0.1 逻辑回归

0.1.1 摘要

本期我们将学习二分类-软输出的一种算法：逻辑回归。该算法主要是依托于一个激活函数：sigmoid，因为这个函数的值域为 $(0,1)$，因此可以近似表示概率值。

0.1.2 本质

以下为 shuhuai 老师的讲义上的解释：

> 有时候我们只要得到一个类别的概率，那么我们需要一种能输出 $(0,1)$ 区间的值的函数。考虑两分类模型，我们利用判别模型，希望对 $p(C|x)$ 建模，利用贝叶斯定理：
>
> $$p\left(C_1 \mid x\right)=\frac{p\left(x \mid C_1\right) p\left(C_1\right)}{p\left(x \mid C_1\right) p\left(C_1\right)+p\left(x \mid C_2\right) p\left(C_2\right)}$$
>
> 取 $a=\ln \frac{p(x|C_1)p(C_1)}{p(x|C_2)p(C_2)}$，于是：
>
> $$p\left(C_1 \mid x\right)=\frac{1}{1+\exp(-a)}$$
>
> 上面的式子叫 Logistic Sigmoid 函数，其参数表示了两类联合概率比值的对数。在判别式中，不关心这个参数的具体值，模型假设直接对 a 进行。

当然了，老师高端的解释看不懂也没关系，我们只需要知道，现在我们有了一个激活函数 sigmoid，它可以用来得到一个类别的概率。

0.1.3 算法

首先我们给出逻辑回归的模型假设：

$$f(x)=\sigma(w^T x)$$

其中，$\sigma(a)=sigmoid(a)$，我们一般用 σ 来表示激活函数
于是，通过寻找 w 的最佳值，则可以确定在该模型假设下的最佳模型。
概率判别模型常用极大似然估计来确定参数。
为了确定似然函数，我们先做一些标记：

$$p_1=\sigma(w^T x) \quad p_0=1-p_1$$

其中 p_1 为 x 属于 1 类的概率，p_0 为 x 属于 0 类的概率
下面我们就可以给出该模型的似然函数了：

$$p(y|w;x)=p_1^y p_0^{1-y}$$

这个似然函数看上去操作有点骚，看不懂，其实也蛮合理的：

- 当 y 为 1 时：$p(y|w;x) = p_1^1 p_0^0 = p_1$
- 当 y 为 0 时：$p(y|w;x) = p_1^0 p_0^1 = p_0$

好，下面我们就可以使用极大似然估计来确定参数了

$$
\begin{aligned}
\hat{w} = argmax(J(w)) &= argmax(p(Y|w;X)) \\
&= argmax(log(p(Y|w;X))) \\
&= argmax(log(\prod_{i=1}^{n} p(y_i|w;x_i))) \\
&= argmax(\sum_{i=1}^{n} log(p(y_i)|w;x_i)) \\
&= argmax(\sum_{i=1}^{n} y\ log\, p_1 + (1-y) log\, p_0)
\end{aligned} \tag{1}
$$

注意到，这个表达式是交叉熵表达式的相反数乘 N,MLE 中的对数也保证了可以和指数函数相匹配, 从而在大的区间汇总获取稳定的梯度。
对上式求导，我们注意到：

$$p_1' = p_1(1-p_1)$$

当然这个也很容易得到，就是链式法则嘛，稍微细心一点就可以求出来了。
最后我们求出结果：

$$\frac{\partial}{\partial w} J(w) = \sum_{i=1}^{N} (y_i - p_1)\, x_i$$

最后还有一点要注意，我们是要求得 $p(p|w;x)$ 的最大值，因此我们需要使用梯度上升，而不是梯度下降，当然两者也差不多，加个负号而已。

0.1.4 实作

```
import os
os.chdir("../")
from models.linear_models import Logistic_regression
import numpy as np
import warnings
warnings.filterwarnings("ignore")


epsilon = 1
num_test = 100
num_base = 1000
ratio = 0.6
k1, k2 = 3, 5
b1, b2 = 1, 2
X = np.linspace(0, 100, num_base)
```

```
X_train = X[:-num_test]
X_test = X[-num_test:]
v1 = X_train[:round(len(X_train) * ratio)] * k1 + b1
v2 = X_train[round(len(X_train) * ratio):] * k2 + b2
v1 += np.random.normal(scale=epsilon, size=v1.shape)
v2 += np.random.normal(scale=epsilon, size=v2.shape)
value = np.r_[v1, v2]
data = np.c_[X_train, value]
l1 = np.ones_like(v1)
l2 = np.zeros_like(v2)
label = np.r_[l1, l2]
v_test_c1 = X_test * k1 + b1
l_test_c1 = np.ones_like(v_test_c1)
data_test = np.c_[X_test, v_test_c1]

model = Logistic_regression(10, 1000, lr=1e-3)
model.fit(data, label)
print(model.get_params())
print(model.predict(data_test, l_test_c1))
```