朴素贝叶斯

1 朴素贝叶斯

1.1 后验概率

假设训练集为

$$T = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$$

其中 $x_i \in R^n, y_i \in \{c_1, c_2, ..., c_K\}, i = 1, 2, ..., N$。

对于后验概率 $P(Y = c_k|X = x)$，可以由**贝叶斯定理**计算：

$$P(Y = c_k|X = x) = \frac{P(X = x|Y = c_k)P(Y = c_k)}{\sum_{k=1}^{K} P(X = x|Y = c_k)P(Y = c_k)}$$

因此后验概率 $P(Y = c_k|X = x)$ 的计算等价于计算先验概率 $P(Y = c_k)$ 和条件概率 $P(X = x|Y = c_k)$。

1.2 条件独立性

朴素贝叶斯假定数据各属性间的**条件独立性**，即：

$$\begin{aligned} P(X = x|Y = c_k) &= P(X^1 = x^1, X^2 = x^2, ..., X^n = x^n|Y = c_k) \\ &= \prod_{j=1}^{n} P(X^j = x^j|Y = c_k) \end{aligned}$$

因此后验概率的计算又等价于先验概率 $P(Y = c_k)$ 和条件概率 $P(X^j = x^j|Y = c_k)$ 的计算。

1.3 极大似然估计

对于先验概率 $P(Y = c_k)$ 和条件概率 $P(X^j = x^j|Y = c_k)$，我们均可使用极大似然法来进行估计。两者的极大似然估计分别为：

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

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

1.4 朴素贝叶斯

朴素贝叶斯算法如下：

输入：数据集 $T=\{(x_1,y_1),(x_2,y_2),...,(x_N,y_N)\}$，其中 $x_i\in R^n, y_i\in\{c_1,c_2,...,c_K\}, i=1,2,...,N$ 和实例 x。

1. 计算先验概率 $P(Y=c_k), k=1,2,...,K$

$$P(Y=c_k)=\frac{\sum_{i=1}^{N}I(y_i=c_k)}{N}$$

2. 计算条件概率 $P(X^j=x^j|Y=c_k), j=1,2,...,n; k=1,2,...,K$

$$P(X^j=x^j|Y=c_k)=\frac{\sum_{i=1}^{N}I(x_i^j=x^j,y_i=c_k)}{\sum_{i=1}^{N}I(y_i=c_k)}$$

3. 计算后验概率 $P(Y=c_k|X=x), k=1,2,...,K$

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

4. 确定 x 的类 $y=argmax_k P(Y=c_k|X=x)$

输出：x 的类 y。

1.5 拉普拉斯平滑

为防止出现极大似然估计出现估计概率为 0 的情况，常使用**平滑方式**，其方法如下：

$$P(Y=c_k)=\frac{\sum_{i=1}^{N}I(y_i=c_k)+\lambda}{N+K\lambda}$$

$$P(X^j=x^j|Y=c_k)=\frac{\sum_{i=1}^{N}I(x_i^j=x^j,y_i=c_k)+\lambda}{\sum_{i=1}^{N}I(y_i=c_k)+S_j\lambda}$$

其中 S_j 为 X^j 的取值个数，$\lambda\geq 0$ 为平滑因子。当 $\lambda=1$ 时为**拉普拉斯平滑**。

2 代码实现

考虑以下数据：

X^1	X^2	Y
1	S	-1
1	M	-1
1	M	1
1	S	1
1	S	-1
2	S	-1
2	M	-1
2	M	1
2	L	1
2	L	1
3	L	1
3	M	1
3	M	1
3	L	1
3	L	-1

2.1 sklearn 实现

准备数据：

```
import numpy as np
X = np.array([[1,'S'],[1,'M'],[1,'M'],[1,'S'],[1,'S'],\
              [2,'S'],[2,'M'],[2,'M'],[2,'L'],[2,'L'],\
              [3,'L'],[3,'M'],[3,'M'],[3,'L'],[3,'L']])
Y = np.array([-1,-1,1,1,-1,-1,-1,1,1,1,1,1,1,1,-1]).reshape(-1,1)
```

朴素贝叶斯的接口位于 sklearn.naive_bayes 下，该类下有适用于离散型的 CategoricalNB 接口和连续型的 GaussianNB 接口等。此处使用 CategoricalNB 进行分类，其文档可见**此处**。其中较为重要的参数有：

- alpha: 平滑参数 λ

注意到 CategoricalNB 接口只接收数值型的输入，所以需要使用 sklearn.preprocessing.LabelEncoder 对其进行编码。

```
from sklearn.naive_bayes import CategoricalNB
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
le.fit(X[:,1])
```

```
X[:,1] = le.transform(X[:,1])
clf = CategoricalNB(alpha = 0)
clf.fit(X,Y)
```

对于新样本 $(2,S)^T$，预测结果如下：

```
new_x = np.array([[2,'S']])
new_x[:,1] = le.transform(new_x[:,1])
clf.predict_proba(new_x)
```

```
## array([[0.75, 0.25]])
```

2.2 朴素贝叶斯实现

定义一个 NaiveBayes 类进行实现。与 sklearn 中的接口类似，此处也定义了 fit 函数和 predict 函数分别进行拟合和预测。实现中为了便于实现，使用了 pandas 的计数功能。同时此处预测只能对单个新数据进行预测，事实上对代码稍加改动便可以实现同时预测多个新样本的功能。

```
import pandas as pd

class NaiveBayes:
    def __init__(self,lam=1):
        self.lam = lam

    def fit(self,X,Y):
        data = pd.concat([pd.DataFrame(X,columns=[str(i) for i in range(X.shape[1])]),\
        pd.DataFrame(Y,columns=['Y'])],axis=1)
        ## 计算先验概率
        self.piror = dict(data['Y'].value_counts())
        self.K = len(self.piror)
        ## 计算条件概率
        self.S = {}
        self.CondProb = {}
        for col in data.columns[:-1]:
            tmp = data[[col,'Y']]
```

```
            tmp = tmp.groupby([col,'Y']).agg({'Y':'count'})
            tmp.columns=['count']
            tmp = tmp.reset_index()
            self.CondProb[col] = tmp
            self.S[col] = len(tmp[col].value_counts())

    def predict(self,new_X):
        posterior = []
        for k in self.piror.keys():
            piror = (self.piror[k]+self.lam)/(sum(self.piror.values())+self.lam*self.K)
            for i in range(new_X.shape[1]):
                S = self.S[str(i)]
                cond = self.CondProb[str(i)]
                count = cond.loc[(cond[str(i)]==new_X[0,i])&(cond['Y']==k),['count']]
                count = (count.values[0,0]+self.lam)/(self.piror[k]+S*self.lam)
                piror *= count
            posterior.append(piror)
        posterior = [round(p/sum(posterior),2) for p in posterior]
        return dict(zip(self.piror.keys(),posterior))
```

其预测结果如下所示 (该段代码在 Jupyter 下可正常运行，但在 Rmd 中报错，经检查代码无错误):

```
clf = NaiveBayes(lam=0)
clf.fit(X,Y)
clf.predict(np.array([[2,'S']]))
```

```
## {1: 0.25, -1: 0.75}
```