第4章 朴素贝叶斯

1．朴素贝叶斯法是典型的生成学习方法。生成方法由训练数据学习联合概率分布 $P(X,Y)$，然后求得后验概率分布$P(Y|X)$。具体来说，利用训练数据学习$P(X|Y)$和$P(Y)$的估计，得到联合概率分布：

$P(X,Y)$＝$P(Y)P(X|Y)$

概率估计方法可以是极大似然估计或贝叶斯估计。

2．朴素贝叶斯法的基本假设是条件独立性，

$$\begin{aligned} P(X=x | Y=c_{k}) &=P\left(X^{(1)}=x^{(1)}, \cdots, X^{(n)}=x^{(n)} | Y=c_{k}\right) \\ &=\prod_{j=1}^{n} P\left(X^{(j)}=x^{(j)} | Y=c_{k}\right) \end{aligned}$$

这是一个较强的假设。由于这一假设，模型包含的条件概率的数量大为减少，朴素贝叶斯法的学习与预测大为简化。因而朴素贝叶斯法高效，且易于实现。其缺点是分类的性能不一定很高。

3．朴素贝叶斯法利用贝叶斯定理与学到的联合概率模型进行分类预测。

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

将输入x分到后验概率最大的类y。

$$y=\arg \max _{c_{k}} P\left(Y=c_{k}\right) \prod_{j=1}^{n} P\left(X_{j}=x^{(j)} | Y=c_{k}\right)$$

后验概率最大等价于0-1损失函数时的期望风险最小化。

模型：

- 高斯模型
- 多项式模型
- 伯努利模型

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

from collections import Counter
import math
```

```python
# data
def create_data():
    iris = load_iris()
    df = pd.DataFrame(iris.data, columns=iris.feature_names)
    df['label'] = iris.target
    df.columns = [
        'sepal length', 'sepal width', 'petal length', 'petal width', 'label'
    ]
    data = np.array(df.iloc[:100, :])
    # print(data)
    return data[:, :-1], data[:, -1]
```

```python
X, y = create_data()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

```python
X_test[0], y_test[0]
```

```
(array([5.7, 2.6, 3.5, 1. ]), 1.0)
```

参考：https://machinelearningmastery.com/naive-bayes-classifier-scratch-python/

GaussianNB 高斯朴素贝叶斯

特征的可能性被假设为高斯

概率密度函数：$P(x_i|y_k) = \frac{1}{\sqrt{2\pi\sigma^2_{yk}}}exp(-\frac{(x_i-\mu_{yk})^2}{2\sigma^2_{yk}})$

数学期望(mean)：μ

方差：$\sigma^2 = \frac{\sum(X-\mu)^2}{N}$

```python
class NaiveBayes:
    def __init__(self):
        self.model = None

    # 数学期望
    @staticmethod
    def mean(X):
        return sum(X) / float(len(X))

    # 标准差（方差）
    def stdev(self, X):
        avg = self.mean(X)
```

```
        return math.sqrt(sum([pow(x - avg, 2) for x in X]) / float(len(X)))

    # 概率密度函数
    def gaussian_probability(self, x, mean, stdev):
        exponent = math.exp(-(math.pow(x - mean, 2) /
                              (2 * math.pow(stdev, 2))))
        return (1 / (math.sqrt(2 * math.pi) * stdev)) * exponent

    # 处理X_train
    def summarize(self, train_data):
        summaries = [(self.mean(i), self.stdev(i)) for i in zip(*train_data)]
        return summaries

    # 分类别求出数学期望和标准差
    def fit(self, X, y):
        labels = list(set(y))
        data = {label: [] for label in labels}
        for f, label in zip(X, y):
            data[label].append(f)
        self.model = {
            label: self.summarize(value)
            for label, value in data.items()
        }
        return 'gaussianNB train done!'

    # 计算概率
    def calculate_probabilities(self, input_data):
        # summaries:{0.0: [(5.0, 0.37),(3.42, 0.40)], 1.0: [(5.8, 0.449),(2.7,
0.27)]}
        # input_data:[1.1, 2.2]
        probabilities = {}
        for label, value in self.model.items():
            probabilities[label] = 1
            for i in range(len(value)):
                mean, stdev = value[i]
                probabilities[label] *= self.gaussian_probability(
                    input_data[i], mean, stdev)
        return probabilities

    # 类别
    def predict(self, X_test):
        # {0.0: 2.9680340789325763e-27, 1.0: 3.5749783019849535e-26}
        label = sorted(
            self.calculate_probabilities(X_test).items(),
            key=lambda x: x[-1])[-1][0]
        return label

    def score(self, X_test, y_test):
        right = 0
```

```
        for X, y in zip(X_test, y_test):
            label = self.predict(X)
            if label == y:
                right += 1

        return right / float(len(X_test))
```

```
model = NaiveBayes()
```

```
model.fit(X_train, y_train)
```

```
'gaussianNB train done!'
```

```
print(model.predict([4.4,  3.2,  1.3,  0.2]))
```

```
0.0
```

```
model.score(X_test, y_test)
```

```
1.0
```

scikit-learn实例

```
from sklearn.naive_bayes import GaussianNB
```

```
clf = GaussianNB()
clf.fit(X_train, y_train)
```

```
GaussianNB()
```

```
clf.score(X_test, y_test)
```

```
1.0
```

```
clf.predict([[4.4,  3.2,  1.3,  0.2]])
```

```
array([0.])
```

```
from sklearn.naive_bayes import BernoulliNB, MultinomialNB # 伯努利模型和多项式模型
```

第4章朴素贝叶斯法-习题

习题4.1

用极大似然估计法推出朴素贝叶斯法中的概率估计公式(4.8)及公式 (4.9)。

解答：

第1步： 证明公式(4.8)：$P(Y=c_k) = \frac{\displaystyle \sum_{i=1}^N I(y_i=c_k)}{N}$

由于朴素贝叶斯法假设Y是定义在输出空间$\mathcal{Y}$上的随机变量，因此可以定义$P(Y=c_k)$概率为p。

令$m=\sum_{i=1}^N I(y_i=c_k)$，得出似然函数：$L(p)=f_D(y_1,y_2,\cdots,y_n|\theta)=\binom{N}{m}p^m(1-p)^{(N-m)}$使用微分求极值，两边同时对$p$求微分：

$$\begin{aligned} 0 &= \binom{N}{m}\left[mp^{(m-1)}(1-p)^{(N-m)}-(N-m)p^m(1-p)^{(N-m-1)}\right] \\ &= \binom{N}{m}\left[p^{(m-1)}(1-p)^{(N-m-1)}(m-Np)\right] \end{aligned}$$

可求解得到

$p=0,p=1,p=\frac{m}{N}$

显然$P(Y=c_k)=p=\frac{m}{N}=\frac{\displaystyle \sum_{i=1}^N I(y_i=c_k)}{N}$，公式(4.8)得证。

第2步：证明公式(4.9)：$P(X^{(j)}=a_{jl}|Y=c_k)=\dfrac{\displaystyle\sum_{i=1}^N I(x_i^{(j)}=a_{jl},y_i=c_k)}{\displaystyle\sum_{i=1}^N I(y_i=c_k)}$

令$P(X^{(j)}=a_{jl}|Y=c_k)=p$，令$m=\displaystyle\sum_{i=1}^N I(y_i=c_k), q=\sum_{i=1}^N I(x_i^{(j)}=a_{jl},y_i=c_k)$，得出似然函数：$L(p)=\binom{m}{q}p^q(i-p)^{m-q}$使用微分求极值，两边同时对$p$求微分：

$$\begin{aligned} 0 &= \binom{m}{q}\left[qp^{(q-1)}(1-p)^{(m-q)}-(m-q)p^q(1-p)^{(m-q-1)}\right] \\ &= \binom{m}{q}\left[p^{(q-1)}(1-p)^{(m-q-1)}(q-mp)\right] \end{aligned}$$ 可求解得到$p=0,p=1,p=\dfrac{q}{m}$

显然$P(X^{(j)}=a_{jl}|Y=c_k)=p=\dfrac{q}{m}=\dfrac{\displaystyle\sum_{i=1}^N I(x_i^{(j)}=a_{jl},y_i=c_k)}{\displaystyle\sum_{i=1}^N I(y_i=c_k)}$，公式(4.9)得证。

习题4.2

用贝叶斯估计法推出朴素贝叶斯法中的慨率估计公式(4.10)及公式(4.11)

解答：

第1步：证明公式(4.11)：$P(Y=c_k)=\dfrac{\displaystyle\sum_{i=1}^N I(y_i=c_k)+\lambda}{N+K\lambda}$

加入先验概率，在没有任何信息的情况下，可以假设先验概率为均匀概率（即每个事件的概率是相同的）。

可得$p=\dfrac{1}{K}\Leftrightarrow pK-1=0\quad(1)$

根据习题4.1得出先验概率的极大似然估计是$pN-\displaystyle\sum_{i=1}^N I(y_i=c_k)=0\quad(2)$

存在参数λ使得$(1)\cdot\lambda+(2)=0$

所以有$\lambda(pK-1)+pN-\sum_{i=1}^N I(y_i=c_k)=0$可得$P(Y=c_k)=\dfrac{\displaystyle\sum_{i=1}^N I(y_i=c_k)+\lambda}{N+K\lambda}$，公式(4.11)得证。

第2步：证明公式(4.10)：$P_{\lambda}(X^{(j)}=a_{jl}|Y=c_k)=\dfrac{\displaystyle\sum_{i=1}^N I(x_i^{(j)}=a_{jl},y_i=c_k)+\lambda}{\displaystyle\sum_{i=1}^N I(y_i=c_k)+S_j\lambda}$

根据第1步，可同理得到$P(Y=c_k,x^{(j)}=a_{jl})=\dfrac{\displaystyle\sum_{i=1}^N I(y_i=c_k,x_i^{(j)}=a_{jl})+\lambda}{N+KS_j\lambda}$

$$
\begin{aligned}
P(x^{(j)}=a_{jl}|Y=c_k) &= \frac{P(Y=c_k,x^{(j)}=a_{jl})}{P(y_i=c_k)} \\
&= \frac{\dfrac{\displaystyle\sum_{i=1}^N I(y_i=c_k,x_i^{(j)}=a_{jl})+\lambda}{N+KS_j\lambda}}{\dfrac{\displaystyle\sum_{i=1}^N I(y_i=c_k)+\lambda}{N+K\lambda}} \\
&= (\lambda\text{可以任意取值，于是取}\lambda=S_j\lambda) \\
&= \frac{\dfrac{\displaystyle\sum_{i=1}^N I(y_i=c_k,x_i^{(j)}=a_{jl})+\lambda}{N+KS_j\lambda}}{\dfrac{\displaystyle\sum_{i=1}^N I(y_i=c_k)+\lambda}{N+KS_j\lambda}} \\
&= \frac{\displaystyle\sum_{i=1}^N I(y_i=c_k,x_i^{(j)}=a_{jl})+\lambda}{\displaystyle\sum_{i=1}^N I(y_i=c_k)+\lambda} (\text{其中}\lambda=S_j\lambda) \\
&= \frac{\displaystyle\sum_{i=1}^N I(x_i^{(j)}=a_{jl},y_i=c_k)+\lambda}{\displaystyle\sum_{i=1}^N I(y_i=c_k)+S_j\lambda}
\end{aligned}
$$

公式(4.11)得证。

参考代码：https://github.com/wzyonggege/statistical-learning-method

本文代码更新地址：https://github.com/fengdu78/lihang-code

习题解答：https://github.com/datawhalechina/statistical-learning-method-solutions-manual

中文注释制作：机器学习初学者公众号：ID:ai-start-com

配置环境：python 3.5+

代码全部测试通过。