朴素贝叶斯 $(Bayes)$ 分类器

一、实验目的

1、了解贝叶斯分类器的基本原理和应用场景。
2、学习如何使用 Python 实现贝叶斯分类器。
3、掌握贝叶斯分类器的评估方法和性能指标。
4、通过实验对比不同数据集在贝叶斯分类器下的分类效果，探究贝叶斯分类器的优缺点。

二、实验原理

朴素贝叶斯分类器是一种基于贝叶斯定理的概率分类器，它假设特征之间是相互独立的。在分类任务中，给定一个样本数据集，朴素贝叶斯分类器通过计算每个类别的后验概率来将样本分配到最可能的类别中。假设有一个样本数据集 $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_n, y_n)\}$，其中 $\boldsymbol{x}_i$ 是一个 d 维特征向量，$y_i \in \{1, 2, \cdots, K\}$ 是样本 i 的类别标签。朴素贝叶斯分类器的目的是学习一个分类器 $f(\boldsymbol{x})$，使得对于任意一个新的输入特征向量 $\boldsymbol{x}$，能够预测其属于哪个类别。朴素贝叶斯分类器的基本思想是利用训练数据集 D 估计每个类别的先验概率 $P(y_k)$ 和每个特征在各个类别下的条件概率 $P(\boldsymbol{x}_i|y_k)$，然后根据贝叶斯定理计算后验概率 $P(y_k|\boldsymbol{x})$，即：

$$P(y_k|\boldsymbol{x}) = \frac{P(y_k)P(\boldsymbol{x}|y_k)}{\sum_{j=1}^{K} P(y_j)P(\boldsymbol{x}|y_j)}$$

其中，$P(y_k)$ 是先验概率，表示在没有任何特征信息的情况下，一个样本属于类别 y_k 的概率；$P(\boldsymbol{x}|y_k)$ 是条件概率，表示在已知样本属于类别 y_k 的情况下，特征向量 $\boldsymbol{x}$ 出现的概率。由于朴素贝叶斯分类器假设特征之间是相互独立的，因此可以将条件概率拆分为每个特征的条件概率的乘积，即：

$$P(\boldsymbol{x}|y_k) = \prod_{j=1}^{d} P(x^j|y_k)$$

其中，$P(x_i|y_k)$ 表示在已知样本属于类别 y_k 的情况下，第 i 个特征出现的概率。在训练阶段，朴素贝叶斯分类器需要估计先验概率 $P(y_k)$ 和条件概率 $P(x_i|y_k)$。对于先验概率 $P(y_k)$，可以使用样本数据集中每个类别出现的频率来进行估计，即：

$$P(y_k) = \frac{\sum_{i=1}^{n} I[y_i = y_k]}{n}$$

其中，$I[y_i = y_k]$ 是指示函数，当 $y_i = y_k$ 时取值为 1，否则为 0。对于条件概率 $P(x_i|y_k)$，可以根据训练数据集中每个类别下第 i 个特征的分布来进行估计。具体地，假设第 i 个特征是一个离散型变量，可以

使用样本数据集中每个类别下第 i 个特征的频率来估计条件概率，即：

$$P(x_i^j = v|y_k) = \frac{\sum_{i=1}^{n} I[x_i^j = v,\ y_i = y_k]}{\sum_{i=1}^{n} I[y_i = y_k]}$$

其中，x_i^j 表示样本 i 的第 j 个特征的取值。如果第 i 个特征是一个连续型变量，可以假设它服从高斯分布，并使用样本数据集中每个类别下第 i 个特征的均值和方差来估计条件概率，即：

$$P(x_i|y_k) = \frac{1}{\sqrt{2\pi\sigma_{k,i}^2}} \exp\left(-\frac{(x_i - \mu_{k,i})^2}{2\sigma_{k,i}^2}\right)$$

其中，$\mu_{k,i}$ 和 $\sigma_{k,i}$ 分别是样本数据集中所有属于类别 y_k 的样本的第 i 个特征的均值和方差。在预测阶段，朴素贝叶斯分类器根据贝叶斯定理计算每个类别的后验概率，然后将样本分配到后验概率最大的类别中，即：

$$\hat{y} = \arg\max_{y_k} P(y_k|\boldsymbol{x})$$

其中，$\hat{y}$ 是样本的预测类别。朴素贝叶斯分类器的优点是具有较好的可解释性、训练和预测速度较快，适用于大规模数据集和高维数据集。但是，它假设特征之间是相互独立的，可能会导致分类效果较差。

此外，朴素贝叶斯分类器还存在概率估计为 0 的问题，即某个特征在某个类别下没有出现过，导致条件概率为 0，从而影响分类结果。为了解决这个问题，通常采用拉普拉斯平滑或者加权平滑等方法来对条件概率进行平滑处理。由于本次实验并未涉及，这里我们也就不再展开。

此外，朴素贝叶斯分类器在文本分类、垃圾邮件过滤、情感分析等领域也有非常广泛的广泛应用。

三、实验步骤

1、利用 python 完成数据的生成（均匀分布，等距划分）。

2、完成 *Bayes* 分类器中先验概率，条件概率以及联合概率的计算，选择可能性最大的一个类别作为模型的输出。

四、实验代码

定义各个变量

```
import numpy as np
import matplotlib.pyplot as plt
import csv

np.random.seed(25565)
total_samplenum=0# valid training sample number
total_testnum=0#valid test sample number
sample=np.random.rand(2000,2)*10# generate training sample
features=np.zeros([2000,2])# features of each training sample, (1,1) for kind1, (1,2)
↪  for kind2...
```

```
tag=np.zeros(2000)#tag of training sample
test=np.random.rand(100,2)*10
test_features=np.zeros([100,2])
test_tag=np.zeros(100)
noise=np.random.rand(1000,2)*10
noise_features=np.zeros([1000,2])
noise_tag=np.random.randint(1,10,1000)
noise_num=len(noise)
default_color= ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd', '#8c564b',
↪  '#e377c2', '#7f7f7f', '#bcbd22', '#17becf']
data_row:list=[]#record data
```

数据的可视化以及划分

```
i=0
while i <len(sample):
    if((sample[i][0]>=3 and sample[i][0]<=3.5) or (sample[i][0]>=6.5 and
    ↪  sample[i][0]<=7)
     or (sample[i][1]>=3 and sample[i][1]<=3.5) or (sample[i][1]>=6.5 and
     ↪  sample[i][1]<=7)):
        sample=np.delete(sample,i,0)
        tag=np.delete(tag,i,0)
        features=np.delete(features,i,0)
        #num-=1
        continue
    if(sample[i][0]>=0 and sample[i][0]<=3 and sample[i][1]>=0 and sample[i][1]<=3):
        features[i]=np.array([1,1])
        tag[i]=1
        plt.scatter(sample[i][0],sample[i][1],marker='3',c=default_color[0])
    elif(sample[i][0]>=3.5 and sample[i][0]<=6.5 and sample[i][1]>=0 and
    ↪  sample[i][1]<=3):
        features[i]=np.array([2,1])
        tag[i]=2
        plt.scatter(sample[i][0],sample[i][1],marker='3',c=default_color[1])
    elif(sample[i][0]>=7 and sample[i][0]<=10 and sample[i][1]>=0 and
    ↪  sample[i][1]<=3):
        features[i]=np.array([3,1])
        tag[i]=3
        plt.scatter(sample[i][0],sample[i][1],marker='3',c=default_color[2])
    elif(sample[i][0]>=0 and sample[i][0]<=3 and sample[i][1]>=3.5 and
    ↪  sample[i][1]<=6.5):
```

```
        features[i]=np.array([1,2])
        tag[i]=4
        plt.scatter(sample[i][0],sample[i][1],marker='3',c=default_color[3])
    elif(sample[i][0]>=3.5 and sample[i][0]<=6.5 and sample[i][1]>=3.5 and
    ↪ sample[i][1]<=6.5):
        features[i]=np.array([2,2])
        tag[i]=5
        plt.scatter(sample[i][0],sample[i][1],marker='3',c=default_color[4])
    elif(sample[i][0]>=7 and sample[i][0]<=10 and sample[i][1]>=3.5 and
    ↪ sample[i][1]<=6.5):
        features[i]=np.array([3,2])
        tag[i]=6
        plt.scatter(sample[i][0],sample[i][1],marker='3',c=default_color[5])
    elif(sample[i][0]>=0 and sample[i][0]<=3 and sample[i][1]>=7 and
    ↪ sample[i][1]<=10):
        features[i]=np.array([1,3])
        tag[i]=7
        plt.scatter(sample[i][0],sample[i][1],marker='3',c=default_color[6])
    elif(sample[i][0]>=3.5 and sample[i][0]<=6.5 and sample[i][1]>=7 and
    ↪ sample[i][1]<=10):
        features[i]=np.array([2,3])
        tag[i]=8
        plt.scatter(sample[i][0],sample[i][1],marker='3',c=default_color[7])
    elif(sample[i][0]>=7 and sample[i][0]<=10 and sample[i][1]>=7 and
    ↪ sample[i][1]<=10):
        features[i]=np.array([3,3])
        tag[i]=9
        plt.scatter(sample[i][0],sample[i][1],marker='3',c=default_color[8])
    total_samplenum+=1
    i+=1

i=0
while i <len(test):
    if((test[i][0]>=3 and test[i][0]<=3.5) or (test[i][0]>=6.5 and test[i][0]<=7)
     or (test[i][1]>=3 and test[i][1]<=3.5) or (test[i][1]>=6.5 and test[i][1]<=7)):
        test=np.delete(test,i,0)
        test_features=np.delete(test_features,i,0)
        test_tag=np.delete(test_tag,i,0)
        continue
    if(test[i][0]>=0 and test[i][0]<=3 and test[i][1]>=0 and test[i][1]<=3):
```

```
        test_features[i]=np.array([1,1])
        test_tag[i]=1
    elif(test[i][0]>=3.5 and test[i][0]<=6.5 and test[i][1]>=0 and test[i][1]<=3):
        test_features[i]=np.array([2,1])
        test_tag[i]=2
    elif(test[i][0]>=7 and test[i][0]<=10 and test[i][1]>=0 and test[i][1]<=3):
        test_features[i]=np.array([3,1])
        test_tag[i]=3
    elif(test[i][0]>=0 and test[i][0]<=3 and test[i][1]>=3.5 and test[i][1]<=6.5):
        test_features[i]=np.array([1,2])
        test_tag[i]=4
    elif(test[i][0]>=3.5 and test[i][0]<=6.5 and test[i][1]>=3.5 and
    ↪  test[i][1]<=6.5):
        test_features[i]=np.array([2,2])
        test_tag[i]=5
    elif(test[i][0]>=7 and test[i][0]<=10 and test[i][1]>=3.5 and test[i][1]<=6.5):
        test_features[i]=np.array([3,2])
        test_tag[i]=6
    elif(test[i][0]>=0 and test[i][0]<=3 and test[i][1]>=7 and test[i][1]<=10):
        test_features[i]=np.array([1,3])
        test_tag[i]=7
    elif(test[i][0]>=3.5 and test[i][0]<=6.5 and test[i][1]>=7 and test[i][1]<=10):
        test_features[i]=np.array([2,3])
        test_tag[i]=8
    elif(test[i][0]>=7 and test[i][0]<=10 and test[i][1]>=7 and test[i][1]<=10):
        test_features[i]=np.array([3,3])
        test_tag[i]=9
    total_testnum+=1
    i+=1

    i=0
    while i <len(noise):
        if((noise[i][0]>=3 and noise[i][0]<=3.5) or (noise[i][0]>=6.5 and
        ↪  noise[i][0]<=7)
         or (noise[i][1]>=3 and noise[i][1]<=3.5) or (noise[i][1]>=6.5 and
         ↪  noise[i][1]<=7)):
            noise=np.delete(noise,i,0)
            noise_tag=np.delete(noise_tag,i,0)
            noise_features=np.delete(noise_features,i,0)
            noise_num-=1
```

```
            continue
        else:
            plt.scatter(noise[i][0], noise[i][1], marker='3',
            ↪  c=default_color[int(noise_tag[i])-1])
            noise_features[i]=np.array([int(noise[i][0]/3.5)+1,
            ↪  int(noise[i][1]/3.5)+1])
        i+=1
    plt.show()
```

贝叶斯分类算法的实现

```
def prior_probability(ntag:int,record:bool=True)->float:
    ntag_num=0
    for i in range(0,len(sample)):
        if tag[i]==ntag:
            ntag_num+=1
    for k in range(0,len(noise)):
        if noise_tag[k]==ntag:
            ntag_num+=1
    result=ntag_num/(total_samplenum+noise_num)
    if record:
        data_row.append(result)
    return result

def conditional_probability(conclusion:int,condition:int,seq:int)->float:
    Ixy:int=0
    Iy=prior_probability(conclusion,False)*(total_samplenum+noise_num)

    for k in range(0,len(sample)):
        if features[k][seq]==condition and tag[k]==conclusion:
            Ixy+=1
    for k in range(0,len(noise)):
        if noise_features[k][seq]==condition and noise_tag[k]==conclusion:
            Ixy+=1
    result=Ixy/Iy
    data_row.append(result)
    return result

def united_probability(x_features:np.array,conclusion:int):
    result=conditional_probability(conclusion,x_features[0], 0) *
           conditional_probability(conclusion,x_features[1], 1) *
           ↪  prior_probability(conclusion)
```

```
    data_row.append(result)
    return result

def bayes_arg_max(x_features:np.array):
    max_probability:float=0
    max_conclusion:int=0
    for k in range(1,10):
        p=united_probability(x_features,k)
        if p>max_probability:
            max_probability=p
            max_conclusion=k

    data_row.append(max_conclusion)
    data_row.append(max_probability)
    writer.writerow(data_row)
    data_row.clear()
    return max_conclusion
```

其中，prior_probability()是计算先验概率 $P(y_k)$；conditional_probability()是计算某一个样本 $\boldsymbol{x}$ 在推测其标签为 y_k 的条件下，其第 j 个特征空间上，取值为 v 的概率；而united_probability() 则表示某一个样本所有特征空间 $\mathcal{X}^D$ 上取某一个特征向量 $\boldsymbol{v}$ 的条件概率，这个条件概率可以写成特征空间中不同维度条件概率之积，也就是实验原理中提到的 $P(\boldsymbol{x}|y_k) = \prod_{j=1}^{d} P(x^j|y_k)$；最后的bayes_arg_max()则是比较某个样本在推测为各个不同的标签时的概率，选择概率最大的标签作为预测结果返回。

结果验证部分

```
for k in range(0,len(test)):
    pred=bayes_arg_max(test_features[k])
    plt.scatter(test[k][0], test[k][1], marker='s', c=default_color[pred-1], s=15,
    ↪  alpha=0.5)
    if pred==test_tag[k]:
        correct+=1

print("acc is "+str(correct/len(test)))
print("total sample number: "+str(total_samplenum))
print("total test number: "+str(total_testnum))
print("total noise number: "+str(noise_num))
file.close()
```

在我们的实验中，设置了 9 个不同的类，它们分别位于不同的区间。由于每个点都有一个二维的坐标，于是我们将其按某一个轴的坐标值所在不同的区间来赋予不同的特征值。我们规定 x 轴的坐标值位于 $[0, 3]$ 的点，其第一个特征为 1，即 $(1, Y)$；x 坐标值位于 $[3.5, 6.5]$ 的点，其第一个特征值为 2，即 $(2, Y)$；x

坐标值位于 $[7, 10]$ 的点，其第一个特征值为 3，即 $(3, Y)$。第二个特征值同理检测 y 轴的值即可。

五、实验结果

在设置总样本点数为 2000，总测试样本数为 100，总噪声点数为 1000 时，结果如下

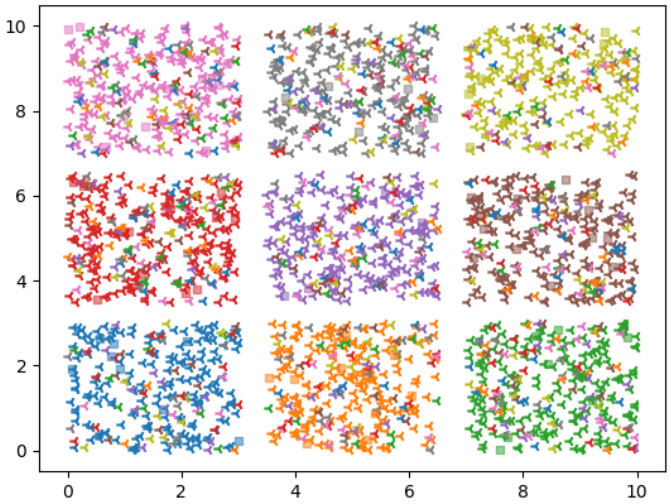

图 1: 样本生成之后的可视化图

图中，方框表示为测试样本，其颜色表示贝叶斯分类的预测结果，总样本点和测试样本点统一使用三叉形点表示，同样，颜色表示其标签类别。预测的结果为

```
acc is 1.0
total sample number: 1590
total test number: 79
total noise number: 821
```

为了更加直观地显示算法的计算过程，我们给出预测过程中各个概率的数值表

表 1: 测试样本为 $(6.42, 7.82)$，特征为 $(2, 3)$，实际 $tag = 8$

各个概率的取值	$P(x_i^1 = v^1 \mid y_k)$	$P(x_i^2 = v^2 \mid y_k)$	$P(y_k)$	$P(\boldsymbol{x_i} = \boldsymbol{v} \mid y_k)$
$y_k = 1$	0.125954	0.125954	0.108668602	0.001724
$y_k = 2$	0.760618	0.111969	0.107424305	0.009149
$y_k = 3$	0.09845	0.125	0.109498134	0.001347988
$y_k = 4$	0.111111111	0.097222222	0.119452509	0.001290382
$y_k = 5$	0.787545788	0.117216117	0.113231024	0.010452702
$y_k = 6$	0.116197183	0.10915493	0.117793447	0.001494033
$y_k = 7$	0.138790036	0.775800712	0.11654915	0.012549244
$y_k = 8$	0.78	0.78	0.103691414	0.063085856
$y_k = 9$	0.128	0.76	0.103691414	0.010087101

表 2: 测试样本为 $(2.06, 3.71)$，特征为 $(1, 2)$，实际 $tag = 4$

各个概率的取值	$P(x_i^1 = v^1 \mid y_k)$	$P(x_i^2 = v^2 \mid y_k)$	$P(y_k)$	$P(\boldsymbol{x_i} = \boldsymbol{v} \mid y_k)$
$y_k = 1$	0.77480916	0.13740458	0.108668602	0.011569112
$y_k = 2$	0.111969112	0.131274131	0.107424305	0.001578992
$y_k = 3$	0.136363636	0.09469697	0.109498134	0.001413974
$y_k = 4$	0.784722222	0.770833333	0.119452509	0.072255634
$y_k = 5$	0.113553114	0.761904762	0.113231024	0.00979637
$y_k = 6$	0.091549296	0.785211268	0.117793447	0.008467645
$y_k = 7$	0.743772242	0.131672598	0.11654915	0.011414174
$y_k = 8$	0.124	0.088	0.103691414	0.001131481
$y_k = 9$	0.132	0.12	0.103691414	0.001642472

贝叶斯分类器总是选择最终联合概率最大的一个作为预测的输出，对于上述两个样本来说，分别为 $y_{kpred} = 8$ 和 $y_{kpred} = 4$，与其正确标签相符。

附加题：增加噪声点，观察噪声比对算法的影响，报告观察结果；对比 K 近邻模型，报告哪一个算法对噪声更鲁棒

为了测试贝叶斯分类器的健壮性，我们将噪声点从原来的 1000 调整至 2000

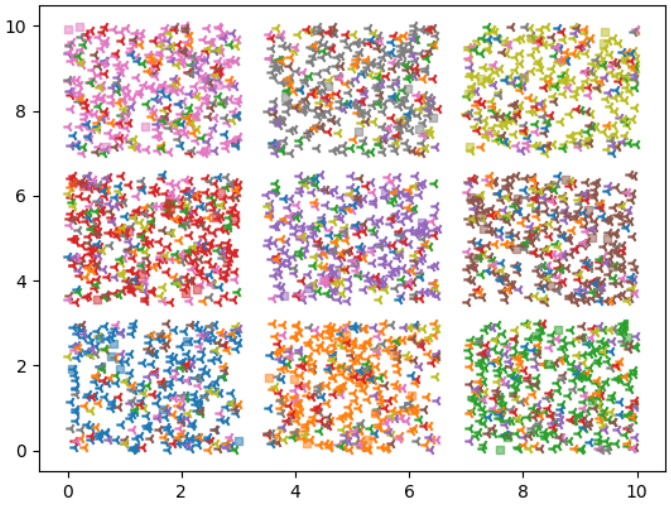

图 2: 噪声点数量为 2000，样本点数量为 2000

最终报告的准确率

```
acc is 1.0
total sample number: 1590
total test number: 79
total noise number: 1607
```

可以看到最终的预测结果没有变化，依然是 100%，这是因为对于某一个类来说，本类的样本数依然占据大多数，尽管噪声的数量和本类的样本数相当，但是噪声点的标签都是随机生成，导致具体到某一个类的噪声点的数量远比本类的数量要少。这里我们同样也给出各个概率数据。

表 3: 测试样本为 $(6.42, 7.82)$，特征为 $(2, 3)$，实际 $tag = 8$

各个概率的取值	$P(x_i^1 = v^1 \mid y_k)$	$P(x_i^2 = v^2 \mid y_k)$	$P(y_k)$	$P(\boldsymbol{x_i} = \boldsymbol{v} \mid y_k)$
$y_k = 1$	0.127388535	0.133757962	0.098217079	0.001673543
$y_k = 2$	0.642473118	0.163978495	0.116359087	0.012258636
$y_k = 3$	0.166204986	0.1966759	0.112918361	0.003691134
$y_k = 4$	0.183462532	0.170542636	0.121050985	0.003787465
$y_k = 5$	0.64109589	0.167123288	0.114169534	0.012232358
$y_k = 6$	0.140921409	0.173441734	0.115420707	0.002821073
$y_k = 7$	0.163841808	0.672316384	0.110728808	0.012197169
$y_k = 8$	0.690690691	0.693693694	0.10416015	0.049906021
$y_k = 9$	0.19005848	0.634502924	0.106975289	0.012900435

虽然分类的结果是正确的，但是很明显可以观察到正确分类 $(y_k = 8)$ 的概率与其他分类的概率之间

的差距减小，说明噪声的增强确实给模型带来了更大的影响。根据我们的估算，如果希望噪声能够影响最终的分类结果，那么至少在某一个类的区间中，噪声点的数量要达到本类数据点的 9 倍以上（对应噪声中某一个其他类的数据点的数量与本类的数据点的数量一致）。为了缩短计算时间，我们仅降低信噪比，数据点总量也下降到样本数据点 200，噪声数据点 2000，测试样本点保持 100.。

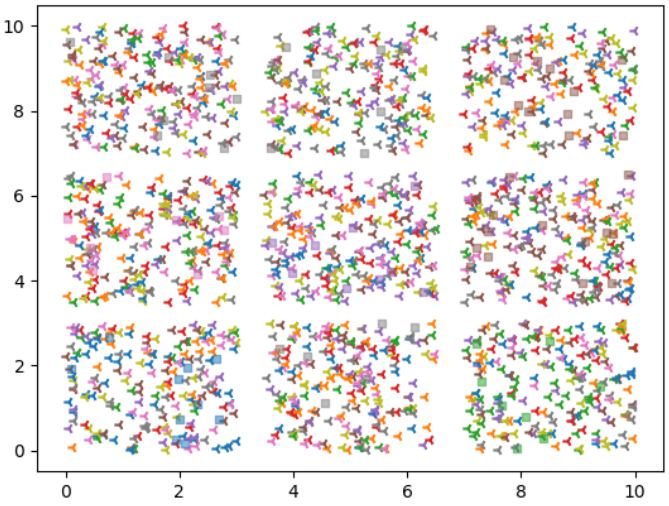

图 3: 噪声点数量为 2000，样本点数量为 200

```
acc is 0.5853658536585366
total sample number: 162
total test number: 82
total noise number: 1581
```

可以看到准确率显著下降，我们再对各个概率值进行分析。

表 4: 测试样本为 $(2.51, 8.55)$，特征为 $(1, 3)$，实际 $tag = 7$

各个概率的取值	$P(x_i^1 = v^1 \mid y_k)$	$P(x_i^2 = v^2 \mid y_k)$	$P(y_k)$	$P(\boldsymbol{x_i} = \boldsymbol{v} \mid y_k)$
$y_k = 1$	0.417525773	0.278350515	0.111302352	0.012935394
$y_k = 2$	0.364102564	0.292307692	0.111876076	0.011906969
$y_k = 3$	0.263736264	0.296703297	0.104417671	0.008170831
$y_k = 4$	0.367231638	0.293785311	0.101549053	0.010955849
$y_k = 5$	0.25	0.325	0.114744693	0.009323006
$y_k = 6$	0.339901478	0.315270936	0.116465863	0.012480605
$y_k = 7$	0.413978495	0.333333333	0.106712565	0.014725569
$y_k = 8$	0.34841629	0.384615385	0.126792886	0.016991041
$y_k = 9$	0.318918919	0.351351351	0.106138841	0.011893132

这里显然由于噪声占比过高，各个分类的概率几乎都是一个水平，虽然结果正确，但是从数据上来看，分类器的效果是显著下降的。

附加问题的另一部分是对比 KNN 分类和 $Bayes$ 分类的性能。这里我们仍然沿用第一次实验中的 KNN 分类器代码。给出样本点 2000，噪声点 1000 时的结果图。

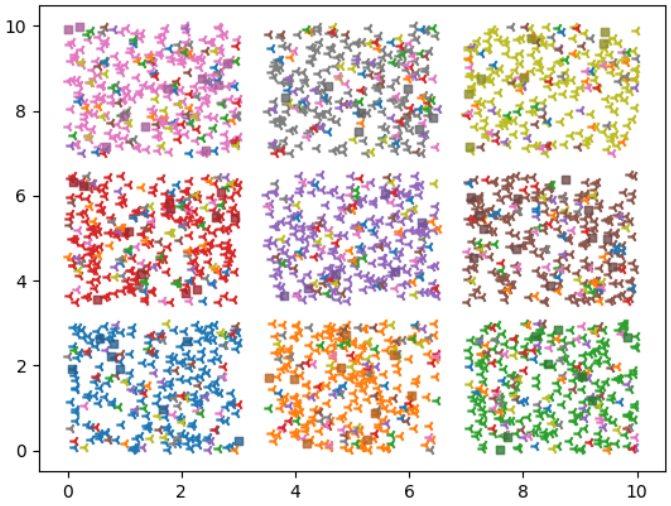

图 4: 噪声点数量为 1000，样本点数量为 2000

```
total sample number is 1590
total noise number is 821
total test number is 79
the acc is 1.0
```

这时，KNN 的预测准确率仍然保持在 1.0，因为 KNN 也是一种基于多数占优的规律的分类器，对于一个类来说，本类的样本点数量远大于噪声中其他类的数据点数量。如果我们增大噪声占比，调整为 5000。

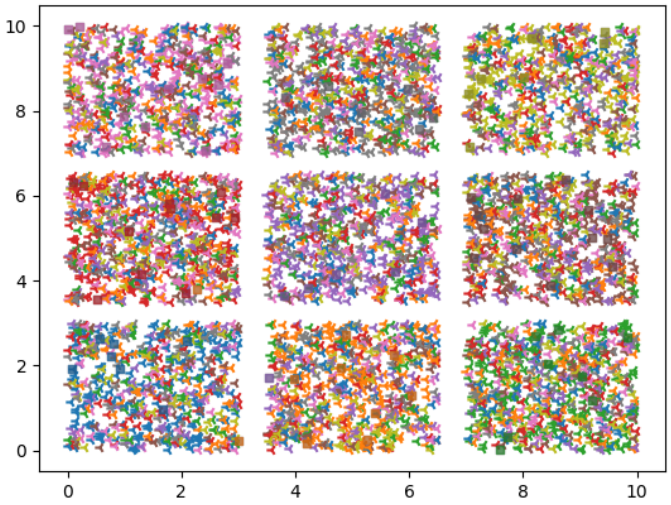

图 5: 噪声点数量为 5000，样本点数量为 2000

```
total sample number is 1590
total noise number is 4057
total test number is 79
the acc is 0.9620253164556962
```

显然准确率已经开始下降，不过下降并不明显。如果我们将信噪比继续提高，同样使样本点为 200，噪声点为 2000，结果如下。

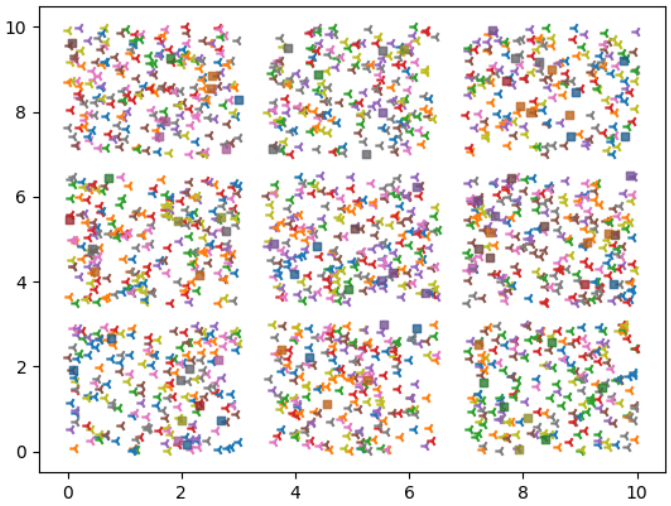

图 6: 噪声点数量为 2000，样本点数量为 200

```
    total sample number is 162
    total noise number is 1581
    total test number is 82
    the acc is 0.36585365853658536
```

准确率下降更加剧烈。

综合对比 $Bayes$ 分类器，KNN 在噪声忍耐程度上表现得结果更加差。不过考虑到如果在一个类中如果本类数据点占比比某一个噪声类得数据点占比还少，那么分类就失去了意义，因为本类数据可以看作噪声。这里我们使用 $1:10$ 的信噪比也仅供一个性能比较，实际的参考意义并不是非常大。