高斯混合模型

17301730178 宁晨然

本次实验实现了：

- 使用高斯分布构造聚类数据
- 使用高斯混合模型分类数据

实验亮点有：

- 代码封装，注释简洁易懂，调用函数方便
- 原理清晰，按部就班
- 分析了实现的优缺点

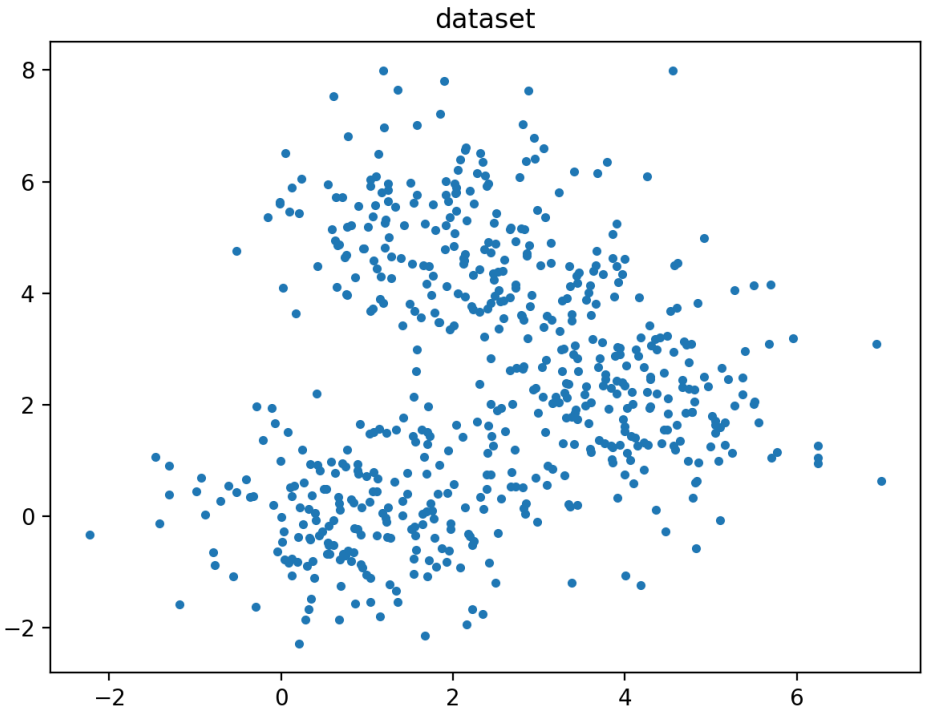
本次实验的`dataset`在实验运行过程中生成。

运行代码命令为：

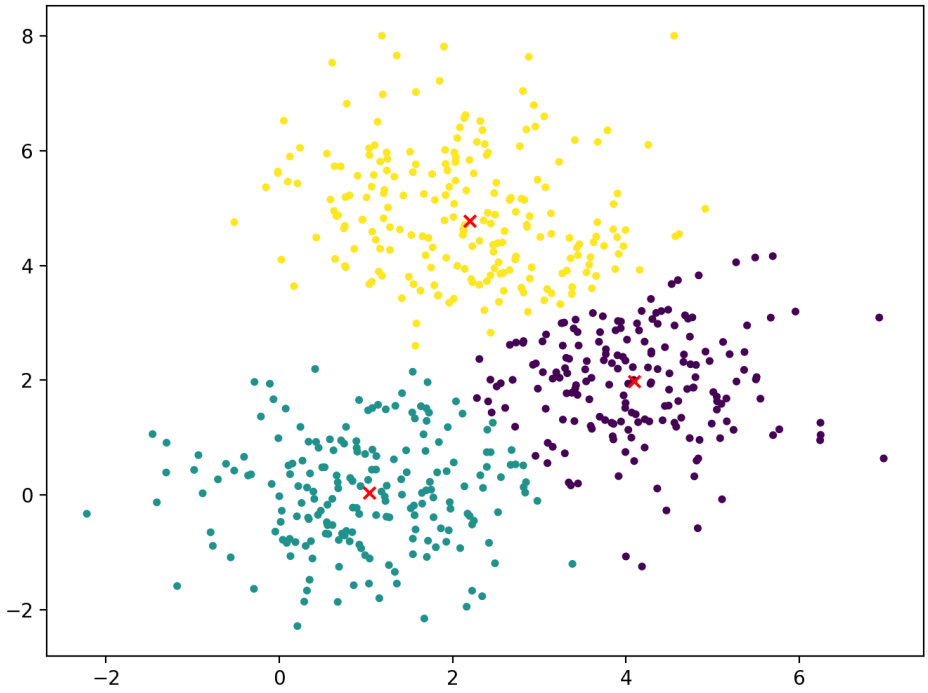
```
python source.py
```

1. 实验结果

使用高斯分布采样出聚类信息



再使用高斯混合模型分类，并标出中心位置。



2. 实验过程

2.1 生成聚类数据

目标：生成无标记的聚类数据

自然而然想到的是使用高斯分布采样，获得不同的聚类数据，由此可复用第一次作业中的高斯分布采样代码。

2.1.1 高斯类

高斯类的实现

高斯类中无需定义`label`标签，使用均值和方差即可。

```
# 定义高斯类
class gaussian():
    mean = None
    cov = None
    sample = None
    G = None
    def __init__(self, mean, cov, sample):
        self.mean = mean
        self.cov = cov
        self.sample = sample
        self.G = np.random.multivariate_normal(mean,cov,sample)
        # self.show()
    def show(self):
        plt.plot(*self.G.T, '.')
        plt.axis('scaled')
        # plt.show()
    def dataset(self):
        x = self.G.T[0]
        y = self.G.T[1]
        return [[x[i], y[i]] for i in range(len(x))]
```

2.1.2 生成数据库

定义三个高斯模型，混杂在一起，做成数据库。

```
# 根据高斯分布生成聚类数据
def gaussian_distribution(tofile=False, showplt=False, sample=500):
    gaussian_A = gaussian(mean[0],cov,sample)
    gaussian_B = gaussian(mean[1],cov,sample)
    gaussian_C = gaussian(mean[2],cov,sample)

    dataset = gaussian_A.dataset() + gaussian_B.dataset() + gaussian_C.dataset()
    random.shuffle(dataset)
    dataset = np.array(dataset)
    print(dataset.T.shape)
    if showplt:
        # plt.show()
        plt.scatter(dataset.T[0],dataset.T[1],marker='.')
        plt.title("dataset")
        plt.show()
    if tofile:
        with open('dataset.data','w') as f:
            print(dataset,file=f)
    else:
        return dataset
```

2.1.3 具体参数

三个高斯模型的均值和方差如下。

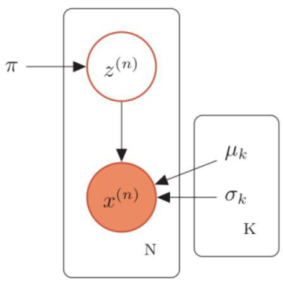
```
mean = np.array([[1,0],[4,2],[2,5]])
cov = np.array([[1,0],[0,1]])
```

2.2 高斯混合模型

目标：设计一个高斯混合模型，实现聚类

$$N_k = \sum_{n=1}^{N} \gamma_{nk}.$$

$$\pi_k = \frac{N_k}{N},$$
$$\mu_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma_{nk} x^{(n)},$$
$$\sigma_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma_{nk} (x^{(n)} - \mu_k)^2,$$



$$\mathcal{N}(x|\mu_k, \sigma_k) = \frac{1}{\sqrt{2\pi}\sigma_k} \exp\left(-\frac{(x-\mu_k)^2}{2\sigma_k^2}\right)$$

实现原理如同课程ppt内容所述，每次`epoch`中根据上述公式更新

- PI：为先验概率，初始化设置为均匀分布
- mu：每个聚类的均值
- sigma：每个聚类的方差
- N：高斯模型

该模型的内容有：

- 初始化：初始化所有参数为0
- 计算方法：
  - 多高斯分布：根据均值、方差，计算高斯分布概率
  - `gamma`计算：根据均值、方差、先验概率，计算`gamma`值
- 训练：
  - 导入数据后，迭代一定次数
  - 每次迭代更新所有均值、方差、先验概率等
- 预测：根据模型中保存的均值、方差、先验概率的值预测，判断属于某个聚类的概率

```
class GMM():
    def __init__(self,n_clusters, iteration=50,D=2):
        self.k = n_clusters
        self.iteration = iteration
        self.mu=0
        self.sigma=0
        self.PI=0
        self.D = D

    def multi_Gaussian(self, x, mu, sigma):
        return 1 / (pow( 2*np.pi, self.D/2) * pow(np.linalg.det(sigma) , 0.5)) *
np.exp(-0.5*(x-mu).T.dot(np.linalg.pinv(sigma)).dot(x-mu))

    def calculate_Gamma(self, x, mu, sigma, PI):
        # Gamma(num_samples,k)
        k = self.k
        D = self.D
        num_samples = x.shape[0]
        gamma = np.zeros((num_samples, k))
        P = np.zeros(k)
        for n in range(num_samples):
            # P(k,)
            for i in range(k):
                P[i] = PI[i] * self.multi_Gaussian(x[n], mu[i], sigma[i] )
            for i in range(k):
                gamma[n,i] = P[i]/np.sum(P)
        return gamma

    def train(self,data):
        # data(n, D)
        n = data.shape[0]
        D = data.shape[1]

        # initialization
        PI = np.ones(self.k) / self.k # 先验
        mu = np.array([data[random.randint(0,len(data))] for i in
range(self.k)]) # 随机选择几个值作为中心点
        sigma = np.full((self.k, D, D), np.diag(np.full(D,0.1)))

        for i in range(self.iteration):
            gamma = self.calculate_Gamma(data, mu, sigma, PI)   # gamma(n,k)
            N = np.sum(gamma,axis=0) # N_k = sum gamma(n)
            PI = N / n
            mu = np.array([1/N[i]*np.sum(data*gamma[:,i].reshape((n,1)),axis=0)
 for i in range(self.k)])
```

```
            for p in range(self.k):
                # calculate sigma_k
                sigma[p] = 0
                for q in range(n):
                    sigma[p] += np.dot( (data[q].reshape(1,D)-mu[p]).T,
(data[q]-mu[p]).reshape(1,D) ) * gamma[q,p]
                sigma[p] = sigma[p] / N[p]
        self.mu = mu
        self.sigma = sigma
        self.PI = PI

    def predict(self, data):
        res = self.calculate_Gamma(data, self.mu, self.sigma, self.PI)
        res = np.argmax(res,axis=1)
        return res
```

然后根据下述代码寻找聚类

```
data = gaussian_distribution(showplt=True,sample=200)
model = GMM(3)
model.train(data)
label = model.predict(data)

plt.scatter(data[:,0],data[:,1],c=label,marker='.')
plt.scatter(model.mu[:,0],model.mu[:,1],marker='x',color='red')
plt.show()
```

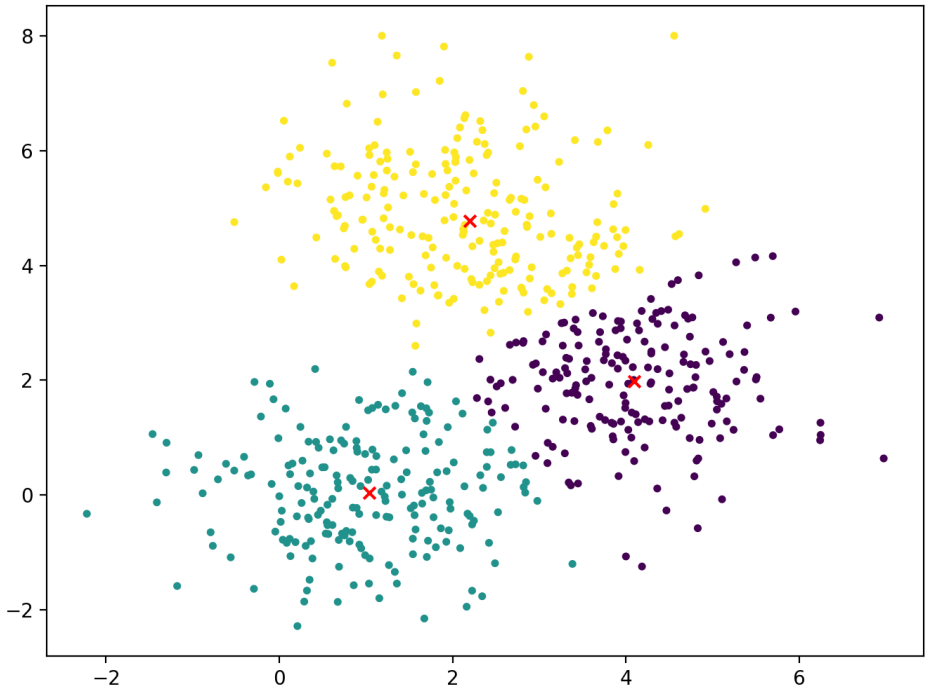
3. 测试

测试了在不同情况下的模型的表现

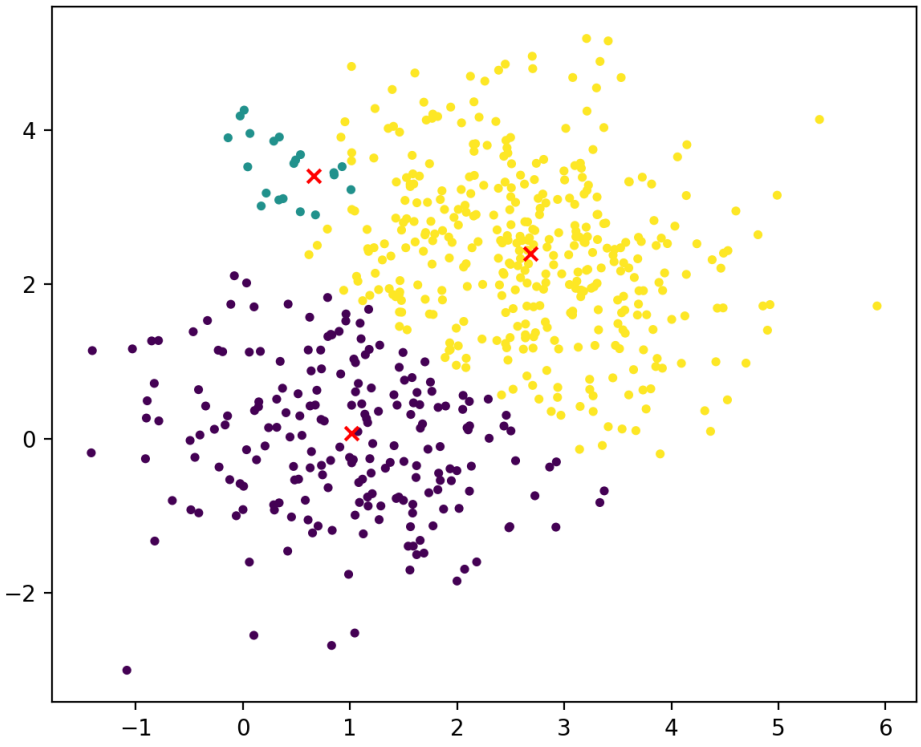
3.1 测试三个高斯分布

下面是不同的高斯分布的情况，当聚类样本混合程度过高时该模型无法正确识别出聚类。

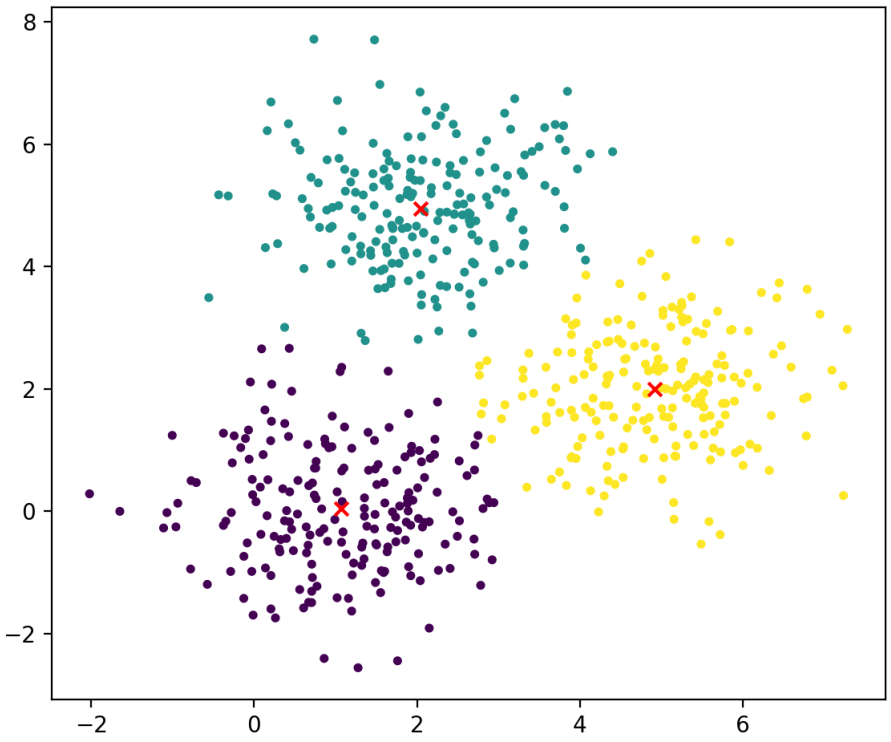
```
mean = np.array([[1,0],[4,2],[2,5]])
cov = np.array([[1,0],[0,1]])
```




```
mean = np.array([[1,0],[3,2],[2,3]])
cov = np.array([[1,0],[0,1]])
```




```
mean = np.array([[1,0],[5,2],[2,5]])
cov = np.array([[1,0],[0,1]])
```

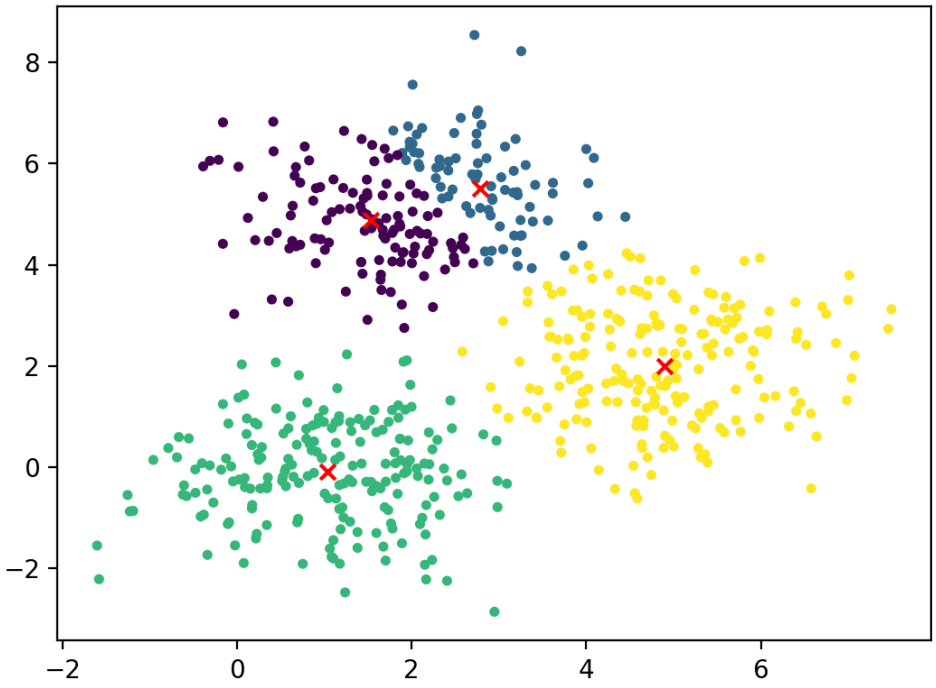


3.2 测试错误的聚类个数

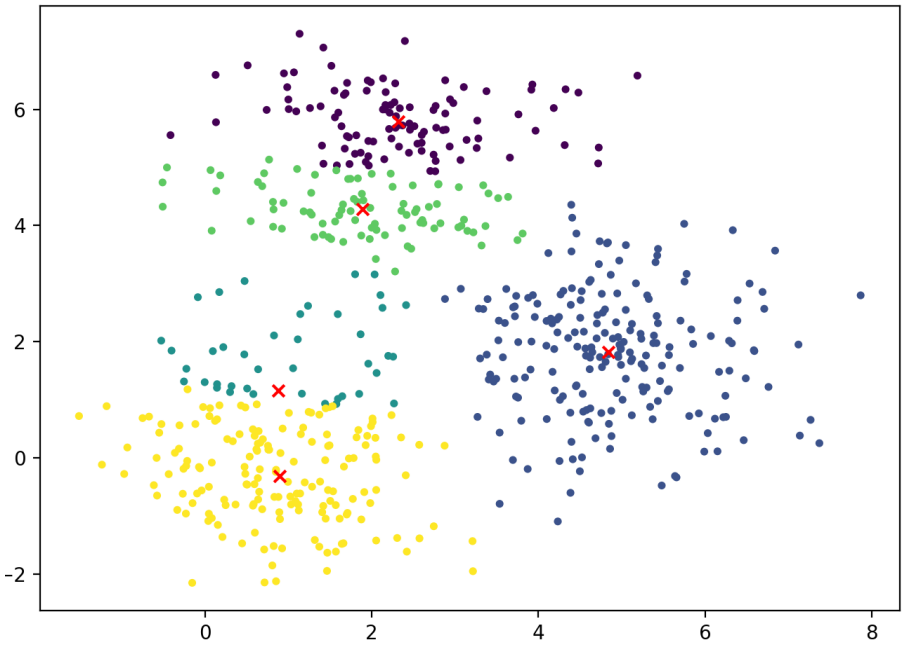
当聚类个数k选择与实际不符时，高斯混合模型也能做出聚类操作

- 实际效果不如k=实际值的效果
- 一般会造成大区域分割和多个小区域分割的情况

当采样的高斯分布为三个时，测试k=4的情况：



再测试k=5的情况：



4. 优缺点

高斯混合模型简单，可以直接根据公式写出代码，然后让其更迭计算即可。

优点：

- 简单，原理清晰
- 对于多个高斯分布的聚类易于实现无监督分类

本次实验缺点：

- 没有实现向量级别的运算，运算速度较慢
- 没有对比其他模型