Central University of Finance and Economics

中央财经大学

数据挖掘课程

代码文档

吴宇翀

2017310836

WUYUCHONG.COM

指导老师：马景义

2020 年 5 月 22 日

目录

1 简介

此文档为两个算法的代码文档，包括了 **Logisitic 回归**和**神经网络**两个模型算法。

1. model 文件夹：存放两个算法模型的类
2. source 文件夹：存放依赖函数
3. main 文件：测试用例 demo

2 Logistic 回归

我们使用**梯度下降**的优化方法构建 logit 模型。

2.1 模型求解步骤

如果 p 是一个事件的概率，这个事件的发生比率就是 p/(1-p)。逻辑回归就是建立模型预测这一比率的对数:

$$\log\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 x_1 + \cdots + \beta_P x_P$$

即：

$$p = \frac{1}{1 + \exp\left[-\left(\beta_0 + \beta_1 x_1 + \cdots + \beta_P x_P\right)\right]}$$

假设我们有 n 个独立的训练样本 {(x1, y1) ,(x2, y2),..., (xn, yn)}，y={0, 1}。那每一个观察到的样本 (xi, yi) 出现的概率是：

$$P\left(\mathrm{y}_i, \mathrm{x}_i\right) = P\left(\mathrm{y}_i = 1|\mathrm{x}_i\right)^{y_i} \left(1 - P\left(\mathrm{y}_i = 1|\mathrm{x}_i\right)\right)^{1-y_i}$$

那我们的整个样本集，也就是 n 个独立的样本出现的似然函数为:

$$L(\theta) = \prod P\left(\mathrm{y}_i = 1|\mathrm{x}_i\right)^{y_i} \left(1 - P\left(\mathrm{y}_i = 1|\mathrm{x}_i\right)\right)^{1-y_i}$$

那么，损失函数（cost function）就是最大似然函数**取对数**。[1]

用 $L(\theta)$ 对 θ 求导，得到：

[1] 最大似然法就是求模型中使得似然函数最大的系数取值 *

$$\frac{\partial L(\theta)}{\partial \theta} = \sum_{i=1}^{n} y_i x_i - \sum_{i}^{n} \frac{e^{\theta^T x_i}}{1 + e^{\theta_{x_i}}} x_i = \sum_{i=1}^{n} \left(y_i - \sigma\left(\theta^T x_i\right)\right) x_i$$

令该导数为 0，无法解析求解。使用梯度下降法 @ 汪宝彬 2011 随机梯度下降法的一些性质，那么：

$$\theta^{t+1} = \theta^t - \alpha \frac{\partial L(\theta)}{\partial \theta} = \theta^t - \alpha \sum_{i=1}^{n} \left(y_i - \sigma\left(\theta^T x_i\right)\right) x_j$$

在进行训练之前，我们对各个变量的数据进行标准化处理。然后，我们让学习率逐步递减进行训练。@LiFeng

2.2 代码实现 - 逐步讲解（Step by Step）

2.2.1 包和数据导入

```
import numpy as np
import random
import math
from sklearn import datasets
```

我们读取经典的 iris 数据集。[2]

```
iris = datasets.load_iris()
X = iris['data']
y = iris['target']
X = X[y!=2]
y = y[y!=2]
X[1:5]
```

```
## array([[4.9, 3. , 1.4, 0.2],
##        [4.7, 3.2, 1.3, 0.2],
##        [4.6, 3.1, 1.5, 0.2],
##        [5. , 3.6, 1.4, 0.2]])
```

```
y[1:5]
```

```
## array([0, 0, 0, 0])
```

2.2.2 定义 sigmoid 函数

sigmoid 函数

[2] 只展示前 5 行

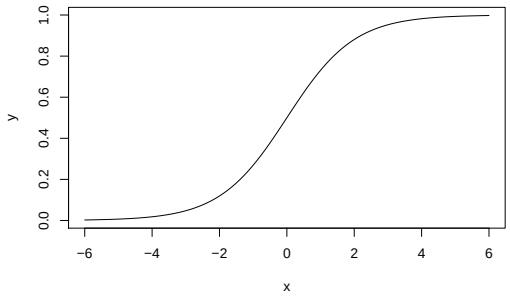

图 1: sigmoid function

```
def sigmoid(Xi,thetas):
    params = - np.sum(Xi * thetas)
    outcome = 1 /(1 + math.exp(params))
    return outcome
```

适用于矩阵的 sigmoid 函数

```
def sigmoidMatrix(Xb,thetas):
    params = - Xb.dot(thetas)
    outcome = np.zeros(params.shape[0])
    for i in range(len(outcome)):
        outcome[i] = 1 /(1 + math.exp(params[i]))
    return outcome
```

带阈值判别的 sigmoid 函数

- 阈值（threshold）：用于给出概率后进行分类，默认为 50%

```
def sigmoidThreshold(Xb,thetas):
    params = - Xb.dot(thetas)
    outcome = np.zeros(params.shape[0])
    for i in range(len(outcome)):
        outcome[i] = 1 /(1 + math.exp(params[i]))
        if outcome[i] >= 0.5:
            outcome[i] = 1
        else:
            outcome[i] = 0
    return outcome
```

2.2.3 定义损失函数

损失函数（cost function）就是最大似然函数**取对数**

```
def costFunc(Xb,y):
    sum = 0.0
    for i in range(m):
        yPre = sigmoid(Xb[i,], thetas)
        if yPre == 1 or yPre == 0:
            return float(-2**31)
        sum += y[i] * math.log(yPre) + (1 - y[i]) * math.log(1-yPre)
    return -1/m * sum
```

2.2.4 用梯度下降法进行训练

初始化：

- 学习率（alpha）：用于调整每次迭代的对损失函数的影响大小
- 准确度（accuracy）：作为终止迭代的评判指标

```
thetas = None
m = 0
alpha = 0.01
accuracy = 0.001
```

在第一列插入 1，构成 Xb 矩阵

```
thetas = np.full(X.shape[1]+1,0.5)
m = X.shape[0]
a = np.full((m, 1), 1)
Xb = np.column_stack((a, X))
n = X.shape[1] + 1
```

使用梯度下降法进行迭代：

```
count = 1
while True:
    before = costFunc(Xb, y)
    c = sigmoidMatrix(Xb, thetas)-y
    for j in range(n):
        thetas[j] = thetas[j] -alpha * np.sum(c * Xb[:,j])
    after = costFunc(Xb, y)
    if after == before or math.fabs(after - before) < accuracy:
        print(" 迭代完成，损失函数值:",after)
        break
    print(" 迭代次数:",count)
```

```
    print(" 损失函数变化",(after - before))
    count += 1
```

```
## 迭代次数: 1
## 损失函数变化 5.025121118513411
## 迭代次数: 2
## 损失函数变化 -3.292032434097819
## 迭代次数: 3
## 损失函数变化 -0.9713865369569756
## 迭代次数: 4
## 损失函数变化 2.649715136143148
## 迭代次数: 5
## 损失函数变化 -6.0552442422478245
## 迭代次数: 6
## 损失函数变化 -0.08147412308098556
## 迭代次数: 7
## 损失函数变化 -0.05224409412488061
## 迭代完成, 损失函数值: 0.00953683385119681
```

迭代完成后得到的参数:

```
thetas
```

```
## array([ 0.05586408, -0.68733466, -1.75042736,  2.95078531,  1.58964498])
```

2.2.5 预测

```
a = np.full((len(X),1),1)
Xb = np.column_stack((a,X))
sigmoidThreshold(Xb, thetas)
```

```
## array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
##        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
##        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1.,
##        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
##        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
##        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

2.3 代码实现 - 类封装

可以调整的参数包括：

- 学习率（alpha）：用于调整每次迭代的对损失函数的影响大小
- 准确度（accuracy）：作为终止迭代的评判指标
- 阈值（threshold）：用于给出概率后进行分类，默认为 50%

```
# ----------------- 导入基本模块 -----------------
import numpy as np
import math

# ------------ 导入 source 中定义的函数 ------------
from source.sigmoid import sigmoid
from source.sigmoidMatrix import sigmoidMatrix
from source.sigmoidThreshold import sigmoidThreshold

# ----------------- 定义 base 类 -----------------
class Regression(object):
    def __init__(self, X, y, threshold = 0.5):
        self.thetas = None
        self.X = X
        self.y = y

# ---------------- 定义逻辑回归类 ----------------
class LogsiticRegression(Regression):
    def __init__(self, X, y, threshold = 0.5):
        Regression.__init__(self, X, y, threshold = 0.5) # 继承 Regression 类
        self.m = 0
        self.threshold = threshold
        self.epoch = 1

    def fit(self, alpha = 0.01, accuracy = 0.001):
        self.thetas = np.full(self.X.shape[1] + 1,0.5)
        self.m = self.X.shape[0]
        a = np.full((self.m, 1), 1)
        Xb = np.column_stack((a,self.X))
        n  = self.X.shape[1]+1

        while True:
            before = self.costFunc(Xb, y)
```

```
            c = sigmoidMatrix(Xb, self.thetas) - y
            for j in range(n):
                self.thetas[j] = self.thetas[j] - alpha * np.sum(c * Xb[:,j])
            after = self.costFunc(Xb, y)
            if after == before or math.fabs(after - before) < accuracy:
                break
            self.epoch += 1

    def costFunc(self, Xb, y):
        sum = 0.0
        for i in range(self.m):
            yPre = sigmoid(Xb[i,], self.thetas)
            if yPre == 1 or yPre == 0:
                return float(-2**31)
            sum += y[i] * math.log(yPre) + (1 - y[i]) * math.log(1 - yPre)
        return -1/self.m * sum

    def predict(self):
        a = np.full((len(X), 1), 1)
        Xb = np.column_stack((a, X))
        return sigmoidThreshold(Xb, self.thetas, self.threshold)
```

2.4 测试用例

我们使用经典的 iris 数据集作为测试用例。

```
iris = datasets.load_iris()
X = iris['data']
y = iris['target']
X = X[y!=2]
y = y[y!=2]

Logstic = LogsiticRegression(X, y)
Logstic.fit()
print("epoch:",Logstic.epoch)
```

```
## epoch: 8
```

```
print("theta:",Logstic.thetas)
```

```
## theta: [ 0.05586408 -0.68733466 -1.75042736  2.95078531  1.58964498]
```

```
y_predict = Logstic.predict()
y_predict
```

```
## array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
##        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
##        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1.,
##        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
##        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
##        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

3 神经网络

我们使用**随机梯度下降**的优化方法构建一个较为简单的神经网络模型。[3]

3.1 模型求解步骤

4 参考文献

[3]考虑到神经网络是较为复杂的一类模型，在此我们只构造它的初始版本，限制较多，不具备广泛实用性。