本节主要介绍非监督学习的聚类算法。

# 常见的聚类算法

聚类问题中，我们给定训练集合，目的是将训练样本聚合几个类中。由于问题过程中，y并没有指定，所以这是一个非监督问题。

## k-means

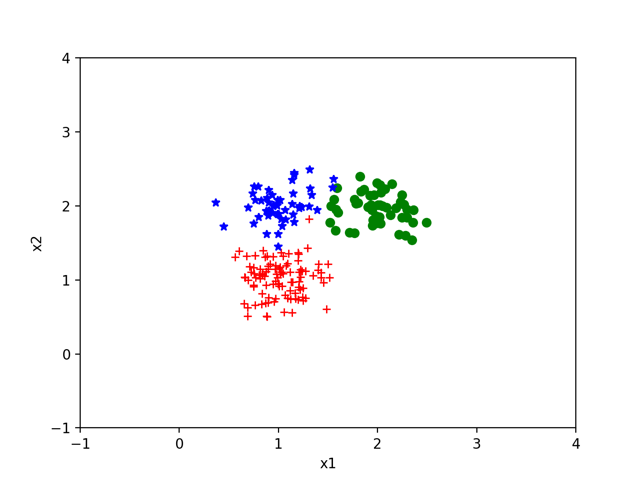
下面介绍k-means算法。算法的主要流程如下：

1. 随机初始化重心(假设有k个分类)
2. 根据当前的，计算距离每个样本距离最近的中心，即为所属类。
3. 然后根据2中得到新的所属类关系，更新一组新的重心值。
4. 重复2,3直到某个截止条件。

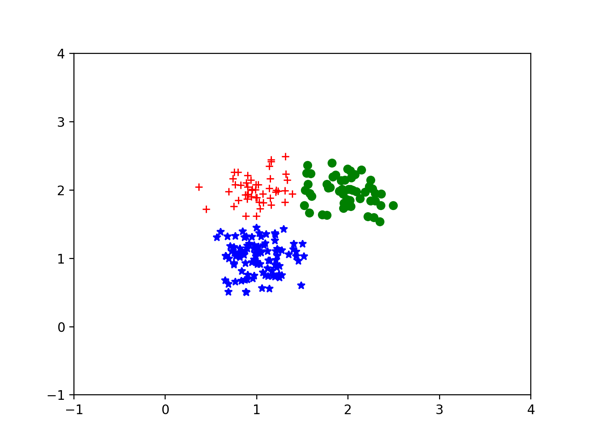
下面我们随机制造以三个点为高斯分布的一组数据，试图从该组数据完成聚类操作，具体代码如下：

**import** numpy **as** np  
**from** matplotlib **import** pyplot **as** plt  
**import** sys  
  
**def** makeData():  
 *# 0 make data* mean\_1 = [1, 1]  
 mean\_2 = [2, 2]  
 mean\_3 = [1, 2]  
 cov = [[0.05, 0], [0, 0.05]]  
 arr1 = np.random.multivariate\_normal(mean\_1, cov, 100)  
 arr2 = np.random.multivariate\_normal(mean\_2, cov, 50)  
 arr3 = np.random.multivariate\_normal(mean\_3, cov, 50)  
 figure, ax = plt.subplots()  
 ax.set\_xlim(left=-1, right=4)  
 ax.set\_ylim(bottom=-1, top=4)  
 **for** i **in** range(len(arr1)):  
 plt.plot(arr1[i][0], arr1[i][1], **'b--'**, marker=**'+'**, color=**'r'**)  
 **for** i **in** range(len(arr2)):  
 plt.plot(arr2[i][0], arr2[i][1], **'b--'**, marker=**'o'**, color=**'g'**)  
 **for** i **in** range(len(arr3)):  
 plt.plot(arr3[i][0], arr3[i][1], **'b--'**, marker=**'\*'**, color=**'b'**)  
 plt.xlabel(**"x1"**)  
 plt.ylabel(**"x2"**)  
 **return** np.vstack((arr1,arr2,arr3))  
  
**def** updateSingleLabel(x,center):  
 n = len(center)  
 min = sys.maxsize  
 minIndex = -1  
 **for** i **in** range(n):  
 tmp = np.linalg.norm(x - center[i])  
 **if** tmp < min:  
 minIndex=i  
 min = tmp  
 **return** minIndex  
  
**def** updateLable(data,label,center):  
 numChanged = 0;  
 n=len(data)  
 label\_new = np.zeros(n);  
 **for** i **in** range(n):  
 label\_new[i] = updateSingleLabel(data[i], center)  
 **if** label\_new[i] != label[i]:  
 numChanged = numChanged + 1  
 **for** i **in** range(n):  
 label[i] = label\_new[i]  
 **return** numChanged;  
  
**def** updateCenter(data,label,center):  
 newCenter=np.array([[0.0,0.0],[0.0,0.0],[0.0,0.0]])  
 newCenterSum=np.array([0,0,0])  
 **for** i **in** range(len(data)):  
 **if** label[i]==0:  
 newCenter[0] = newCenter[0] + data[i]  
 newCenterSum[0] = newCenterSum[0] + 1  
 **elif** label[i] == 1:  
 newCenter[1] = newCenter[1] + data[i]  
 newCenterSum[1] = newCenterSum[1] + 1  
 **elif** label[i] == 2:  
 newCenter[2] = newCenter[2] + data[i]  
 newCenterSum[2] = newCenterSum[2] + 1  
 **if** newCenterSum[0] > 0 :  
 center[0] = newCenter[0]/newCenterSum[0]  
 **if** newCenterSum[1] > 0 :  
 center[1] = newCenter[1]/newCenterSum[1]  
 **if** newCenterSum[2] > 0 :  
 center[2] = newCenter[2]/newCenterSum[2]  
  
**def** showPic(x,label,label1):  
 figure, ax = plt.subplots()  
 ax.set\_xlim(left=-1, right=4)  
 ax.set\_ylim(bottom=-1, top=4)  
 **for** i **in** range(len(x)):  
 **if** label[i] == 0:  
 plt.plot(x[i][0], x[i][1], **'b--'**, marker=**'+'**, color=**'r'**)  
 **elif** label[i] == 1:  
 plt.plot(x[i][0], x[i][1], **'b--'**, marker=**'o'**, color=**'g'**)  
 **elif** label[i] == 2:  
 plt.plot(x[i][0], x[i][1], **'b--'**, marker=**'\*'**, color=**'b'**)  
 figure, ax = plt.subplots()  
 ax.set\_xlim(left=-1, right=4)  
 ax.set\_ylim(bottom=-1, top=4)  
 **for** i **in** range(len(x)):  
 **if** label1[i] == 0:  
 plt.plot(x[i][0], x[i][1], **'b--'**, marker=**'+'**, color=**'r'**)  
 **elif** label1[i] == 1:  
 plt.plot(x[i][0], x[i][1], **'b--'**, marker=**'o'**, color=**'g'**)  
 **elif** label1[i] == 2:  
 plt.plot(x[i][0], x[i][1], **'b--'**, marker=**'\*'**, color=**'b'**)  
 plt.show()  
  
**if** \_\_name\_\_==**"\_\_main\_\_"**:  
  
 *# 0 make data* data=makeData()  
  
 *# 1 initial* n = len(data)  
 label = np.zeros(n) *# 0,1,2 represent center[0],center[1],center[2]* center = np.array([[0.0,3.0],[3.0,3.0],[0.0,0.0]])  
 label1 = np.zeros(n)  
 center1 = np.array([[0.0,10.0],[2.0,3.0],[0.5,5.0]])  
  
 *# 2. trainning  
 # 2.1 trainning for good initial* **while True**:  
 *# update label* numChanged = updateLable(data,label,center)  
 *# update center* updateCenter(data,label,center)  
 **if** numChanged==0:  
 **break** *# 2.1 trainning for bad initial* **while True**:  
 *# update label* numChanged = updateLable(data,label1,center1)  
 *# update center* updateCenter(data,label1,center1)  
 **if** numChanged==0:  
 **break** *# 3. showData* showPic(data,label,label1)

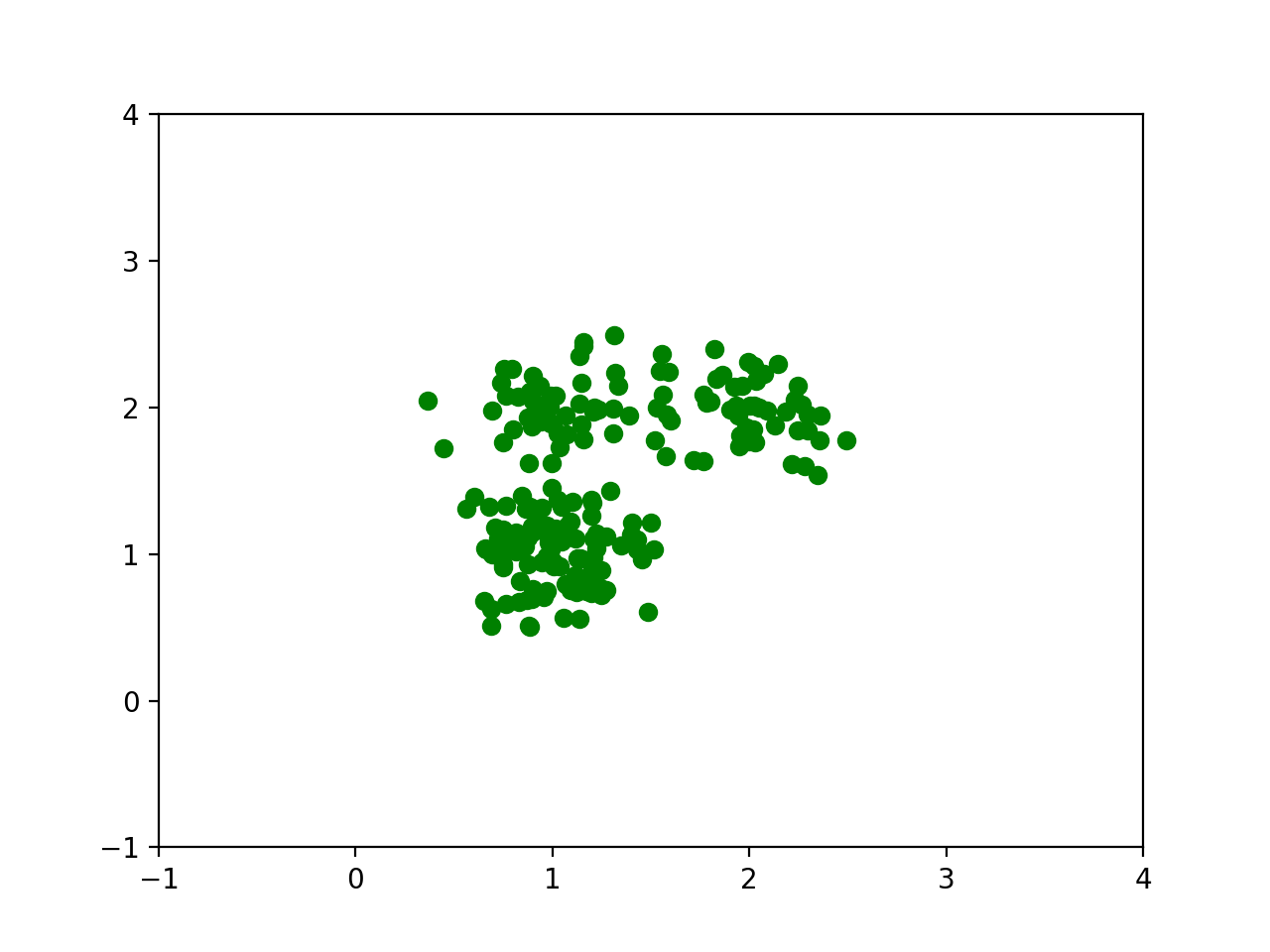
下面是随机产生的基于（1，1），（2，2），（1，2）的高斯分布。



然后我们从(0.0,3.0),(3.0,3.0),(0.0,0.0)开始迭代得到如下效果，可以看出结果还是非常理想的。



然后我们试图从(0.0,10.0),(2.0,3.0),(0.5,5.0)开始迭代,则会得到这样的结果。



可以从上文中看出，k-means对初始值的敏感度很高。对于现实问题，也许我们并不知道训练样本中本身存在多个分类，我们可以设置多个分类，如果某一个分类里面的样本过少，就删除分类，这样就不再依赖于事先知道分类的数量了。

## 1.2高斯混合聚类

假设我们的分类都服从于各自的高斯分布，我们试图从样本中对其分类。该问题与之前的高斯判别分析类似，区别仅仅在于该问题没有样本标签。

我们设置z表示样本的距离分类，可以知道他服从一个多项式分布（），其中。然后已经z之后，x服从的是一个高斯分布，。

下面直接写出算法过程(具体的算法推导见EM算法小节)：

1. 随机初始化,,
2. 遍历样本，计算得,如下：





1. 根据计算得到的w重新更新各个分类的分布，如下：





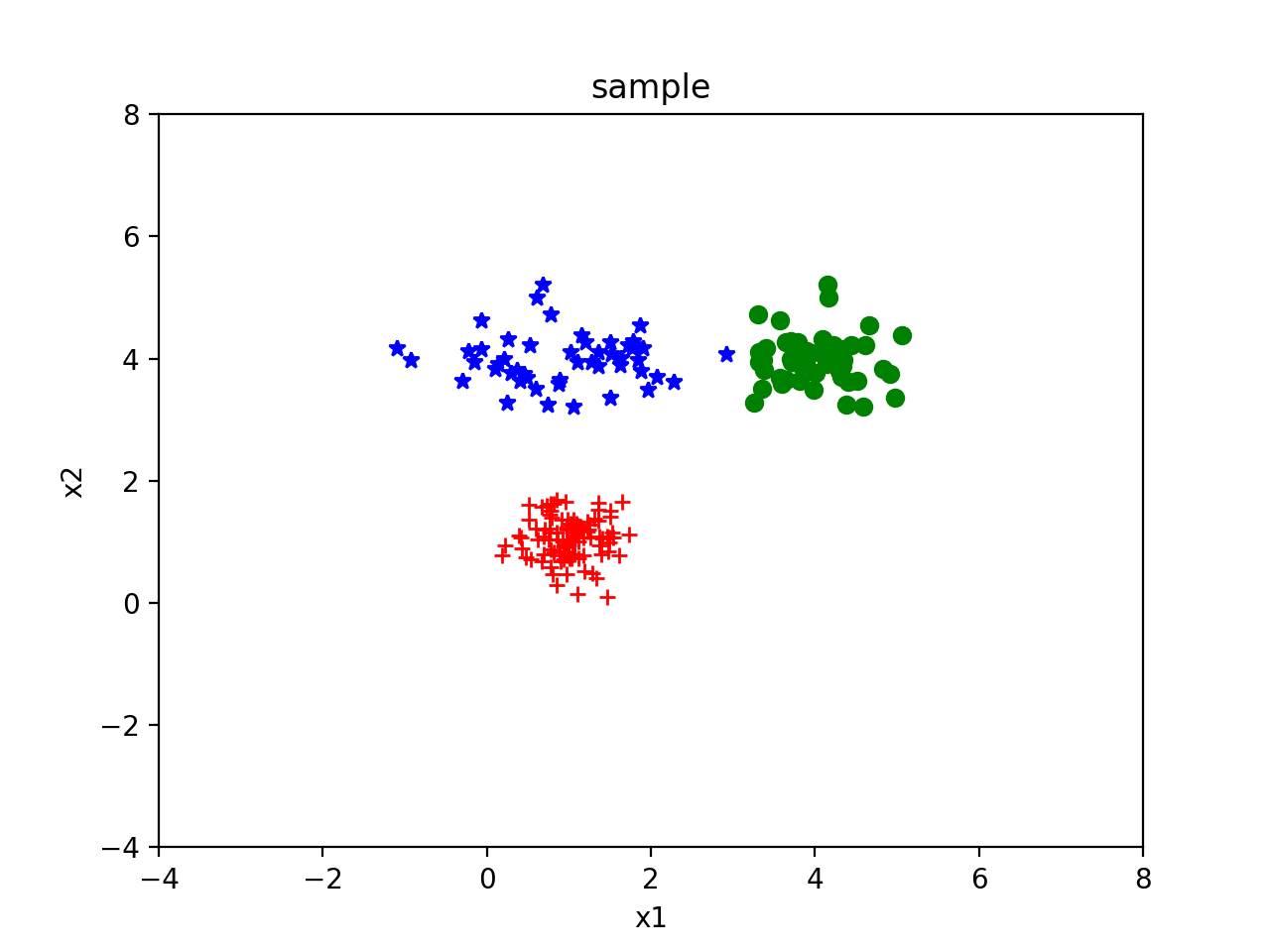


1. 重复2,3知道达到截止条件。

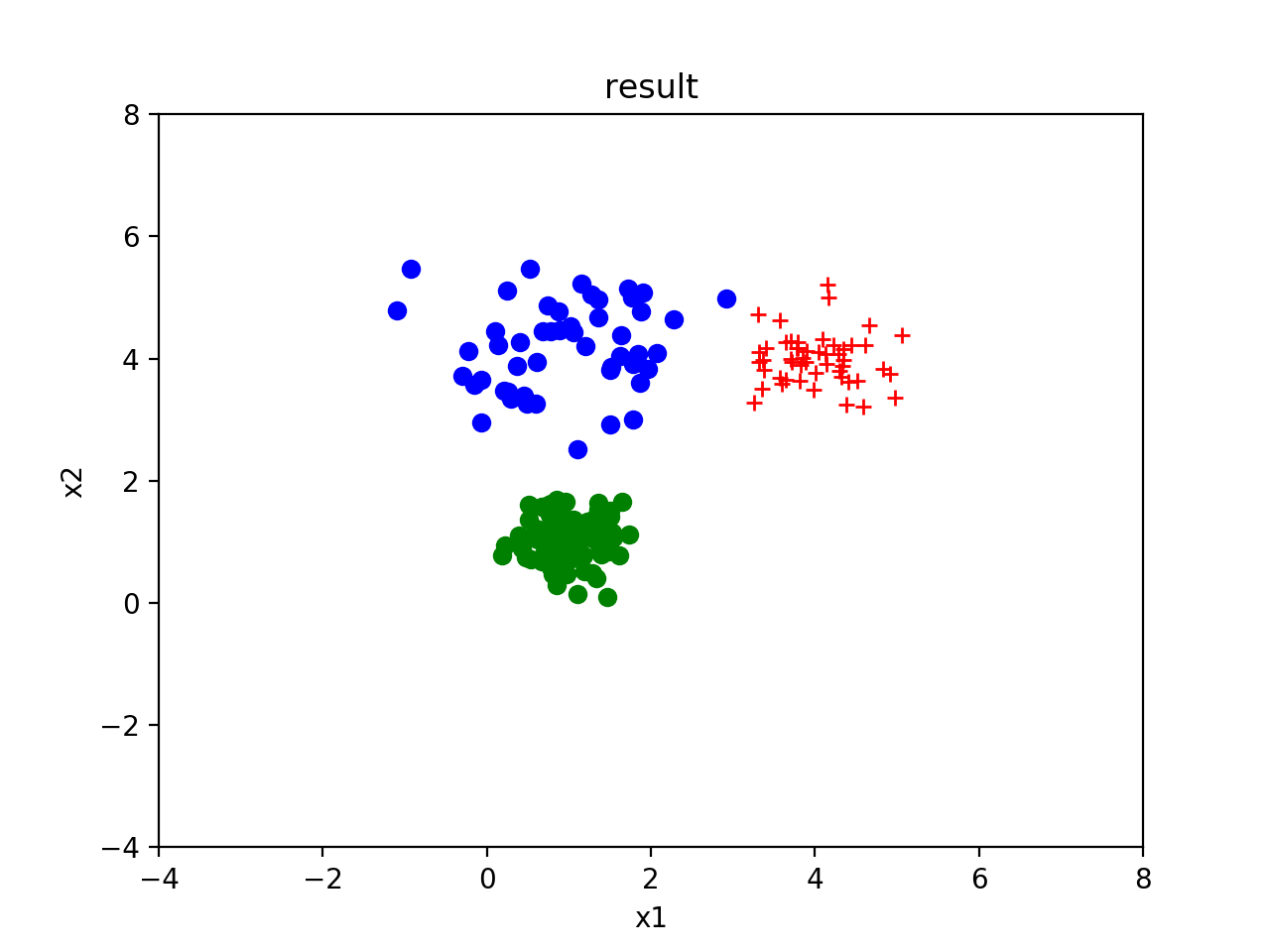
下面我们制作一组由三个高斯分布组成的样本数据，对其进行聚类。代码如下:

*# coding=utf-8***import** numpy **as** np  
**from** matplotlib **import** pyplot **as** plt  
**from** scipy.stats **import** multivariate\_normal  
  
**def** make\_data():  
 mean\_1 = [1,1]  
 mean\_2 = [4,4]  
 mean\_3 = [1,4]  
 cov1 = [[0.1,0],[0,0.1]]  
 cov2 = [[0.2,0],[0,0.2]]  
 cov3 = [[0.6,0],[0,0.6]]  
 arr1 = np.random.multivariate\_normal(mean\_1, cov1, 100)  
 arr2 = np.random.multivariate\_normal(mean\_2, cov2, 50)  
 arr3 = np.random.multivariate\_normal(mean\_3, cov3, 50)  
 figure, ax = plt.subplots()  
 ax.set\_xlim(left=-4, right=8)  
 ax.set\_ylim(bottom=-4, top=8)  
 **for** i **in** range(len(arr1)):  
 plt.plot(arr1[i][0], arr1[i][1], **'b--'**, marker=**'+'**, color=**'r'**)  
 **for** i **in** range(len(arr2)):  
 plt.plot(arr2[i][0], arr2[i][1], **'b--'**, marker=**'o'**, color=**'g'**)  
 **for** i **in** range(len(arr3)):  
 plt.plot(arr3[i][0], arr2[i][1], **'b--'**, marker=**'\*'**, color=**'b'**)  
 plt.xlabel(**"x1"**)  
 plt.ylabel(**"x2"**)  
 plt.plot()  
 plt.title(**"sample"**)  
 *#plt.show()* **return** np.vstack((arr1, arr2, arr3))  
  
**def** updatePhi(w):  
 n1 = len(w)  
 phi=np.zeros(n1)  
 **for** i **in** range(n1):  
 sum = 0  
 n2 = len(w[i])  
 **for** j **in** range(n2):  
 sum = sum + w[i][j]  
 phi[i] = sum/n2  
 **return** phi  
  
**def** updateW(data,w,phi,mu,sigma):  
 n1=len(w) *# 3* n2 = len(w[0]) *# 200* var = []  
 **for** k **in** range(n1):  
 var.append(multivariate\_normal(mean=mu[k].tolist(), cov=sigma[k].tolist()))  
  
 **for** j **in** range(n2):  
 sum = 0  
 **for** i **in** range(n1):  
 sum = sum + var[i].pdf(data[j])\*phi[i]  
 **for** i **in** range(n1):  
 w[i][j] = var[i].pdf(data[j])\*phi[i]/sum  
  
**def** updateMu(data,w,mu):  
 changed=**False** n1 = len(w) *# 3* n2 = len(w[0]) *# 200* **for** i **in** range(n1):  
 sumW = 0.0  
 sumX = np.array([0.0,0.0])  
 **for** j **in** range(n2):  
 sumW = sumW + w[i][j]  
 sumX = sumX + w[i][j]\*data[j]  
 mu\_new = sumX / sumW  
 **if** np.dot(mu\_new-mu[i],mu\_new-mu[i]) > 0.001:  
 changed = **True** mu[i] = mu\_new  
 **return** changed  
  
**def** updateSigma(data,w,mu,sigma):  
 n1 = len(w) *# 3* n2 = len(w[0]) *# 200* sum=np.array([[0,0],[0,0]])  
 **for** i **in** range(n1):  
 sumW = 0.0  
 sumX = np.array([0.0,0.0])  
 **for** j **in** range(n2):  
 sumW = sumW + w[i][j]  
 z0 = np.array([data[j] - mu[i]])  
 z0T = np.array([data[j] - mu[i]]).transpose()  
 sumX = sumX + w[i][j]\*np.dot(z0T, z0)  
 sigma[i] = sumX/sumW  
  
**def** classify(data,mu,sigma):  
 n1=len(w) *# 3* n2 = len(w[0]) *# 200* var=[]  
 **for** k **in** range(n1):  
 var.append(multivariate\_normal(mean=mu[k].tolist(), cov=sigma[k].tolist()))  
  
 figure, ax = plt.subplots()  
 ax.set\_xlim(left=-4, right=8)  
 ax.set\_ylim(bottom=-4, top=8)  
  
 **for** i **in** range(n2):  
 tmp\_arr=np.array([])  
 **for** j **in** range(n1):  
 tmp\_arr=np.append(tmp\_arr,var[j].pdf(data[i]))  
 index = tmp\_arr.argmax()  
 **if** index == 0:  
 plt.plot(data[i][0], data[i][1], **'b--'**, marker=**'+'**, color=**'r'**)  
 **elif** index == 1:  
 plt.plot(data[i][0], data[i][1], **'b--'**, marker=**'o'**, color=**'g'**)  
 **elif** index == 2:  
 plt.plot(data[i][0], data[i][1], **'b--'**, marker=**'o'**, color=**'b'**)  
 plt.xlabel(**"x1"**)  
 plt.ylabel(**"x2"**)  
 plt.title(**"result"**)  
 plt.plot()  
 plt.show()  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 *# 1 make data* classN=3  
 data=make\_data()  
 n=len(data)  
 **for** k **in** range(n):  
 w = np.array([np.zeros(n), np.zeros(n), np.zeros(n)])  
 *# w = array[classN][n] =array[3][200]* **for** i **in** range(classN):  
 **for** j **in** range(n):  
 w[i][j]= 1.0/classN  
  
 phi=updatePhi(w)  
 mu = np.array([[6.0, 6.0], [4.0, -1.0], [-2.0, 2.0]])  
 sigma=np.array([[[0.1,0],[0,0.1]],[[0.1,0],[0,0.1]],[[0.1,0],[0,0.1]]])  
  
 *# 2 training* **while True**:  
 updateW(data,w,phi,mu,sigma)  
 updatePhi(w)  
 updateSigma(data,w,mu,sigma)  
 changed = updateMu(data,w,mu)  
 **if** changed == **False**:  
 print(**"迭代完成"**)  
 **break** *# 3 show* print(**"mu="**,mu,**", sigma="**,sigma)  
 classify(data,mu,sigma)

我们分别以[6.0, 6.0], [4.0, -1.0], [-2.0, 2.0]为均值，以[[0.1,0],[0,0.1]],[[0.1,0],[0,0.1]],[[0.1,0],[0,0.1]]为标准差生成一组高斯分布如下：



然后通过训练的到的训练均值分别为[4.00729834, 3.99848889], [1.01089497, 1.05225006], [0.92949217, 4.18380895]]。标准差为[[0.23196114,-0.01896477],[-0.01896477, 0.17165715]], [[0.10205901, 0.00157192], [0.00157192, 0.11477843]], [[0.7010517, 0.04783335], [0.04783335, 0.51147277]]。具体结果如下:

**

# 2 EM算法

## 2.1 Jensen不等式

对于一个严格凸函数(即)，我们容易得到下式:



当且仅当x为常数的时候，上式等号成立。对于严格凹函数，则正好相反。

## 2.2 EM算法模型建立

2.2.1 EM算法公式推导

假定我们的数据有k个分类。我们聚类的目标是，样本在自己分类中出现的概率最大。或者换句话说，让其在所属分类的分布(可以用高斯分类假想该问题)中出现的概率最大。可是对于非监督学习问题，我们不知道具体的分类。因此，我们可以将模型假定为找到给定参数对应分布，是的x在分布中出现的概率足够大，这说明对应的分布能够充分的表示某一组分类。因此可以构造如下的最大释然函数：



进入推导最大释然函数：



上面k为分类的个数。对上面的为一个概率分布，有。对于,我们知道,可知其为一个严格凹函数。上式可以写成：



根据Jensen不等式，我们有:



因此我们得到最大释然函数的下确定，即为上式子中后面的部分。我们只要保证下确定随着迭代的方向单调递增即可，具体的要保证。这里我们设是已知的情况下，最大释然函数的下确定关于的函数。我们看如下公式:



事实上，我们只要保证上面的式子我们就可以保证迭代方向是正确的。其中，第一个不等号是必然成立的。那我们分头来构造条件是后面的式子成立。

首先如果保证最后一个等号的成立，根据Jensen不等式，只有自变量为常数才能事等号成立，这样我们设置如下式子(其中c为常数):



然后对上面的式子按照分类累积求和得:



因此得:



这样我们完成EM算法的E步骤。然后解决中间大于等于号的问题。这个就比较好解决了，我们只需要计算下确定函数关于求最大值，最大值对应的参数，可保证不等号的成立，即为迭代的正确方向。具体公式如下：



综上，我们来重新整理一下EM算法，具体如下：

1. 初始化相关参数
2. E步骤:计算，如下：



1. M步骤：更新参数，如下：



1. 重复2,3直到截止条件

注: 上面的例子是不断的更新迭代和。我们完全可以使用梯度上升发不断从各个方向更新和，来完成最大值的逼近。

## 2.3 混合高斯分布的公式推导

对比之前的内容，我们可以发现混合高斯分布的聚类问题为EM算法的一个特例。可以通过通用的EM算法来证明，下面我们来证明这一过程。

注： 这里只简单地提示计算，不展开了，因为与之前的高斯判别分析类似。

根据上一节，我们已得到E步骤的公式：



然后对于M步骤，我们将为一个已知值的方式对进行求导，从而求得下确定的最大值，记为下一个迭代值。将最大释然函数展开记为如下函数：



然后对,,,即可得到M步骤的更新公式。