朴素贝叶斯

一、概述

1. 真正的概率分类器

在许多分类算法应用中，特征和标签之间的关系并非是决定性的。比如说，我们想预测一个人究竟是否会在泰坦尼克号海难中生存下来，那我们可以建一棵决策树来学习我们的训练集。在训练中，其中一个人的特征为：30岁，男，普通舱，他最后在泰坦尼克号海难中去世了。当我们测试的时候，我们发现有另一个人的特征也为：30岁，男，普通舱。基于在训练集中的学习，我们的决策树必然会给这个人打上标签：去世。然而这个人的真实情况一定是去世了吗？并非如此。

也许这个人是心脏病患者，得到了上救生艇的优先权。又有可能，这个人就是挤上了救生艇，活了下来。对分类算法来说，基于训练的经验，这个人“很有可能”是没有活下来，但算法永远也无法确定”这个人一定没有活下来“。即便这个人最后真的没有活下来，算法也无法确定基于训练数据给出的判断，是否真的解释了这个人没有存活下来的真实情况。这就是说，**算法得出的结论，永远不是100%确定的，更多的是判断出了一种“样本的标签更可能是某类的可能性”，而非一种“确定”**。我们通过某些规定，比如说，在决策树的叶子节点上占比较多的标签，就是叶子节点上所有样本的标签，来强行让算法为我们返回一个固定结果。但许多时候，我们也希望能够理解算法判断出的可能性本身。

每种算法使用不同的指标来衡量这种可能性。比如说，决策树使用的就是叶子节点上占比较多的标签所占的比例（接口predict_proba调用），逻辑回归使用的是sigmoid函数压缩后的似然（接口predict_proba调用）。但这些指标的本质，其实都是一种“类概率”的表示，我们可以通过归一化或sigmoid函数将这些指标压缩到0~1之间，让他们表示我们的模型对预测的结果究竟有多大的把握（置信度）。但无论如何，我们都希望使用真正的概率来衡量可能性，因此就有了真正的概率算法：朴素贝叶斯。

2. 朴素贝叶斯是如何工作的

2.1 基本原理概述

在所有的机器学习分类算法中，朴素贝叶斯和其他绝大多数的分类算法都不同。对于大多数的分类算法，比如决策树、KNN、逻辑回归、支持向量机等，他们都是判别方法，也就是直接学习出特征输出Y和特征X之间的关系，要么是决策函数 $Y = f(X)$，要么是条件分布 $P(Y|X)$。但是朴素贝叶斯确实生成方法，也就是直接找出特征输出Y和特征X的联合分布 $P(X,Y)$，然后用 $P(Y|X)$，然后用 $P(Y|X) = P(X,Y)/P(X)$ 的得出。

朴素贝叶斯很直观，计算量也不大，在很多领域有广泛的应用，首先我们对朴素贝叶斯算法原理做一梳理。

2.2 贝叶斯相关统计学原理

在了解朴素贝叶斯的算法之前，我们需要对相关必须的统计学知识做一个回顾。

贝叶斯学派很古老，但是从诞生到一百年前一直不是主流。主流是频率学派。频率学派的权威皮尔逊和费歇尔都对贝叶斯学派不屑一顾，但是贝叶斯学派硬是凭借在现代特定领域的出色应用表现为自己赢得了半壁江山。

贝叶斯学派的思想可以概括为「先验概率+数据=后验概率」。也就是说我们在实际问题中需要得到的后验概率，可以通过先验概率和数据一起综合得到。数据大家好理解，被频率学派攻击的是先验概率，一般来说先验概率就是我们对于数据所在领域的历史经验，但是这个经验常常难以量化或者模型化，于是贝叶斯学派大胆的假设先验分布的模型，比如正态分布、beta分布等。这个假设一般没有特定的依据，因此一直被频率学派认为很荒谬。虽然难以从严密的数学逻辑里推出贝叶斯学派的逻辑，但是在很多实际应用中，贝叶斯理论很好用，比如垃圾邮件分类，文本分类。

我们先看看条件独立公式，如果X和Y相互独立，它们的联合概率 $P(X=x,Y=y)$ 是指X取值x 且 Y 取值y 的概率。我们接着看看条件概率公式：

$$P(Y|X)=\frac{P(X,Y)}{P(X)}$$

$$P(X|Y)=\frac{P(X,Y)}{P(Y)}$$

在概率论中，我们可以证明，两个事件的联合概率等于这两个事件任意条件概率 * 这个条件事件本身的概率。

$$P(X,Y)=P(Y|X)*P(X)=P(X|Y)*P(Y)$$

接着看看全概率公式：

$$P(X)=\sum_k P(X|Y=Y_k)P(Y_k)$$

其中 $\sum_k P(Y_k)=1$ ，由上面的式子，我们可以得到贝叶斯公式：

$$P(Y|X)=\frac{P(X|Y)}{P(X)}P(Y)$$

$$P(Y_k|X)=\frac{P(X|Y_k)P(Y_k)}{\sum_k P(X|Y=Y_k)P(Y_k)}$$

(1) A_1, A_2, A_3两两互不相容，

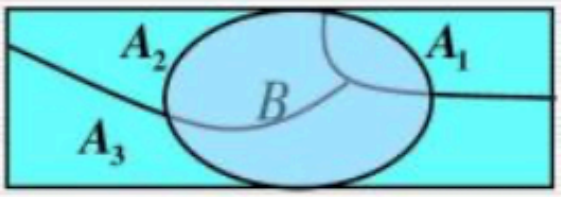
(2) $A_1 \cup A_2 \cup A_3 = \Omega \qquad P(A_i) > 0,$

则对事件B，

$$P(B)=P(A_1B)+P(A_2B)+P(A_3B)$$

$$=\sum_{i=1}^{3} P(A_i)P(B \mid A_i),$$

称满足(1)和(2)的A_1，A_2，A_3为完备事件组或样本空间的一个划分.



而这个式子，就是我们一切贝叶斯算法的根源理论。我们可以把我们的特征X当成是我们的条件事件，而我们要求解的标签Y当成是我们被满足条件后会被影响的结果，而两者之间的概率关系就是$P(Y|X)$，这个概率在机器学习中，被我们称之为是标签的**后验概率**（posterior probability），即是说我们先知道了条件，再去求解结果。而标签Y在没有任何条件限制下取值为某个值的概率，被我们写作$P(Y)$，与后验概率相反，这是完全没有任何条件限制的，标签的**先验概率**（prior probability）。而我们的$P(X|Y)$被称为**"类的条件概率"**，表示当Y的取值固定的时候，X为某个值的概率。那现在，有趣的事情就出现了。

$P(Y)$ **称为「先验概率」**，即在Y事件发生之前，我们对X事件概率的一个判断。

$P(Y|X)$ **称为「后验概率」**，即在Y事件发生之后，我们对X事件概率的重新评估。

$P(X|Y)/P(X)$ **称为「可能性函数」**，这是一个调整因子，使得预估概率更接近真实概率。

所以条件概率可以理解为：**后验概率=先验概率 * 调整因子**

如果可能性函数>1，意味着先验概率被增强，事件Y的发生的可能性变大；

如果可能性函数=1，意味着X 事件无助于判断事件Y 的可能性；

如果可能性函数<1，意味着先验概率被削弱，事件Y可能性变小。

2.3 嫁？还是不嫁？这是一个问题🤔

为了加深对朴素贝叶斯的理解，我们.....

序号	颜值	性格	上进否	嫁与否
1	帅	好	上进	嫁
2	不帅	好	一般	不嫁
3	不帅	不好	不上进	不嫁
4	帅	好	一般	嫁
5	不帅	好	上进	嫁
6	帅	不好	一般	不嫁
7	帅	好	不上进	嫁
8	不帅	不好	上进	不嫁
9	帅	不好	上进	嫁
10	不帅	好	不上进	不嫁

现在，假如某男（帅，性格不好，不上进）向女生求婚，该女生嫁还是不嫁？

根据贝叶斯公式：

$$P(Y|X) = \frac{P(X|Y)}{P(X)}P(Y)$$

转换成分类任务的表达式：

$$P(\text{类别}|\text{特征}) = \frac{P(\text{特征}|\text{类别})}{P(\text{特征})}P(\text{类别})$$

我们这个栗子，按照朴素贝叶斯的求解，可以转换为计算$P(\text{嫁}|\text{帅，性格不好，不上进})$和$P(\text{不嫁}|\text{帅，性格不好，不上进})$，最终选择嫁与不嫁的答案。

根据贝叶斯公式可知：

$$P(\text{嫁}|\text{帅，性格不好，不上进}) = P(\text{嫁})\frac{P(\text{帅}|\text{嫁})P(\text{性格不好}|\text{嫁})P(\text{不上进}|\text{嫁})}{P(\text{帅，性格不好，不上进})}$$

$$P(\text{不嫁}|\text{帅，性格不好，不上进}) = P(\text{不嫁})\frac{P(\text{帅}|\text{不嫁})P(\text{性格不好}|\text{不嫁})P(\text{不上进}|\text{不嫁})}{P(\text{帅，性格不好，不上进})}$$

对于分母我们可以求得：

$$P(X) = \sum_k P(X|Y = Y_k)P(Y_k)$$

所以

$$\begin{aligned}P(\text{帅，性格不好，不上进}) = &P(\text{嫁})P(\text{帅}|\text{嫁})P(\text{性格不好}|\text{嫁})P(\text{不上进}|\text{嫁}) \\ &+ P(\text{不嫁})P(\text{帅}|\text{不嫁})P(\text{性格不好}|\text{不嫁})P(\text{不上进}|\text{不嫁})\end{aligned}$$

由上表可以得到：

$$P(\text{嫁}) = \frac{5}{10} = \frac{1}{2}$$

$$P(\text{不嫁}) = \frac{5}{10} = \frac{1}{2}$$

$$P(\text{帅}|\text{嫁}) * P(\text{性格不好}|\text{嫁}) * P(\text{不上进}|\text{嫁}) = \frac{4}{5} * \frac{1}{5} * \frac{1}{5} = \frac{4}{125}$$

$$P(\text{帅}|\text{不嫁}) * P(\text{性格不好}|\text{不嫁}) * P(\text{不上进}|\text{不嫁}) = \frac{1}{5} * \frac{3}{5} * \frac{2}{5} = \frac{6}{125}$$

对于类别「嫁」的贝叶斯分子为：

$$P(\text{嫁}) * P(\text{帅}|\text{嫁}) * P(\text{性格不好}|\text{嫁}) * P(\text{不上进}|\text{嫁}) = \frac{1}{2} * \frac{4}{125} = \frac{2}{125}$$

对于类别「不嫁」的贝叶斯分子为：

$$P(\text{不嫁}) * P(\text{帅}|\text{不嫁}) * P(\text{性格不好}|\text{不嫁}) * P(\text{不上进}|\text{不嫁}) = \frac{1}{2} * \frac{6}{125} = \frac{3}{125}$$

所以最终结果为：

$$P(\text{嫁}|\text{帅，性格不好，不上进}) = \frac{\frac{2}{125}}{\frac{2}{125} + \frac{3}{125}} = 40\%$$

$$P(\text{不嫁}|\text{帅，性格不好，不上进}) = \frac{\frac{3}{125}}{\frac{2}{125} + \frac{3}{125}} = 60\%$$

60%>40%，该女生选择不嫁。

这里也可以理解为对于「嫁」的来说，我们设定了阈值为0.5，假设大于0.5的就被认为嫁，小于0.5的就被认为是不嫁。根据我们的计算，我们认为对于一名长得帅，但性格不好，又不上进的男生，女生是选择不嫁的。这就完成了一次预测。但是这样，有趣的地方又来了。刚才的预测过程是没有问题的。但我们总是好奇，这个过程如何对应sklearn当中的fit和predict呢？这个决策过程中，我们的训练集和我的测试集分别在哪里？

二、不同分布下的贝叶斯

1. 高斯朴素贝叶斯GaussianNB

class `sklearn.naive_bayes.GaussianNB`(*priors=None*, *var_smoothing=1e-09*)

如果我们的X_i是连续值，我们通常X_i的先验概率为高斯分布（也就是正态分布），即在样本类别C_k中，X_i 的值符合正态分布。以此来估计每个特征下每个类别上的条件概率。对于每个特征下的取值，高斯朴素贝叶斯有如下公式：

$$\begin{aligned} P(x_i = X_i^{(test)}|Y = C_k) &= f(x_i;\mu_k,\sigma_k) \ *\epsilon \\ &= \frac{1}{\sqrt{2\pi\sigma_k^2}}exp(-\frac{(x_i-\mu_k)^2}{2\sigma_k^2}) \end{aligned}$$

其中，μ_k 和 σ_k^2 是正态分布的期望和方差，可通过极大似然估计求得。

对于任意一个Y的取值，贝叶斯都以求解最大化的$P(x_i = X_i^{(test)}|Y = C_k)$ 为目标，这样我们才能够比较在不同标签下我们的样本究竟更靠近哪一个取值。

μ_k 为样本类别C_k中，所有X_i 的平均值。σ_k^2 为在样本类别C_k 中，所有X_i 的方差。对于一个连续的样本值，带入正态分布的公式，就能够得到一个$P(x_i = X_i^{(test)}|Y = C_k)$ 的概率取值。

这个类包含两个参数：

参数	含义
prior	可输入任何类数组结构，形状为（n_classes，） 表示类的先验概率。如果指定，则不根据数据调整先验，如果不指定，则自行根据数据计算先验概率$P(Y)$。
var_smoothing	浮点数，可不填（默认值= 1e-9） 在估计方差时，为了追求估计的稳定性，将所有特征的方差中最大的方差以某个比例添加到估计的方差中。这个比例，由var_smoothing参数控制。

但在实例化的时候，我们不需要对高斯朴素贝叶斯类输入任何的参数，调用的接口也全部sklearn中比较标准的一些搭配，可以说是一个非常轻量级的类，操作非常容易。但过于简单也意味着贝叶斯没有太多的参数可以调整，因此贝叶斯算法的成长空间并不是太大，如果贝叶斯算法的效果不是太理想，我们一般都会考虑换模型。

无论如何，先来进行一次预测试试看吧：

1. **导入需要的库和数据**

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.naive_bayes import GaussianNB
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split

digits = load_digits()
X, y = digits.data, digits.target

Xtrain,Xtest,Ytrain,Ytest =
train_test_split(X,y,test_size=0.3,random_state=420)
```

2. **建模，探索建模结果**

```
gnb = GaussianNB().fit(Xtrain,Ytrain)

#查看分数
acc_score = gnb.score(Xtest,Ytest)

acc_score

#查看预测结果
Y_pred = gnb.predict(Xtest)

#查看预测的概率结果
prob = gnb.predict_proba(Xtest)

prob.shape

prob.shape #每一列对应一个标签下的概率

prob[1,:].sum() #每一行的和都是一

prob.sum(axis=1)
```

3. **使用混淆矩阵来查看贝叶斯的分类结果**

```
from sklearn.metrics import confusion_matrix as CM
CM(Ytest,Y_pred)

# 多分类状况下最佳的模型评估指标是混淆矩阵和整体的准确度
```

2. 多项式朴素贝叶斯以及其变化

2.1 多项式朴素贝叶斯MultinomialNB

多项式贝叶斯可能是除了高斯之外，最为人所知的贝叶斯算法了。它也是基于原始的贝叶斯理论，但假设概率分布是服从一个简单多项式分布。多项式分布来源于统计学中的多项式实验，这种实验可以具体解释为：实验包括n次重复试验，每项试验都有不同的可能结果。在任何给定的试验中，特定结果发生的概率是不变的。

举个例子，比如说一个特征矩阵$\boldsymbol{X}$表示投掷硬币的结果，则得到正面的概率为$P(X = \text{正面}|Y)$ = 0.5，得到反面的概率为$P(X = \text{反面}|Y)$ = 0.5，只有这两种可能并且两种结果互不干涉，并且两个随机事件的概率加和为1，这就是一个二项分布。这种情况下，适合于多项式朴素贝叶斯的特征矩阵应该长这样：

测试编号	X_1：出现正面	X_2：出现反面
0	0	1
1	1	0
2	1	0
3	0	1

假设另一个特征X'表示投掷骰子的结果，则i就可以在[1,2,3,4,5,6]中取值，六种结果互不干涉，且只要样本量足够大，概率都为$1/6$，这就是一个多项分布。多项分布的特征矩阵应该长这样：

测试编号	出现1	出现2	出现3	出现4	出现5	出现6
0	1	0	0	0	0	0
1	0	0	0	0	0	1
2	0	0	1	0	0	0
…						
m	0	0	0	0	0	1

可以看出：

1. **多项式分布擅长的是分类型变量**，在其原理假设中，$P(x_i|Y)$的概率是离散的，并且不同x_i下的$P(x_i|Y)$相互独立，互不影响。虽然sklearn中的多项式分布也可以处理连续型变量，但现实中，如果我们真的想要处理连续型变量，我们应当使用高斯朴素贝叶斯。
2. 多项式实验中的实验结果都很具体，它所涉及的特征往往是次数，频率，计数，出现与否这样的概念，这些概念都是离散的正整数，因此**sklearn中的多项式朴素贝叶斯不接受负值的输入**。

在sklearn中，用来执行多项式朴素贝叶斯的类MultinomialNB包含如下的参数和属性：

class `sklearn.naive_bayes.MultinomialNB`(*alpha=1.0*, *fit_prior=True*, *class_prior=None*)

参数
alpha : 浮点数, 可不填 (默认为1.0) 拉普拉斯或利德斯通平滑的参数λ，如果设置为0则表示完全没有平滑选项。但是需要注意的是，平滑相当于人为给概率加上一些噪音，因此λ 设置得越大，多项式朴素贝叶斯的精确性会越低（虽然影响不是非常大）。
fit_prior : 布尔值, 可不填 (默认为True) 是否学习先验概率$P(Y=c)$。如果设置为false，则所有的样本类别输出都有相同的类别先验概率。即认为每个标签类出现的概率是$\frac{1}{n_classes}$。
class_prior：形似数组的结构，结构为(n_classes,)，可不填（默认为None） 类的先验概率$P(Y=c)$。如果没有给出具体的先验概率则自动根据数据来进行计算。

布尔参数fit_prior表示是否要考虑先验概率，如果是False，则所有的样本类别输出都有相同的类别先验概率。否则，可以自己用第三个参数class_prior输入先验概率，或者不输入第三个参数class_prior让MultinomialNB自己从训练集样本来计算先验概率，此时的先验概率为$P(Y=C_k)=m_k/m$。其中m为训练集样本总数量，m_k为输出为第k个类别的训练集样本数。总结如下：

fit_prior	**class_prior**	**最终先验概率**
False	填或者不填没有意义	$P(Y=C_k)=1/k$
True	不填	$P(Y=C_k)=m_k/m$
True	填	$P(Y=C_k)=class\_prior$

通常我们在实例化多项式朴素贝叶斯的时候，我们会让所有的参数保持默认。先来简单建一个多项式朴素贝叶斯的例子试试看：

1. **导入需要的模块和库**

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_blobs
```

2. **建立数据集**

```
class_1 = 500
class_2 = 500 #两个类别分别设定500个样本
centers = [[0.0, 0.0], [2.0, 2.0]] #设定两个类别的中心
clusters_std = [0.5, 0.5] #设定两个类别的方差
X, y = make_blobs(n_samples=[class_1, class_2],
                  centers=centers,
                  cluster_std=clusters_std,
                  random_state=0, shuffle=False)

Xtrain, Xtest, Ytrain, Ytest = train_test_split(X,y
                                                ,test_size=0.3
                                                ,random_state=420)
```

3. **归一化，确保输入的矩阵不带有负数**

```
#先归一化，保证输入多项式朴素贝叶斯的特征矩阵中不带有负数
mms = MinMaxScaler().fit(Xtrain)
Xtrain_ = mms.transform(Xtrain)
Xtest_ = mms.transform(Xtest)
```

4. **建立一个多项式朴素贝叶斯分类器吧**

```
mnb = MultinomialNB().fit(Xtrain_, Ytrain)

#重要属性：调用根据数据获取的，每个标签类的对数先验概率log(P(Y))
#由于概率永远是在[0,1]之间，因此对数先验概率返回的永远是负值
mnb.class_log_prior_

np.unique(Ytrain)

(Ytrain == 1).sum()/Ytrain.shape[0]

mnb.class_log_prior_.shape

#可以使用np.exp来查看真正的概率值
np.exp(mnb.class_log_prior_)

#重要属性：返回一个固定标签类别下的每个特征的对数概率log(P(Xi|y))
mnb.feature_log_prob_
```

```
mnb.feature_log_prob_.shape

#重要属性：在fit时每个标签类别下包含的样本数。当fit接口中的sample_weight被设置时，该接口返回的值也会受到加权的影响
mnb.class_count_

mnb.class_count_.shape
```

5. **那分类器的效果如何呢?**

```
#一些传统的接口
mnb.predict(Xtest_)

mnb.predict_proba(Xtest_)

mnb.score(Xtest_,Ytest)
```

7. **效果不太理想，思考一下多项式贝叶斯的性质，我们能够做点什么呢?**

```
#来试试看把Xtiain转换成分类型数据吧
#注意我们的Xtrain没有经过归一化，因为做哑变量之后自然所有的数据就不会又负数了
from sklearn.preprocessing import KBinsDiscretizer
kbs = KBinsDiscretizer(n_bins=10, encode='onehot').fit(Xtrain)
Xtrain_ = kbs.transform(Xtrain)
Xtest_ = kbs.transform(Xtest)

mnb = MultinomialNB().fit(Xtrain_, Ytrain)

mnb.score(Xtest_,Ytest)
```

可以看出，多项式朴素贝叶斯的基本操作和代码都非常简单。同样的数据，如果采用哑变量方式的分箱处理，多项式贝叶斯的效果会突飞猛进。

2.2 伯努利朴素贝叶斯BernoulliNB

多项式朴素贝叶斯可同时处理二项分布（抛硬币）和多项分布（掷骰子），其中二项分布又叫做伯努利分布，它是一种现实中常见，并且拥有很多优越数学性质的分布。因此，既然有着多项式朴素贝叶斯，我们自然也就又专门用来处理二项分布的朴素贝叶斯：伯努利朴素贝叶斯。

伯努利贝叶斯类BernoulliN假设数据服从多元伯努利分布，并在此基础上应用朴素贝叶斯的训练和分类过程。多元伯努利分布简单来说，就是数据集中可以存在多个特征，但每个特征都是二分类的，可以以布尔变量表示，也可以表示为{0，1}或者{-1，1}等任意二分类组合。因此，这个类要求将样本转换为二分类特征向量，如果数据本身不是二分类的，那可以使用类中专门用来二值化的参数binarize来改变数据。

伯努利朴素贝叶斯与多项式朴素贝叶斯非常相似，都常用于处理文本分类数据。但由于伯努利朴素贝叶斯是处理二项分布，所以它更加在意的是“存在与否”，而不是“出现多少次”这样的次数或频率，这是伯努利贝叶斯与多项式贝叶斯的根本性不同。在文本分类的情况下，伯努利朴素贝叶斯可以使用单词出现向量（而不是单词计数向量）来训练分类器。文档较短的数据集上，伯努利朴素贝叶斯的效果会更加好。如果时间允许，建议两种模型都试试看。

来看看伯努利朴素贝叶斯类的参数：

class `sklearn.naive_bayes.BernoulliNB`(*alpha=1.0*, *binarize=0.0*, *fit_prior=True*, *class_prior=None*)

伯努利朴素贝叶斯
alpha : 浮点数, 可不填 (默认为1.0) 拉普拉斯或利德斯通平滑的参数λ，如果设置为0则表示完全没有平滑选项。但是需要注意的是，平滑相当于人为给概率加上一些噪音，因此λ设置得越大，多项式朴素贝叶斯的精确性会越低（虽然影响不是非常大），布里尔分数也会逐渐升高。
binarize : 浮点数或None，可不填，默认为0 将特征二值化的阈值，如果设定为None，则会假定说特征已经被二值化完毕
fit_prior : 布尔值, 可不填 (默认为True) 是否学习先验概率$P(Y=c)$。如果设置为false，则不使用先验概率，而使用统一先验概率（uniform prior），即认为每个标签类出现的概率是$\frac{1}{n_classes}$。
class_prior：形似数组的结构，结构为(n_classes,)，可不填（默认为None） 类的先验概率$P(Y=c)$。如果没有给出具体的先验概率则自动根据数据来进行计算。

在sklearn中，伯努利朴素贝叶斯的实现也非常简单：

```
from sklearn.naive_bayes import BernoulliNB

#普通来说我们应该使用二值化的类sklearn.preprocessing.Binarizer来将特征一个个二值化
#然而这样效率过低，因此我们选择归一化之后直接设置一个阈值

mms = MinMaxScaler().fit(Xtrain)
Xtrain_ = mms.transform(Xtrain)
Xtest_ = mms.transform(Xtest)

#不设置二值化
bnl_ = BernoulliNB().fit(Xtrain_, Ytrain)
bnl_.score(Xtest_,Ytest)

#设置二值化阈值为0.5
bnl = BernoulliNB(binarize=0.5).fit(Xtrain_, Ytrain)
bnl.score(Xtest_,Ytest)
```

和多项式贝叶斯一样，伯努利贝叶斯的结果也受到数据结构非常大的影响。因此，根据数据的模样选择贝叶斯，是贝叶斯模型选择中十分重要的一点。

2.3 探索贝叶斯：贝叶斯的样本不均衡问题

接下来，我们来探讨一个分类算法永远都逃不过的核心问题：样本不平衡。贝叶斯由于分类效力不算太好，因此对样本不平衡极为敏感，我们接下来就来看一看样本不平衡如何影响了贝叶斯。

1. **导入需要的模块，建立样本不平衡的数据集**

```
from sklearn.naive_bayes import MultinomialNB, GaussianNB, BernoulliNB
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_blobs
from sklearn.preprocessing import KBinsDiscretizer
from sklearn.metrics import recall_score,roc_auc_score as AUC

class_1 = 50000 #多数类为50000个样本
class_2 = 500 #少数类为500个样本
centers = [[0.0, 0.0], [5.0, 5.0]] #设定两个类别的中心
clusters_std = [3, 1] #设定两个类别的方差
X, y = make_blobs(n_samples=[class_1, class_2],
                  centers=centers,
                  cluster_std=clusters_std,
                  random_state=0, shuffle=False)
X.shape

np.unique(y)
```

2. **查看所有贝叶斯在样本不平衡数据集上的表现**

```
name = ["Multinomial","Gaussian","Bernoulli"]
models = [MultinomialNB(),GaussianNB(),BernoulliNB()]

for name,clf in zip(name,models):
    Xtrain, Xtest, Ytrain, Ytest = train_test_split(X,y
                                                    ,test_size=0.3
                                                    ,random_state=420)
    if name != "Gaussian":
        kbs = KBinsDiscretizer(n_bins=10, encode='onehot').fit(Xtrain)
        Xtrain = kbs.transform(Xtrain)
        Xtest = kbs.transform(Xtest)

    clf.fit(Xtrain,Ytrain)
    y_pred = clf.predict(Xtest)
    proba = clf.predict_proba(Xtest)[:,1]
    score = clf.score(Xtest,Ytest)
    print(name)
    print("\tAccuracy:{:.3f}".format(score))
    print("\tRecall:{:.3f}".format(recall_score(Ytest,y_pred)))
```

```
    print("\tAUC:{:.3f}".format(AUC(Ytest,proba)))
```

从结果上来看，多项式朴素贝叶斯判断出了所有的多数类样本，但放弃了全部的少数类样本，受到样本不均衡问题影响最严重。高斯比多项式在少数类的判断上更加成功一些，至少得到了43.8%的recall。伯努利贝叶斯虽然整体的准确度不如多项式和高斯朴素贝叶斯和，但至少成功捕捉出了77.1%的少数类。可见，伯努利贝叶斯最能够忍受样本不均衡问题。

可是，伯努利贝叶斯只能用于处理二项分布数据，在现实中，强行将所有的数据都二值化不会永远得到好结果，在我们有多个特征的时候，我们更需要一个个去判断究竟二值化的阈值该取多少才能够让算法的效果优秀。这样做无疑是非常低效的。那如果我们的目标是捕捉少数类，我们应该怎么办呢？高斯朴素贝叶斯的效果虽然比多项式好，但是也没有好到可以用来帮助我们捕捉少数类的程度——43.8%，还不如抛硬币的结果。因此，孜孜不倦的统计学家们改进了朴素贝叶斯算法，修正了包括无法处理样本不平衡在内的传统朴素贝叶斯的众多缺点，得到了一些新兴贝叶斯算法，比如补集朴素贝叶斯等。

三、概率分类模型评估指标

接受者操作特征（Receiver Operating Characteristic Curve，ROC）曲线是显示分类器真正率和假正率之间折中的一种图形化方法，也是最常用的评估分类模型好坏的可视化图形，在介绍ROC曲线之前，先回归之前介绍过的混淆矩阵。

1. 回顾混淆矩阵

		Hypothesis output	
		Y	**N**
True class	**p**	TP (True Positive)	FN (False Negative)
	n	FP (False Positive)	TN (True Negative)

2. ROC曲线

该曲线的横坐标为假正率（False Positive Rate, FPR），N是真实负样本的个数，FP是N个负样本中被分类器预测为正样本的个数。

		预测值		
		1	0	
真实值	1	11	10	11 + 10
	0	01	00	01 + 00
		11 + 01	10 + 00	11 + 10 + 01 + 00

假正率
False positive rate

$$FPR = \frac{FP}{N} = \frac{01}{01 + 00}$$

纵坐标为真正率（True Positive Rate, TPR）：

$$TPR = \frac{TP}{P} = \frac{11}{11 + 10}$$

P是真实正样本的个数，TP是P个正样本中被分类器预测为正样本的个数。

四、Kaggle竞赛案例：旧金山犯罪分类预测

此案例中使用的数据集为Kaggle比赛数据集，内容是12年内旧金山城内的犯罪报告。犯罪报告里面包括日期、描述、星期、所属警区、处理结果、地址、GPS定位等信息。分类问题有很多分类器可以选择，此处我们使用朴素贝叶斯算法来进行犯罪类型预测。

1. 导入相关包

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import BernoulliNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import log_loss
```

2. 导入数据集

```
train = pd.read_csv('Kaggle/train.csv',parse_dates = ['Dates'])
test = pd.read_csv('Kaggle/test.csv',parse_dates = ['Dates'],index_col=0)
```

train.csv中的数据时间跨度为12年，包含了将近90w的记录。另外，这部分数据大部分都是分类型，比如犯罪类型、星期几等。

每一列的含义：

- Date：日期
- Category：犯罪类型（标签）
- Descript：对于犯罪更详细的描述
- DayOfWeek：星期几
- PdDistrict：所属警区
- Resolution：处理结果
- Address：发生街区位置
- X and Y：GPS坐标

3. 特征预处理

sklearn.preprocessing模块中的`LabelEncoder`函数可以对类别做编号，我们用它对犯罪类型做编号：

```
#对犯罪类别:Category; 用LabelEncoder进行编号
leCrime = LabelEncoder()
crime = leCrime.fit_transform(train.Category) #39种犯罪类型

#用get_dummies因子化星期几、街区、小时等特征
days=pd.get_dummies(train.DayOfWeek)
district = pd.get_dummies(train.PdDistrict)
hour = train.Dates.dt.hour
hour = pd.get_dummies(hour)

#组合特征形成训练集
trainData = pd.concat([hour, days, district], axis = 1) #将特征进行左右拼接
trainData['crime'] = crime #追加标签列

#得到测试集
days = pd.get_dummies(test.DayOfWeek)
district = pd.get_dummies(test.PdDistrict)
hour = test.Dates.dt.hour
hour = pd.get_dummies(hour)
testData = pd.concat([hour, days, district], axis=1)
testData.head()
```

4. 切分数据集并建模

在机器学习算法中，我们常常使用损失函数来衡量模型预测的好坏。损失函数越小，模型拟合就越好。

这里我们使用对数损失函数（logarithmic loss function）：

$$L(Y,P(Y|X)) = -logP(Y|X)$$

$P(Y|X)$ 通俗的解释就是：在当前模型的基础上，对于样本X，其预测值为Y，也就是预测正确的概率。由于概率之间同时满足需要使用乘法，为了将其转化为加法，我们将其取对数。最后由于是损失函数，所以预测正确的概率越高，其损失值应该是越小，因此再加个负号取相反数。

```
#切分数据集
Xtrain, Xtest, Ytrain, Ytest =
train_test_split(trainData.iloc[:,:-1],trainData.iloc[:,-1],test_size=0.2)

#训练模型
BNB = BernoulliNB()
BNB.fit(Xtrain, Ytrain)

# 计算损失函数
proba = BNB.predict_proba(Xtest)
logLoss=log_loss(Ytest, proba)
logLoss

#使用模型预测testData
BNB.predict(testData)
```

五、朴素贝叶斯算法总结

朴素贝叶斯算法的主要原理基本已经做了总结，这里对朴素贝叶斯的优缺点做一个总结。

1. 朴素贝叶斯的优点

- 既简单又快速，预测表现良好；
- 直接使用概率预测，通常很容易理解；
- 如果变量满足独立条件，相比逻辑回归等其他分类方法，朴素贝叶斯分类器性能更优，且只需少量训练数据；

- 相较于数值变量，朴素贝叶斯分类器在多个分类变量的情况下表现更好。若是数值变量，需要正态分布假设。

这些有点可以使得朴素贝叶斯分类器通常很适合作为分类的初始解。如果分类效果满足要求，那么万事大吉，你获得了一个非常快速且容易解释的分类器。但如果分类效果不够好，那么你可以尝试更复杂的分类模型，与朴素贝叶斯分类器的分类效果进行对比，看看复杂模型的分类效果究竟如何。

2. 朴素贝叶斯的缺点

- 理论上，朴素贝叶斯模型与其他分类方法相比具有最小的误差率。但是实际上并非总是如此，这是因为朴素贝叶斯模型给定输出类别的情况下，假设属性之间相互独立，这个假设在实际应用中往往是不成立的，在属性个数比较多或者属性之间相关性较大时，分类效果不好。而在属性相关性较小时，朴素贝叶斯性能最为良好。对于这一点，有关朴素贝叶斯之类的算法通过考虑部分关联性适度改进。
- 需要知道先验概率，且先验概率很多时候取决于假设，假设的模型可以有很多种，因此在某些时候会由于假设的先验模型的原因导致预测效果不佳。
- 由于我们是通过先验和数据来决定后验的概率从而决定分类，所以分类决策存在一定的错误率。
- 对输入数据的表达形式很敏感。

3. 朴素贝叶斯的4种应用

实时预测：毫无疑问，朴素贝叶斯很快。

多类预测：这个算法以多类别预测功能闻名，因此可以用来预测多类目标变量的概率。

文本分类/垃圾邮件/情感分析：相比其他算法，朴素贝叶斯的应用主要集中在文本分类（变量类型多，且更独立），具有较高的成功率。因此被广泛应用于垃圾邮件过滤（识别垃圾邮件）和情感分析（在社交媒体平台分辨积极情绪和消极情绪的用户）。

推荐系统：朴素贝叶斯分类器和协同过滤结合使用可以过滤出用户想看到和不想看到的东西。

4. 关于朴素贝叶斯分类器的Tips

以下是一些可以提升朴素贝叶斯分类器性能的小方法：

- 如果连续特征不是正态分布的，我们应该使用各种不同的方法将其转换为正态分布。
- 如果测试数据集具有 "零概率" 的问题，应用平滑技术 "拉普拉斯估计" 修正数据集。
- 删除重复出现的高度相关的特征，可能会丢失频率信息，影响效果。
- 朴素贝叶斯分类在参数调整上选择有限。建议把重点放在数据预处理和特征工程。