使用贝叶斯方法进行推断

2021 年 4 月 11 日

1　独立性和条件独立性

1.1　独立性

对于事件 A 和事件 B，我们说事件 A 和事件 B 是互相独立的，如果：

$$\Pr(A \cap B) = \Pr(A)\Pr(B). \tag{1}$$

1.2　条件概率

当已知事件 A 发生时，事件 B 发生的概率，称为事件 B 关于事件 A 的条件概率，记做：

$$\Pr(B|A). \tag{2}$$

容易证明，对任意事件 A、事件 B，恒有：

$$\Pr(A)\Pr(B|A) = \Pr(A \cap B) \tag{3}$$

如此一来，我们得到了一种计算条件概率的方法：

$$\Pr(B|A) = \frac{\Pr(A \cap B)}{\Pr(A)} = \frac{\Pr(A|B)\Pr(B)}{\Pr(A)} \tag{4}$$

式中，我们要求 $\Pr(A) \neq 0$.

事件的独立性可以用条件概率来表述，具体地，我们有

定理 1. 设 A, B 是事件，则

$$\Pr(A \cap B) = \Pr(A)\Pr(B) \iff \Pr(A) = \Pr(A|B). \tag{5}$$

证明. 充分性. 设给定 $\Pr(A) = \Pr(A|B)$. 那么

$$\Pr(A)\Pr(B) = \Pr(A|B)\Pr(B) = \Pr(A \cap B) \tag{6}$$

从而充分性成立.

必要性. 设给定 $\Pr(A \cap B) = \Pr(A)\Pr(B)$. 那么, 假设 $\Pr(B) \neq 0$, 则

$$\frac{\Pr(A \cap B)}{\Pr(B)} = \frac{\Pr(A)\Pr(B)}{\Pr(B)} = \Pr(A) \tag{7}$$

依式4, 我们得到

$$\Pr(A|B) = \Pr(A) \tag{8}$$

从而必要性成立. □

1.3 条件独立性

条件独立性是说, 当某个事件发生时, 另外两个 (或多个) 事件互相独立. 具体地, 设 A, B, C 是事件, 则事件 A, 事件 B 关于事件 C 条件独立是指:

$$\Pr(A \cap B|C) = \Pr(A|C)\Pr(B|C) \tag{9}$$

例如:「在吸烟的人群中, 男性患肺癌的风险和女性患肺癌的风险相当」就是一个条件独立性的陈述. 对于一个受访者, 用 A 表示受访者是男性, $\neg A$ 表示受访者是女性, 用 B 表示受访者患肺癌, $\neg B$ 表示受访者不患肺癌, 用 C 表示吸烟, 则上述论断可写为:

$$\Pr(A \cap B|C) = \Pr(A|C)\Pr(B|C) \tag{10}$$

我们亦有条件独立性的等价表述:

定理 2. *设 A, B, C 是事件, 则*

$$\Pr(A \cap B|C) = \Pr(A|C)\Pr(B|C) \iff \Pr(A|B \cap C) = \Pr(A|C) \tag{11}$$

证明. 充分性. 设给定 $\Pr(A|B \cap C) = \Pr(A|C)$. 注意到

$$\Pr(A|B \cap C) = \frac{\Pr(A \cap B \cap C)}{\Pr(B \cap C)} = \frac{\Pr(A \cap B \cap C)}{\Pr(B|C)\Pr(C)} \tag{12}$$

所以

$$\frac{\Pr(A \cap B \cap C)}{\Pr(B|C)\Pr(C)} = \Pr(A|C) \tag{13}$$

所以

$$\frac{\Pr(A\cap B\cap C)}{\Pr(C)}=\Pr(A|C)\Pr(B|C) \tag{14}$$

对上列等式左边应用式4，得

$$\Pr(A\cap B|C)=\Pr(A|C)\Pr(B|C) \tag{15}$$

于是充分性得证．从上述证明过程倒推回去可得到必要性． □

2 朴素贝叶斯分类器

设我们有表 1 所示的数据集：

表 1: 数据集

$\boldsymbol{x}_1$	$\cdots$	$\boldsymbol{x}_m$	$\boldsymbol{y}$
$x_1^{(1)}$	$\cdots$	$x_m^{(1)}$	y_1
$\vdots$		$\vdots$	$\vdots$
$x_1^{(N)}$	$\cdots$	$x_m^{(N)}$	y_N

我们希望利用表 1 所提供的数据训练得到一个分类器，它能够对于一个新的输入 $\boldsymbol{x}^\star$，判断（预测）出该输入所对应的标签 $y^\star$.

设标签变量 $\boldsymbol{y}$ 的取值范围是在 $\{c_1,c_2,\cdots,c_k\}$，而变量 $\boldsymbol{x}_1,\cdots,\boldsymbol{x}_m$ 的取值都是离散的，那么每当得到一组输入 $\boldsymbol{x}^\star=(x_1^\star,\cdots,x_m^\star)$，我们能够计算出后验概率：

$$\Pr(c_j|\boldsymbol{x}^\star). \tag{16}$$

$y_\star$ 则是选取自

$$y_\star=\underset{c\in\{c_1,\cdots,c_k\}}{\arg\max}\ \Pr(c|\boldsymbol{x}^\star) \tag{17}$$

所以说现在的重点，就是如何去计算 $\Pr(c|\boldsymbol{x}^\star)$.

2.1 条件独立性假设

首先我们继续上一节的计算过程，展开 $\Pr(c|\boldsymbol{x}^\star)$：

$$\Pr(c|\boldsymbol{x}^\star) = \frac{\Pr(c)\Pr(\boldsymbol{x}^\star|c)}{\Pr(\boldsymbol{x}^\star)} \tag{18}$$

$$\propto \Pr(c)\Pr(\boldsymbol{x}^\star|c) \tag{19}$$

$$= \Pr(c)\Pr(\bigcap_{j=1}^{m} x_j^\star|c) \tag{20}$$

在贝叶斯方法中，有这样一个条件独立性假设：

$$\Pr(\bigcap_{j=1}^{m} x_j^\star|c) = \prod_{j=1}^{m} \Pr(x_j^\star|c) \tag{21}$$

于是就有

$$\Pr(c|\boldsymbol{x}^\star) \propto \Pr(c)\prod_{j=1}^{m} \Pr(x_j^\star|c) \tag{22}$$

于是就有

$$y^\star = \mathop{\arg\max}_{c\in\{c_1,\cdots,c_k\}} \Pr(c)\prod_{j=1}^{m} \Pr(x_j^\star|c). \tag{23}$$

这就是贝叶斯分类器的基本工作原理.

2.2 拉普拉斯平滑

为了避免连乘操作中出现的某个 $\Pr(x_j^\star|c)$ 为 0 使得整个连乘式的结果为 0，可以引入拉普拉斯平滑操作. 具体地，当在计算 $\Pr(x_j^\star|c)$ 时，需要先找到数据集（也就是表 1）中，那些 $\boldsymbol{y}$ 列取值为 c 的行，然后再在这些行里边，找出变量 $\boldsymbol{x}_j$ 取值为 $x_j^\star$ 的哪些行，并且统计这些行的行数，当统计出来的行数为 0 时，就将 0 替换为 1，这就是拉普拉斯平滑操作.

3 在西瓜数据集上进行推断

我们从互联网上收集来了西瓜数据集（表 2）：

表 2: 西瓜数据集

id	color	root	knock	texture	navel	touch	quality
1	青绿	蜷缩	浊响	清晰	凹陷	硬滑	是
2	乌黑	蜷缩	沉闷	清晰	凹陷	硬滑	是
3	乌黑	蜷缩	浊响	清晰	凹陷	硬滑	是
4	青绿	蜷缩	沉闷	清晰	凹陷	硬滑	是
5	浅白	蜷缩	浊响	清晰	凹陷	硬滑	是
6	青绿	稍蜷	浊响	清晰	稍凹	软粘	是
7	乌黑	稍蜷	浊响	稍糊	稍凹	软粘	是
8	乌黑	稍蜷	浊响	清晰	稍凹	硬滑	是
9	乌黑	稍蜷	沉闷	稍糊	稍凹	硬滑	否
10	青绿	硬挺	清脆	清晰	平坦	软粘	否
11	浅白	硬挺	清脆	模糊	平坦	硬滑	否
12	浅白	蜷缩	浊响	模糊	平坦	软粘	否
13	青绿	稍蜷	浊响	稍糊	凹陷	硬滑	否
14	浅白	稍蜷	沉闷	稍糊	凹陷	硬滑	否
15	乌黑	稍蜷	浊响	清晰	稍凹	软粘	否
16	浅白	蜷缩	浊响	模糊	平坦	硬滑	否
17	青绿	蜷缩	沉闷	稍糊	稍凹	硬滑	否

id 那一列是每个西瓜的唯一编号，不用做变量，quality 那列则是目标变量（也就是标签），第 2 列到第 7 列都是自变量．有了这些数据，再加上我们刚刚学过的朴素贝叶斯方法，使我们能够实现一个模型，这个模型能够根据西瓜的外表（颜色、根蒂、敲声、纹理等）来预判一个西瓜是否是好瓜．

相比于支持向量机、随机森林、决策树和神经网络，朴素贝叶斯分类器的训练非常简单——仅仅是将数据原样保存就可以了，计算工作主要是在推断是进行的．我们下面基于 Pandas 和 Numpy 实现一个简单的朴素贝叶斯分类器，它能够接受离散取值的自变量和离散取值的因变量．它的初始化过程是这样的：读入一个 pd.DataFrame 并将其保存，以第 1 列到倒数第 2 列作为自变量，最后 1 列作为因变量．

```
import pandas as pd
from typing import Union
```

```
from typing import Any
import numpy as np

watermelons = pd.read_csv('watermelon.csv')
watermelons.iloc[:, 0] = None

del watermelons['id']
del watermelons['density']
del watermelons['sugar']

# 朴素贝叶斯分类器
class NaiveBayesianClassifier:

    # 目标变量默认为最后 1 列
    # 第 1 列到倒数第 2 列是自变量
    def __init__(self, data: pd.DataFrame):
        self.data = data

    # target 为目标变量取值
    # 返回先验概率 Pr(target)
    def prior_prob(self, target: Union[str, int]) -> float:
        # 总变量个数
        n_var = self.data.shape[1]

        # 标签为 target 的记录的个数
        n_target = (self.data.iloc[:, n_var-1] == target).sum()

        # 样本量
        n_samples = self.data.shape[0]

        # 先验概率
        pr_target = n_target / n_samples

        return pr_target

    # target 为目标变量的取值
    # input_variables 为输入自变量向量
    # 输出为 Pr(target | input_variable) * Pr(input_variable)
    def probability(
```

```
        self,
        target: Any,
        input_variables: np.array
    ) -> float:

        # 先验概率
        pr_target = self.prior_prob(target)

        return pr_target

    # 根据条件独立性计算条件概率 Pr(input_variables | target)
    def cond_prob(
        self,
        target: Any,
        input_variables: np.array
    ) -> float:
        col_idx = 0
        n_cols = self.data.shape[1]
        prob = 1
        while col_idx <= n_cols-2:
            prob = prob * self.single_cond_prob(
                col_idx,
                input_variables[col_idx],
                target
            )
            col_idx = col_idx + 1

        return prob

    # 计算单个变量的条件概率 Pr(input_variable | target)
    def single_cond_prob(
        self,
        col_idx: int,
        input_variable: Any,
        target: Any
    ) -> float:

        n_cols = self.data.shape[1]
        last_col = self.data.iloc[:, n_cols-1]
```

```python
        selector = last_col == target
        subset = self.data.loc[selector, :].iloc[:, col_idx]

        match_selector = subset == input_variable
        n_matches = match_selector.sum() or 1
        subset_size = subset.shape[0]

        return n_matches / subset_size

    # 根据输入预测所属类别
    def predict(self, input_variables: np.array) -> Any:

        n_cols = self.data.shape[1]
        unique_classes = self.data.iloc[:, n_cols-1].unique()
        odds = []
        for c in unique_classes:
            odds.append(
                self.cond_prob(c, input_variables)
            )
        odds_np = np.array(odds)
        max_idx = odds_np.argmax()
        return unique_classes[max_idx]

f = NaiveBayesianClassifier(watermelons)

row_idx = 0
n_cols = f.data.shape[1]
input_variables = f.data.iloc[row_idx, range(0, n_cols-1)]

print(f.predict(input_variables))
```