ASSIGNMENT 3

Part 0

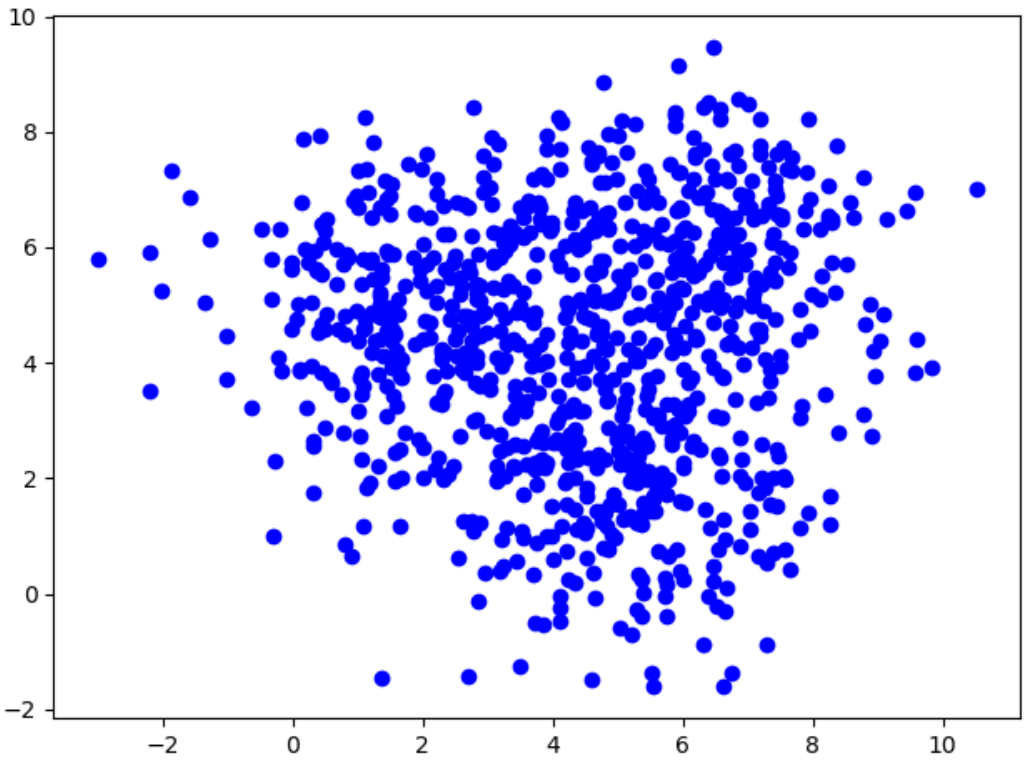
本实验按要求生成了无标记的数据集，完成了高斯混合模型，并针对聚类形状、初始化方法、k 值误差进行了进一步讨论。(source.py 最终上交状态里生成的是修改聚类形状后的数据集)

Part I

本实验设定数据集为 2 维，设计三个高斯分布，参数如下：

```
mean = np.array([[2, 5], [5, 2], [6, 6]])
cov = np.array([
    [[2, 0],
     [0, 2]],
    [[2, 0],
     [0, 2]],
    [[2, 0],
     [0, 2]]
])
```

混合三个高斯分布形成一个无标记的数据集，每个分布取 300 个点（source.py 中包含了生成数据的代码，由于是随机生成，每一次执行 source.py 都会有略微差异）



Part II

1，理论推导

给定N个由高斯混合模型生成的数据$\boldsymbol{x}^{(1)},\boldsymbol{x}^{(2)}\cdots\boldsymbol{x}^{(N)}$，希望能学习参数$\pi_k,\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k,1\le k\le K$。对于每个样本$x^{(n)}$，其对数边际分布为

$$\log P\left(x^{(n)}\right)=\log\sum_{k=1}^{k}\pi_k\,N\left(x^{(n)};u_k,\Sigma_k\right)$$

根据 EM 算法，参数估计可以分为两步进行迭代:

（1） **E 步**

先固定参数$\boldsymbol{\mu},\boldsymbol{\Sigma}$，计算后验分布$P\left(z^{(n)}|\,x^{(n)}\right)$，其中$z^{(n)}$表示第$n$个数据所属的类，即

$$\gamma_{nk}=P\left(z^{(n)}=k|\,x^{(n)}\right)=\frac{P\left(z^{(n)}\right)P\left(\,x^{(n)}|z^{(n)}\right)}{P(x^{(n)})}=\frac{\pi_k N\left(x^{(n)};u_k,\Sigma_k\right)}{\sum_{k=1}^{K}\pi_k N(x^{(n)};u_k,\Sigma_k)}$$

其中γ_{nk}定义了样本$x^{(n)}$属于第k个高斯分布的后验概率。

（2） **M 步**

令$q(z=k)=\gamma_{nk}$，训练集D的证据下界为

$$ELBO(\gamma,D;\boldsymbol{\pi},\boldsymbol{\mu},\boldsymbol{\Sigma})=\sum_{n=1}^{N}\sum_{k=1}^{K}\gamma_{nk}\log\frac{P\left(z^{(n)}=k,x^{(n)}\right)}{\gamma_{nk}}$$

$$=\sum_{n=1}^{N}\sum_{k=1}^{K}\gamma_{nk}\left(\log N\left(x^{(n)};u_k,\Sigma_k\right)+\log\frac{\pi_k}{\gamma_{nk}}\right)$$

将参数估计问题转化为优化问题:

$$\max_{\boldsymbol{\pi},\boldsymbol{\mu},\boldsymbol{\Sigma}}ELBO(\gamma,D;\boldsymbol{\pi},\boldsymbol{\mu},\boldsymbol{\Sigma}),$$

$$s.t.\quad\sum_{k=1}^{K}\pi_k=1$$

利用拉格朗日乘数法求解上面的等式约束优化问题，可得

$$\pi_k=\frac{N_k}{N}$$

$$\boldsymbol{\mu}_k=\frac{1}{N_k}\sum_{n=1}^{N}\gamma_{nk}\boldsymbol{x}^{(n)}$$

$$\boldsymbol{\Sigma}_k=\frac{1}{N_k}\sum_{n=1}^{N}\gamma_{nk}(\boldsymbol{x}^{(n)}-\boldsymbol{\mu}_k)(\boldsymbol{x}^{(n)}-\boldsymbol{\mu}_k)^T$$

其中

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

2，代码

由上述公式，可以编写高斯混合模型代码如下：

```
class GMM():
    def __init__(self, dim, k, iteration):
        self.dim = dim
        self.k = k
        self.iteration = iteration
        self.pi = np.empty((k, 1))
        self.mu = np.empty((k, dim))
        self.sigma = np.empty((dim, dim))
        self.gamma = None

    #第一种初始化方法
    def initmethod1(self, n, x):
        self.pi = np.ones(self.k, dtype=np.float) / self.k
        self.mu = np.array([x[np.random.randint(0, len(x))] for i in ra
nge(self.k)], dtype=np.float)
        self.sigma = np.full((self.k, self.dim, self.dim), np.identity(
self.dim), dtype=np.float)

    #计算 gamma
    def Gamma(self, n, x):
        p_x_n = np.zeros(self.k)
        gamma = np.empty((n, self.k))
        for j in range(n):
            for m in range(self.k):
                p_x_n[m] = self.pi[m] * st.multivariate_normal.pdf(x[j]
, mean=self.mu[m], cov=self.sigma[m])
            for m in range(self.k):
                gamma[j][m] = p_x_n[m] / np.sum(p_x_n)
        return gamma

    #训练获得参数
    def train(self, n, x, method):
        if method == 1:
            self.initmethod1(n, x)
        for i in range(self.iteration):
            self.gamma = self.Gamma(n, x)
            N = np.sum(self.gamma, axis=0)
            self.pi = N / n
```

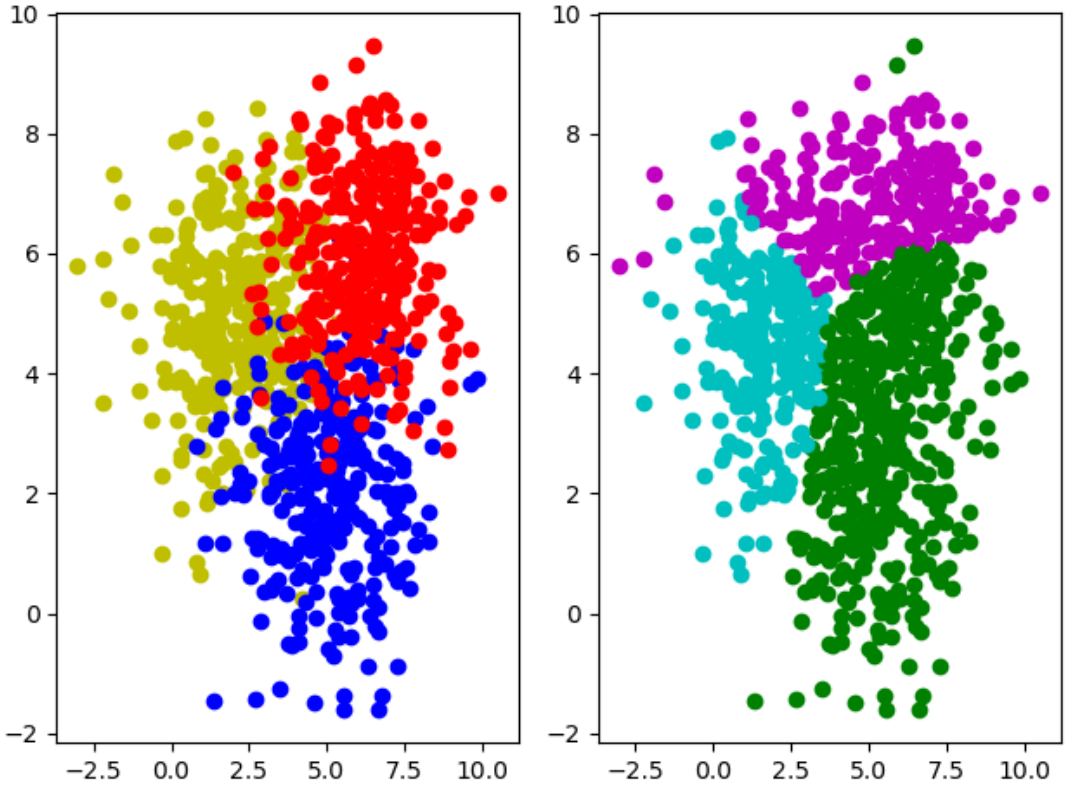
```
            self.mu = np.array([1 / N[p] * np.sum(x * self.gamma[:, p].
reshape((n, 1)), axis=0) for p in range(self.k)])
            for m in range(self.k):
                sum_sigma = np.zeros((self.k, self.dim, self.dim))
                for j in range(n):
                    sum_sigma[m] += self.gamma[j][m]*np.dot((x[j] - sel
f.mu[m])[:, None], (x[j] - self.mu[m])[None, :])
                self.sigma[m] = sum_sigma[m]/N[m]
        return self.gamma

    #对数据集分类返回标记数组
    def classify(self, n, x):
        t = np.empty(n)
        for j in range(n):
            t[j] = np.argmax(self.gamma[j])
        return t
```

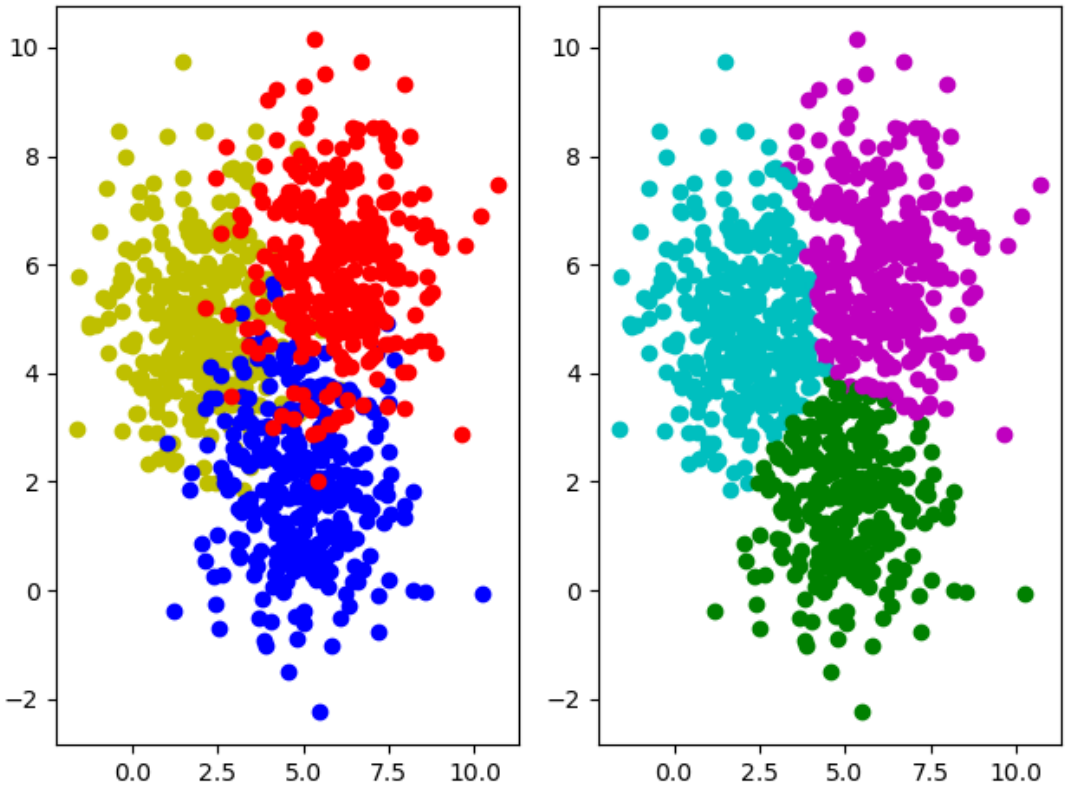
此时采取的初始化方法为随机在数据集中选择三个点组成$\boldsymbol{\mu}$的初始值，使用三个二维单位阵组成$\boldsymbol{\Sigma}$的初始值，π初始采取均匀分布。后续将会**针对初始化方法**进行进一步讨论，并为 GMM 类添加新的初始化方法

3， 基本实验

迭代 100 代，结果如下:



迭代 200 代，结果如下:



其中用红黄蓝标识的为数据集真实分类情况，用青紫绿标识的为模型给出的分类，观察两张图可以发现，模型已经具备基本的分类能力，但是对于三个类交界上的点分类情况不好。

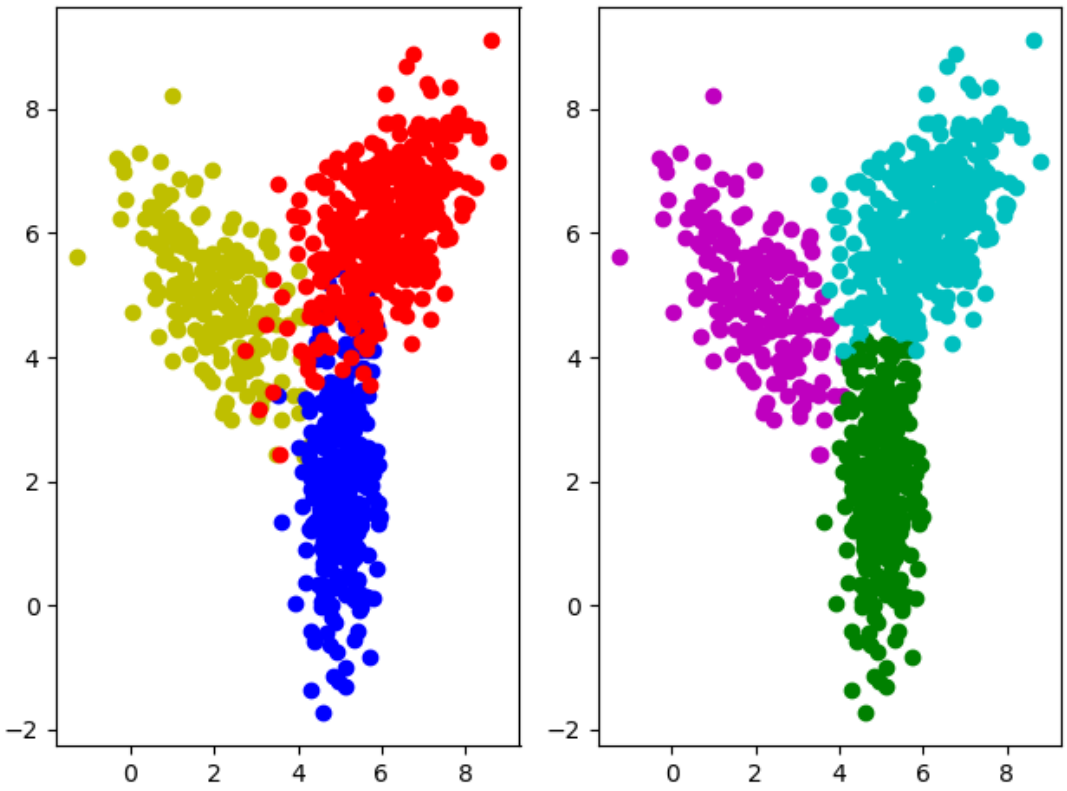
4， 讨论

4.1 改变聚类形状

初始数据集我们设置的三个高斯分布协方差矩阵相同且规则，不妨更换协方差矩阵如下

```
cov = np.array([
    [[1, -0.5],
     [-0.5, 1]],
    [[0.2, 0],
     [0, 2]],
    [[1, 0.5],
     [0.5, 1]]
])
```

同时更改数据集中各高斯分布的数据量，由 300，300，300 改为 200，300，400。
迭代 200 代，结果如下



由于重合度降低，比第一个数据集上的分类效果还要好一些。由此可以推理，聚类形状对于高斯混合模型影响有限，但是对于重合边界上的点，高斯分布的效果就要差一些。当然，用高斯混合模型进行分类是一种**软分类**，对于每一个点都会先给出一个概率分布，对于边界上的点留下了进一步处理的空间。

4.2 不同的初始化方法

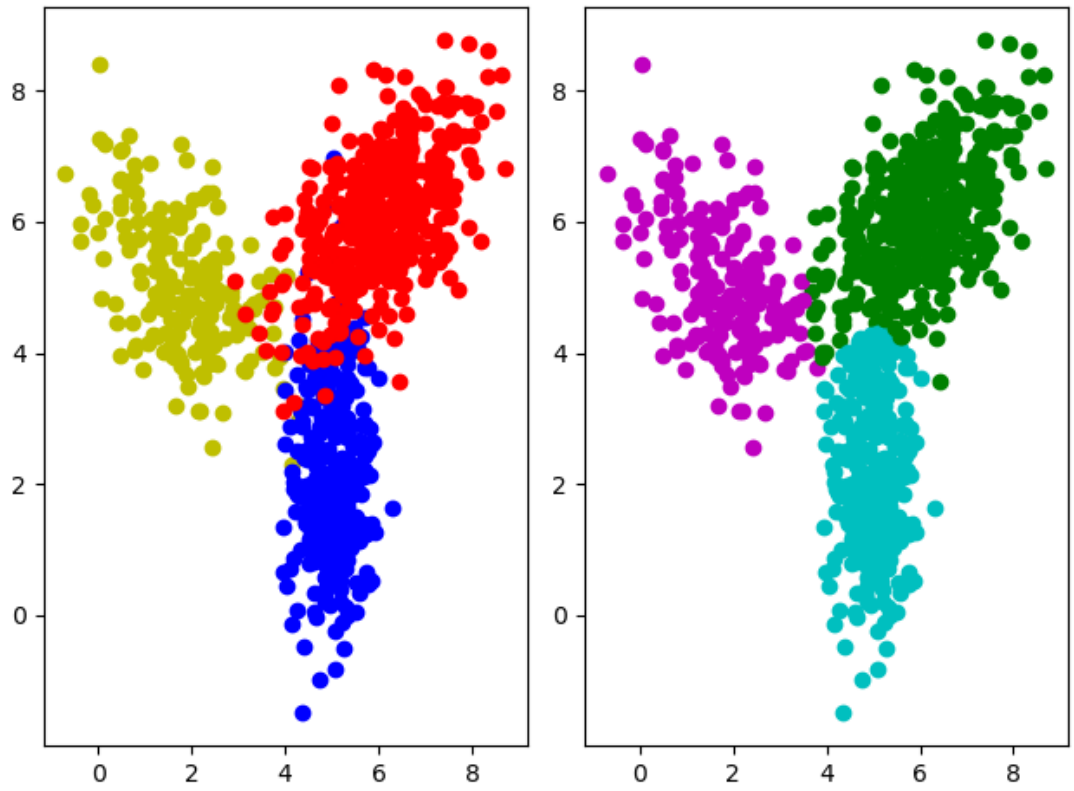
查阅资料可知，EM 算法对于参数初始值十分敏感，虽然目前的实验还没有表现出这一点，但是初始化方法依然是值得讨论的主题。我们在这里尝试 kmeans++算法首先找到 k 个聚类中心，然后参考 https://arxiv.org/pdf/1312.5946v2.pdf 这篇文章的方法计算协方差矩阵（并没有实现论文中的算法，仅仅借用了一个算协方差的方法）。
代码如下：

```
#第二种初始化方法:kmeans++
    def initmethod2(self, n, x):
        #随机选取第一个聚类中心
        self.mu[0] = x[np.random.randint(0, len(x))]
        p = np.empty(n)
        for q in range(1, self.k, 1):
            for i in range(n):
                d = np.array([np.linalg.norm(x[i] - self.mu[j]) for j i
n range(q)])
```

```
                p[i] = d[np.argmin(d)]
            p_sum = np.sum(p)
            #轮盘法依次选择 k-1 个聚类中心
            select = np.random.rand()
            for i in range(n):
                select = select - p[i]/p_sum
                if select < 0:
                    self.mu[q] = x[i]

        pi = np.zeros((self.k, 1))
        mu = np.zeros((self.k, self.dim))
        sigma = np.zeros((self.k, self.dim, self.dim))
        t = np.empty(n, dtype=int)
        while(True):
            for i in range(n):
                d = np.array([np.linalg.norm(x[i]-
self.mu[p]) for p in range(self.k)])
                t[i] = np.argmin(d)        #找到最近的聚类中心
                pi[t[i]] += 1
                mu[t[i]] += x[i]
            for i in range(self.k):
                if pi[i] == 0:
                    mu[i] = self.mu[i]
                else:
                    mu[i] = mu[i]/pi[i]
            if((mu==self.mu).all()):       #收敛条件
                break
            else:
                self.mu = mu
                self.pi = pi/n
            pi = np.zeros((self.k, 1))
            mu = np.zeros((self.k, self.dim))
        #根据初始分类计算 sigma
        for i in range(n):
            sigma[t[i]] += np.dot((x[i] - self.mu[t[i]])[:, None], (x[i
] - self.mu[t[i]])[None, :])
        for i in range(self.k):
            self.sigma[i] = sigma[i]/pi[i]
            if np.linalg.det(self.sigma[i]) == 0:
                self.sigma[i] = np.identity(self.dim)
```

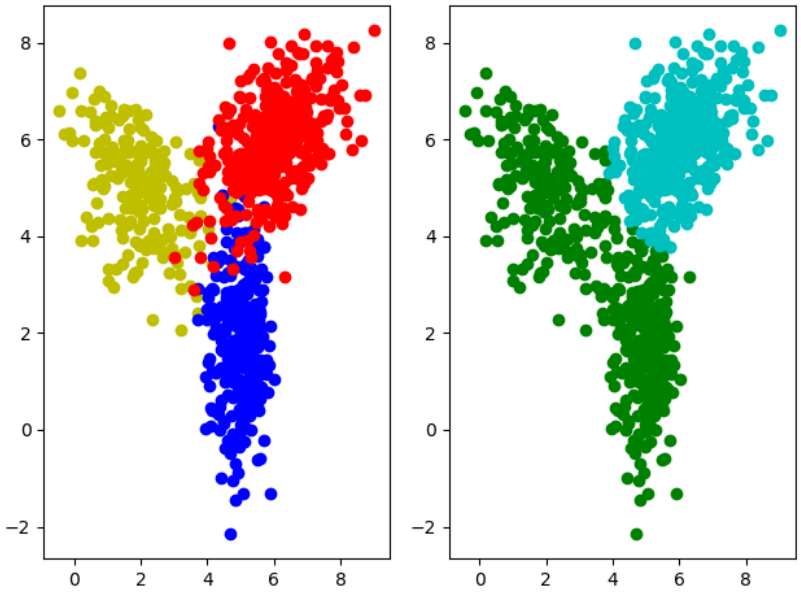
使用这种初始化方法，使用**改变聚类形状后**的数据集，**迭代 200 次**，结果如下



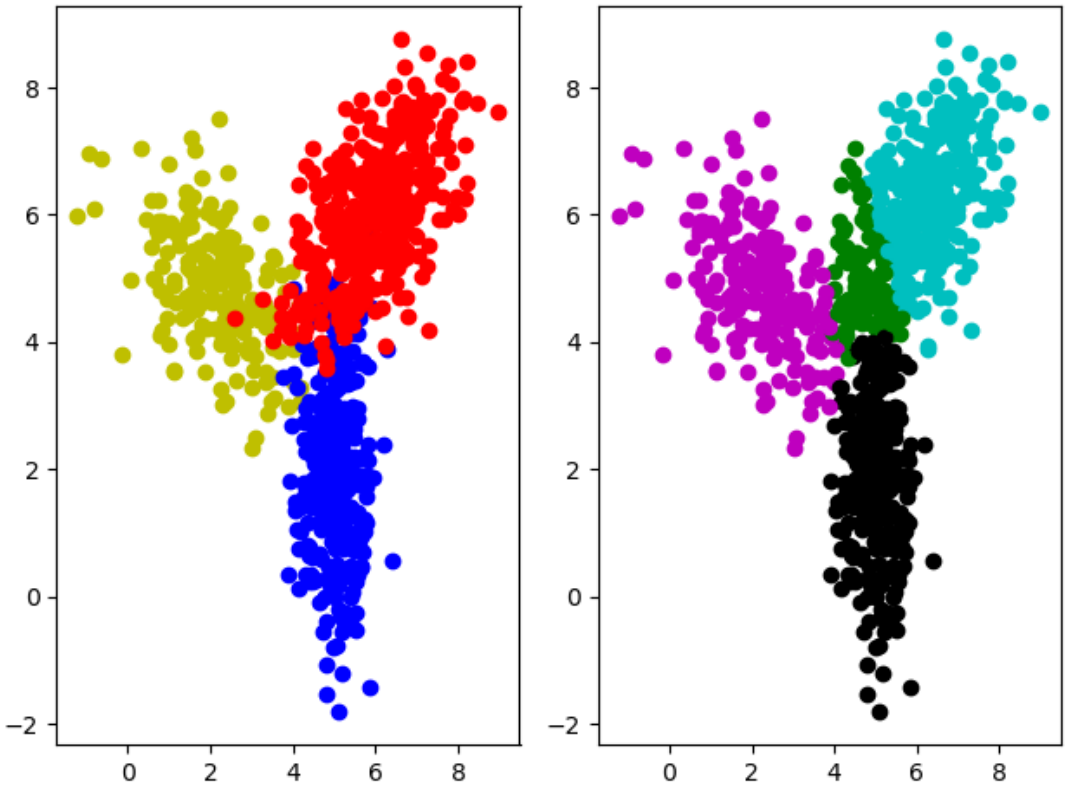
效果与第一种初始化方法相差不大。可以推理，不同的初始化方法可能无法很大提升模型对于边界区域点的处理能力，但是可以提高模型稳定性，避免出现收敛到局部极值点的情况。

4.3 不同 k 值

之前讨论中 k 值均设定与数据集中高斯分布的个数相同，这里我们讨论假如出现误差的情况。设置 k 为 **2**，使用**改变聚类形状后**的数据集，使用第一种初始化方法，**迭代 200 次**，结果如下：



设置 k 为 **4**，使用**改变聚类形状后**的数据集，**迭代 200 次**，结果如下：



可以看到，k 值的少许偏离不会对模型的能力产生太大影响，只不过由于 k 值的限制，某两个类会合并到一个类，或是一个类分成两个类，不会产生混乱。

5 启动命令

python source.py

(训练过程需要等待一点时间)