Machine Learning Homework 11

Hairui Yin

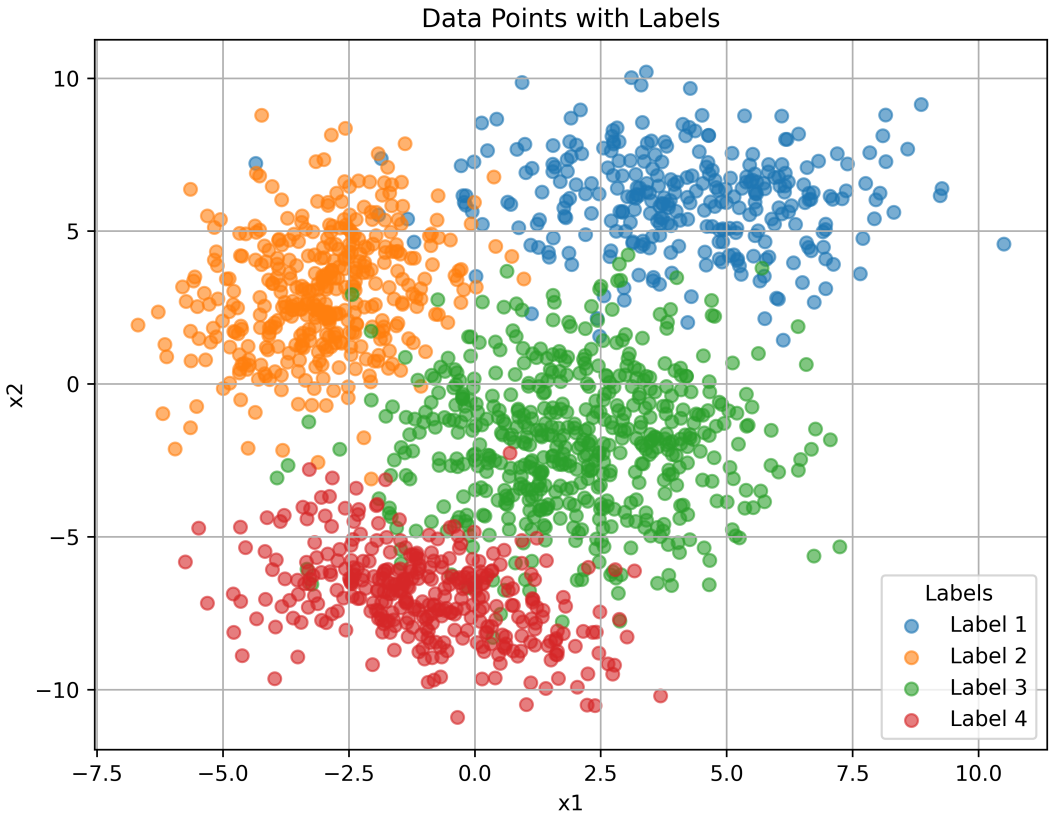
For the whole problem, code is uploaded in file code.ipynb.

1.

(a) We load the data and plot data points using different colors for each label with the following code:

```python
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

df = pd.read_csv('HW11-ClusteringData.csv', header=None, names=['x1', 'x2', 'label'])
plt.figure(figsize=(8, 6))
for label in df['label'].unique():
    subset = df[df['label'] == label]
    plt.scatter(subset['x1'], subset['x2'], label=f"Label {label}", alpha=0.6)
plt.title("Data Points with Labels")
plt.xlabel("x1")
plt.ylabel("x2")
plt.legend(title="Labels")
plt.grid(True)
# plt.show()
plt.savefig('q1a.png', dpi=600)
```

The result is shown in the figure:



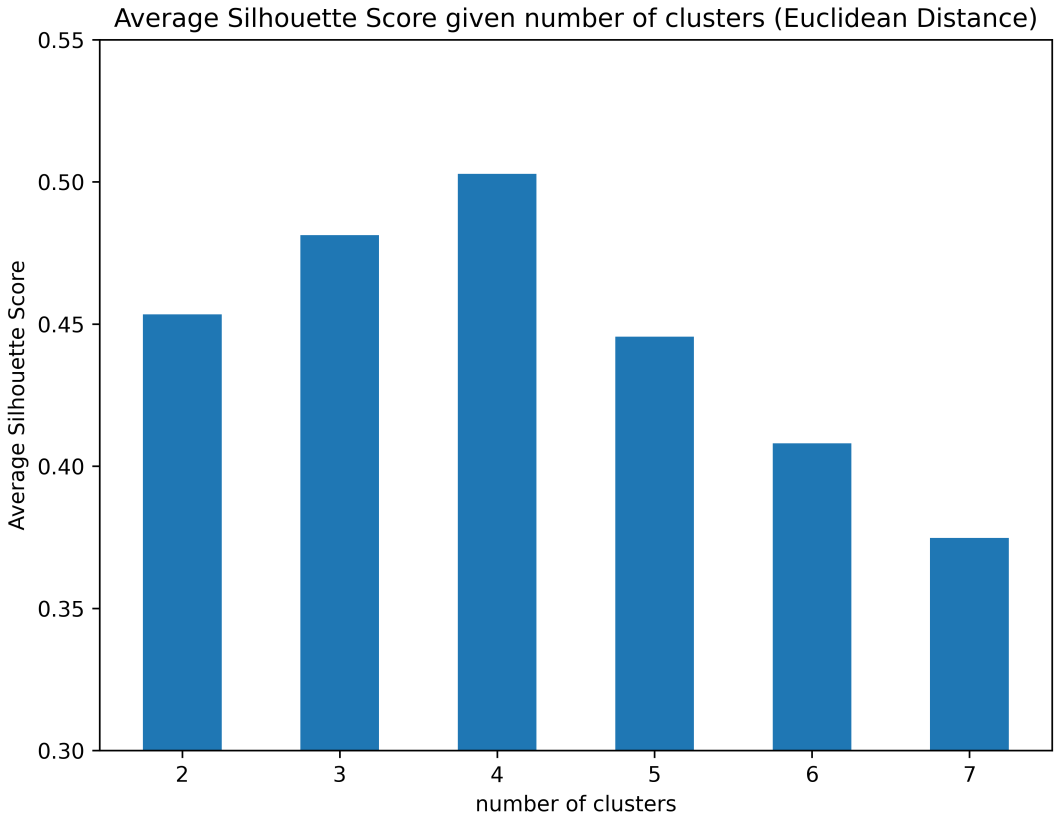
(b) In this problem, we set $k \in \{2, 3, ..., 7\}$ and perform KMeans to the data using two distance measurement, Euclidean Distance and Manhattan Distance. And then plot average silhouette coefficient of the data points. After experiment, the optimal value of k is $k = 4$, the same as number of data labels. The code is followed the above one. For Euclidean Distance,

```
def EuclideanDistance(v, axis):
    return np.linalg.norm(v, axis=axis)
def ManhattanDistance(v, axis):
    return np.sum(np.abs(v), axis=axis)
def AvgSilhouetteScore(data, labels, distance):
    AvgS = 0
    unique_labels = np.unique(labels)
    for i in range(len(labels)):
        cur_sample = data[i]
        cur_label = labels[i]

        same_cluster = data[labels == cur_label]
        a = np.mean([distance(cur_sample - sample, axis=0)\
         for sample in same_cluster if not np.array_equal(cur_sample, sample)])
```

```
        b = np.inf
        for label in unique_labels:
            if label != cur_label:
                other_cluster = data[labels == label]
                b = min(b, np.mean([distance(cur_sample - sample, axis=0)\
                 for sample in other_cluster]))

        AvgS += (b - a) / max(a, b)
    return AvgS / len(labels)
```

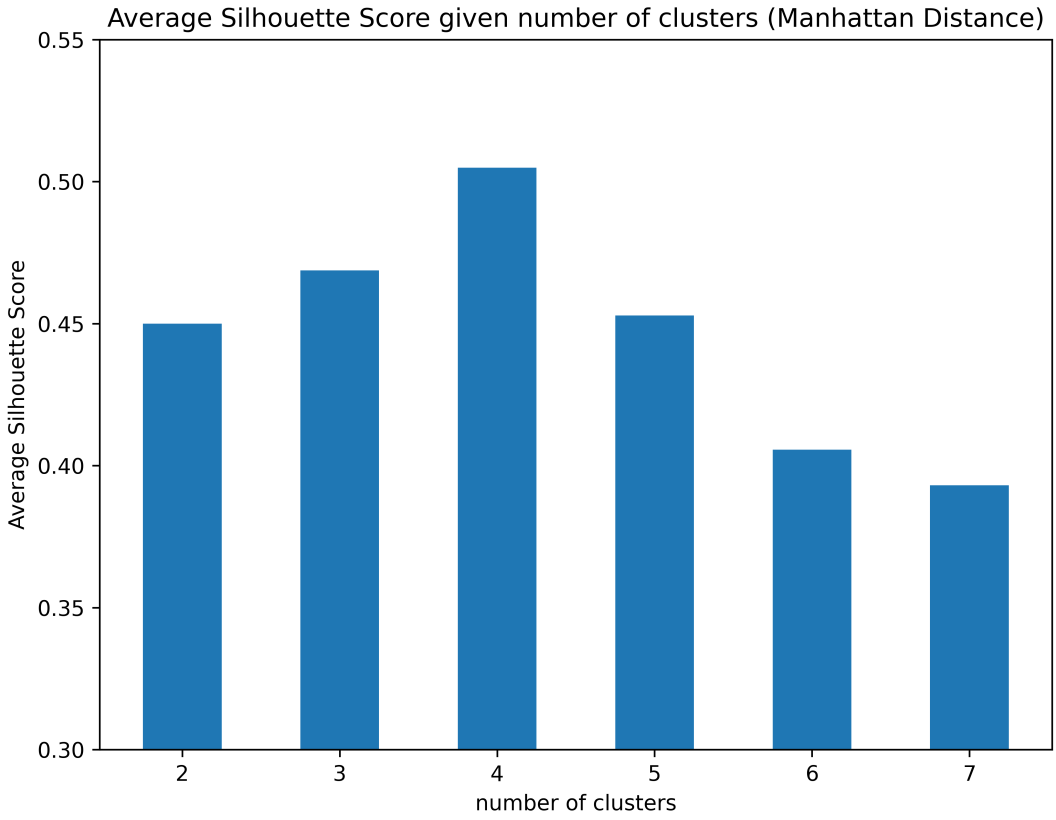
```
class Kmeans:
def __init__(self, k, distance, max_iters=10000, tol=1e-4):
    self.k = k
    self.distance = distance
    self.max_iters = max_iters
    self.tol = tol
    self.centroids = None

    def _init_centroids(self, data):
        indices = np.random.choice(data.shape[0], self.k, replace=False)
        return data[indices]

    def _assign_clusters(self, data):
        distances = self.distance(data[:, np.newaxis] - self.centroids, axis=2)
        return np.argmin(distances, axis=1)

    def _updata_centroids(self, data, labels):
        return np.array([data[labels == i].mean(axis=0) for i in range(self.k)])

    def fit(self, data):
        self.centroids = self._init_centroids(data)
        for i in range(self.max_iters):
            labels = self._assign_clusters(data)
            new_centroids = self._updata_centroids(data, labels)
            if np.all(self.distance(new_centroids - self.centroids, axis=1) < self.tol):
                break
            self.centroids = new_centroids

    def predict(self, data):
        return self._assign_clusters(data)
```

```
np.random.seed(42)
data = df.iloc[:, :2].to_numpy()
K = [i for i in range(2, 8, 1)]
AvgSScore_Euc = []
for i in K:
    model = Kmeans(i, EuclideanDistance)
    model.fit(data)
    labels = model.predict(data)
    AvgSScore_Euc.append(float(AvgSilhouetteScore(data, labels, EuclideanDistance)))

plt.figure(figsize=(8, 6))
plt.bar(K, AvgSScore_Euc, width=0.5)
plt.title('Average Silhouette Score given number of clusters (Euclidean Distance)')
plt.xlabel('number of clusters')
plt.ylabel('Average Silhouette Score')
plt.ylim(0.3, 0.55)
plt.show()
```



For Manhattan Distance,

```
AvgSScore_Man = []
for i in K:
    model = Kmeans(i, EuclideanDistance)
    model.fit(data)
    labels = model.predict(data)
    AvgSScore_Man.append(float(AvgSilhouetteScore(data, labels, ManhattanDistance)))

plt.figure(figsize=(8, 6))
plt.bar(K, AvgSScore_Man, width=0.5)
plt.title('Average Silhouette Score given number of clusters (Manhattan Distance)')
plt.xlabel('number of clusters')
plt.ylabel('Average Silhouette Score')
plt.ylim(0.3, 0.55)
plt.show()
```



(c) In this problem, we set $k \in \{1, 2, 3, 4\}$ and do EM algorithm. We firstly initialize weight, mean and covariance matrix, then repeat e-step, m-step until either reach max iteration or converge. The code is as follows (follow the above):

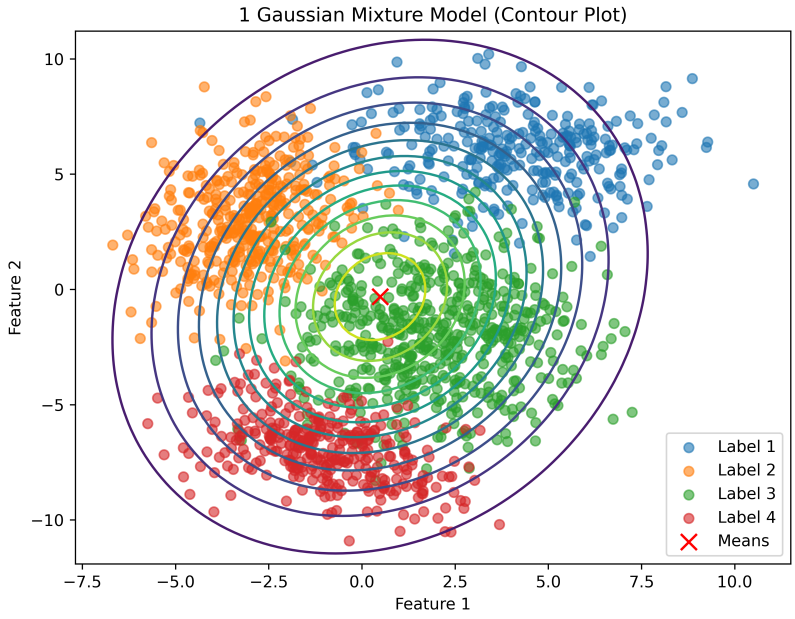
```
from scipy.stats import multivariate_normal
class EM:
    def __init__(self, k, max_iter=1000, tol=1e-3):
        self.k = k
        self.max_iter=max_iter
        self.tol = tol
        self.weight = None
        self.mean = None
        self.covariances = None

    def _init_parameters(self, data):
        self.weight = np.ones(self.k) / self.k
        indices = np.random.choice(data.shape[0], self.k, replace=False)
        self.mean = data[indices]
        self.covariances = [np.eye(data.shape[1]) for _ in range(self.k)]
        # print(self.weight, self.mean, self.covariances)

    def fit(self, data):
        self._init_parameters(data)
        for _ in range(self.max_iter):
            pre_likelihood = self.log_Marginallikelihood(data)
            post_prob = self.e_step(data)
            self.m_step(data, post_prob)
            log_likelihood = self.log_Marginallikelihood(data)
            if np.abs(log_likelihood - pre_likelihood) < self.tol:
                break


    def log_Marginallikelihood(self, data):
        log_marginallikelihood = 0
        for k in range(self.k):
            log_marginallikelihood += np.log(self.weight[k] *\
              multivariate_normal(self.mean[k], self.covariances[k]).pdf(data).sum())
        return log_marginallikelihood

    def e_step(self, data):
        post_prob = np.zeros((data.shape[0], self.k))
        for k in range(self.k):
            post_prob[:, k] = self.weight[k] *\
              multivariate_normal(self.mean[k], self.covariances[k]).pdf(data)
        post_prob /= post_prob.sum(axis=1, keepdims=True)
        return post_prob

    def m_step(self, data, post_prob):
        self.weight = post_prob.sum(axis=0) / data.shape[0]
        self.mean = np.dot(post_prob.T, data) / post_prob.sum(axis=0)[:, np.newaxis]
```

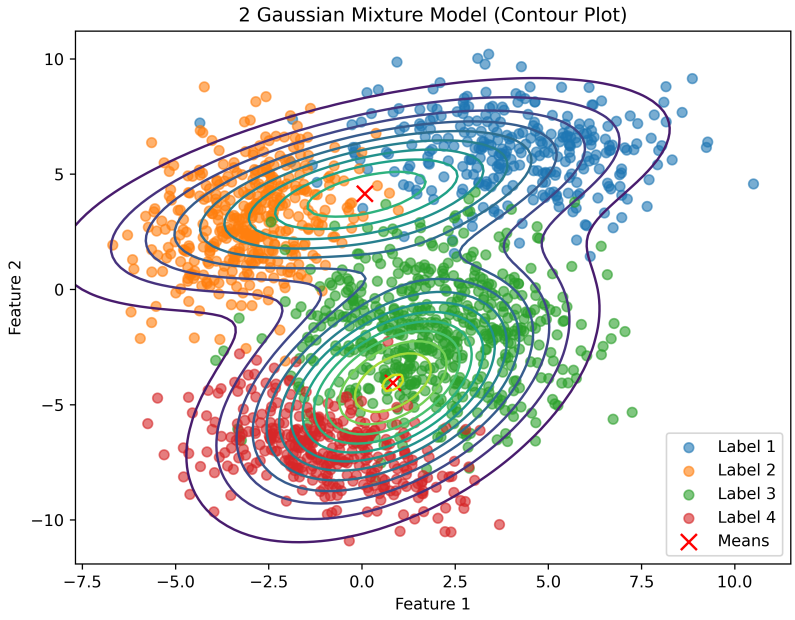
```
        data_centered = data[:, np.newaxis, :] - self.mean
        post_prob_extended = post_prob[:, :, np.newaxis]  # Shape: (n_samples, n_components, 1)
        self.covariances = np.einsum(
        'ijk,ijl->jkl', data_centered * post_prob_extended, data_centered
        ) / post_prob.sum(axis=0)[:, np.newaxis, np.newaxis]

k = [1, 2, 3, 4]
for i in k:
    np.random.seed(42)
    model = EM(i)
    model.fit(data)
    print(f'{i} Gaussian has weight {model.weight}, mean {model.mean}')
    print('----------------------------------------------------------------')
```

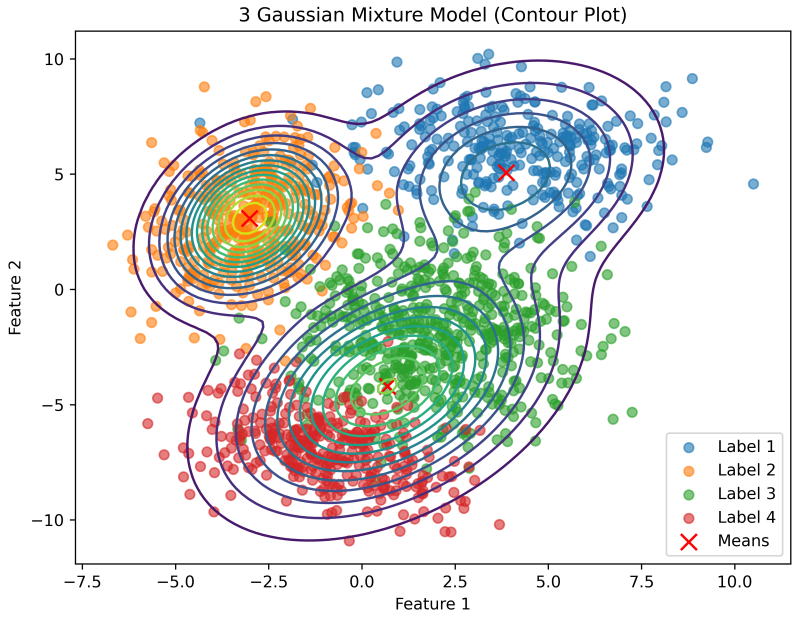
The result of weight and mean is as follows:

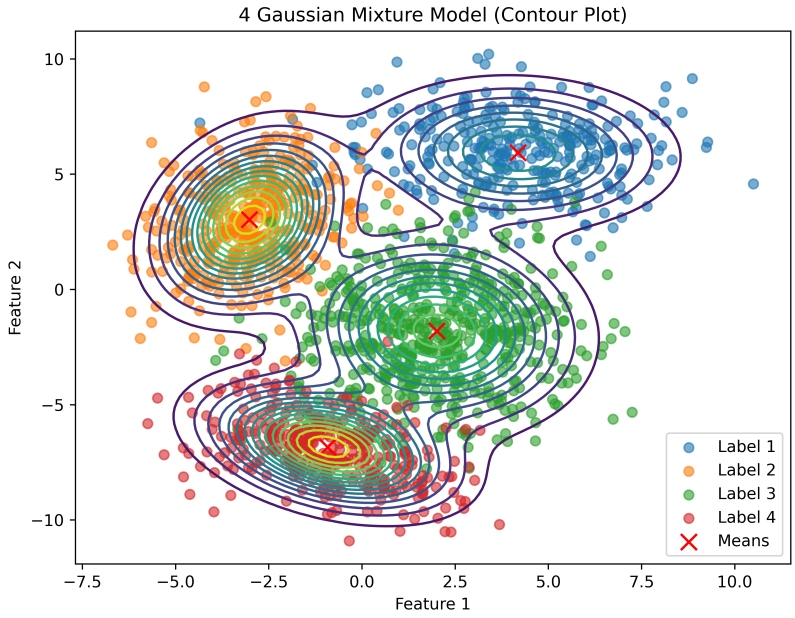
```
1 Gaussian has weight [1.]
mean [[ 0.487239   -0.31368064]]
----------------------------------------------------------------
2 Gaussian has weight [0.45573887 0.54426113]
mean [[ 0.07674448  4.14786009]
 [ 0.83096796 -4.0495667 ]]
----------------------------------------------------------------
3 Gaussian has weight [0.24776738 0.52700064 0.22523198]
mean [[-3.01428079  3.07826986]
 [ 0.68580732 -4.2057353 ]
 [ 3.87448791  5.06166878]]
----------------------------------------------------------------
4 Gaussian has weight [0.24928593 0.23223658 0.18818314 0.33029435]
mean [[-3.01962034  3.02734944]
 [-0.91764389 -6.84225433]
 [ 4.18756722  5.92714792]
 [ 2.0135639  -1.80058744]]
----------------------------------------------------------------
```

We can see with the increasing of k, the estimated values are approaching the true values. The contour of density function is shown below:


1 Gaussian Mixture Model (Contour Plot)
Feature 2
Feature 1
10
5
0
−5
−10
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
Label 1
Label 2
Label 3
Label 4
Means


2 Gaussian Mixture Model (Contour Plot)
Feature 2
Feature 1
10
5
0
−5
−10
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
Label 1
Label 2
Label 3
Label 4
Means


3 Gaussian Mixture Model (Contour Plot)
Feature 2
Feature 1
10
5
0
−5
−10
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
Label 1
Label 2
Label 3
Label 4
Means


4 Gaussian Mixture Model (Contour Plot)
Feature 2
Feature 1
10
5
0
−5
−10
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
Label 1
Label 2
Label 3
Label 4
Means