基本要求 (4') 在两个数据集合上分别应用“似然率测试规则”、“最大后验概率规则” 进行分类实验，计算分类错误率，分析实验结果。

因为两个分布的参数都是已经确定的，不需要确定，所以似然率测试规则和最大后验测验规则唯一的区别就在于

```
 Gaussian_function(X[i][0:2], mean[j], cov)  # 计算样本i决策到j类的概率
gmax(p_temp) + 1  # 得到样本i决策到的类
数的索引
[i][2]:
+= 1
 num



X, mean, cov, P):
ape[0]  # 类的个数 3
hape[0]

:
os(3)
class_num):
 Gaussian_function(X[i][0:2], mean[j], cov) * P[j]  # 计算样本i是j类的后验概率
gmax(p_temp) + 1  # 得到样本i分到的类
```

结果：

```
error_ave:
[[0.07037037 0.0729    ]
 [0.07037037 0.0665    ]]
```

中级要求 (1') 在两个数据集合上使用高斯核函数估计方法，应用“似然率测试规则”分类，在 [0.1, 0.5, 1, 1.5, 2] 范围内交叉验证找到最优 h 值，分析实验结果。

其中交叉验证法使用的是留一法，每次用核函数算的时候，都把被测试的它本身给排除掉。

其中用core_func调用Cal_gussWin(核函数)，累加每个类的核函数结果，最终累加最大的那个类就是估计的结果。

```
def Cal_gussWin(test,X,n,h): # 求的是和每一类中所有的点进行一个核函数的求值，然后哪个类最大，就选哪个类
    t=np.zeros(3)
    start=0
    for lable in range(1,4):
        for i in range(start,n):
            if X[i][0]==test[0] and X[i][1]==test[1]:
                continue
            if X[i][2] == lable:

                temp = test-X[i][0:2]
                norm = np.linalg.norm(temp)
                t[lable-1] += 1/np.sqrt(2*np.pi*h**2)*np.exp(-0.5*norm/h**2)
            else:
                start = i+1
                break

        lable += 1
    return t/n


def Core_func(X,class_num,h):
    num = np.array(X).shape[0] # 获得x中的点数量
    error_rate = 0
    for i in range(num):
        p_temp = Cal_gussWin(X[i][0:2],X,num,h)  # 计算样本i决策到j类的概率
        p_class = np.argmax(p_temp) + 1  # 得到样本i决策到的类
        if p_class != X[i][2]:
            error_rate += 1
    return error_rate/num
```

结果：

```
average core error 1 is [[0.06836837 0.06826827 0.06886887 0.06976977]]
average core error 2 is [[0.0707 0.1161 0.3197 0.4   ]]
```

- 其中第一行是分布1 的预测结果，第二行是分布2的预测结果
- h的大小分别取了0.5,1，1.5,2，顺序对应了每行中的四个值
- 分析：
    - 第一个分布中，因为是均匀分布的，所以各个结果相差不大
    - 第二个分布相对更加随机，随着h的增大，错误率越来越大，说明h增大之后，把很多空间结构都给抹去了，所以估计效果会相对差一些。这也和我们的各个点之间距离较小有关系。

提高要求(1')在两个数据集合上使用进行k-近邻概率密度估计，计算并分析 k=1，3，5 时的概率密度估计结果

我在这里使用的是后验密度的方式来计算：

后验密度为：

$$p(w_i|x)=\frac{p(x|w_i)p(w_i)}{p(x)}=\frac{\frac{k_i}{n_iV}\cdot\frac{n_i}{N}}{\frac{k}{NV}}=\frac{k_i}{k}$$

仅仅需要计算在我们的k值中，哪一类占有k的概率最大（也采用留1法交叉验证）

代码：

```
def kneighbor_est (test,X,k):
    topx = []
    info={}
    maxdis = 0
    #分别找出距离top k的点，存下他们的距离和类别
    for i in X:
        if i[0]==test[0] and i[1]==test[1]:
            continue
        dis=np.linalg.norm(test-i[0:2])
        if len(topx)<k:
            topx.append(dis)
            info[dis]=i[2]
            topx=sorted(topx)
            maxdis=topx[len(topx)-1]
        elif maxdis > dis:
            topx.append(dis)
            info[dis]=i[2]
            topx=sorted(topx)
            todel=topx.pop(k) # del the minimum
            info.pop(todel)
            maxdis=topx[k-1]

    lable=[0,0,0]
    #找出类别总数最多的那个类作为估计结果。
    for i in info.keys():
        if info[i]==1:
            lable[0]+=1
        elif info[i]==2:
            lable[1]+=1
        else:
```

```
            lable[2]+=1;

    return lable.index(max(lable))


def knn(mean, cov, P1, P2,k):
    error1=0
    error2=0
    X1 = Generate_DataSet(mean, cov, P1)
    X2 = Generate_DataSet(mean, cov, P2)
    for i in X1:
        lable=kneighbor_est(i[0:2],X1,k)
        if lable!=i[2]-1:
            error1+=1
    for i in X2:
        lable=kneighbor_est(i[0:2],X2,k)
        if lable!=i[2]-1:
            error2+=1
    return error1,error2
```

分别计算了当k=1,3,5的时候：

```
---------k= 1
erro1: 0.095
erro2: 0.118
---------k= 3
erro1: 0.093
erro2: 0.083
---------k= 5
erro1: 0.094
erro2: 0.096
```

- 分析：
    - 可以发现，在k=1时，第二个分布中的erro率非常高，这可能是以为第二个分布随机性太大的原因，只用最近邻的方式，很容易以偏概全出现错误。
    - 表现最好的是k=3的时候，这个时候有了类似投票的机制。可以规避k=1时出现的问题
    - 当k=5的时候，erro2又有轻微反弹，这个可能是因为太多的点，让一些距离较远的negative点也获得了同样的投票权，结果适得其反。