Author: 陆星宇 Date: 2021/2/1

第四章 朴素贝叶斯法

一、交流讨论

问题1：独立性假设使得朴素贝叶斯算法的适用范围如何？在目前已知的应用领域中，哪些效果较好，原因是什么？

分类问题，如新闻分类或临床诊断，这类问题特征之间独立性较强，朴素贝叶斯算法的效果较好。

问题2：条件概率分布有指数级数量的参数，面对特征向量位数较高和输入空间较大的情况直接计算分布是不可行的，请问如何在实际算法中改进？

特征向量的各分量取所取不同值相似性随数值相近更强，能够通过少量样本拟合出缺少数据段的近似值。

问题3：什么是贝叶斯网络？

一个贝叶斯网络是一个有向无环图(Directed Acyclic Graph,DAG),由代表变量结点及连接这些结点有向边构成。结点代表随机变量，结点间的有向边代表了结点间的互相关系(由父结点指向其子结点)，用条件概率进行表达关系强度，没有父结点的用先验概率进行信息表达。结点变量可以是任何问题的抽象，如：测试值，观测现象，意见征询等。适用于表达和分析不确定性和概率性的事件，应用于有条件地依赖多种控制因素的决策，可以从不完全、不精确或不确定的知识或信息中做出推理。

问题4：属性独立性假设是一个较强的假设，在实际情况中很难成立，但为什么朴素贝叶斯仍能取得较好的效果？

一方面，表达式相乘的前一项于对应类类样本数量占比有关，提高了常见问题的正确率。另一方面，虽然现实中大部分问题不同特征分量之间存在联系，但诸如新闻分类的问题，使用词袋模型时容易发现，词汇元素之间的关系较之于词汇与主题之间的关系较弱，独立性是一个良好的假设。

二、内容概要

1. 先验概率

$$P(Y=c_k)=\frac{\sum_{i=1}^{N}I(y_i=c_k)}{N},k=1,2,\dots,K$$

2. 条件独立性假设

$$P(X=x|Y=c_k)=\prod_{j=1}^{n}P(X^{(j)}=x^{(j)}|Y=c_k),k=1,2,\dots,K$$

3. 条件概率

$$P(X^{(j)}=a_{jl}|Y=c_k)=\frac{\sum_{i=1}^{N}I(x_i^{(j)}=a_{jl},y_i=c_k)}{\sum_{i=1}^{N}I(y_i=c_k)}$$
$$j=1,2,\dots,n;\ l=1,2,\dots,S_j;\ k=1,2,\dots,K$$

4. 贝叶斯估计

$$P(X^{(j)}=a_{jl}|Y=c_k)=\frac{\sum_{i=1}^{N}I(x_i^{(j)}=a_{jl},y_i=c_k)+\lambda}{\sum_{i=1}^{N}I(y_i=c_k)+\lambda S_j}$$
$$j=1,2,\dots,n;\ l=1,2,\dots,S_j;\ k=1,2,\dots,K$$

其中，取$\lambda=0$时，就是极大似然估计；取$\lambda=1$时就称为拉普拉斯平滑。

5. 后验概率

$$P(Y=c_k|X=x)=\frac{P(Y=c_k)\prod_j P(X^{(j)}=x^{(j)}|Y=c_k)}{\sum_k P(Y=c_k)\prod_j P(X^{(j)}=x^{(j)}|Y=c_k)},k=1,2,\ldots,K$$

6. 朴素贝叶斯分类器

$$f(x)=arg\ \max_{c_k} P(Y=c_k)\prod_{j=1}^{n} P(X^{(j)}=x^{(j)}|Y=c_k)$$

三、算法实现

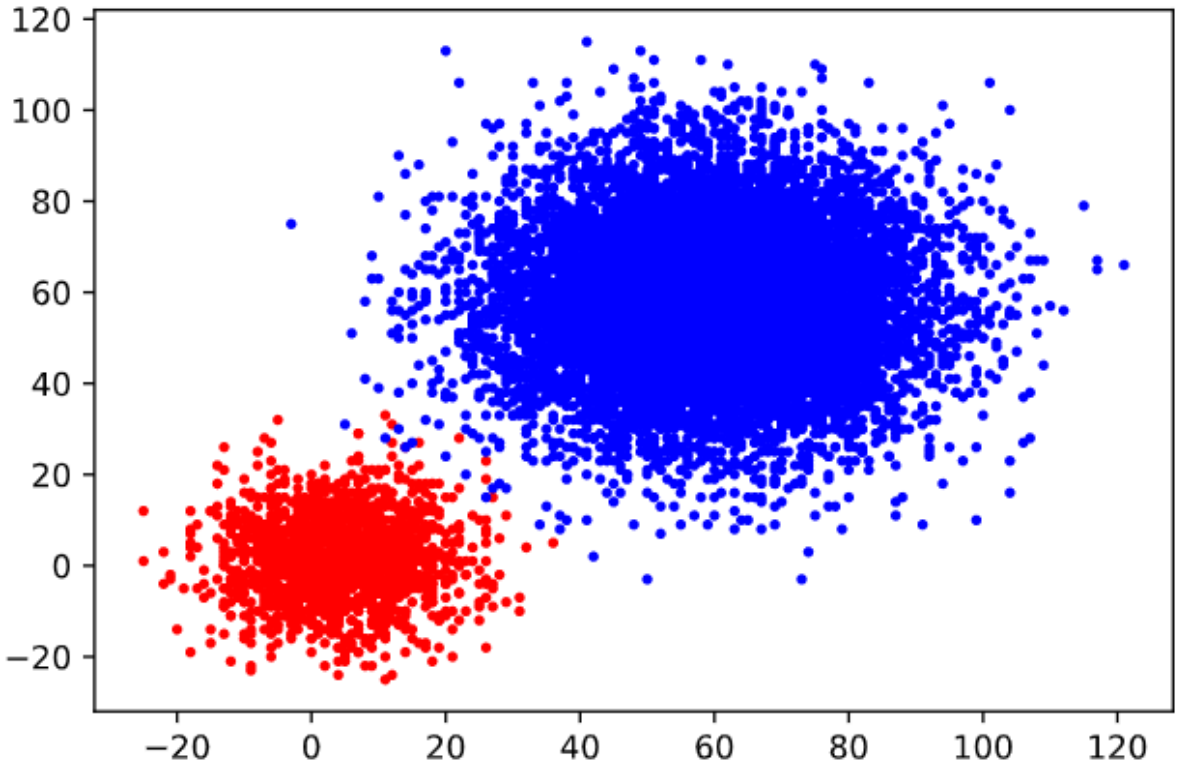
```
#随机生成训练集
import random as rd
import matplotlib.pyplot as plt

mu1_x = rd.randrange(0,10)
mu1_y = rd.randrange(0,10)
sigma1 = rd.randrange(50,200)/10

mu2_x = rd.randrange(0,10) + 50
mu2_y = rd.randrange(0,10) + 50
sigma2 = rd.randrange(50,200)/10

print(mu1_x,mu1_y)
print(mu2_x,mu2_y)

s1_x = []
s1_y = []
s1_n = rd.randint(1000,2000)
s2_x = []
s2_y = []
s2_n = rd.randint(10000,20000)

for i in range(s1_n):
    s1_x.append(round(rd.normalvariate(mu1_x, sigma1)))
    s1_y.append(round(rd.normalvariate(mu1_y, sigma1)))
for i in range(s2_n):
    s2_x.append(round(rd.normalvariate(mu2_x, sigma2)))
    s2_y.append(round(rd.normalvariate(mu2_y, sigma2)))

plt.scatter(s1_x,s1_y,c="red",s=5)
plt.scatter(s2_x,s2_y,c="blue",s=5)
plt.show()
#朴素贝叶斯分类器训练
N = s1_n + s2_n
p1 = s1_n / N
p2 = s2_n / N
n1x = {}
n1y = {}
n2x = {}
n2y = {}

for i in range(s1_n):
    if(n1x.get(s1_x[i]) == None):
```

```
        n1x[s1_x[i]] = 1
    else:
        n1x[s1_x[i]] = n1x[s1_x[i]] + 1

for i in range(s1_n):
    if(n1y.get(s1_y[i]) == None):
        n1y[s1_y[i]] = 1
    else:
        n1y[s1_y[i]] = n1y[s1_y[i]] + 1

for i in range(s2_n):
    if(n2x.get(s2_x[i]) == None):
        n2x[s2_x[i]] = 1
    else:
        n2x[s2_x[i]] = n2x[s2_x[i]] + 1

for i in range(s2_n):
    if(n2y.get(s2_y[i]) == None):
        n2y[s2_y[i]] = 1
    else:
        n2y[s2_y[i]] = n2y[s2_y[i]] + 1
#随机生成测试集
ts1_x = []
ts1_y = []
ts1_n = rd.randint(1000,2000)
ts2_x = []
ts2_y = []
ts2_n = rd.randint(1000,2000)

for i in range(ts1_n):
    ts1_x.append(round(rd.normalvariate(mu1_x, sigma1)))
    ts1_y.append(round(rd.normalvariate(mu1_y, sigma1)))
for i in range(ts2_n):
    ts2_x.append(round(rd.normalvariate(mu2_x, sigma2)))
    ts2_y.append(round(rd.normalvariate(mu2_y, sigma2)))

plt.scatter(ts1_x,ts1_y,c="red",s=5)
plt.scatter(ts2_x,ts2_y,c="blue",s=5)
plt.show()
#测试
n_correct = 0
for i in range(ts1_n):
    pc1 = p1
    if(n1x.get(ts1_x[i]) != None):
        pc1 *= n1x[ts1_x[i]]
    if(n1y.get(ts1_y[i]) != None):
        pc1 *= n1y[ts1_y[i]]
    pc2 = p2
    if(n2x.get(ts1_x[i]) != None):
        pc2 *= n2x[ts1_x[i]]
    if(n2y.get(ts1_y[i]) != None):
        pc2 *= n2y[ts1_y[i]]
    if(pc1 > pc2):
        n_correct += 1


for i in range(ts2_n):
    pc1 = p1
```

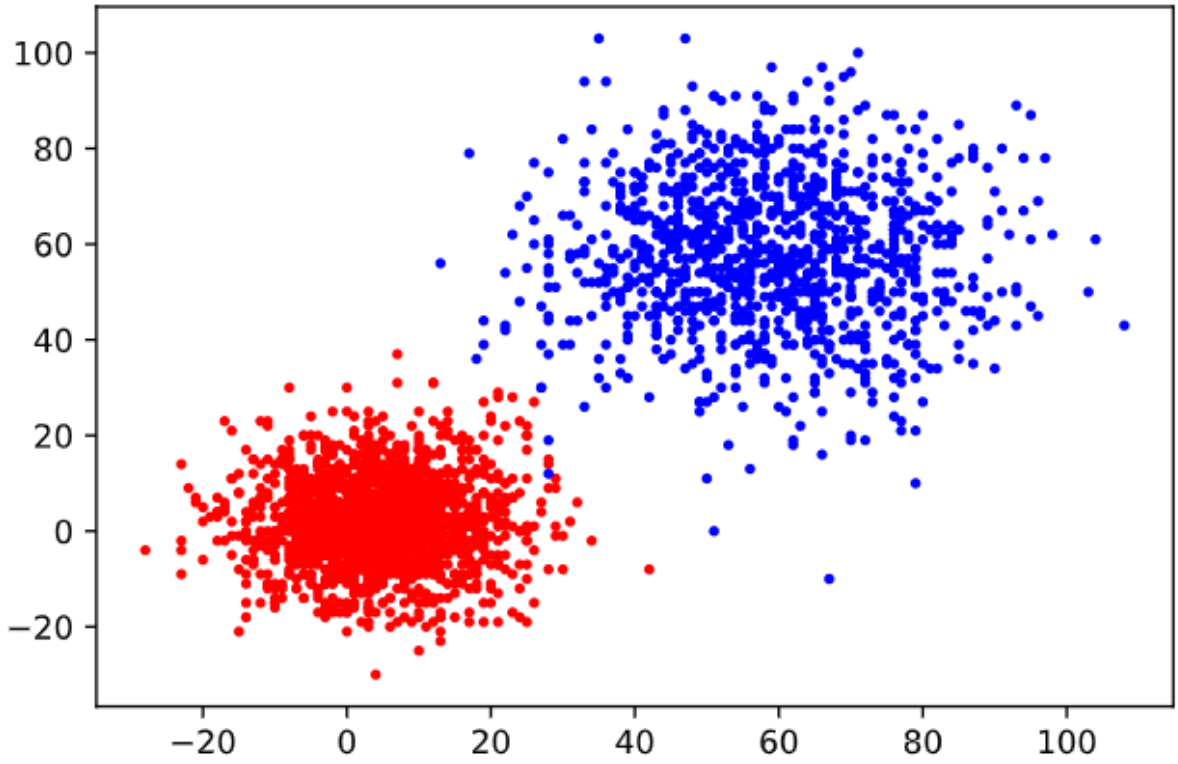
```
    if(n1x.get(ts2_x[i]) != None):
        pc1 *= n1x[ts2_x[i]]
    if(n1y.get(ts2_y[i]) != None):
        pc1 *= n1y[ts2_y[i]]
    pc2 = p2
    if(n2x.get(ts2_x[i]) != None):
        pc2 *= n2x[ts2_x[i]]
    if(n2y.get(ts2_y[i]) != None):
        pc2 *= n2y[ts2_y[i]]
    if(pc1 < pc2):
        n_correct += 1

p_c = n_correct/(ts1_n + ts2_n)
print(p_c)
```

四、实验结果

训练集



测试集



分类正确率

```
测试正确率为：  97.34219269102991 %
```

五、阅读拓展

六、阅读计划