariances: corresponds to  $\sigma$ , size: (K, D, D)
    :return prob_matrix: a matrix filled by posterior probability  $h_{ik}$ , size_
    ↪ (N, K)
    """
    num_data = len(data)
    num_clusters = len(means)
    prob_matrix = np.zeros((num_data, num_clusters))
    ##### START YOUR CODE HERE #####
    cov_inv = np.zeros_like(covariances)
    cov_det = np.zeros(num_clusters)
    for k in range(num_clusters):
        cov_inv[k] = np.linalg.inv(covariances[k])
        cov_det[k] = np.linalg.det(covariances[k])
    cov_det = np.power(cov_det, [-1/2])
    for l in range(num_data):
        exp_value = np.zeros(num_clusters)
        for i in range(num_clusters):
            DM = data[l] - means[i]
            exp_value[i] = -1 / 2 * (np.matrix.transpose(DM) @ cov_inv[i] @ DM)
        exp_value = np.exp(exp_value)
        denominator = np.matrix.transpose(cov_det * exp_value) @ weights
        numerator = cov_det * exp_value * weights
```

```

        prob_matrix[l] = numerator / denominator
    ##### END YOUR CODE HERE #####
    return prob_matrix

```

```

[65]: # M-step: update weights \pi, means \mu, and covariances \sigma
def updates(data, prob_matrix, weights, means, covariances):
    num_clusters = len(means)
    num_data = len(data)
    dim = data.shape[1]

    ##### START YOUR CODE HERE #####
    prob_sum = np.sum(prob_matrix, axis=0)
    weights = prob_sum / num_data
    for i in range(num_clusters):
        means[i] = np.matrix.transpose(prob_matrix[:, i]) @ data / prob_sum[i]
        numerator = np.zeros([dim, dim])
        for l in range(num_data):
            DM = np.expand_dims(data[l] - means[i], axis=1)
            numerator += prob_matrix[l, i] * (DM @ np.matrix.transpose(DM))
        covariances[i] = numerator / prob_sum[i]
    ##### END YOUR CODE HERE #####

    return weights, means, covariances

```

To help us develop and test our implementation, we will generate some observations from a mixture of Gaussians and then run our EM algorithm to discover the mixture components. We'll begin with a function to generate the data, and a quick plot to visualize its output for a 2-dimensional mixture of three Gaussians.

```

[66]: # Generate dataset

def generate_MoG_data(num_data, means, covariances, weights):
    """ Creates a list of data points """
    num_clusters = len(weights)
    data = []
    for i in range(num_data):
        # Use np.random.choice and weights to pick a cluster id greater than
        # or equal to 0 and less than num_clusters.
        k = np.random.choice(len(weights), 1, p=weights)[0]
        # Use np.random.multivariate_normal to create data from this cluster
        x = np.random.multivariate_normal(means[k], covariances[k])
        data.append(x)
    return data

# Model parameters
data_means = np.array([
    [5, 0], # mean of cluster 1

```



```

    [1, 1], # mean of cluster 2
    [0, 5] # mean of cluster 3
])
data_covariances = np.array([
    [[.5, 0.], [0, .5]], # covariance of cluster 1
    [[.92, .38], [.38, .91]], # covariance of cluster 2
    [[.5, 0.], [0, .5]] # covariance of cluster 3
])
data_weights = np.array([1 / 4., 1 / 2., 1 / 4.]) # weights of each cluster

np.random.seed(4)
data = np.array(generate_MoG_data(100, data_means, data_covariances,
    ↪data_weights))

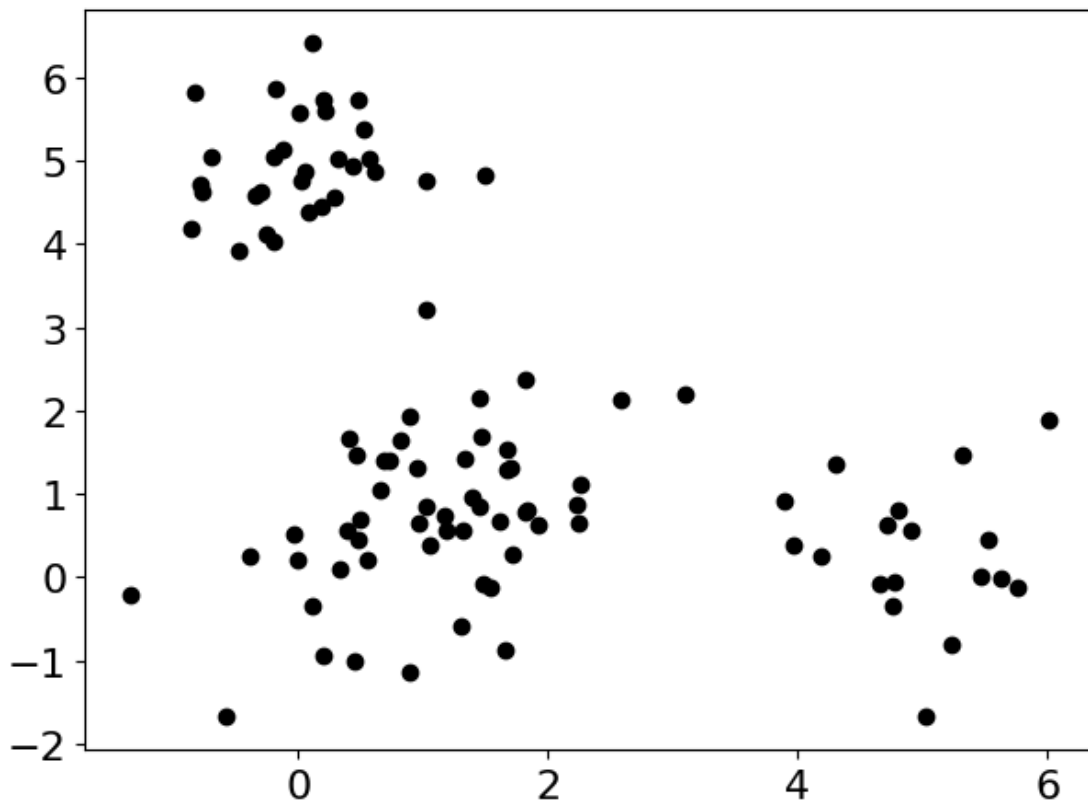
```

Now plot the data you created above. The plot should be a scatterplot with 100 points that appear to roughly fall into three clusters.

```

[67]: plt.figure()
      d = np.vstack(data)
      plt.plot(d[:, 0], d[:, 1], 'ko')
      plt.rcParams.update({'font.size': 16})
      plt.tight_layout()
      plt.show()

```



Now we'll fit a mixture of Gaussians to this data using our implementation of the EM algorithm. As with k-means, it is important to ask how we obtain an initial configuration of mixing weights and component parameters. In this simple case, we'll take three random points to be the initial cluster means, use the empirical covariance of the data to be the initial covariance in each cluster (a clear overestimate), and set the initial mixing weights to be uniform across clusters.

```
[68]: # Initialization of parameters
np.random.seed(4)
chosen = np.random.choice(len(data), 3, replace=False)
initial_means = np.array([data[x] for x in chosen])
initial_covariances = np.array([np.cov(data, rowvar=False)] * 3)
initial_weights = np.array([1 / 3., 1 / 3., 1 / 3.])
```

We will use the following `plot_contours()` function to visualize the Gaussian components over the data at three different points in the algorithm's execution:

1. At initialization (using `initial_means`, `initial_covariances`, and `initial_weights`)
2. After running the algorithm to completion
3. After 20 iterations

```
[69]: def bivariate_normal(X, Y, sigmax=1.0, sigmay=1.0,
                           mux=0.0, muy=0.0, sigmaxy=0.0):

    Xmu = X - mux
    Ymu = Y - muy
    rho = sigmaxy / (sigmax * sigmay)
    z = Xmu ** 2 / sigmax ** 2 + Ymu ** 2 / sigmay ** 2 - 2 * rho * Xmu * Ymu /
    → (sigmax * sigmay)
    denom = 2 * np.pi * sigmax * sigmay * np.sqrt(1 - rho ** 2)
    return np.exp(-z / (2 * (1 - rho ** 2))) / denom

def plot_contours(data, means, covs, title):
    plt.figure()
    plt.plot([x[0] for x in data], [y[1] for y in data], 'ko') # data

    delta = 0.025
    k = len(means)
    x = np.arange(-2.0, 7.0, delta)
    y = np.arange(-2.0, 7.0, delta)
    X, Y = np.meshgrid(x, y)
    col = ['green', 'red', 'indigo']
    for i in range(k):
        mean = means[i]
        cov = covs[i]
        sigmax = np.sqrt(cov[0][0])
        sigmay = np.sqrt(cov[1][1])
```

```

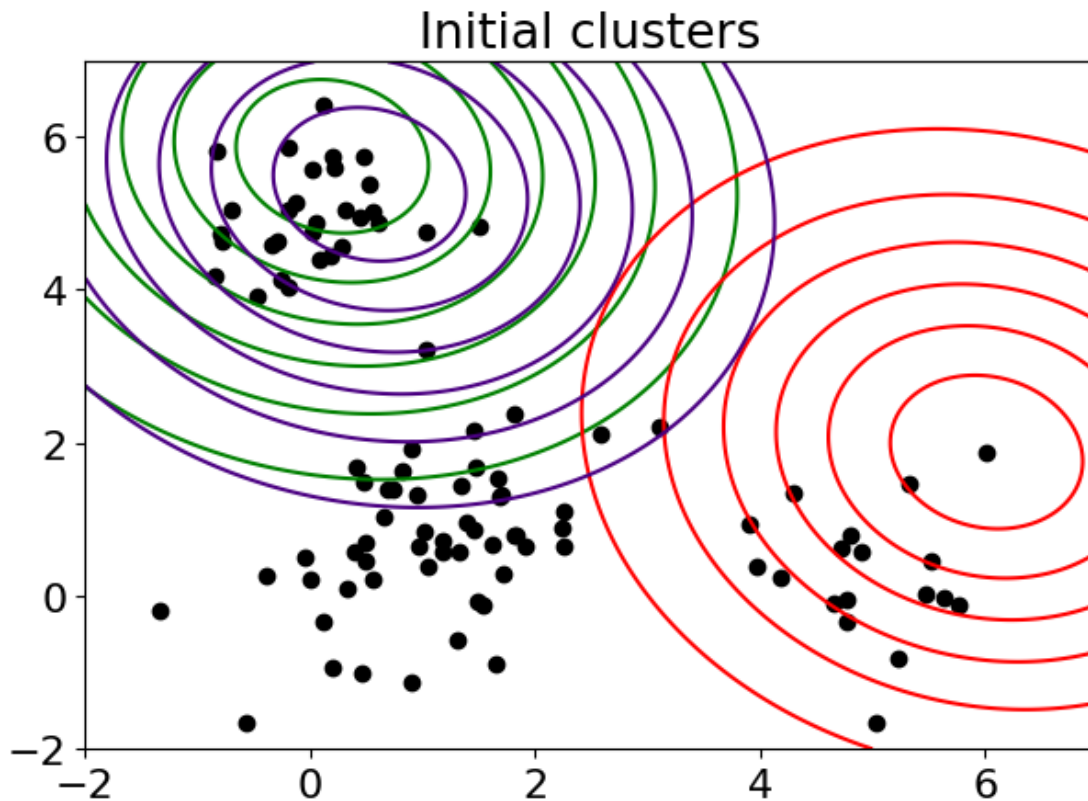
    sigmaxy = cov[0][1] / (sigmax * sigmay)
    Z = bivariate_normal(X, Y, sigmax, sigmay, mean[0], mean[1], sigmaxy)
    plt.contour(X, Y, Z, colors=col[i])
    plt.title(title)
plt.rcParams.update({'font.size': 16})
plt.tight_layout()
plt.show()

```

```

[70]: # Parameters after initialization
plot_contours(data, initial_means, initial_covariances, 'Initial clusters')

```



Now run the EM algorithm and plot contours afterwards.

```

[71]: # Run EM
my_means = initial_means
my_weights = initial_weights
my_covariances = initial_covariances
iters = 20
for iter in range(iters):
    # Here we run 20 iterations. You can use a more strict convergence
    ↪ criterion,

```

```

    # such as that the increment of the log-likelihood function is less than
    ↪ 1e-3
    prob_matrix = probabilities(data, my_weights, my_means, my_covariances)
    [my_weights, my_means, my_covariances] = updates(data, prob_matrix,
    ↪ my_weights, my_means, my_covariances)
    print("my_weights =\n{},\n{}\nmy_means =\n{},\n{}\nmy_covariances =\n{}".
    ↪ format(my_weights, my_means, my_covariances))

```

```

my_weights =
[0.30100743 0.17993525 0.51905733],

```

```

my_means =
[[0.02238127 4.94603136]
 [4.94240545 0.31364134]
 [1.08184868 0.73761235]],

```

```

my_covariances =
[[[ 0.29398303  0.04872587]
   [ 0.04872587  0.35540776]]

 [[ 0.35562857 -0.01493194]
  [-0.01493194  0.66694074]]

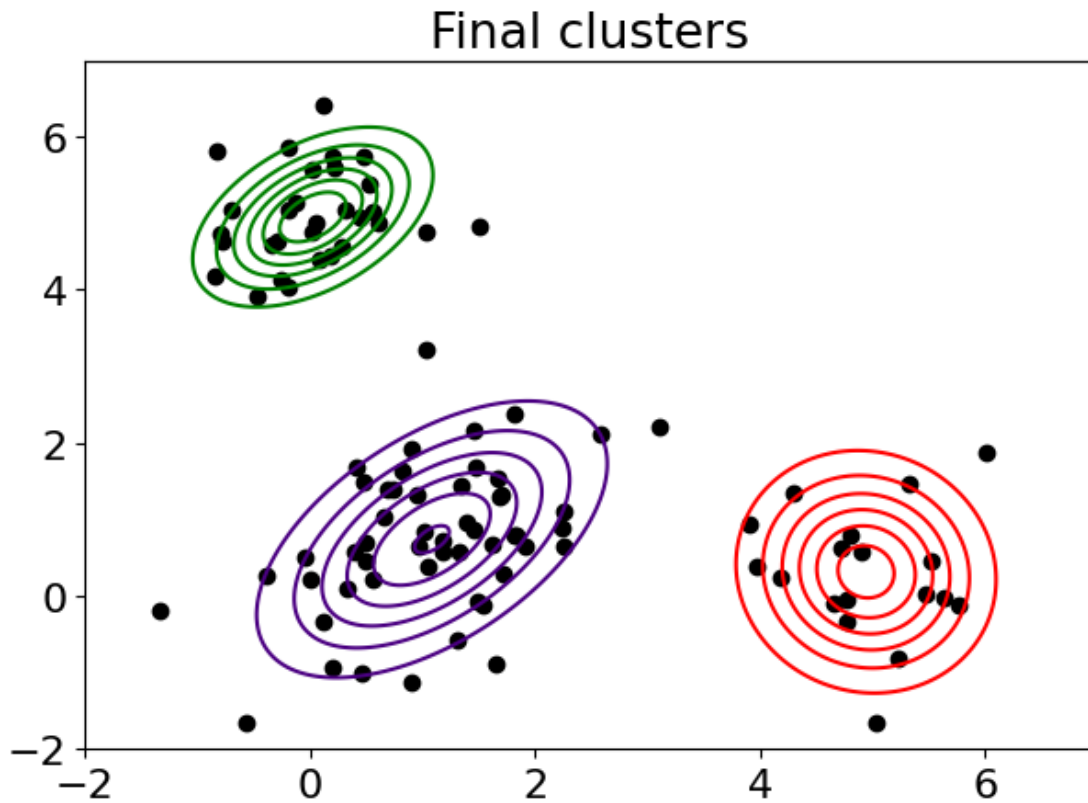
 [[ 0.67155363  0.3308503 ]
  [ 0.3308503  0.90125037]]]

```

```

[72]: # Parameters after running EM to convergence
      plot_contours(data, my_means, my_covariances, 'Final clusters')

```



Now call the `GaussianMixture()` method in `sklearn` package, and see its performance.

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

gmm_sklearn = GaussianMixture(n_components=3, covariance_type='full',
                               max_iter=20, weights_init=initial_weights,
                               means_init=initial_means).fit(data)
print("weights_sklearn =\n{0},\n\nmeans_sklearn =\n{0},\n\ncovariance_sklearn_\n{0}"
      "\n{0}".format(gmm_sklearn.weights_,
                      gmm_sklearn.means_,
                      gmm_sklearn.covariances_))
```

```
weights_sklearn =
[0.30089166 0.17993589 0.51917245],
```

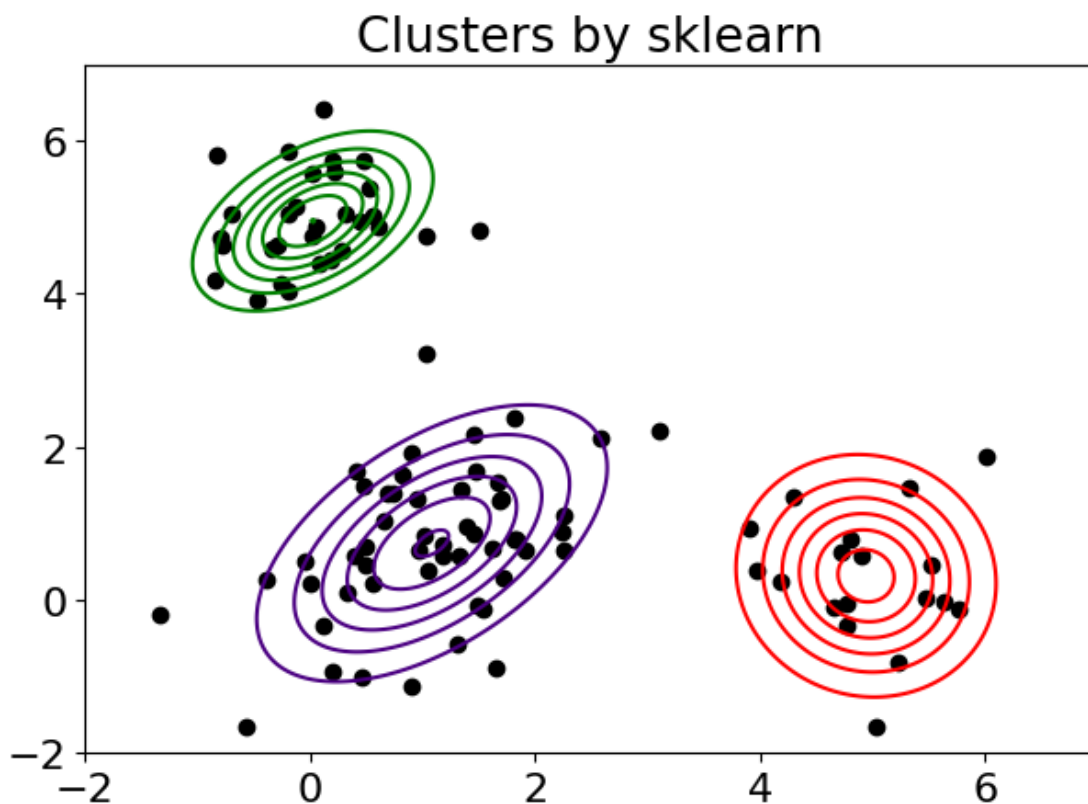
```
means_sklearn =
[[0.02199279 4.94669312]
 [4.9424009  0.31364545]
 [1.08183436 0.73816636]],
```

```
covariance_sklearn =
[[[ 0.29370389  0.04941099]
  [ 0.04941099  0.3543986 ]]

 [[ 0.35563482 -0.01493779]
 [-0.01493779  0.66694506]]

 [[ 0.67139816  0.33074842]
 [ 0.33074842  0.90243587]]]
```

```
[74]: # Parameters after running EM to convergence
plot_contours(data, gmm_sklearn.means_, gmm_sklearn.covariances_, 'Clusters by_
↳sklearn')
```



Now see the differences of parameters of your algorithm and that of sklearn. You should see the precision is at least **1e-3**.

```
[75]: print("After 20 iterations, differences between our algorithm and sklearn:\n")
print("delta_weights =\n{}",\n\ndelta_means =\n{}",\n\ndelta_covariances=\n{}".
↳format(my_weights - gmm_sklearn.weights_,

↳ my_means - gmm_sklearn.means_,
```

```
↪ my_covariances - gmm_sklearn.covariances_))
```

After 20 iterations, differences between our algorithm and sklearn:

```
delta_weights =  
[ 1.15770227e-04 -6.45837833e-07 -1.15124389e-04],
```

```
delta_means =  
[[ 3.88483410e-04 -6.61764743e-04]  
 [ 4.55441691e-06 -4.11036858e-06]  
 [ 1.43250214e-05 -5.54005410e-04]],
```

```
delta_covariances=  
[[[ 2.79146961e-04 -6.85120337e-04]  
 [-6.85120337e-04  1.00916157e-03]]
```

```
 [[-6.25209629e-06  5.85161700e-06]  
 [ 5.85161700e-06 -4.31164803e-06]]
```

```
 [[ 1.55477197e-04  1.01874776e-04]  
 [ 1.01874776e-04 -1.18549656e-03]]]
```