哈尔滨工业大学计算机科学与技术学院

实验报告

课程名称： 机器学习

课程类型： 选修

实验题目： k-means 聚类和混合高斯模型

学号： 1170300520

姓名： 郭子阳

一、实验目的

实现一个k-means算法和混合高斯模型，并且用EM算法估计模型中的参数。

二、实验要求及实验环境

实验要求

用高斯分布产生k个高斯分布的数据（不同均值和方差）（其中参数自己设定）。

（1）用k-means聚类，测试效果；

（2）用混合高斯模型和你实现的EM算法估计参数，看看每次迭代后似然值变化情况，考察EM算法是否可以获得正确的结果（与你设定的结果比较）。

（3）可以UCI上找一个简单问题数据，用你实现的GMM进行聚类。

实验环境

- Python 3.7.0
- JupyterLab 1.1.3

三、设计思想（本程序中的用到的主要算法及数据结构）

算法原理

k-means算法是一种迭代求解的聚类分析算法，其步骤是随机选取K个对象作为初始的聚类中心，然后计算每个对象与各个种子聚类中心之间的距离，把每个对象分配给距离它最近的聚类中心。聚类中心以及分配给它们的对象就代表一个聚类。每分配一个样本，聚类的聚类中心会根据聚类中现有的对象被重新计算。这个过程将不断重复直到满足某个终止条件。终止条件可以是没有（或最小数目）对象被重新分配给不同的聚类，没有（或最小数目）聚类中心再发生变化，误差平方和局部最小。

伪代码：

```
选择k个点作为初始质心。
repeat
   将每个点指派到最近的质心，形成k个簇
   重新计算每个簇的质心
until 质心不发生变化
```

高斯混合模型是一个概率模型，它假设数据来自由若干个未知的高斯（正态）分布组成的混合分布。

EM算法求解伪代码：

```
确定K值，随机初始各个参数值
repeat
  E步：根据当前参数，计算每个点由某个分模型生成的概率
  M步：E步估计出的概率越大，则相对应的模型权重越大，用此作为权重来计算加权的均值和方差来替代其原本的均值和方差
until 均值、方差收敛
```

算法实现

初始化数据点（三个类别）：

```
X = np.zeros((2, 300))
for i in range(100):
    X[0, i] = random.gauss(0, 2)
    X[1, i] = random.gauss(0, 2)
for i in range(100, 200):
    X[0, i] = random.gauss(0, 2)
    X[1, i] = random.gauss(10, 2)
for i in range(200, 300):
    X[0, i] = random.gauss(-6, 2)
    X[1, i] = random.gauss(6, 2)
plt.scatter(X[0, 0:100], X[1, 0:100], marker=".")
plt.scatter(X[0, 100:200], X[1, 100:200], marker=".")
plt.scatter(X[0, 200:300], X[1, 200:300], marker=".")
plt.show()
```

使用k-means方法聚类（返回各个类的中心点）：

```
def kmeans(generation, x):
    c1_center = x[random.randint(0, x.shape[0])]
    c2_center = x[random.randint(0, x.shape[0])]
    c3_center = x[random.randint(0, x.shape[0])]
    for i in range(generation):
        c1 = []
        c2 = []
        c3 = []
        for i in range(x.shape[0]):
            current = x[i]
            c1_dis = calDis(c1_center, current)
            c2_dis = calDis(c2_center, current)
            c3_dis = calDis(c3_center, current)
            if (c1_dis <= c2_dis) & (c1_dis <= c3_dis):
                c1.append(current)
            elif (c2_dis <= c1_dis) & (c2_dis <= c3_dis):
                c2.append(current)
            else:
                c3.append(current)
        sumx = 0.0
```

```
        sumy = 0.0
        if len(c1) != 0:
            for i in range(len(c1)):
                sumx = sumx + c1[i][0]
                sumy = sumy + c1[i][1]
            c1_center = [sumx / len(c1), sumy / len(c1)]
        sumx = 0.0
        sumy = 0.0
        if len(c2) != 0:
            for i in range(len(c2)):
                sumx = sumx + c2[i][0]
                sumy = sumy + c2[i][1]
            c2_center = [sumx / len(c2), sumy / len(c2)]
        sumx = 0.0
        sumy = 0.0
        if len(c3) != 0:
            for i in range(len(c3)):
                sumx = sumx + c3[i][0]
                sumy = sumy + c3[i][1]
            c3_center = [sumx / len(c3), sumy / len(c3)]
    return c1_center, c2_center, c3_center
```

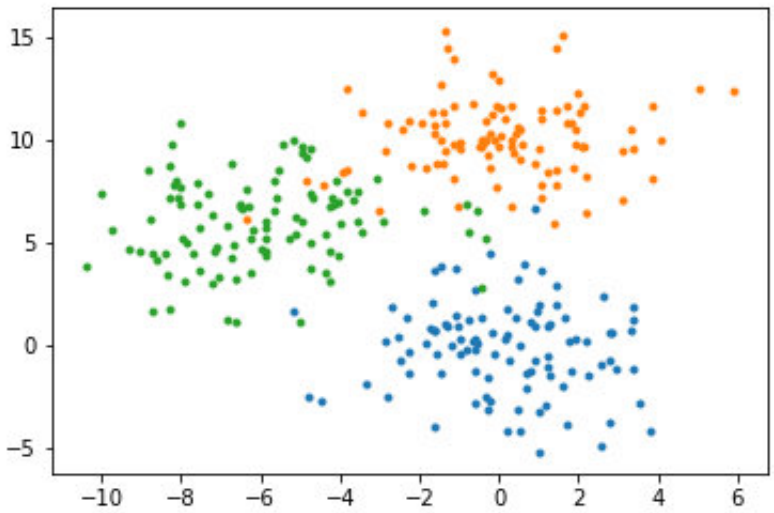
用于GMM的EM算法：

```
def update_W(X, Mu, Var, Pi):
    n_points, n_clusters = len(X), len(Pi)
    pdfs = np.zeros(((n_points, n_clusters)))
    for i in range(n_clusters):
        pdfs[:, i] = Pi[i] * multivariate_normal.pdf(X, Mu[i], np.diag(Var[i]))
    W = pdfs / pdfs.sum(axis=1).reshape(-1, 1)
    return W

def update_Pi(W):
    Pi = W.sum(axis=0) / W.sum()
    return Pi

def logLH(X, Pi, Mu, Var):
    n_points, n_clusters = len(X), len(Pi)
    pdfs = np.zeros(((n_points, n_clusters)))
    for i in range(n_clusters):
        pdfs[:, i] = Pi[i] * multivariate_normal.pdf(X, Mu[i], np.diag(Var[i]))

def plot_clusters(X, Mu, Var):
    colors = ['r', 'g', 'b']
    n_clusters = len(Mu)
    XT = X.T
    plt.scatter(XT[0, 0:100], XT[1, 0:100], marker=".", color="r")
    plt.scatter(XT[0, 100:200], XT[1, 100:200], marker=".", color="g")
    plt.scatter(XT[0, 200:300], XT[1, 200:300], marker=".", color="b")
```

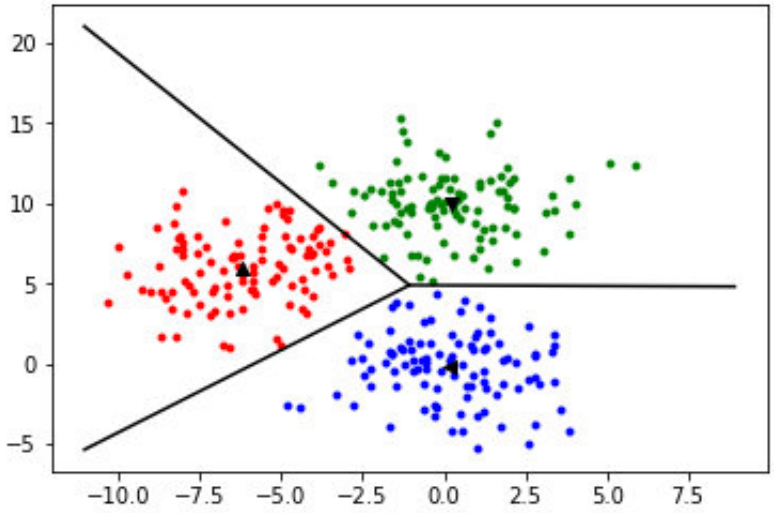
```
    ax = plt.gca()
    for i in range(n_clusters):
        plot_args = {'fc': 'None', 'lw': 2, 'edgecolor': colors[i], 'ls': ':'}
        ellipse = Ellipse(Mu[i], 3 * Var[i][0], 3 * Var[i][1], **plot_args)
        ax.add_patch(ellipse)
    plt.show()

def update_Mu(X, W):
    n_clusters = W.shape[1]
    Mu = np.zeros((n_clusters, 2))
    for i in range(n_clusters):
        Mu[i] = np.average(X, axis=0, weights=W[:, i])
    return Mu

def update_Var(X, Mu, W):
    n_clusters = W.shape[1]
    Var = np.zeros((n_clusters, 2))
    for i in range(n_clusters):
        Var[i] = np.average((X - Mu[i]) ** 2, axis=0, weights=W[:, i])
    return Var
```

四、实验结果与分析

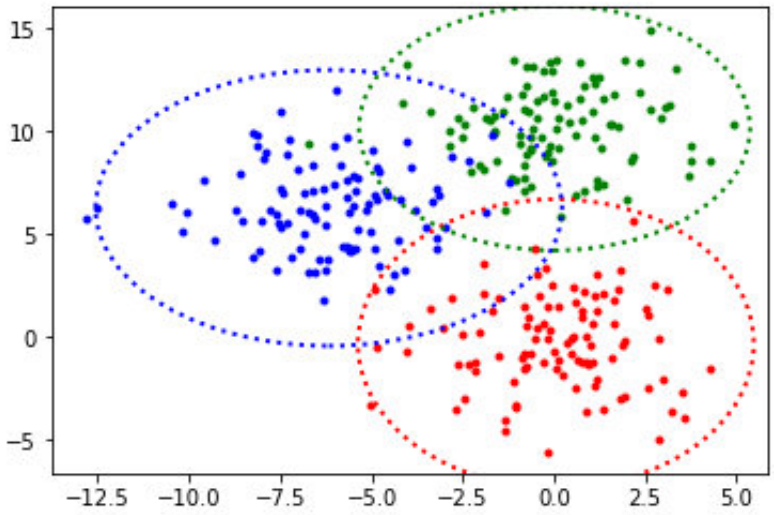
随机生成数据点：



使用k-means聚类结果：



使用EM算法聚类：



五、结论

K-means的聚类结果受制于最初中心点的生成，最初中心点效果不好可能会导致聚类陷于局部最优而无法达到全局最优。

六、参考文献

[1]. K-means clustering. https://en.wikipedia.org/wiki/K-means_clustering

[2]. Expectation–maximization algorithm. https://en.wikipedia.org/wiki/Expectation%E2%80%93maximization_algorithm