机器学习讲义 (L14)：逻辑回归

授课教师：王贝伦 / 助教：张嘉琦，黄旭，谈笑，徐浩卿

1 贝叶斯分类器

贝叶斯学派认为，可以把特征样本数据 X 与对应的类标签 y 都看作随机变量，以统计学的方式对标签进行估计。训练集中的样本分布恰能用来作为先验分布参与估计的计算，一个常见的估计法被称为最大后验概率估计。

1.1 最大后验概率估计

考虑含参总体分布 $f(\mathbf{x};\theta)$，如果把 θ 当作参数，一个经常使用的估计是最大似然估计法。现在我们可以从训练集中获得 θ 的先验分布，意味着我们可以把 θ 当作随机变量来估计。具体来说，我们可以将训练集当作条件，找到这个条件下拥有最大概率的 θ 值，此值即为对 θ 的估计值。利用贝叶斯公式，我们可以作以下推导：

$$\begin{aligned} f(\theta|\mathbf{x}_1,\mathbf{x}_2,\cdots,\mathbf{x}_n) &= \frac{f(\mathbf{x}_1,\mathbf{x}_2,\cdots,\mathbf{x}_n|\theta)f(\theta)}{f(\mathbf{x}_1,\mathbf{x}_2,\cdots,\mathbf{x}_n)} \\ \Rightarrow \hat{\theta} &= \operatorname{argmax}\ f(\theta|\mathbf{x}_1,\mathbf{x}_2,\cdots,\mathbf{x}_n) \\ &= \operatorname{argmax}\ f(\mathbf{x}_1,\mathbf{x}_2,\cdots,\mathbf{x}_n|\theta)f(\theta). \end{aligned} \tag{1}$$

上式表明，获得 θ 的最大后验概率估计估计，只需要求解最大化联合分布问题。

1.2 贝叶斯分类器

将最大后验概率估计应用到分类问题上，可以将 θ 换为样本的标签 y。即，在样本 $\mathbf{x}_i$ 的条件下，使条件概率 $P(y_i=y|\mathbf{x}_i)$ 最大的 y 值成为对 y_i 的估计 $\hat{y}_i$。数学定义如下：

$$\hat{y}_i = \operatorname{argmax}_y\ P(y_i=y|\mathbf{x}_i) \tag{2}$$

类似最大后验概率估计的推导，我们可以得到：

$$\hat{y}_i = \operatorname{argmax}_y\ f(\mathbf{x}_i|y)P(y_i=y) \tag{3}$$

此处 y_i 的先验分布 $P(y_i=y)$ 可以由计算训练集的频率来近似，而条件概率 $f(\mathbf{x}_i|y)$ 通常认为服从某正态分布。

2 逻辑回归

在上一节的贝叶斯分类器中，我们试图训练联合分布从而获得条件概率 $P(y_i=y|\mathbf{x}_i)$，之后选择概率最大的类别 y。理论上，这种方法是可行的，并且得到联合分布后可以进一步分析不同类的特征，这种模型被称为生成模型 (Generative Model)。名字由来是因为这种模型"生成"了样本的总体分布，并根据这个总体分布来考虑某个特定的样本分类。

但是实际应用中，可能没有足够的数据量训练联合分布，或者训练得到的假设分布与实际分布相差甚远，效果很差。我们反过来思考，如果直接考虑对条件概率 $P(y_i=y|\mathbf{x}_i)$ 建模，实现起来更加简单和直接。对于这种输入为某样本，输出为一系列条件概率 $P(y_i=y|\mathbf{x}_i)$ 的模型，我们称之判别模型 (Discriminative Model)。这个名字是因为这种模型只考虑样本"是否"属于某一类，最可能"是"的那一类即成为模型的预测分类。

2.1 几率与逻辑函数

概率值均属于 [0,1] 的实数区间内，虽然对人来说表示起来非常简单易懂，但在计算机程序中可能会导致指数衰退等问题，我们希望能用一个实数来与概率值一一对应，并且最好能保持概率值越大越容易发生的性质。数学上，[0,1] 与 $\mathbb{R}$ 等势，这种双射的构造当然是可能的。下面介绍一种最常用的构造方法：

定义：几率

几率指示了一个事件 A 发生概率 p 与不发生概率 $1-p$ 之间的比，其数学形式是：

$$o_A = \frac{p}{1-p} \tag{4}$$

注意到几率的值域为非负实数，并且几率越大，其发生概率越大，这一性质得到保留。进一步，我们可以用对数函数将非负实数单调映射到整个实数集中。

定义：逻辑函数

逻辑函数又称对数几率，由几率取对数得到，其意义是将一个概率值 p 单调映射到整个实数集上，便于进一步操作。

$$\operatorname{logit}(p) = \log\frac{p}{1-p} \tag{5}$$

2.2 逻辑回归

这一节我们将用回归的思路来求解分类问题。前面提到，使概率最大的 y 可以作为模型的预测结果，而又可以通过逻辑函数将概率映射到实数集上，自然我们可以将分类问题转化为一个回归问题。最简单的情况，我们考虑条件概率 $P(y_i=y|\mathbf{x}_i)$ 的对数几率满足

$$\log\frac{P(y_i=y|\mathbf{x}_i)}{1-P(y_i=y|\mathbf{x}_i)} = \theta^\top\mathbf{x}_i \tag{6}$$

解出 $P(y_i=y|\mathbf{x}_i)$：

$$P(y_i=y|\mathbf{x}_i) = \frac{1}{1+e^{-\theta^\top\mathbf{x}_i}} \tag{7}$$

对于二分类问题，假设两类的标签 y 分别为 0,1，则可以写为

$$\begin{aligned} P(y_i=1|\mathbf{x}_i) &= \frac{1}{1+e^{-\theta^\top\mathbf{x}_i}} \\ P(y_i=0|\mathbf{x}_i) &= 1-P(y_i=1|\mathbf{x}_i) \end{aligned} \tag{8}$$

3 二分类逻辑回归

我们现在来详细解释一下二分类问题中的逻辑回归。我们之前介绍过，对于二分类问题，一个分类器的目标就是找到一个超平面来将两类数据点分开。图1展示了二维空间中的一个二分类问题。

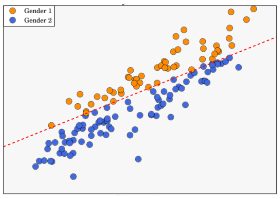

Figure 1: 二维空间中的二分类问题。

假设我们有样本 $\mathbf{x} \in \mathbb{R}^p$，样本对应的类别标签为 $y \in \{0,1\}$。我们可以把决策边界表示为 $\theta^T\mathbf{x} = 0$。如果一个样本的 $h(\mathbf{x}) = \theta^T\mathbf{x} > 0$，那么这个样本就会被分类为 1，否则会被分类为 0。这种简单的分类过程就是我们之前介绍的**感知机**。我们可以看到感知机输出的结果就是 0 或者 1，但在一些情况下我们需要知道一个样本有多大的概率会被分为类别 1 或者类别 0。这在一些用概率来指导决策的问题中是非常重要的需求。为了满足这样的要求，我们不再关注 y 和 $\mathbf{x}$ 的关系，而是考虑 $P(y=1 \mid \mathbf{x})$ 与 $\mathbf{x}$ 的关系（因为是二分类问题，$P(y=1 \mid \mathbf{x}) + P(y=0 \mid \mathbf{x}) = 1$，所以我们只需要关注其中一类就可以）。

因为我们仍然需要找到线性的决策边界，我们可以沿用之前的决策边界定义来计算 $P(y=1 \mid \mathbf{x})$

$$P(y=1 \mid \mathbf{x}) = g(\theta^T\mathbf{x}) = \theta^T\mathbf{x} \tag{9}$$

这里我们定义 $g(z) = z$ 并且 $z = \theta^T\mathbf{x}$。但是线性函数的值域是整个实数域，这不满足概率值属于 $[0,1]$ 的要求。所以我们需要修改一下 $g(z)$ 的定义来将 $\theta^T\mathbf{x}$ 映射到需要的范围内。最简单的一种 $g(z)$ 是阶跃函数

$$g(z) = \begin{cases} 0, & z < 0 \\ 0.5, & z = 0 \\ 1, & z > 0 \end{cases} \tag{10}$$

但是阶跃函数并不可导，这就为我们后续的计算与优化带来了麻烦。所以在逻辑回归中，我们选择的是 **Sigmoid 函数**

$$g(z) = \frac{1}{1+e^{-z}} \tag{11}$$

这样的定义除了使得优化问题可导，还有一个好处就是“分类为类别 1”的对数几率恰好就是样本点的线性变换

$$\ln \frac{P(y=1 \mid \mathbf{x})}{1 - P(y=1 \mid \mathbf{x})} = \theta^T\mathbf{x} \tag{12}$$

所以我们可以看到逻辑回归是在用线性回归的预测值（$\theta^T\mathbf{x}$）来拟合分类的对数几率

这里我们只介绍了二分类问题，但是逻辑回归还可以应用与多分类问题。假设我们有三个类别（A,B,C），逻辑回归采用一种**一对多**的分类方法，即每次把其中一个类别作为正类，而剩下的所有类别作为负类，然后对所有情况都划分出一个分类平面。比如第一次将 A 作为正类，把 B 和 C 作为负类，那么这时找到的分类平面可以判断一个样本是否属于类别 A；第二次将 B 作为正类，A 和 C 作为负类，以此类推。这种针对多分类问题的分类模型被称作 **OVR 逻辑回归（one-vs-rest logistic regression）**，而之前介绍的二分类模型被称为 **OVO 逻辑回归（one-vs-one logistic regression）**。这种把多分类转化为二分类的方法不仅适用于逻辑回归模型，还适用于所有本来只能解决二分类问题的其它分类器。

4 逻辑回归的求解

逻辑回归模型的参数 θ 是通过最大似然函数（MLE）来求解的。为了简化符号，我们令 $p(\mathbf{x}) = P(y=1 \mid \mathbf{x})$，则 $1-p(\mathbf{x}) = P(y=0 \mid \mathbf{x})$。假设我们有数据集 $\mathcal{D} = \{(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_i, y_i), (\mathbf{x}_n, y_n)\}$，其中 $y_i \in \{0,1\}$ 且 $i = 1, 2, \cdots, n$，则似然函数为

$$L(\theta) = \prod_i^n [p(\mathbf{x}_i)]^{y_i}[1 - p(\mathbf{x}_i)]^{1-y_i} \tag{13}$$

我们选择把 y_i 作为指数是因为正好 $y_i \in \{0,1\}$。我们取对数得到对数似然函数为

$$\begin{aligned} \ln L(\theta) &= \sum_i^n [y_i \ln p(\mathbf{x}_i) + (1-y_i)\ln(1-p(\mathbf{x}_i))] \\ &= \sum_i^n [y_i \ln \frac{p(\mathbf{x}_i)}{1-p(\mathbf{x}_i)} + \ln(1-p(\mathbf{x}_i))] \\ &= \sum_i^n [y_i(\theta^T\mathbf{x}_i) - \ln(1+e^{\theta^T\mathbf{x}_i})] \end{aligned} \tag{14}$$

我们的目标是最大化对数似然函数 (14)。这个问题有很多种方法求解，在这里我们只介绍梯度上升法。与在线性回归中介绍的梯度下降法类似，这里因为我们想要求最大值，因此采用梯度上升法。对于参数 θ，我们可以求出 $\ln L(\theta)$ 关于它的梯度

$$\frac{\partial \ln L(\theta)}{\theta} = \sum_i^n = [y_i - \frac{1}{1+e^{-\theta^T\mathbf{x}_i}}]\mathbf{x}_i \tag{15}$$

所以我们可以迭代更新 θ

$$\theta^{t+1} = \theta^t + \eta \cdot \sum_i^n [y_i - \frac{1}{1+e^{-\theta^T\mathbf{x}_i}}]\mathbf{x}_i \tag{16}$$

其中 η 表示学习速率。下面我们给出用梯度上升法求解逻辑回归的伪代码

Algorithm 1: 梯度上升法求解逻辑回归

输入：样本矩阵 $\mathbf{X} \in \mathbb{R}^{n\times p}$，样本对应的标签 $y \in \mathbb{R}^n$，最大迭代次数 T 和学习率 $\eta > 0$。随机初始化系数向量 θ^0；
for $t \leftarrow 1$ *to* T **do**
  $\theta^{t+1} = \theta^t + \eta \cdot \sum_i^n [y_i - \frac{1}{1+e^{-\theta^T x_i}}]x_i$
end
输出：预测系数 θ。

除了梯度上升法，我们之前介绍的牛顿法也可以用来最大化逻辑回归的对数似然函数。

4.1 编程实现

我们这里用 Python 实现梯度上升法求解逻辑回归的算法。

```python
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(12)
num_observations = 5000

# 依据均值和协方差生成数据
# np.random.multivariate_normal方法用于根据实际情况生成一个多
    元正太分布矩阵
x1 = np.random.multivariate_normal([0, 0], [[1, .75], [.75,
    1]], num_observations)
```

```python
x2 = np.random.multivariate_normal([1, 4], [[1, .75], [.75, 1]], num_observations)
# [0, 0]和[1, 4]为其各自均值坐标, [[1, .75], [.75, 1]]为协方差矩阵, num_observations为生成数量

# 方便做散点图做的变换
simulated_separableish_features = np.vstack((x1, x2)).astype(np.float32)
# 生成10000x2的矩阵
simulated_labels = np.hstack((np.zeros(num_observations), np.ones(num_observations)))
# 生成1x10000的向量
plt.figure(figsize=(12, 8))
plt.scatter(simulated_separableish_features[:, 0], simulated_separableish_features[:, 1],
            c=simulated_labels, alpha=.4)

precision = 1e-10  # 精准度


# 逻辑回归
def sigmoid_eg(x1, x2, theta_1, theta_2, theta_3):
    z = (theta_1 * x1 + theta_2 * x2 + theta_3).astype("float_")
    return 1.0 / (1.0 + np.exp(-z))


def gradient_eg(x1, x2, y, theta_1, theta_2, theta_3):
    sigmoid_probs = sigmoid_eg(x1, x2, theta_1, theta_2, theta_3)
    return 1 / len(y) * np.sum((y - sigmoid_probs) * x1), 1 / len(y) * np.sum((y - sigmoid_probs) * x2), 1 / len(
        y) * np.sum((y - sigmoid_probs))


def GradDe_eg(x1, x2, y, Max_Loop=20, alpha=0.1):
    # alpha = 0.00000001
    # Max_Loop = 200
    # 初始值
    theta_1 = 0.1
    theta_2 = -0.4
    theta_3 = 0.56
    for l in range(Max_Loop):
        theta_1_last = theta_1
        theta_2_last = theta_2
        theta_3_last = theta_3
        delta1, delta2, delta3 = gradient_eg(x1, x2, y, theta_1, theta_2, theta_3)  # 梯度
        theta_1 = theta_1 + alpha * delta1
        theta_2 = theta_2 + alpha * delta2
        theta_3 = theta_3 + alpha * delta3
        # 计算theta变化情况, 判断是否应该停止
        mse = (theta_1_last - theta_1)**2 + (theta_2_last - theta_2)**2 + (theta_3_last - theta_3)**2
        if mse <= precision:
            break
        if l % 1000 == 0:
            print('delta%d =' % (l), [delta1, delta2, delta3])
            print('theta%d =' % (l), [theta_1, theta_2, theta_3], '\n')
    return [theta_1, theta_2, theta_3]


weights = GradDe_eg(simulated_separableish_features[:, 0], simulated_separableish_features[:, 1], simulated_labels,
                    200000, 0.9)

# 做出决策边界
x_values = [-4, 4]
y_values = - (weights[2] + np.dot(weights[0], x_values)) / weights[1]

plt.plot(x_values, y_values, label='Decision Boundary')
plt.show()
print(weights)print(weights)
```

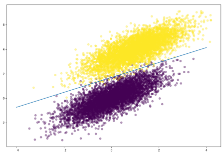

Figure 2: 决策边界

引用

[1] Logistic Regression Step-By-Step Guide in Python：https://learn-ml.com/index.php/2019/05/29/logistic-regression-st

[2] Multi-Class Classification Using Logistic Regression: https://utkuufuk.com/2018/06/03/one-vs-all-classification/

[3] https://blog.csdn.net/daycym/article/details/80472015