机器学习任务 1 实验报告

2017211416 张志博

1 任务定义：目标是将二维数据集 cluster.dat 分为几个集群。

2 完成任务（共 60 分）

2.1 方法 1：K-means（20 分）

2.2 方法 1：额外内容（10 分）

2.3 方法 2：高斯混合模型（Gaussian Mixture Model）（30 分）

3 输入输出

3.1 输入：二维数据集

3.2 输出：数据分类结果（以及相关过程量）

4 方法描述

4.1 方法 1：K-means

4.1.1 目标：在 cluster.dat 上实现 K-means 方法并尝试不同数量的集群。

4.1.2 算法

4.1.2.1 SSE 计算函数 **kmeans_2d_sse**

4.1.2.1.1 功能：SSE 计算功能，输入数据集、分类标签及中心点，计算并输出 SSE

4.1.2.1.2

```
def kmeans_2d_sse(data, labels, centroids):
    sse = 0
    for i in range(0, int(labels.max()) + 1):
        if not np.isnan(centroids[i]).any():
            sse += np.sqrt((np.power(data[labels == i] - centroids[i],
                                     2)).sum(axis=1)).sum()
    return sse
```

4.1.2.2 预测函数 **kmeans_2d_predict_sse**

4.1.2.2.1 功能：输入数据集和中心点，预测并输出分类标签结果

4.1.2.2.2

```
def kmeans_2d_predict_sse(data, centroids):
    labels = np.zeros(data.shape[0])
    for i, point in enumerate(data):
        labels[i] = np.argmin(
            np.sqrt((np.power(point - centroids, 2)).sum(axis=1)))
    return kmeans_2d_sse(data, labels, centroids)
```

4.1.2.3 中心点随机生成函数 **init_centroid_2d**

4.1.2.3.1 功能：输入数据集和数量 k，随机生成 k 个中心点，也用于 GMM

```
def init_centroid_2d(data, k):
    min_x = np.min(data[:, 0])
    max_x = np.max(data[:, 0])
    min_y = np.min(data[:, 1])
    max_y = np.max(data[:, 1])

    centroid_x = random.uniform(min_x, max_x)
    centroid_y = random.uniform(min_y, max_y)
    centroids = np.array([centroid_x, centroid_y])
    for i in range(1, k):
        centroid_x = random.uniform(min_x, max_x)
        centroid_y = random.uniform(min_y, max_y)
        centroids = np.vstack([centroids, [centroid_x, centroid_y]])
    return centroids
```

4.1.2.3.2

4.1.2.4 函数 **kmeans_2d**

4.1.2.4.1 功能：输入数据集和分类数量 k，输出分类标签结果、中心点和用于评估的结果损失

```
def kmeans_2d(data, k):
    centroids = init_centroid_2d(data, k)

    labels = np.zeros(data.shape[0])
    for i, point in enumerate(data):
        labels[i] = np.argmin(
            np.sqrt((np.power(point - centroids, 2)).sum(axis=1)))
    new_centroids = centroids.copy()

    for i in range(0, k):
        new_centroids[i] = np.mean(data[labels == i], axis=0)

    while kmeans_2d_predict_sse(data, new_centroids) < kmeans_2d_sse(
            data, labels, centroids):
        if np.isnan(new_centroids).any():
            return kmeans_2d(data, k)  # try again
        centroids = new_centroids
        for i, point in enumerate(data):
            labels[i] = np.argmin(
                np.sqrt((np.power(point - centroids, 2)).sum(axis=1)))
        for i in range(0, k):
            new_centroids[i] = np.mean(data[labels == i], axis=0)

    labels = np.zeros(data.shape[0])
    for i, point in enumerate(data):
        labels[i] = np.argmin(
            np.sqrt((np.power(point - centroids, 2)).sum(axis=1)))

    return labels, new_centroids, kmeans_2d_sse(data, labels, new_centroids)
```

4.1.2.4.2

4.2 方法 2：高斯混合模型（Gaussian Mixture Model）

4.2.1 目标：在 cluster.dat 上对混合的高斯实施 EM 拟合并尝试使用不同数量的混合。

4.2.2 算法

4.2.2.1 E 步和 M 步

4.2.2.1.1 功能：E 步通过数据集、均值、协方差矩阵、权重计算出 responsibilities，M 步通过 E 步计算出的数据集、均值、协方差矩阵和新的 responsibilities 再计算出新的均值、协方差矩阵和权重。

```
def E_step(data, means, covs, weights):
    n_data = data.shape[0]
    n_clusters = means.shape[0]
    responsibilities = np.zeros([n_data, n_clusters])
    for c in range(n_clusters):
        responsibilities[:, c] = multivariate_normal.pdf(
            data, means[c], covs[c])
    responsibilities = weights * responsibilities
    responsibilities /= responsibilities.sum(axis=1)[:, np.newaxis]
    return responsibilities


def M_step(data, responsibilities, means, covs):
    n_data, n_clusters = responsibilities.shape
    weights = responsibilities.sum(axis=0)
    for c in range(n_clusters):
        resp = responsibilities[:, c][:, np.newaxis]
        means[c] = (resp * data).sum(axis=0) / resp.sum()
        covs[c] = ((data - means[c]).T).dot(
            (data - means[c]) * resp) / weights[c]
    weights /= weights.sum()
    return means, covs, weights
```

4.2.2.1.2

4.2.2.2 高斯混合

4.2.2.2.1 功能：输入数据集、初始的中心点（由其他方法，得出可提升性能，故没有把随机生成中心点的过程嵌入函数）和迭代次数，初始化 EM 算法所需的参数并执行 n 次 EM 迭代，计算并输出分类结果。

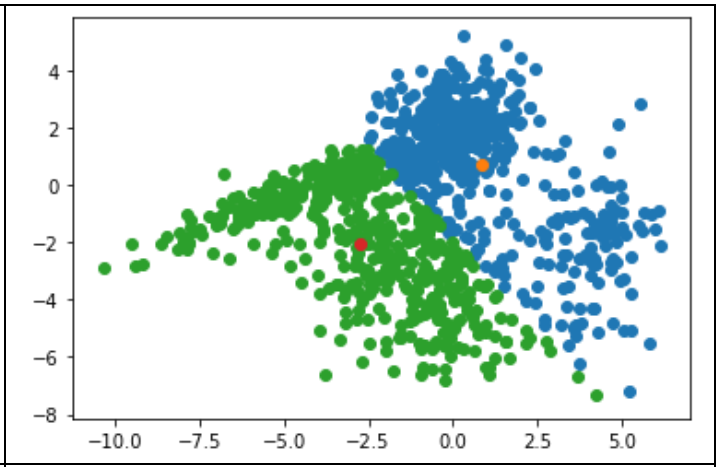
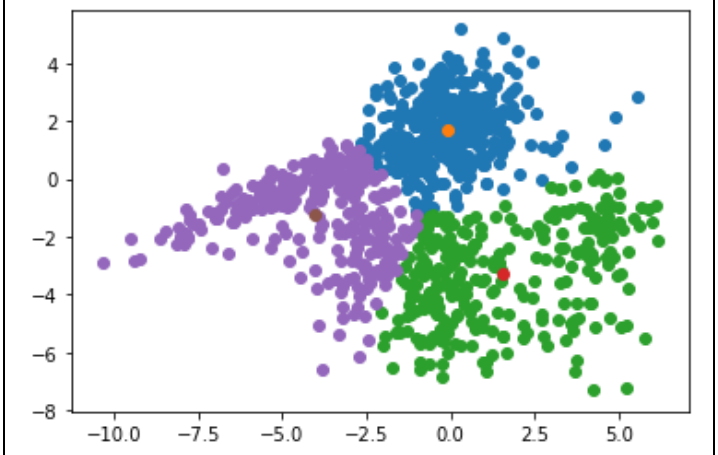
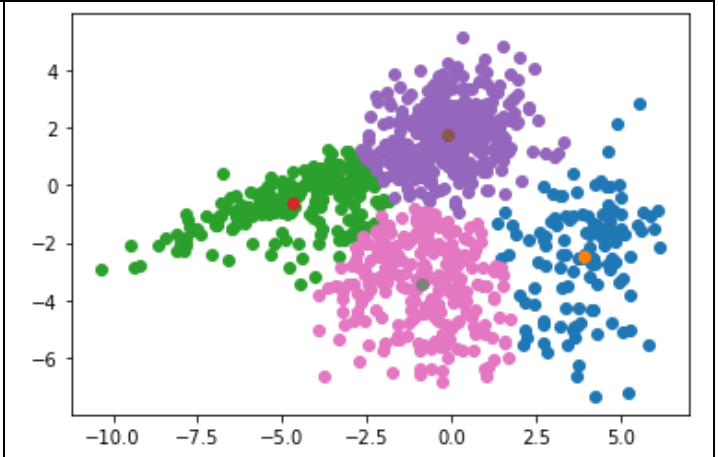
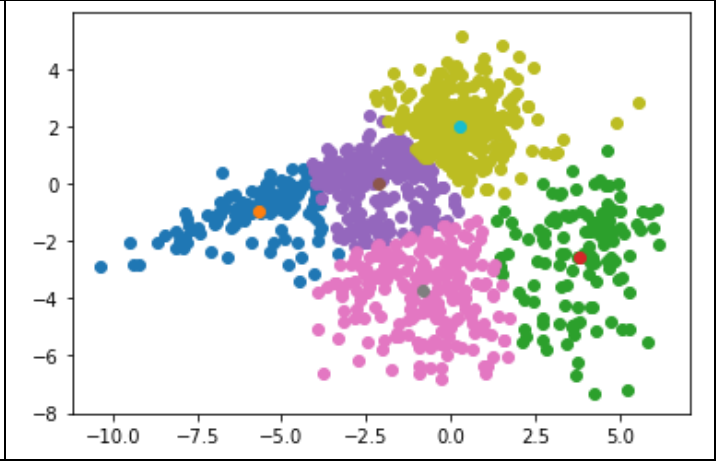
```
def Gaussian_Mixture(data, centroids, n_iterations=99):
    k = centroids.shape[0]
    means = centroids
    weights = np.ones(k) / k
    weights[0] += 1
    covs = np.array([np.cov(data.T)] * k)
    weights /= weights.sum()

    for i in range(n_iterations):
        responsibilities = E_step(data, means, covs, weights)
        means, covs, weights = M_step(data, responsibilities, means, covs)
        labels = responsibilities.argmax(axis=1)
    return labels
```

4.2.2.2.2

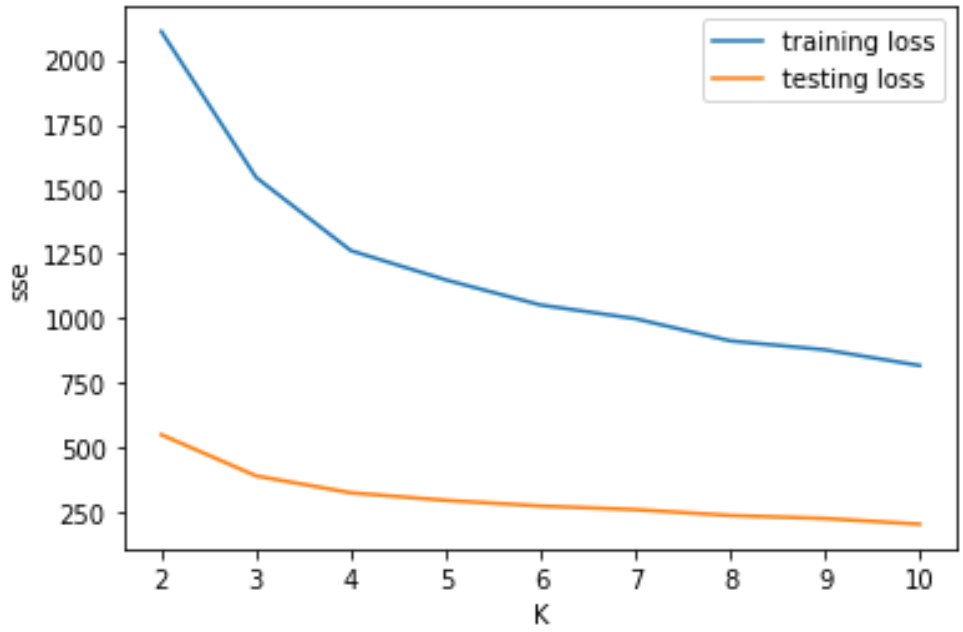
5 结果分析（性能评价）

5.1 方法 1：K-means

5.1.1 不同 k 时分类结果和中心点

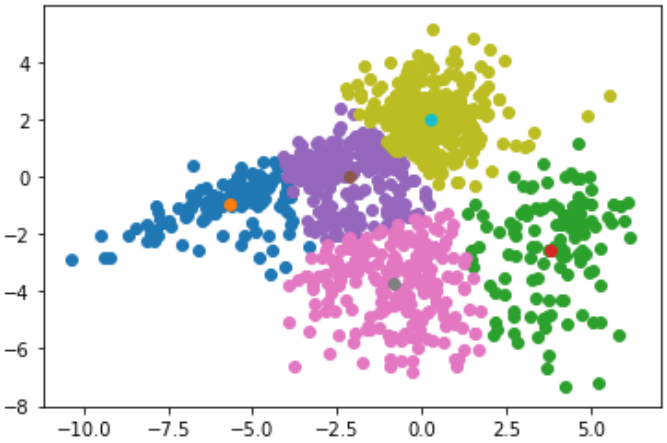
K = 2	 4 2 0 −2 −4 −6 −8 −10.0 −7.5 −5.0 −2.5 0.0 2.5 5.0
K = 3	 4 2 0 −2 −4 −6 −8 −10.0 −7.5 −5.0 −2.5 0.0 2.5 5.0
K = 4	 4 2 0 −2 −4 −6 −10.0 −7.5 −5.0 −2.5 0.0 2.5 5.0
K = 5	 4 2 0 −2 −4 −6 −8 −10.0 −7.5 −5.0 −2.5 0.0 2.5 5.0

5.1.2 SSE 和 K 的关系图:



5.1.3 结果评价：几乎不存在最佳的 K

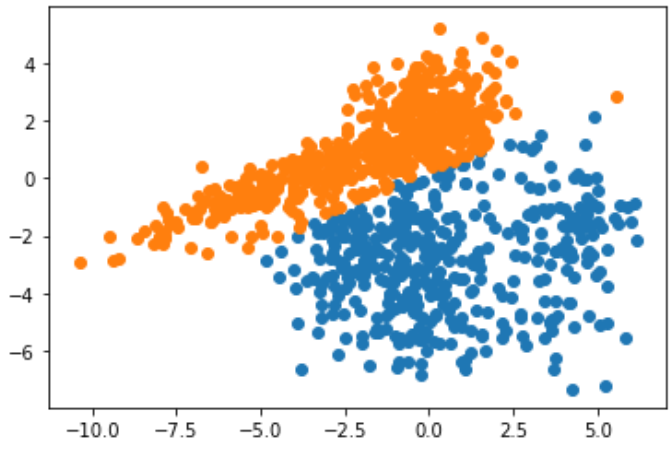
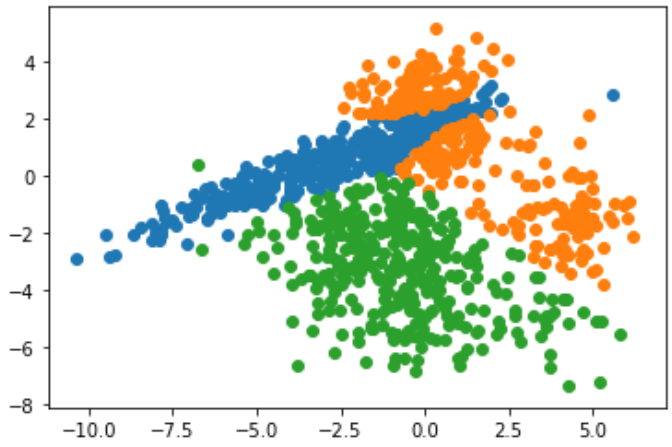
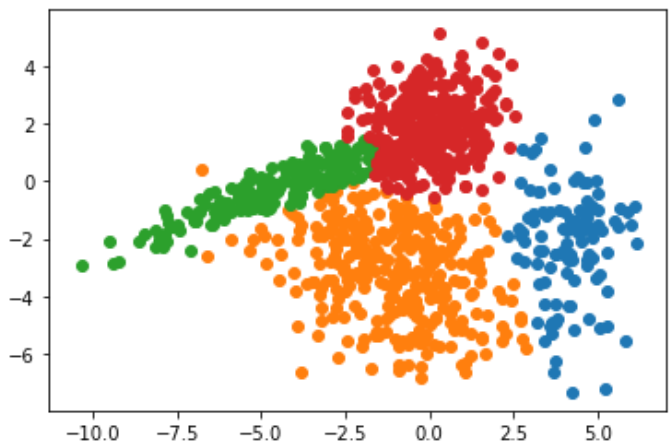
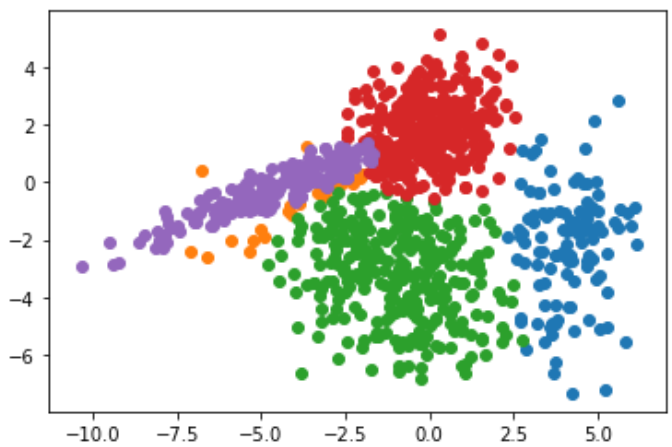
5.1.3.1 光是从这组数据本身其实还是很难看出这组数据的结构的，因为数据点本身分布密集程度就很高，即使是把它们全分成 1 类也是合理的，因此，我认为在聚类关系不明显时使用强行聚类有时并不能正确地揭示数据地结构特性。但如果一定要选的话，k=5 时分成的 5 组直观上更好，同时训练和测试 SSE 都比较小，且更高的 k 对 SSE 的提升效果也变更加弱。



5.1.3.2

5.2 方法 2：高斯混合模型（Gaussian Mixture Model）

5.2.1 迭代次数 99 次不同 k 时分类结果和中心点

K = 2	 4 2 0 −2 −4 −6 −10.0 −7.5 −5.0 −2.5 0.0 2.5 5.0
K = 3	 4 2 0 −2 −4 −6 −8 −10.0 −7.5 −5.0 −2.5 0.0 2.5 5.0
K = 4	 4 2 0 −2 −4 −6 −10.0 −7.5 −5.0 −2.5 0.0 2.5 5.0
K = 5	 4 2 0 −2 −4 −6 −10.0 −7.5 −5.0 −2.5 0.0 2.5 5.0