02+LogisticRegression

2019 年 3 月 12 日

1 logistic regression

1.1 一、概念

逻辑回归是应用于**二分类**的方法。相对的应用于多分类的方法叫 softmax。

逻辑回归的实现依赖于 sigmoid 函数：

$$y = \frac{1}{1+e^{-z}}$$

可以看出这是一个单调递增，当 x 趋向于负无穷时函数值趋向于 0，x 趋向于正无穷时函数值趋向于 1，对应于某一样本属于某一类的概率。

对于一个有 p 个变量的样本而言，模型的形式为：

$$\hat{y}_i = P(y_i = 1) = \frac{1}{1+e^{-(\theta_0+\theta_1 x_1+...+\theta_p x_p)}}$$

1.2 二、损失函数

逻辑回归的损失函数是交叉熵，从数理统计的角度出发，则是极大似然函数。

假设样本来自于伯努利分布，$y_i \in \{0,1\}$，对概率的估计值 $\hat{y}_i \in (0,1)$，则估计的似然函数是：

$$L = \prod_{i=1}^{n} \hat{y}_i^{y_i}(1-\hat{y}_i)^{1-y_i}$$

对其求以自然底数为底数的对数：

$$ln(L) = \sum_{i=1}^{n}(y_i ln(\hat{y}_i) + (1-y_i)ln(1-\hat{y}_i))$$

根据极大似然估计的定义，似然函数的函数值越大，表示随机事件更可能服从当前的概率分布。由于伯努利分布的似然函数值在 0，1 之间，取对数后不改变增减性，极大化似然函数则等价于极小化损失函数。

因此损失函数就定义为：

$$loss = -\frac{1}{m}\sum_{i=1}^{n} y_i ln(\hat{y}_i) + (1-y_i)ln(1-\hat{y}_i))$$

1.3 三、优化

和多元线性回归类似，逻辑回归也存在相应假定

尽管逻辑回归的形式像是把多元线性回归套进 sigmoid 函数，但逻辑回归没有像多元线性回归那样的正规方程（据我所知没有），因此逻辑回归的参数求解和优化主要靠梯度下降。

梯度下降的核心在于求导，样本为常数，参数为变量，把损失函数视为参数的函数对其求导。

首先，由于真实值已知，所以损失函数是关于估计值的函数：

$$\frac{\partial loss}{\partial \hat{y}_i} = -\frac{1}{n}(\frac{y_i}{\hat{y}_i} - \frac{1-y_i}{1-\hat{y}_i})$$

而 $\hat{y}_i$ 是数据各变量线性组合 z_i 的 sigmoid 函数：

$$\begin{aligned}
\hat{y}_i &= \frac{1}{1+e^{-z_i}} \\
\frac{\partial \hat{y}_i}{\partial z_i} &= \frac{e^{-z_i}}{(1+e^{-z_i})^2} = \frac{1}{1+e^{-z_i}}(1 - \frac{1}{1+e^{-z_i}}) = \hat{y}_i(1-\hat{y}_i) \\
z_i &= \theta_0 + \theta_1 x_1 + ... + \theta_p x_p \\
\frac{\partial z_i}{\partial \theta_j} &= x_{ij}
\end{aligned}$$

接下来求梯度有两种方法：1. 求分量的导数：

$$\frac{\partial loss}{\partial \theta_j} = \sum_{i=1}^{n} \frac{\partial loss}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial z_i}\frac{\partial z_i}{\partial \theta_j} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)x_{ij}$$

然后迭代：

$$\theta_j := \theta_j - step \times \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)x_{ij}$$

2. 直接求梯度矩阵

损失是一个标量，$\hat{y}_i$，z_i 和 Θ 分别是 $n \times 1$，$n \times 1$ 和 $p \times 1$ 维向量，根据雅可比（Jacob）矩阵的定义，最终的梯度 $\frac{\nabla loss}{\nabla \Theta}$ 应该是一个 $p \times 1$ 维的向量。

首先，损失对估计值向量的导数（梯度）是一个雅可比矩阵：

$$\frac{\nabla loss}{\nabla \hat{Y}} = \begin{bmatrix} \frac{\partial loss}{\partial \hat{y}_1} \\ \frac{\partial loss}{\partial \hat{y}_2} \\ ... \\ \frac{\partial loss}{\partial \hat{y}_n} \end{bmatrix} = \begin{bmatrix} -\frac{1}{n}(\frac{y_1}{\hat{y}_1} - \frac{1-y_1}{1-\hat{y}_1}) \\ -\frac{1}{n}(\frac{y_2}{\hat{y}_2} - \frac{1-y_2}{1-\hat{y}_2}) \\ ... \\ -\frac{1}{n}(\frac{y_n}{\hat{y}_n} - \frac{1-y_n}{1-\hat{y}_n}) \end{bmatrix}$$

其次，$\hat{Y}$ 和 Z 的函数关系是分量之间的函数关系，于是：

$$\frac{\nabla \hat{Y}}{\nabla Z} = \begin{bmatrix} \hat{y}_1(1-\hat{y}_1) \\ \hat{y}_2(1-\hat{y}_2) \\ ... \\ \hat{y}_n(1-\hat{y}_n) \end{bmatrix}$$

然后，$Z = X\Theta$ 需要求矩阵对矩阵的导数，其中的数学很复杂，我比较赞成的是这个方法：https://www.zhihu.com/question/39523290/answer/100057066 ，即导数的矩阵形式按需自取，也可以说是“凑”，先前求得的导数都是 $n \times 1$ 维的，因此只需一个 $p \times n$ 维的矩阵参与，即可求得合适的梯度，即：

$$\frac{\nabla Z}{\nabla \Theta} = X^T$$

总结起来：

$$\frac{\nabla loss}{\nabla \Theta} = \frac{\nabla Z}{\nabla \Theta}(\frac{\nabla loss}{\nabla \hat{Y}} * \frac{\nabla \hat{Y}}{\nabla Z}) = X^T \begin{bmatrix} -\frac{1}{n}(y_1 - \hat{y}_1) \\ -\frac{1}{n}(y_2 - \hat{y}_2) \\ \dots \\ -\frac{1}{n}(y_n - \hat{y}_n) \end{bmatrix}$$

其中 * 是按元素相乘。

最后按照梯度下降的定义进行迭代即可。

1.4 四、应用

选用的是数据集是经典的鸢尾花数据集：http://archive.ics.uci.edu/ml/datasets/Iris ，一共五列变量，分别是花萼长度、花萼宽度、花瓣长度、花瓣宽度和鸢尾花的品种，数据简洁易于快速应用算法。

```
In [1]: import numpy as np
        import pandas as pd

        cols = ['sepal.l','sepal.w','petal.l','petal.w','class']
        iris = pd.read_csv('data_set/Iris.data', header=None, names=cols)
        iris.insert(0, 'bias', 1)
        iris.head(5)
```

Out[1]:

	bias	sepal.l	sepal.w	petal.l	petal.w	class
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	1	4.9	3.0	1.4	0.2	Iris-setosa
2	1	4.7	3.2	1.3	0.2	Iris-setosa
3	1	4.6	3.1	1.5	0.2	Iris-setosa
4	1	5.0	3.6	1.4	0.2	Iris-setosa

Iris 数据集中有三类花的数据，logistic regression 适用于二分类，因此我们从 Iris 中选取两种花的数据作为训练数据。

```
In [29]: iris.index = iris['class']
         iris['class'].unique()
```

```
Out[29]: array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)
```

```
In [30]: iris = iris.loc[['Iris-setosa', 'Iris-versicolor'],:]
         iris['class'].unique()
```

```
Out[30]: array(['Iris-setosa', 'Iris-versicolor'], dtype=object)
```

```
In [34]: n = iris.shape[0] # 样本数
         p = iris.shape[1]-1 # 变量数

         np.random.seed(2099)
         index = np.random.permutation(n) # 打乱样本索引

         train_index = index[0: int(0.7*n)]
         test_index = index[int(0.7*n): n]

         y = iris['class']
         y = y=='Iris-setosa'
         x = iris.drop(['class'], axis=1)

         train_x = x.iloc[train_index]
         train_y = y.iloc[train_index]
         test_x = x.iloc[test_index]
         test_y = y.iloc[test_index]

         train_y = np.array(train_y).reshape([int(0.7*n), 1])
         test_y = np.array(test_y).reshape([n-int(0.7*n), 1])
```

```
In [36]: def initialize(p):
             return np.random.randn(p).reshape([p, 1])

         theta = initialize(p)
         print(theta)
```

```
[[-0.24336871]
 [-1.07204116]
 [-0.06854123]
 [-0.44725433]
 [ 0.4089976 ]]
```

```
In [37]: def sigmoid(x, theta):
             return 1/(1 + np.exp(-np.dot(x, theta)))
```

```
        train_y_hat = sigmoid(train_x, theta)
        print(train_y_hat)
```

```
[[1.18168242e-04]
 [1.19431968e-03]
 [1.82891796e-03]
 [1.29628065e-03]
 [5.16319355e-04]
 [2.08947873e-04]
 [7.74229880e-04]
 [1.47235670e-03]
 [3.88988777e-04]
 [4.18409328e-04]
 [2.72819821e-04]
 [1.80808517e-03]
 [1.28280893e-03]
 [1.33698913e-04]
 [2.06605462e-03]
 [7.53311575e-04]
 [2.06030984e-04]
 [1.50865120e-03]
 [1.08666180e-04]
 [4.28638128e-04]
 [2.07475411e-04]
 [1.61678645e-03]
 [1.71438708e-04]
 [4.46981250e-04]
 [1.67512631e-03]
 [2.65918245e-04]
 [1.58908648e-03]
 [5.39683701e-04]
 [3.45110196e-03]
 [5.23442192e-04]
 [1.83672973e-03]
 [2.16082610e-03]
 [2.62919074e-03]
 [9.37060879e-04]
 [1.01036263e-03]
 [7.65099459e-04]
 [1.19288691e-04]
```

```
 [2.53172907e-03]
 [4.57807848e-04]
 [4.03040687e-03]
 [1.43730724e-04]
 [1.00974263e-03]
 [1.57153490e-03]
 [1.85718071e-04]
 [1.03215318e-03]
 [5.25866338e-04]
 [1.95491545e-03]
 [7.08187601e-04]
 [2.16011534e-04]
 [2.55116949e-04]
 [1.14881894e-03]
 [1.19571914e-03]
 [1.45266640e-03]
 [3.53491311e-04]
 [1.54426209e-03]
 [1.93376524e-03]
 [4.81962211e-04]
 [8.01941035e-05]
 [3.70775033e-04]
 [3.90393587e-04]
 [2.47034216e-03]
 [2.24176665e-03]
 [1.14327589e-03]
 [2.70146749e-03]
 [1.91520442e-03]
 [3.12368695e-04]
 [1.35169580e-03]
 [2.75026341e-04]
 [1.67443275e-04]
 [1.67774987e-03]]
```

```
In [43]: def loss(y, y_hat):
             n = y.shape[0]
             return -1/n*(np.dot(y.T, np.log(y_hat)) + np.dot(1-y.T, np.log(1-y_hat)))

         train_loss = loss(train_y, train_y_hat)
```

```
        print(train_loss)
```

```
[[4.77718095]]
```

```
In [48]: def train(y, x, theta, step=0.1, delta = 0.0001):
             decay = 1
             loss_pre = None
             loss_aft = float('inf')
             y_hat = sigmoid(x, theta)

             while decay > delta:
                 gradient = np.dot(x.T, 1/n*(y_hat - y))

                 theta -= step*gradient
                 y_hat = sigmoid(x, theta)

                 loss_pre = loss_aft
                 loss_aft = loss(y, y_hat)
                 decay = loss_pre-loss_aft

                 #print(decay)

             return theta, loss_aft

         theta, train_loss = train(train_y, train_x, theta)
         print(train_loss)
```

```
[[0.03819359]]
```

```
In [53]: test_y_hat = sigmoid(test_x, theta)
         test_loss = loss(test_y, test_y_hat)
         print(test_loss)
```

```
[[0.04520658]]
```

1.5 五、阈值和评价指标

在分类问题中，我们求出的估计值通常是一个概率值，是属于 0 到 1 的连续实数 $(0,1)$，而真实值的取值集合 $\{0,1\}$。因此我们通常需要选取阈值，即确定概率值大于多少时认为取值是 1，反之取值为 0。同时，直接的损失函数并不直观，通常我们还用准确率和一些其他评价指标。

1.5.1 1. 阈值的选取

1.5.2 2. 评价指标

最常见的评价指标是准确率，$accuracy = \frac{1}{n}\sum_{i=1}^{n} I(\hat{y}_i = y_i)$，准确率非常直观，即预测结果正确的占比。

尽管准确率较直观，但仍然有些粗糙，除了准确率之外我们还有查准率、召回率和 F1 分数作为评价指标。

混淆矩阵	预测正例-1	预测反例-0
真实正例	TP（真正例）	FN（假反例）
真实反例	FP（假正例）	TN（真反例）

- TP：预测为真，实际为真
- FP：预测为真，实际为假
- FN：预测为假，实际为真
- TN：预测为假，实际为假

查准率：$P = precision = \frac{TP}{TP+FP}$
召回率：$R = recall = \frac{TP}{TP+FN}$
F1 分数则是二者的调和平均数：$F1 = \frac{2}{\frac{1}{P}+\frac{1}{R}}$

根据不同的问题选用不同的指标是较为明智的选择，譬如犯罪嫌疑人识别，我们希望尽可能多得识别，即模型判断是犯罪嫌疑人的人数占所有犯罪嫌疑人人数的比例尽可能高，此时选用查准率；而疾病检测则不同，误诊会给病人带来极大的困扰，因此我们“希望”诊断患有某种疾病的人“确实”患有该疾病，此时选用召回率更合适；但也有很多情况中，并没有明显的偏好，因此选用 F1 分数将会是能够兼顾二者的一个较好选择

```
In [57]: def accuracy(y, y_hat, threshold=0.5):
             y_hat = y_hat > threshold
             return np.sum(y == y_hat)/y.shape[0]

         train_y_hat = sigmoid(train_x, theta)
         train_accuracy = accuracy(train_y, train_y_hat)
         test_accuracy = accuracy(test_y, test_y_hat)

         print('accuracy on train set:'+str(train_accuracy))
         print('accuracy on test set:'+str(test_accuracy))
```

```
accuracy on train set:1.0
accuracy on test set:1.0
```

由于数据量较少，纬度较低，因此准确率较高，下面计算查准率、召回率和 F1 分数（尽管根据准确率来看，三者肯定都是 1）。

```
In [72]: def precision(y, y_hat):
             TP_FP_index = np.where(y_hat == 1)[0]
             TP_FP = len(TP_FP_index)

             TP = np.sum(y_hat[TP_FP_index] == y[TP_FP_index])

             return TP/TP_FP

         test_y_hat = test_y_hat > 0.5
         test_precision = precision(test_y, test_y_hat)
         print(test_precision)
```

```
1.0
```

```
In [73]: def recall(y, y_hat):
             TP_FN_index = np.where(y == 1)[0]
             TP_FN = len(TP_FN_index)

             TP = np.sum(y[TP_FN_index] == y_hat[TP_FN_index])

             return TP/TP_FN

         test_recall = recall(test_y, test_y_hat)
         print(test_recall)
```

```
1.0
```

```
In [74]: def F1(y, y_hat):
             return 2/(1/precision(y, y_hat) + 1/recall(y, y_hat))

         test_F1 = F1(test_y, test_y_hat)
         print(test_F1)
```

```
1.0
```

1.6 六、实际应用

在工业界，很少直接将连续值作为逻辑回归模型的特征输入，而是将连续特征离散化为一系列 0、1 特征交给逻辑回归模型，这样做的优势有以下几点：

0. 离散特征的增加和减少都很容易，易于模型的快速迭代；
1. 稀疏向量内积乘法运算速度快，计算结果方便存储，容易扩展；
2. 离散化后的特征对异常数据有很强的鲁棒性：比如一个特征是年龄 >30 是 1，否则 0。如果特征没有离散化，一个异常数据 “年龄 300 岁” 会给模型造成很大的干扰；
3. 逻辑回归属于广义线性模型，表达能力受限；单变量离散化为 N 个后，每个变量有单独的权重，相当于为模型引入了非线性，能够提升模型表达能力，加大拟合；
4. 离散化后可以进行特征交叉，由 M+N 个变量变为 M*N 个变量，进一步引入非线性，提升表达能力；
5. 特征离散化后，模型会更稳定，比如如果对用户年龄离散化，20-30 作为一个区间，不会因为一个用户年龄长了一岁就变成一个完全不同的人。当然处于区间相邻处的样本会刚好相反，所以怎么划分区间是门学问；
6. 特征离散化以后，起到了简化了逻辑回归模型的作用，降低了模型过拟合的风险。

李沐曾经说过：模型是使用离散特征还是连续特征，其实是一个 “海量离散特征 + 简单模型” 同 “少量连续特征 + 复杂模型” 的权衡。既可以离散化用线性模型，也可以用连续特征加深度学习。就看是喜欢折腾特征还是折腾模型了。通常来说，前者容易，而且可以 n 个人一起并行做，有成功经验；后者目前看很赞，能走多远还须拭目以待。

来源：知乎：https://www.zhihu.com/question/31989952/answer/54184582

In [ ]: