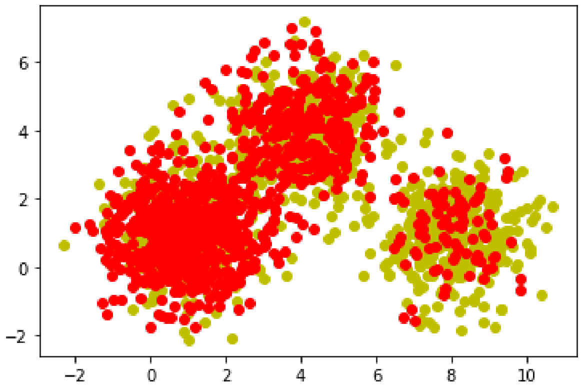
实验三：参数估计

数据集实现

首先可以基于多元正态分布实现出对应的数据集，在这一过程中，使用`np.random.choice`，可以得到一系列随机数后，构造出两个数据集：

```
conv=[[1,0],[0,1]]
x_mean=[1,4,8]
y_mean=[1,4,1]
def do_sample(size,distribution=None):
    X_1=np.random.choice([1,2,3],size=1000,p=distribution)
    normal_sample=[]
    for i in range(size):
        if X_1[i]==1:
            L=list(np.random.multivariate_normal((1,1),cov=conv))
            L.append(0)
            normal_sample.append(L)
        elif X_1[i]==2:
            L=list(np.random.multivariate_normal((4,4),cov=conv))
            L.append(1)
            normal_sample.append(L)
        else:
            L=list(np.random.multivariate_normal((8,1),cov=conv))
            L.append(2)
            normal_sample.append(L)
    return normal_sample
```

其结果可视化为：



最大后验估计实现

在已有的数据集上，根据MAP公式，可以计算出：

$$p(\theta|X) = \frac{P(X|\theta)p(\theta)}{p(X)}$$

即可得到各个类别所对应的似然概率，根据其似然函数的大小比较，即可得到其应该被归于哪一类别：

```
def MAP(x,y,ux,uy,p):
    return math.exp(-0.25*((x-ux)**2+(y-uy)**2))*p/(math.pi*4)#概率计算
def single_evalue(item,probability):
```

```
    score=[]
    for i in np.arange(0,3):
        score.append(MAP(item[0],item[1],x_mean[i],y_mean[i],probability[i]))
    res=max(score)
    for ite in np.arange(0,3):
        if(score[ite]==res):
            return ite
acc1=0
for iter_num in np.arange(0,1000):
    if(single_evalue(sample_2[iter_num],[0.6,0.3,0.1])==sample_2[iter_num][2]):
        acc1=acc1+1
print(acc1/1000)
acc2=0
for iter_num in np.arange(0,1000):
    if(single_evalue(sample_1[iter_num],[1/3,1/3,1/3])==sample_1[iter_num][2]):
        acc2=acc2+1
print(acc2/1000)
```

即完成了MAP的分类，可以观察到，在这一算法中，在两个数据集上，其准确率均达到了95%+,取得了良好的效果。

高斯核函数实现

对应的，使用高斯核函数对概率密度进行计算，并进一步求出了对应的似然值：

在这一步骤下，分别通过普通窗口函数：

$$p(x) = \frac{1}{N}\sum_{n=1}^{N}\frac{1}{h^D}\phi(\frac{x-x_n}{h})$$

与函数：

$$p(x) = \frac{1}{N}\sum_{n=1}^{N}\frac{1}{\sqrt{2\pi h^2}}\exp\{-\frac{||x-x_n||^2}{2h^2}\}$$

进行计算，并进行比较：

```
H=[0.1,0.5,1,1.5,2]
def density_cal_window(k):
    return 1/k**2  #D=2
def gaussian_cal(k,tar_x,tar_y,x_i,y_i):
    return 1/(math.sqrt(2*math.pi*k**2))*np.power(np.e,(-((tar_x-x_i)**2+(tar_y-
y_i)**2)/2*k**2))
ef gaussian_KDE(k,sample):
    predict_result=[]
    acc=0
    for item in range(1000):
        score=[0,0,0]
        count_class=[0,0,0]
        for i in range(1000):
            score[sample[i][2]]=score[sample[i][2]]+gaussian_cal(k,sample[item]
[0],sample[item][1],x_mean[sample[i][2]],y_mean[sample[i][2]])
        res=max(score)
        for i in np.arange(0,3):
            if (res==score[i]):
                predict_result.append(i)
```

```
    for i in range(0,1000):
        if (predict_result[i]==sample[i][2]):
            acc=acc+1
    print('the accuracy is %.3f ' % (acc/1000))
    acc=0

print('for sample 1:')
for item in H:
    gaussian_KDE(item,sample_1)
print('for sample 2:')
for item in H:
    gaussian_KDE(item,sample_2)
#KDE(H[3],sample_1)
def KDE(k,sample):
    predict_result=[]
    acc=0
    for item in range(1000):
        score=[0,0,0]
        for i in range(1000):
            if (item==i):
                continue
            elif(sample[item][0]-sample[i][0]<=k and sample[item][1]-sample[i]
[1]<=k ):
                score[sample[i][2]]=score[sample[i][2]]+density_cal_window(k)
        res=max(score)
        for i in np.arange(0,3):
            if (res==score[i]):
                predict_result.append(i)

    for i in range(0,1000):
        if (predict_result[i]==sample[i][2]):
            acc=acc+1
    print('the accuracy is %.3f ' % (acc/1000))

for item in H:
    KDE(item,sample_1)
print('for sample 2:')
for item in H:
    KDE(item,sample_2)
```

对其使用交叉验证以及最大似然估计，可以计算得出：在h值为1时，取得了最好的效果；而与直接使用窗口函数进行计算的结果相比，使用高斯核函数方法可以取得明显的准确率提升，提升大约20%。

KNN实现

对于两个数据集的分类问题，也可以使用KNN算法进行对数据的分类工作：

```
def knn(input_num, sample, k):
    store={}
    for i in np.arange(1000):
        dist=calculate_distance(sample[input_num],sample[i])
        store[i]=dist
    store[input_num]=10000
    dict1=sorted(store.items(),key=lambda x:x[1])
    dict2=dict1[1:k+1]#序号+距离
    choice=[0,0,0]
    for item in dict2:
```

```
            choice[sample[item[0]][2]]=choice[sample[item[0]][2]] +1
        res=max(choice)
        for i in np.arange(3):
            if choice[i]==res:
                maxlable=i
        return maxlable
    def test(sample,K):
        acc=0
        for i in np.arange(1000):
            labeli=knn(i,sample,K)
            if(labeli==sample[i][2]):
                acc=acc+1
        return acc
    K=[1,3,5]
    for item in K:
        result=test(sample_1,item)
        print(result)
    for item in K:
        result=test(sample_2,item)
        print(result)
```

在这一算法下，可以计算出在K=5时取得了最好的结果，其整体效果也达到了98%，超过了第二部分所使用的高斯核估计算法。

总结

在这一实验过程中，完成了基本，中级，提高要求，分别使用了MAP算法，高斯核估计算法以及KNN算法完成了对数据点的归类任务，在这一过程中，对基本的概率计算相关知识以及算法基本实现有了更进一步的认识。