机器学习笔记

一、离散分类问题

1. 问题的提出

假设某个分类问题有n个特征feature $x_1, x_2, \cdots, x_n$ ，他们可以有不同的取值。

当$w_1x_1 + \cdots + w_nx_n + b > 0$时该问题属于类别标签$y = 1$;

当$w_1x_1 + \cdots + w_nx_n + b < 0$时该问题属于类别标签$y =$0;

其中$w_1, \cdots, w_n, b$是待定权重参量，是需要通过程序和多个样本训练而获得。

令$w_0 = b, x_0 = 1$，则上述分类问题的临界值可以表示为$\vec{W} \cdot \vec{X} = w_0x_0 + w_1x_1 + \cdots + w_nx_n = 0$,

只有两个分类的问题可以用阈值函数Sigmoid函数来表示$f(z) = \dfrac{1}{1+e^{-z}}, \quad f'(z) = f(z)[1 - f(z)]$

$$\sigma(W \cdot X) = \frac{1}{1+e^{-W \cdot X}}, \qquad \frac{\partial \sigma}{\partial w_j} = \sigma(1-\sigma)x_j, \quad j = 0, 1, \cdots, n$$

这个函数可以视为类别$y = 1$发生的概率。可以验证：

当$W \cdot X > 0$时，类别$y = 1$的概率为$P = \sigma(WX) \simeq 1$

当$W \cdot X < 0$时，类别$y = 1$的概率为$P = \sigma(WX) \simeq$0

当$W \cdot X = 0$时，类别$y = 1$的概率为$P = \sigma(WX) = \frac{1}{2}$

属于类别$y(= 0, 1)$的概率可以综合表示为

$$P(y|X, W, b) = \sigma(WX)^y \left\{1 - \sigma(WX)\right\}^{1-y}, \qquad y = 0, 1$$

2. 损失函数

现在，已知该问题有m个已知样本$\left\{(X^1, y^1), (X^2, y^2), \cdots, (X^m, y^m)\right\}$

似然估计函数$L_{W,b}$

$$= \sigma(WX^1)^{y^1} \left\{1 - \sigma(WX^1)\right\}^{1-y^1} \cdot \sigma(WX^2)^{y^2} \left\{1 - \sigma(WX^2)\right\}^{1-y^2} \cdots \sigma(WX^m)^{y^m} \left\{1 - \sigma(WX^m)\right\}^{1-y^m}$$

因为上述m个样本已经发生了，所以这m个概率的乘积一定是取得最大值。下述损失函数（Logistic Regression 算法）必然取得最小值

$$l_{W,b}=-\frac{1}{m}\log(L_{W,b})=$$
$$-\frac{y^1}{m}\log[\sigma(WX^1)]-\frac{1-y^1}{m}\log[1-\sigma(WX^1)]-\cdots$$

在上述函数中的已知量为$X^1,\cdots,X^m$，待定参量为$W=(w_0=b,w_1,w_2)$。

损失函数的负梯度
$$-\frac{\partial l}{\partial w_j}=\frac{1}{m}\left\{y^1-\sigma(WX^1)\right\}x_j^1+\cdots+\frac{1}{m}\left\{y^m-\sigma(WX^m)\right\}x_j^m,\quad j=0,1,\cdots,n$$

3. 最速下降法

给定一个初始参量$W_0=(w_0,w_1,w_2)$, 选择一个步长α，沿着梯度下降方向逐步更新
$$W=W-\alpha\cdot\frac{\partial l}{\partial W}$$

直到满足一定条件，停止更新，此时的W就是最终值。

用分量形式表示
$$w_j=w_j-\alpha\frac{\partial l}{\partial w_j},\qquad j=0,1,\cdots,n$$

4. Python代码

我自己根据直线$0-2x+y=0$将二维平面切分为两部分。在直线上方选取了三个点作为类别$y=1$, 直线下方选取了三个点作为类别$y=0$。 因此权重$W=(0,-2,1)$。

```
# -*- coding: utf-8 -*-

import numpy as np



def sig(x):
    '''Sigmoid函数
    x是m*1矩阵，存在着m个样本的 WX+b的数值
    '''
    return 1.0/(1+np.exp(-x))



def lr_train(feature, label, maxCycle, alpha):
    '''梯度下降法LR模型
    feature是m*n矩阵，m个样本，每个样本n个特征值
    lable是m*1标签矩阵，矩阵元素为1或0
    maxCycle是迭代次数
    alpha是浮点数，代表步长
    '''
```

```
    n = np.shape(feature)[1]
    m = np.shape(feature)[0]
    alpha = alpha / m
    w = np.mat(np.ones((n,1))) # n*1矩阵
    i = 0
    while i<=maxCycle :
        i += 1
        h = sig(feature*w) # m*1矩阵
        err = label - h
        w = w + alpha * feature.T * err

    return w

# 自选六个点，初始化数据
X = np.mat([[1,1,3],[1,2,5],[1,3,8],[1,1,1],[1,2,3],[1,3,4]])
Y = np.mat([[1],[1],[1],[0],[0],[0]])

w = lr_train(X, Y, 10000, 0.001)
print('W=', w)
```

运行结果如下

```
W= [[-0.17284256]
 [-1.48779184]
 [ 0.86448801]]
```

该权重系数可以归一化为$W=(0.20,-1.72,1.00)$，还是比较接近设定值$(0,-2,1)$。

之所以会出现较大误差，应该是训练的数据量（只有6个）不够造成的。