第 5 章 logistic回归

5.1 logistic回归的数学基础

5.1.1 单位阶跃函数

考虑二分类任务, 其输出标记 $y \in \{0,1\}$, 而线性回归模型产生的预测值 $z=\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}+b$是实值, 于是, 我们需将实值 z 转换为 0/ 1 值. 最理想的是"单位阶跃函数" (unit-step function) .

$$y=\begin{cases} 0, & z<0 \\ 0.5, & z=0 \\ 1, & z>0 \end{cases} \tag{3.16}$$

即即若预测值 z 大于零就判为正例, 小于零则判为反例, 预测值为临界值零则可任意判别.

5.1.2 对数几率函数

$$y=\frac{1}{1+e^{-z}} \tag{3.17}$$

将对数几率函数作为 $g^{-}(\cdot)$ 带入到 (3.15), 就可以得到:

$$y=\frac{1}{1+e^{-\left(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}+b\right)}} \tag{3.18}$$

进行变换可得:

$$\ln \frac{y}{1-y}=\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}+b \tag{3.19}$$

若将 y 视为样本 $\boldsymbol{x}$ 作为正例的可能性, 则 $1-y$ 是反例的可能性, 两者的比值

$$\frac{y}{1-y} \tag{3.20}$$

成为"几率"(odds), 反映了 $\boldsymbol{x}$ 作为正例的相对可能性. 对几率取对数则可以得到"对数几率"(log odds,也称 logit)

$$\ln \frac{y}{1-y} \tag{3.21}$$

5.1.3 对数几率回归中参数的求解

若将式(3.18)中的 y 视为类后验概率估计 $p(y=1|\boldsymbol{x})$, 则(3.19) 可以重写为:

$$\ln \frac{p(y=1|\boldsymbol{x})}{p(y=0|\boldsymbol{x})}=\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}+b \tag{3.22}$$

由于 $p(y=1|\boldsymbol{x})+p(y=0|\boldsymbol{x})=1$, 则可以求得:

$$p(y=1|\boldsymbol{x})=\frac{e^{\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}+b}}{1+e^{\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}+b}} \tag{3.23}$$

$$p(y=0|\boldsymbol{x}) = \frac{1}{1+e^{\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b}} \tag{3.24}$$

于是, 我们可通过"极大似然法" (maximum likelihood method)来估计w 和 b. 给定数据集 $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^m$, 对率回归模型最大化"对数似然" (log-likelihood) .

$$\ell(\boldsymbol{w}, b) = \sum_{i=1}^{m} \ln p\left(y_i|\boldsymbol{x}_i; \boldsymbol{w}, b\right) \tag{3.25}$$

和上面一样, 为便于讨论和简写, 令 $\boldsymbol{\beta} = (\boldsymbol{w}; b)$, $\hat{\boldsymbol{x}} = (\boldsymbol{x}; 1)$, 则 $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b$ 可简写为 $\boldsymbol{\beta}^{\mathrm{T}}\hat{\boldsymbol{x}}$. 同时再令 $p_1(\hat{\boldsymbol{x}}; \boldsymbol{\beta}) = p(y=1|\hat{\boldsymbol{x}}; \boldsymbol{\beta})$, $p_0(\hat{\boldsymbol{x}}; \boldsymbol{\beta}) = p(y=0|\hat{\boldsymbol{x}}; \boldsymbol{\beta}) = 1 - p_1(\hat{\boldsymbol{x}}; \boldsymbol{\beta})$, 则式 (3.25)中的似然项可以写成:

$$p\left(y_i|\boldsymbol{x}_i; \boldsymbol{w}, b\right) = y_i p_1\left(\hat{\boldsymbol{x}}_i; \boldsymbol{\beta}\right) + \left(1 - y_i\right) p_0\left(\hat{\boldsymbol{x}}_i; \boldsymbol{\beta}\right) \tag{3.26}$$

> **推导3**: y_i 只能取0或1, 分别带入即可:
>
> $$p\left(y_i|\boldsymbol{x}_i; \boldsymbol{w}, b\right) = \begin{cases} p_1\left(\hat{\boldsymbol{x}}_i; \boldsymbol{\beta}\right) & \text{if } y_i = 1 \\ p_0\left(\hat{\boldsymbol{x}}_i; \boldsymbol{\beta}\right) & \text{if } y_i = 0 \end{cases}$$

将式(3.26)代入到(3.25)中, 且根据(3.23)和(3.24), 则最大化式 (3.25) 等价于**最小化**:

$$\ell(\boldsymbol{\beta}) = \sum_{i=1}^{m} \left(-y_i \boldsymbol{\beta}^{\mathrm{T}} \hat{\boldsymbol{x}}_i + \ln\left(1 + e^{\boldsymbol{\beta}^{\mathrm{T}} \hat{\boldsymbol{x}}_i}\right)\right) \tag{3.27}$$

> **推导4**: 将式 (3.26) 带入式 (3.25) 可以得到:
>
> $$l(\beta) = \sum_{i=1}^{m} \ln(y_i p_1\left(\hat{\boldsymbol{x}}_i; \beta\right) + \left(1 - y_i\right) p_0\left(\hat{\boldsymbol{x}}_i; \beta\right))$$
>
> 同时, $p_1\left(\hat{\boldsymbol{x}}_i; \beta\right) = \frac{e^{\beta^T \hat{\boldsymbol{x}}_i}}{1+e^{\beta^T \hat{\boldsymbol{x}}_i}}$, $p_0\left(\hat{\boldsymbol{x}}_i; \beta\right) = \frac{1}{1+e^{\beta^T \hat{\boldsymbol{x}}_i}}$, 代入到上式可得:
>
> $$\begin{aligned} l(\beta) &= \sum_{i=1}^{m} \ln\left(\frac{y_i e^{\beta^T \hat{\boldsymbol{x}}_i} + 1 - y_i}{1 + e^{\beta^T \hat{\boldsymbol{x}}_i}}\right) \\ &= \sum_{i=1}^{m} \left(\ln\left(y_i e^{\beta^T \hat{\boldsymbol{x}}_i} + 1 - y_i\right) - \ln\left(1 + e^{\beta^T \hat{\boldsymbol{x}}_i}\right)\right) \end{aligned}$$
>
> 由于 $y_i = 0$或1 , 则有:
>
> $$l(\beta) = \begin{cases} \sum\limits_{i=1}^{m} \left(-\ln\left(1 + e^{\beta^T \hat{\boldsymbol{x}}_i}\right)\right), & y_i = 0 \\ \sum\limits_{i=1}^{m} \left(\beta^T \hat{\boldsymbol{x}}_i - \ln\left(1 + e^{\beta^T \hat{\boldsymbol{x}}_i}\right)\right), & y_i = 1 \end{cases}$$
>
> 把两式综合可得:
>
> $$l(\beta) = \sum_{i=1}^{m} \left(y_i \beta^T \hat{\boldsymbol{x}}_i - \ln\left(1 + e^{\beta^T \hat{\boldsymbol{x}}_i}\right)\right)$$
>
> 添加负号即是式(3.27), 也即是最小化

式 (3.27)是关于 β 的高阶可导连续凸函数 , 根据凸优化理论, 用梯度下降法,牛顿法都可以求得最优解. 则可以得到

$$\boldsymbol{\beta}^* = \arg\min_{\boldsymbol{\beta}} \ell(\boldsymbol{\beta}) \tag{3.28}$$

5.2 梯度下降(梯度上升)

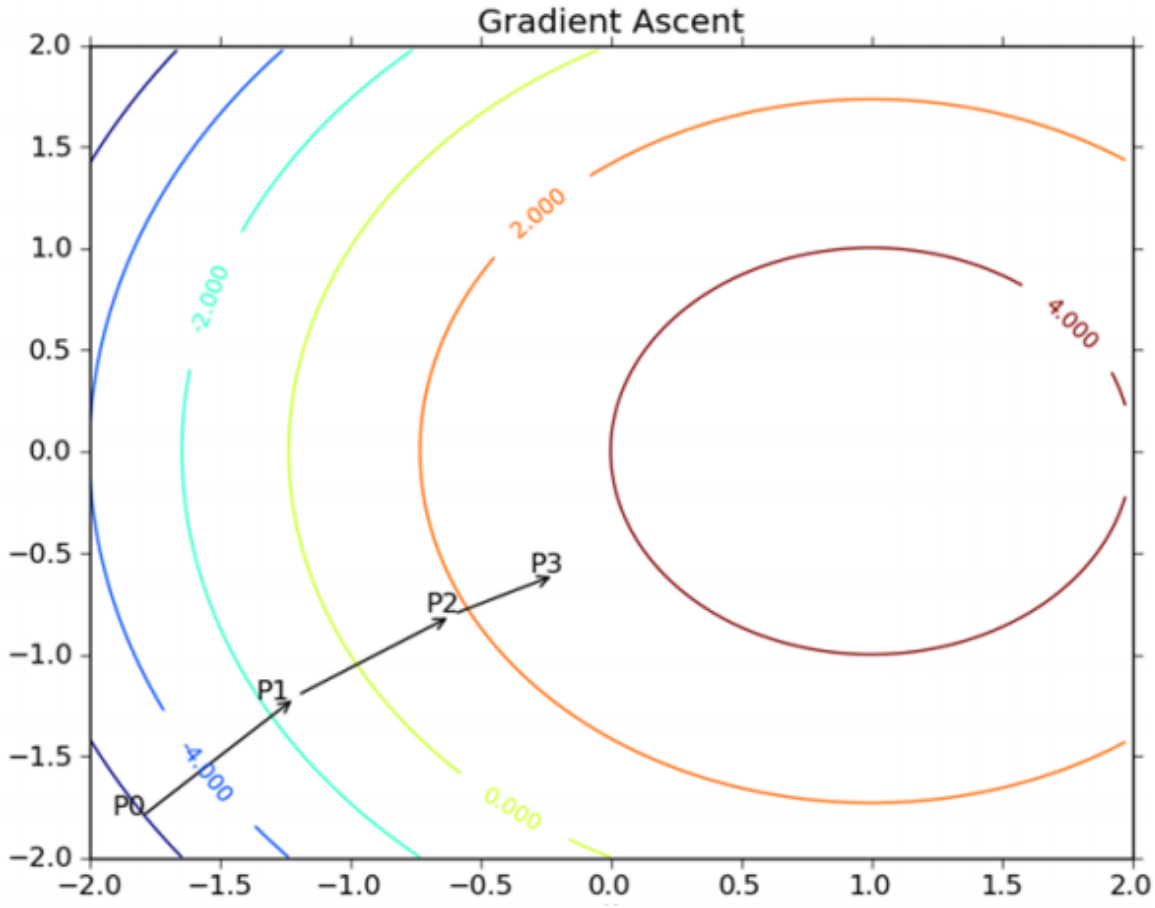
首先, 了解一下梯度数学方面的概念

- 向量 = 值 + 方向
- 梯度 = 向量
- 梯度 = 梯度的值 + 梯度的方向

梯度上升法基于的思想是: 要找到某函数的最大值, 最好的方法是沿着该函数的梯度方向探寻. 如果梯度记为 ∇ , 则函数 $f(x,y)$ 的梯度由下式表示:

$$\nabla f(x,y) = \begin{pmatrix} \frac{\partial f(x,y)}{\partial x} \\ \frac{\partial f(x,y)}{\partial y} \end{pmatrix}$$

这个梯度意味着要沿 x 的方向移动 $\frac{\partial f(x,y)}{\partial x}$, 沿 y 的方向移动 $\frac{\partial f(x,y)}{\partial y}$. 其中, 函数 $f(x,y)$ 必须要在待计算的点上有定义并且可微. 一个具体的函数的例子如下图:



上图展示的, 梯度上升算法到达每个点后都会重新估计移动的方向. 从 p_0 开始, 计算完该点的梯度, 函数就根据梯度移动到下一点 p_1. 在 p_1 点, 梯度再次被重新计算, 并沿着新的梯度方向移动到 p_2. 如此循环迭代, 直到满足停止条件. 迭代过程中, 梯度算子总是保证我们能选取到最佳的移动方向.

注意:

- 梯度是一个**向量**, 有方向有大小
- 梯度的方向是**最大方向导数**的方向
- 梯度的值的最大方向导数的值

梯度即函数在某一点最大的方向导数, 函数沿梯度方向, 函数的变化率最大.

梯度总是指向函数值增长最快的方向. 这里所说的是移动方向, 而未提到移动量的大小. 该量值称为步长, 记做α, 用向量来表示的话, **梯度上升算法**的迭代公式如下:

$$w := w + \alpha \nabla_w f(w)$$

该公式将一直被迭代执行, 直至达到某个停止条件为止, 比如迭代次数达到某个指定值或者算法达到某个可以允许的误差范围.

梯度下降算法, 它与这里的梯度上升算法是一样的, 只是公式中的加法需要变成减法. 因此, 对应的公式可以写成:

$$w := w - \alpha \nabla_w f(w)$$

梯度上升算法用来求函数的最大值, 而梯度下降算法用来求函数的最小值

> 注:关于梯度和梯度下降的概念可以参考以下:
>
> 1. 深入浅出--梯度下降法及其实现: https://www.jianshu.com/p/c7e642877b0e
> 2. 知乎: 如何直观形象的理解方向导数与梯度以及它们之间的关系? https://www.zhihu.com/question/36301367

5.3 logistic的梯度上升法

对于 **5.1** 中的公式 (3.27) , 即是最小化似然函数 $\ell(\boldsymbol{\beta})$

$$\ell(\boldsymbol{\beta}) = \sum_{i=1}^{m} \left(-y_i \boldsymbol{\beta}^{\mathrm{T}} \hat{\boldsymbol{x}}_i + \ln\left(1 + e^{\boldsymbol{\beta}^{\mathrm{T}} \hat{\boldsymbol{x}}_i}\right) \right) \tag{3.27}$$

对 $\boldsymbol{\beta}$ 求导可得:

$$\begin{aligned} \frac{\partial \ell(\boldsymbol{\beta})}{\partial(\boldsymbol{\beta})} &= \sum_{i=1}^{m} \left(-y_i \frac{\partial\left(\boldsymbol{\beta}^{\mathrm{T}} \hat{\boldsymbol{x}}_i\right)}{\partial \boldsymbol{\beta}} + \frac{1}{1 + e^{\boldsymbol{\beta}^{\mathrm{T}} \hat{\boldsymbol{x}}_i}} \cdot \left(e^{\boldsymbol{\beta}^{\mathrm{T}} \hat{\boldsymbol{x}}_i} \cdot \frac{\partial\left(\boldsymbol{\beta}^{\mathrm{T}} \hat{\boldsymbol{x}}_i\right)}{\partial \boldsymbol{\beta}} \right) \right) \\ &= \sum_{i=1}^{m} \left(-y_i \hat{\boldsymbol{x}}_i + \frac{e^{\boldsymbol{\beta}^{\mathrm{T}} \hat{\boldsymbol{x}}_i} \cdot \hat{\boldsymbol{x}}_i}{1 + e^{\boldsymbol{\beta}^{\mathrm{T}} \hat{\boldsymbol{x}}_i}} \right) \end{aligned} \tag{3.30}$$

我们知道, $\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x} + b$ 简写为 $\boldsymbol{\beta}^{\mathrm{T}} \hat{\boldsymbol{x}}$, 且, 由公式 (3.18) 即, $y = \frac{1}{1+e^{-\left(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}+b\right)}}$, 可知: $y = \frac{1}{1+e^{-\boldsymbol{\beta}^{T} \hat{\boldsymbol{x}}_i}}$

注意: 这里的 y 是预测的值, y_i 是真实值, 两者不可混淆. 可以令 $\sigma(\boldsymbol{\beta}^{\mathrm{T}} \hat{\boldsymbol{x}}_i) = y = \frac{1}{1+e^{-\boldsymbol{\beta}^{T} \hat{\boldsymbol{x}}_i}}$

$$e^{\boldsymbol{\beta}^{\top} \hat{\boldsymbol{x}}_i} = \frac{\sigma(\boldsymbol{\beta}^{\mathrm{T}} \hat{\boldsymbol{x}}_i)}{1 - \sigma(\boldsymbol{\beta}^{\mathrm{T}} \hat{\boldsymbol{x}}_i)}$$

将上式带入到公式 (3.30) , 化简可得:

$$\begin{aligned} \frac{\partial \ell(\boldsymbol{\beta})}{\partial(\boldsymbol{\beta})} &= \sum_{i=1}^{m} (-y_i \hat{\boldsymbol{x}}_i + \sigma(\boldsymbol{\beta}^{\mathrm{T}} \hat{\boldsymbol{x}}_i) \hat{\boldsymbol{x}}_i) \\ &= \sum_{i=1}^{m} \left(-y_i + \sigma(\boldsymbol{\beta}^{\mathrm{T}} \hat{\boldsymbol{x}}_i) \right) \hat{\boldsymbol{x}}_i \end{aligned}$$

那么梯度下降的更新公式可以表达为:

$$\boldsymbol{\beta} := \boldsymbol{\beta} - \alpha \sum_{i=1}^{m} \left(-y_i + \sigma(\boldsymbol{\beta}^{\mathrm{T}} \hat{\boldsymbol{x}}_i)\right) \hat{\boldsymbol{x}}_i$$

> 注: 与梯度上升法最后得到的更新公式:
>
> $$\boldsymbol{\beta} := \boldsymbol{\beta} + \alpha \sum_{i=1}^{m} \left(y_i - \sigma(\boldsymbol{\beta}^{\mathrm{T}} \hat{\boldsymbol{x}}_i)\right) \hat{\boldsymbol{x}}_i \tag{3.31}$$
>
> 是一致的.

5.4 项目案例1: 使用 Logistic 回归在简单数据集上的分类

5.4.1 项目概述

在一个简单的数据集上, 采用梯度上升法找到 Logistic 回归分类器在此数据集上的最佳回归系数

5.4.1 开发流程

- (1) 收集数据: 采用任意方法收集数据.
- (2) 准备数据: 由于需要进行距离计算, 因此要求数据类型为数值型. 另外, 结构化数据格式则最佳.
- (3) 分析数据: 采用任意方法对数据进行分析.
- (4) 训练算法: 大部分时间将用于训练, 训练的目的是为了找到最佳的分类回归系数.
- (5) 测试算法: 一旦训练步骤完成, 分类将会很快.
- (6) 使用算法: 首先, 我们需要输入一些数据, 并将其转换成对应的结构化数值; 接着, 基于训练好的回归系数就可以对这些数值进行简单的回归计算, 判定它们属于哪个类别; 在这之后, 我们就可以在输出的类别上做一些其他分析工作.

5.4.2 开发过程

一 加载并解析数据 (loadDataSet函数)

打开testSet.txt数据, 读取并解析数据. 先看下数据的样式:

testSet.txt　　用记事本打开

```
-0.017612   14.053064   0
-1.395634   4.662541    1
-0.752157   6.538620    0
-1.322371   7.152853    0
0.423363    11.054677   0
0.406704    7.067335    1
0.667394    12.741452   0
-2.460150   6.866805    1
0.569411    9.548755    0
-0.026632   10.427743   0
0.850433    6.920334    1
1.347183    13.175500   0
1.176813    3.167020    1
-1.781871   9.097953    0
-0.566606   5.749003    1
0.931635    1.589505    1
-0.024205   6.151823    1
-0.036453   2.690988    1
-0.196949   0.444165    1
1.014459    5.754399    1
1.985298    3.230619    1
-1.693453   -0.557540   1
-0.576525   11.778922   0
-0.346811   -1.678730   1
-2.124484   2.672471    1
1.217916    9.597015    0
-0.733928   9.098687    0
-3.642001   -1.618087   1
0.315985    3.523953    1
1.416614    9.619232    0
-0.386323   3.989286    1
0.556921    8.294984    1
1.224863    11.587360   0
-1.347803   -2.406051   1
1.196604    4.951851    1
0.275221    9.543647    0
0.470575    9.332488    0
```

```python
import numpy as np
"""
函数说明: 加载并解析数据

Parameters:
    无
Returns:
    dataMat - 数据列表(list类型)
    labelMat - 标签列表(list类型)
"""


def loadDataSet():
    #初始化数据列表和标签列表
    dataMat = []
    labelMat = []
    #打开文件
    with open('testSet.txt') as f:
        #读取文件所有内容形成列表,再对列表进行遍历
        for line in f.readlines():
            #剔除字符串首尾空白,并切割元素,形成列表
            lineArr = line.strip().split()
            #将索引0填充1,索引1填充lineArr[0],索引1填充lineArr[1]
            #dataMat第一列为常数项,第二列为x1(即x),第二列为x2(即y)
            dataMat.append([1, float(lineArr[0]), float(lineArr[1])])
            #用lineArr[2]填充labelMat
            labelMat.append(int(lineArr[2]))
    return dataMat, labelMat
if __name__ == '__main__':
    dataMat, labelMat = loadDataSet()
```

```
    print(dataMat)
    print(labelMat)
```

结果如下:

```
In [4]: runfile('C:/Users/HeatonHsu/.spyder-py3/temp.py', wdir='C:/Users/HeatonHsu/.spyder-py3')
[[1, -0.017612, 14.053064], [1, -1.395634, 4.662541], [1, -0.752157, 6.53862], [1, -1.322371, 7.152853], [1, 0.423363,
11.054677], [1, 0.406704, 7.067335], [1, 0.667394, 12.741452], [1, -2.46015, 6.866805], [1, 0.569411, 9.548755], [1, -0.026632,
10.427743], [1, 0.850433, 6.920334], [1, 1.347183, 13.1755], [1, 1.176813, 3.16702], [1, -1.781871, 9.097953], [1, -0.566606,
5.749003], [1, 0.931635, 1.589505], [1, -0.024205, 6.151823], [1, -0.036453, 2.690988], [1, -0.196949, 0.444165], [1, 1.014459,
5.754399], [1, 1.985298, 3.230619], [1, -1.693453, -0.55754], [1, -0.576525, 11.778922], [1, -0.346811, -1.67873], [1,
-2.124484, 2.672471], [1, 1.217916, 9.597015], [1, -0.733928, 9.098687], [1, -3.642001, -1.618087], [1, 0.315985, 3.523953],
[1, 1.416614, 9.619232], [1, -0.386323, 3.989286], [1, 0.556921, 8.294984], [1, 1.224863, 11.58736], [1, -1.347803, -2.406051],
[1, 1.196604, 4.951851], [1, 0.275221, 9.543647], [1, 0.470575, 9.332488], [1, -1.889567, 9.542662], [1, -1.527893, 12.150579],
[1, -1.185247, 11.309318], [1, -0.445678, 3.297303], [1, 1.042222, 6.105155], [1, -0.618787, 10.320986], [1, 1.152083,
0.548467], [1, 0.828534, 2.676045], [1, -1.237728, 10.549033], [1, -0.683565, -2.166125], [1, 0.229456, 5.921938], [1,
-0.959885, 11.555336], [1, 0.492911, 10.993324], [1, 0.184992, 8.721488], [1, -0.355715, 10.325976], [1, -0.397822, 8.058397],
[1, 0.824839, 13.730343], [1, 1.507278, 5.027866], [1, 0.099671, 6.835839], [1, -0.344008, 10.717485], [1, 1.785928, 7.718645],
[1, -0.918801, 11.560217], [1, -0.364009, 4.7473], [1, -0.841722, 4.119083], [1, 0.490426, 1.960539], [1, -0.007194, 9.075792],
[1, 0.356107, 12.447863], [1, 0.342578, 12.281162], [1, -0.810823, -1.466018], [1, 2.530777, 6.476801], [1, 1.296683,
11.607559], [1, 0.475487, 12.040035], [1, -0.783277, 11.009725], [1, 0.074798, 11.02365], [1, -1.337472, 0.468339], [1,
-0.102781, 13.763651], [1, -0.147324, 2.874846], [1, 0.518389, 9.887035], [1, 1.015399, 7.571882], [1, -1.658086, -0.027255],
[1, 1.319944, 2.171228], [1, 2.056216, 5.019981], [1, -0.851633, 4.375691], [1, -1.510047, 6.061992], [1, -1.076637,
-3.181888], [1, 1.821096, 10.28399], [1, 3.01015, 8.401766], [1, -1.099458, 1.688274], [1, -0.834872, -1.733869], [1,
-0.846637, 3.849075], [1, 1.400102, 12.628781], [1, 1.752842, 5.468166], [1, 0.078557, 0.059736], [1, 0.089392, -0.7153], [1,
1.825662, 12.693808], [1, 0.197445, 9.744638], [1, 0.126117, 0.922311], [1, -0.679797, 1.22053], [1, 0.677983, 2.556666], [1,
0.761349, 10.693862], [1, -2.168791, 0.143632], [1, 1.38861, 9.341997], [1, 0.317029, 14.739025]]
[0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1,
0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1,
1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0]
```

二 定义sigmoid函数

比较简单, 直接给出代码:

```
"""
函数说明: sigmoid函数

Parameters:
    inX - 输入数据(数字或者np.ndarray)
Returns:
    sigmoid函数值(数字或者np.ndarray)
"""
def sigmoid(inX):
    return 1 / (1+ np.exp(-inX))
```

三 梯度上升法 (gradAscend函数)

```
"""
函数说明: 梯度上升法

Parameters:
    dataMatIn - 数据集(是一个2维NumPy数组,  每列分别代表每个不同的特征,  每行则代表每个训
练样本)(array_like类型)
    classLabels - 数据标签(是类别标签,  它是一个 1*100 的行向量.  为了便于矩阵计算,
                  需要将该行向量转换为列向量,  做法是将原向量转置,  再将它赋值给
labelMat
                  (array_like类型)
Returns:
    np.array(weights) - 返回权重数组(即最优的参数)(np.ndarray类型)
"""
def gradAscend(dataMatIn, classLabels):
    #转换成矩阵
    dataMatrix = np.mat(dataMatIn)
    #转换成矩阵, 并进行转置
    #转化为矩阵[[0,1,0,1,0,1.....]],  并转置[[0],[1],[0].....]
```

```
    #也就是首先将数组转换为 NumPy 矩阵，  然后再将行向量转置为列向量
    labelMat = np.mat(classLabels).transpose()
    #返回dataMatrix的大小，其中m,n分别为行数和列数
    #也就是m个数据量,即样本数，n个特征
    m, n = np.shape(dataMatrix)
    #设置步长,也就是学习率
    alpha = 0.001
    #设置最大迭代次数
    maxCycles = 500
    #生成一个长度和特征数相同的矩阵，  此处n为3 -> [[1],[1],[1]]
    #weights 代表回归系数，   此处的 ones((n,1)) 创建一个长度和特征数相同的矩阵，  其中的数全部都是 1
    weights = np.ones((n,1))
    for k in range(maxCycles):
        #m*n 的矩阵 * n*1 的矩阵 = m*1的矩阵
        #这个乘法的结果就是通过公式计算得到的理论值(此时还不是label,因为label是0或者1,通过sigmoid函数后,h就是理论label)
        h = sigmoid(dataMatrix*weights)
        #求出错误矩阵
        #labelmat是实际值
        error = (labelMat-h)
        #关于梯度上升中梯度的求解,以及为何矩阵乘积是下面的形式,有详细参考.
        weights = weights + alpha * dataMatrix.transpose() * error
    return np.array(weights)


if __name__ == '__main__':
    dataMat, labelMat = loadDataSet()
    weights = gradAscend(dataMat,labelMat)
    #上面的还可以写成weights = gradAscend(np.array(dataMat),labelMat)
    #或者weights = gradAscenp(np.array(dataMat),np.array(labelMat))都可以
    #因为这两个参数是array_like的都行
    print(weights)
```

注意:本章最难理解的一段

就是代码倒数第二行: `weights = weights + alpha * dataMatrix.transpose() * error` **的说明:**

在<机器学习实战>的书上, 关于这段代码作者是这么说的: `此处略去了一个简单的数学推导,我把它留给有兴趣的读者`

作者说是一个简单的推导, 实际上, 要说明和理解这段计算代码, **需要 5.1-5.3 的数学知识及向量化推导**

具体如下:

1. **基于5.1-5.3 的数学推导和说明.**
2. **将梯度上升法迭代公式 (3.31) 进行向量化**, 可得:

$$\boldsymbol{\beta} := \boldsymbol{\beta} + \alpha \boldsymbol{X}^{\mathrm{T}}(\boldsymbol{y} - \sigma(\boldsymbol{X}\boldsymbol{\beta}))$$

> 注: 具体如何进行矢量化的, 看参考这篇文章
>
> https://blog.csdn.net/achuo/article/details/51160101
>
> 其中, 这篇文章中关于**梯度上升法迭代公式的矢量过程**如下:

3.4 梯度下降过程向量化

关于**θ**更新过程的vectorization，Andrew Ng的课程中只是一带而过，没有具体的讲解。

《机器学习实战》连Cost函数及求梯度等都没有说明，所以更不可能说明vectorization了。但是，其中给出的实现代码确是实现了vectorization的，图4所示代码的32行中weights（也就是**θ**）的更新只用了一行代码，直接通过矩阵或者向量计算更新，没有用for循环，说明确实实现了vectorization，具体代码下一章分析。

文献[3]中也提到了vectorization，但是也是比较粗略，很简单的给出vectorization的结果为：

$$\theta := \theta - \alpha \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(\mathrm{x}^{(i)}) - y^{(i)}\right)x^{(i)}, \quad (j = 0 \ldots n) \qquad (17)$$

且不论该更新公式正确与否，这里的**Σ(...)**是一个求和的过程，显然需要一个for语句循环m次，所以根本没有完全的实现vectorization，不像《机器学习实战》的代码中一条语句就可以完成**θ**的更新。

下面说明一下我理解《机器学习实战》中代码实现的vectorization过程。

约定训练数据的矩阵形式如下，**x**的每一行为一条训练样本，而每一列为不同的特称取值：

$$x = \begin{bmatrix} x^{(1)} \\ x^{(2)} \\ \cdots \\ x^{(\mathrm{m})} \end{bmatrix} = \begin{bmatrix} x_0^{(1)} & x_1^{(1)} & \cdots & x_n^{(1)} \\ x_0^{(2)} & x_1^{(2)} & \cdots & x_n^{(2)} \\ \cdots & \cdots & \cdots & \cdots \\ x_0^{(\mathrm{m})} & x_1^{(\mathrm{m})} & \cdots & x_n^{(\mathrm{m})} \end{bmatrix}, \quad y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \cdots \\ y^{(\mathrm{m})} \end{bmatrix} \qquad (18)$$

约定待求的参数**θ**的矩阵形式为：

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \cdots \\ \theta_n \end{bmatrix} \qquad (19)$$

先求**x.θ**并记为**A**：

$$A = x \bullet \theta = \begin{bmatrix} x_0^{(1)} & x_1^{(1)} & \cdots & x_n^{(1)} \\ x_0^{(2)} & x_1^{(2)} & \cdots & x_n^{(2)} \\ \cdots & \cdots & \cdots & \cdots \\ x_0^{(\mathrm{m})} & x_1^{(\mathrm{m})} & \cdots & x_n^{(\mathrm{m})} \end{bmatrix} \bullet \begin{bmatrix} \theta_0 \\ \theta_1 \\ \cdots \\ \theta_n \end{bmatrix} = \begin{bmatrix} \theta_0 x_0^{(1)} + \theta_1 x_1^{(1)} + \ldots + \theta_n x_n^{(1)} \\ \theta_0 x_0^{(2)} + \theta_1 x_1^{(2)} + \ldots + \theta_n x_n^{(2)} \\ \cdots \\ \theta_0 x_0^{(\mathrm{m})} + \theta_1 x_1^{(\mathrm{m})} + \ldots + \theta_n x_n^{(\mathrm{m})} \end{bmatrix}$$

(20)

求**hθ(x)-y**并记为**E**：

$$E = h_\theta(\mathrm{x}) - y = \begin{bmatrix} g\left(A^{(1)}\right) - y^{(1)} \\ g\left(A^{(2)}\right) - y^{(2)} \\ \\ g\left(A^{(\mathrm{m})}\right) - y^{(\mathrm{m})} \end{bmatrix} = \begin{bmatrix} e^{(1)} \\ e^{(2)} \\ \cdots \\ e^{(\mathrm{m})} \end{bmatrix} = g(A) - y \qquad (21)$$

g(A)的参数**A**为一列向量，所以实现**g**函数时要支持列向量作为参数，并返回列向量。由上式可知**hθ(x)-y**可以由**g(A)-y**一次计算求得。

再来看一下（15）式的**θ**更新过程，当**j=0**时：

$$\theta_0 := \theta_0 - \alpha \sum_{i=1}^{m}\left(h_\theta(\mathrm{x}^{(i)}) - y^{(i)}\right)x_0^{(i)}$$

$$m$$

$$= \theta_0 - \alpha \sum_{i=1} e^{(i)} x_0^{(i)} \quad (22)$$

$$= \theta_0 - \alpha \bullet \left(x_0^{(1)}, x_0^{(2)}, \ldots, x_0^{(m)} \right) \bullet E$$

同样的可以写出**θj**,

$$\theta_j := \theta_j - \alpha \bullet \left(x_j^{(1)}, x_j^{(2)}, \ldots, x_j^{(m)} \right) \bullet E \quad (23)$$

综合起来就是:

$$\begin{bmatrix} \theta_0 \\ \theta_1 \\ \cdots \\ \theta_n \end{bmatrix} := \begin{bmatrix} \theta_0 \\ \theta_1 \\ \cdots \\ \theta_n \end{bmatrix} - \alpha \bullet \begin{bmatrix} x_0^{(1)}, x_0^{(2)}, \ldots, x_0^{(m)} \\ x_1^{(1)}, x_1^{(2)}, \ldots, x_1^{(m)} \\ \\ x_n^{(1)}, x_n^{(2)}, \ldots, x_n^{(m)} \end{bmatrix} \bullet E \quad (24)$$

$$= \theta - \alpha \bullet x^T \bullet E$$

综上所述，vectorization后**θ**更新的步骤如下：

（1）求**A=x.θ**;

（2）求**E=g(A)-y**;

（3）求**θ:=θ-α.x'.E**,x'表示矩阵x的转置。

也可以综合起来写成:

$$\theta := \theta - \alpha \bullet \left(\frac{1}{m} \right) \bullet x^T \bullet \left(g\left(x \bullet \theta \right) - y \right)$$

前面已经提到过：1/m是可以省略的。

还有一个注意的地方就是关于各个输入和输出参数的数据类型.

之前, 一直没有注意这个, 以后, 关于每个函数的输入输出参数的数据类型都会做标注

注2:方法np.mat(data)是把data转化为矩阵. 关于np.mat(data)中data的数据类型如下:

```
In [4]: ?np.mat(a)
Signature: np.mat(data, dtype=None)
Docstring:
Interpret the input as a matrix.

Unlike `matrix`, `asmatrix` does not make a copy if the input is already
a matrix or an ndarray.  Equivalent to ``matrix(data, copy=False)``.

Parameters
----------
data : array_like
    Input data.
```

可以看到data的数据类型是array_like, 那么哪些是属于array_like的呢

array_like包括: array, list, tuple, dict, matrix以及基本数据类型int, string, float以及bool类型

> 注3: weights = gradAscend(dataMat,labelMat) #上面的还可以写成weights = gradAscend(np.array(dataMat),labelMat) #或者weights = gradAscenp(np.array(dataMat),np.array(labelMat))都可以 #因为这两个参数是array_like的都行

将 loadDataSet 和 sigmoid函数和 gradAscend函数结合在一起, 运行可以得到以下结果:

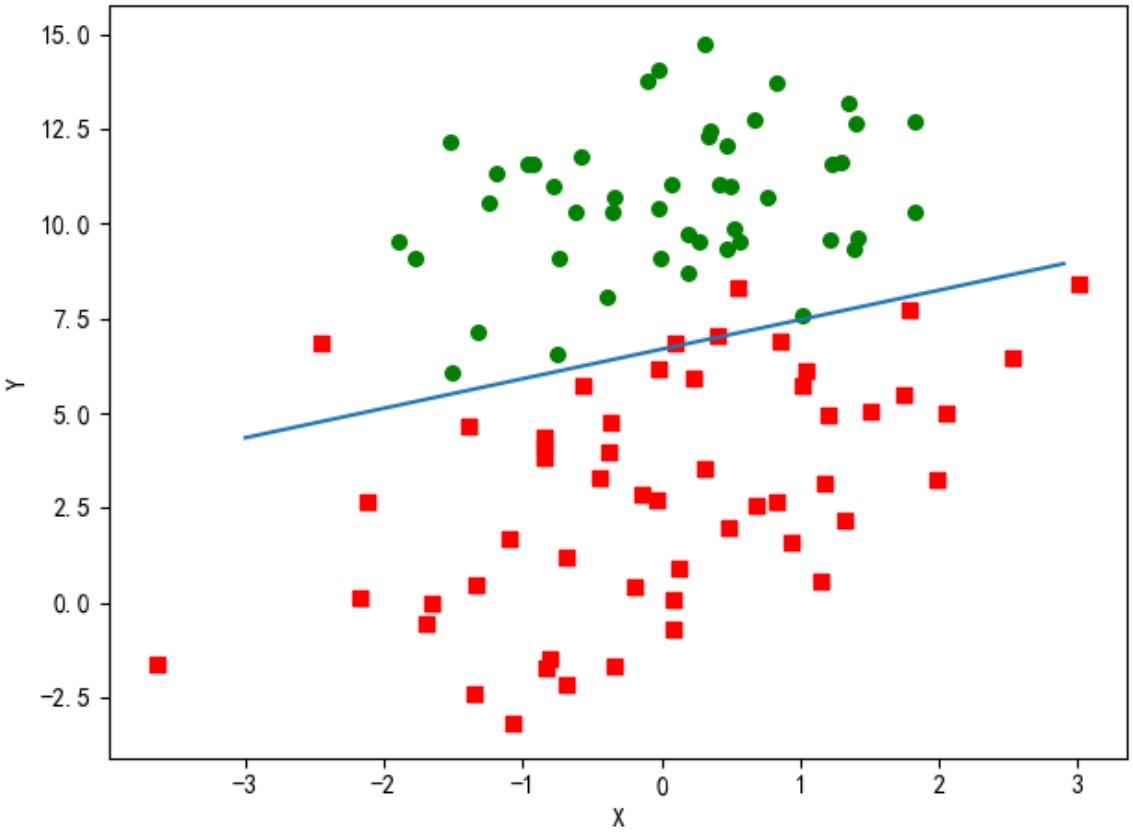
```
In [1]: runfile('C:/Users/HeatonHsu/.spyder-py3/temp.py', wdir='C:/Users/
HeatonHsu/.spyder-py3')
[[ 4.12414349]
 [ 0.48007329]
 [-0.6168482 ]]
```

四 画出决策边界 (plotBestFit函数)

画图就没什么特别要说的了, 直接放代码.

```
"""
函数说明: 绘制数据

Parameters:
    dataMat - 数据(array_like)
    labelMat - 样本的类别标签(array_like)
    weights - 回归系数组(np.ndarray)
Returns:
    无
"""
def plotBestFit(dataMat, labelMat, weights):
    #把dataMat转换成array数组
    dataArr = np.array(dataMat)
    #返回数据的个数
    n = np.shape(dataMat)[0]
    #正样本和负样本
    xcord1=[];ycord1=[]
    xcord2=[];ycord2=[]
    for i in range(n):
        if int(labelMat[i]) == 1:
            xcord1.append(dataArr[i,1]);ycord1.append(dataArr[i,2])
        else:
            xcord2.append(dataArr[i,1]);ycord2.append(dataArr[i,2])
    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.scatter(xcord1,ycord1,s=30,c='red', marker='s')
    ax.scatter(xcord2,ycord2,s=30,c='green')
    x = np.arange(-3.0,3.0,0.1)
    # w0+w1*x+w2*y=0 => y = (-w0-w1*x)/w2
    y = (-weights[0]-weights[1]*x) / weights[2]
    ax.plot(x,y)
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.show()
if __name__ =='__main__':
    dataMat, labelMat = loadDataSet()
    weights = gradAscend(dataMat,labelMat)
    plotBestFit(dataMat,labelMat,weights)
```

运行结果如下:



效果还不错.

五 算法的改进:随机梯度上升 (stocGradAscent0函数)

梯度上升算法在每次更新回归系数时都需要遍历整个数据集, 该方法在处理100个左右的数据集时尚可, 但如果有数十亿样本和成千上万的特征, 那么该方法的计算复杂度就太高了. 一种改进方法是**一次仅用一个样本点来更新回归系数**, 该方法称为**随机梯度上升算法**. 由于可以在新样本到来时对分类器进行增量式更新, 因而随机梯度上升算法是一个在线学习算法. 与"在线学习"相对应, 一次处理所有数据被称作是"批处理"

```
"""
函数说明: 随机梯度上升法

Parameters:
    dataMatrix - 数据集(是一个2维NumPy数组,  每列分别代表每个不同的特征,  每行则代表每个训练样本)(需输入np.ndarray)
    classLabels - 数据标签(是类别标签,  它是一个 1*100 的行向量.  为了便于矩阵计算,
                  需要将该行向量转换为列向量,  做法是将原向量转置,  再将它赋值给labelMat
                  (array_like)
Returns:
    weights- 返回权重(即最优的参数)(np.ndarray)
"""

def stocGradAscent0(dataMatrix, classLabels):
    #返回dataMatrix的大小, 其中m,n分别为行数和列数
    #也就是m个数据量,即样本数, n个特征
```

```
    m,n = np.shape(dataMatrix)
    #初始化alpha的值
    alpha = 0.01
    #函数ones创建一个全1的数组
    #初始化长度为n的数组，  元素全部为 1
    weights = np.ones(n)
    #遍历所有样本
    for i in range(m):
        #sum(dataMatrix[i]*weights)为了求 f(x)的值，   f(x)=a1*x1+b2*x2+..+nn*xn,
        #此处求出的 h 是一个具体的数值，  而不是一个矩阵
        h = sigmoid(sum(dataMatrix[i] * weights))
        #计算真实类别与预测类别之间的差值，  然后按照该差值调整回归系数
        error = classLabels[i] - h
        weights = weights + alpha * error*dataMatrix[i]
    return weights

if __name__ == '__main__':
    dataMat, labelMat = loadDataSet()
    weights = stocGradAscent0(np.array(dataMat),labelMat)
    print(weights)
```

随机梯度上升算法与梯度上升算法在代码上很相似, 但也有一些区别: 第一, 后者的变量 h 和误差error 都是向量, 而前者则全是数值; 第二, 前者没有矩阵的转换过程, 所有变量的数据类型都是NumPy数组.

运行结果如下:

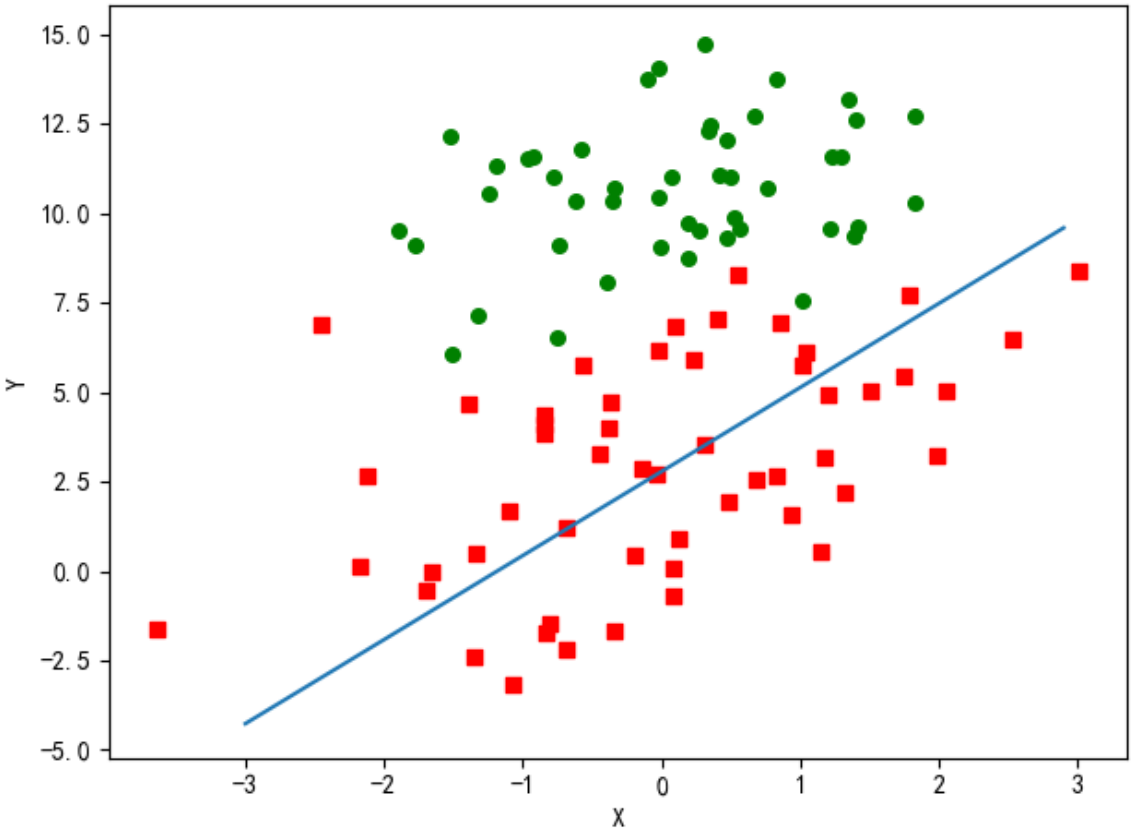
```
C:\SoftWare\Anaconda\python.exe D:/机器学习实战/logistic_hsu03.py
[ 1.01702007  0.85914348 -0.36579921]

Process finished with exit code 0
```

> 注: weights = stocGradAscent0(np.array(dataMat),labelMat)
>
> 这里的第一个参数为什么需要np.array(dataMat)?
>
> 在上一个梯度上升函数gradAscend函数中, 两个参数都是array_like的, 也就是list, array都可以, 而在stocGradAscent0函数中, classLabels是array_like的, 而 dataMatrix则必须是array. 如果dataMatrix传入和gradAscend一样的list会出现什么结果? 如下:
>
> 
>
>
> 报错信息: numpy.float64不能解释为整数.
>
> 其实错误就出现在error*datamatrix[i]这里, error是numpy.float64, 而dataMatrix[i]则是一个含有三个元素的列表.

接着再用绘图函数绘出决策边界,

```
if __name__ == '__main__':
    dataMat, labelMat = loadDataSet()
    weights = stocGradAscent0(np.array(dataMat),labelMat)
    plotBestFit(dataMat,labelMat,weights)
```

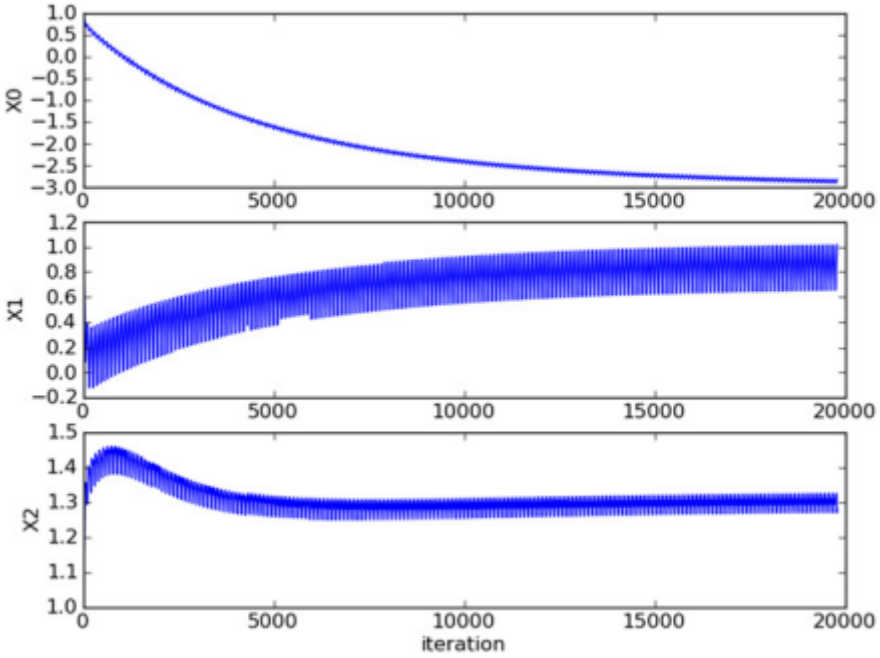
结果如下:



误分类有点多啊.

六 随机梯度的继续改进 (stocGradAscent1函数)

判断优化算法优劣的可靠方法是看它是否收敛, 也就是说参数是否达到了稳定值, 是否还会不断地变化? 下图展示了随机梯度上升算法在 200 次迭代过程中回归系数的变化情况. 其中的系数2, 也就是 X2 只经过了 50 次迭代就达到了稳定值, 但系数 1 和 0 则需要更多次的迭代. 如下图所示:



针对这个问题, 继续改进之前的随机梯度上升算法, 具体如下:

```
"""
函数说明: 随机梯度上升法(改进)

Parameters:
    dataMatrix - 数据集(是一个2维NumPy数组， 每列分别代表每个不同的特征， 每行则代表每个训练样本)(需输入np.ndarray)
    classLabels - 数据标签(是类别标签， 它是一个 1*100 的行向量. 为了便于矩阵计算，
                  需要将该行向量转换为列向量， 做法是将原向量转置， 再将它赋值给 labelMat
                  (array_like)
    numIter - 迭代次数(整数)
Returns:
    weights- 返回权重(即最优的参数)(np.ndarray)
"""

def stocGradAscent1(dataMatrix, classLabels, numIter = 150):
    # 返回dataMatrix的大小, 其中m,n分别为行数和列数
    # 也就是m个数据量,即样本数, n个特征
    m, n = np.shape(dataMatrix)
    # 函数ones创建一个全1的数组
    # 初始化长度为n的数组， 元素全部为 1
    weights = np.ones(n)
    #随机梯度, 循环150次, 观察是否收敛
    for j in range(numIter):
        dataIndex = list(range(m))
        for i in range(m):
            #i和j的不断增大， 导致alpha的值不断减少， 但是不为0
            alpha = 4 / (1+j+i) + 0.0001
            #随机产生一个 0～len()之间的一个整数值
            randIndex = int(random.uniform(0,len(dataIndex)))
            # sum(dataMatrix[i]*weights)为了求 f(x)的值, f(x)=a1*x1+b2*x2+..+nn*xn,
            # 此处求出的 h 是一个具体的数值， 而不是一个矩阵
            h = sigmoid(sum(dataMatrix[dataIndex[randIndex]]*weights))
```

```
            error = classLabels[dataIndex[randIndex]] - h
            weights = weights + alpha * error * dataMatrix[dataIndex[randIndex]]
            del(dataIndex[randIndex])
    return weights

if __name__ == '__main__':
    dataMat, labelMat = loadDataSet()
    weights = stocGradAscent1(np.array(dataMat),labelMat)
    print(weights)
```

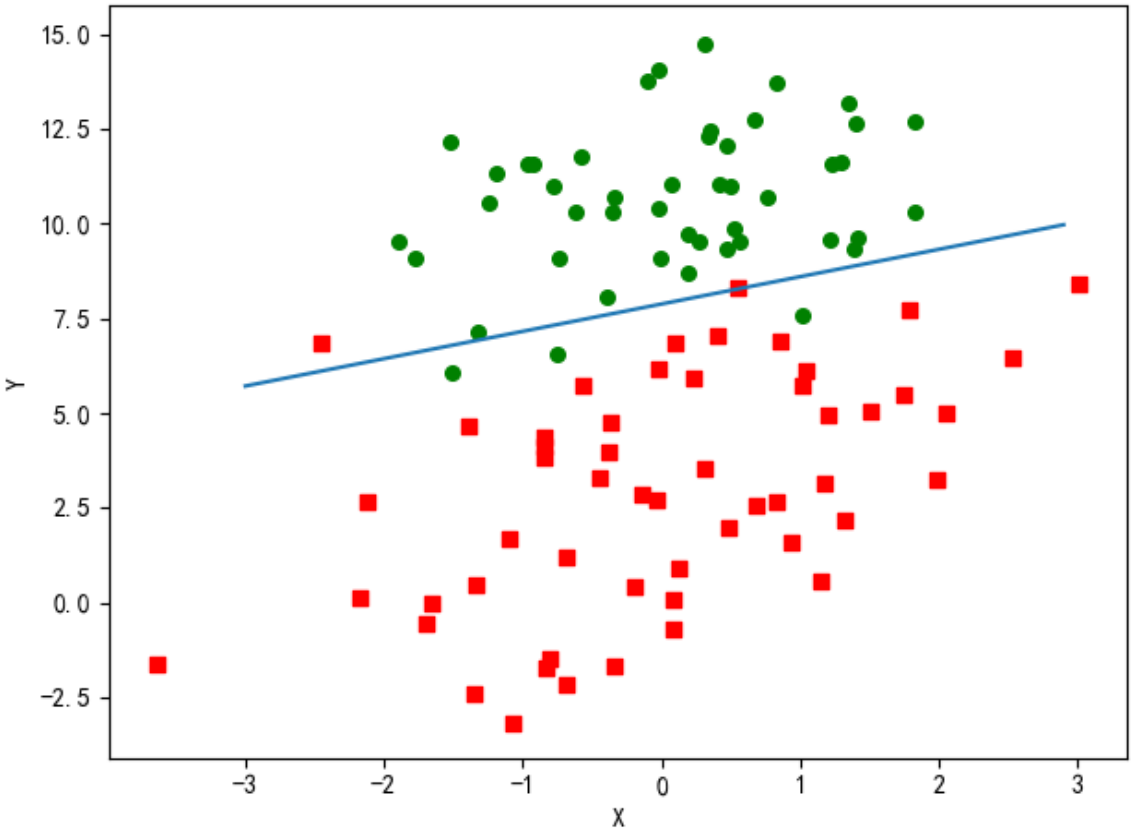
运行结果如下:

```
C:\SoftWare\Anaconda\python.exe D:/机器学习实战/logistic_hsu03.py
[13.95710174  1.17992426 -1.89419112]

Process finished with exit code 0
```

接着再用绘图函数绘出决策边界,

```
if __name__ == '__main__':
    dataMat, labelMat = loadDataSet()
    weights = stocGradAscent1(np.array(dataMat),labelMat)
    plotBestFit(dataMat, labelMat, weights)
```

运行结果如下:



这次就好多了.

5.5 项目案例2: 从疝气病症预测病马的死亡率

5.5.1 项目概述

使用Logistic回归来预测患有疝病的马的存活问题. 这里的数据包含368个样本和28个特征. 疝病是描述马胃肠痛的术语. 然而, 这种病不一定源自马的胃肠问题, 其他问题也可能引发马疝病. 该数据集中包含了医院检测马疝病的一些指标, 有的指标比较主观, 有的指标难以测量, 例如马的疼痛级别.

5.5.2 开发流程

- (1) 收集数据: 给定数据文件.
- (2) 准备数据: 用Python解析文本文件并填充缺失值.
- (3) 分析数据: 可视化并观察数据.
- (4) 训练算法: 使用优化算法, 找到最佳的系数.
- (5) 测试算法: 为了量化回归的效果, 需要观察错误率. 根据错误率决定是否回退到训练 阶段, 通过改变迭代的次数和步长等参数来得到更好的回归系数.
- (6) 使用算法: 实现一个简单的命令行程序来收集马的症状并输出预测结果并非难事,

5.5.3 开发过程

一 缺失值的处理

特征的缺失:

数据中的缺失值是个非常棘手的问题, 有很多文献都致力于解决这个问题

一些可行的做法如下:

- 使用可用特征的均值来填补缺失值;
- 使用特殊值来填补缺失值, 如-1;
- 忽略有缺失值的样本;
- 使用相似样本的均值添补缺失值;
- 使用另外的机器学习算法预测缺失值.

标签的缺失:

如果在测试数据集中发现了一条数据的类别标签已经缺失, 简单做法是将该条数据丢弃. 这是因为类别标签与特征不同, 很难确定采用某个合适的值来替换.

二 分类函数 (classifyVector函数)

```
"""
函数说明: 最终的分类函数,根据回归系数和特征向量来计算sigmoid的值, 大于0.5返回1,否则返回0

Parameters:
    inX - 特征向量,features (np.ndarray)
    weights - 根据梯度下降或者随机梯度下降, 计算的回归系数(np.ndarray)
```

```
Returns:
    返回0或者1
"""
def classifyVector(inX, weights):
    prob = sigmoid(sum(inX*weights))
    if prob > 0.5:
        return 1
    else:
        return 0
```

三 打开数据, 进行处理和分类 (colicTest函数)

```
"""
函数说明: 打开测试数据集合训练数据集, 并对数据进行格式化处理

Parameters:
    无
Returns:
    errorRate - 分类错误率
"""
def colicTest():
    frTrain = open('horseColicTraining.txt')
    frTest = open('horseColicTest.txt')
    trainingSet = []; trainingLabels = []
    for line in frTrain.readlines():
        currLine = line.strip().split('\t')
        lineArr = []
        for i in range(21):
            lineArr.append(float(currLine[i]))
        trainingSet.append(lineArr)
        trainingLabels.append(float(currLine[21]))
    #使用改进后的随机梯度下降算法求得在此数据集上的最佳回归系数 trainWeights
    #trainWeights1 = gradAscend(np.array(trainingSet), trainingLabels)
    trainWeights2 = stocGradAscent1(np.array(trainingSet), trainingLabels)
    errorCount = 0
    numTestVec = 0
    #读取测试数据集进行测试,  计算分类错误的样本条数和最终的错误率
    for line in frTest.readlines():
        #每一行作为一个样本,样本数加一
        numTestVec += 1
        currLine = line.strip().split('\t')
        lineArr = []
        for i in range(21):
            lineArr.append(float(currLine[i]))
            #trainWeights1[:,0]可以将(21,1)转换成(21,)
        #if int(classifyVector(np.array(lineArr),trainWeights1[:,0])) !=
int(currLine[21]):
        if int(classifyVector(np.array(lineArr),trainWeights2)) !=
int(currLine[21]):
            errorCount += 1
    errorRate = float(errorCount) / numTestVec
    print('错误率为: '+ str(errorRate))
    return errorRate
```

这里注意一点, 就是, 如果调用的是gradAscend进行迭代的话, 下面的if语句需要调整为:

`if int(classifyVector(np.array(lineArr),trainWeights1[:,0])) != int(currLine[21])`

trainWeights1[:,0]可以将(21,1)转换成(21,)

四 调用函数colicTest()10次并求结果的平均值

来一个代码大整合

```
import numpy as np
import random
import matplotlib.pyplot as plt
"""
函数说明: 加载数据

Parameters:
    无
Returns:
    dataMat - 数据列表(list)
    labelMat - 标签列表(list)
"""


def loadDataSet():
    #初始化数据列表和标签列表
    dataMat = []
    labelMat = []
    #打开文件
    with open('testSet.txt') as f:
        #读取文件所有内容形成列表,再对列表进行遍历
        for line in f.readlines():
            #剔除字符串首尾空白,并切割元素,形成列表
            lineArr = line.strip().split()
            #将索引0填充1,索引1填充lineArr[0],索引1填充lineArr[1]
            # dataMat第一列为常数项,第二列为x1(即x),第二列为x2(即y)
            dataMat.append([1, float(lineArr[0]), float(lineArr[1])])
            #用lineArr[2]填充labelMat
            labelMat.append(int(lineArr[2]))
    return dataMat, labelMat

"""
函数说明: sigmoid函数

Parameters:
    inX - 输入数据(数字或者np.ndarray)
Returns:
    sigmoid函数(数字或者np.ndarray)
"""
def sigmoid(inX):
    return 1 / (1+ np.exp(-inX))
```

```
"""
函数说明: 梯度上升法

Parameters:
    dataMatIn - 数据集(是一个2维NumPy数组， 每列分别代表每个不同的特征， 每行则代表每个训练样本)(array_like类型)
    classLabels - 数据标签(是类别标签， 它是一个 1*100 的行向量. 为了便于矩阵计算，
                  需要将该行向量转换为列向量， 做法是将原向量转置， 再将它赋值给 labelMat
                  (array_like类型)
Returns:
    np.array(weights) - 返回权重数组(即最优的参数)(np.ndarray类型)
"""
def gradAscend(dataMatIn, classLabels):
    #转换成矩阵
    dataMatrix = np.mat(dataMatIn)
    #转换成矩阵, 并进行转置
    #转化为矩阵[[0,1,0,1,0,1.....]], 并转置[[0],[1],[0].....]
    #也就是首先将数组转换为 NumPy 矩阵， 然后再将行向量转置为列向量
    labelMat = np.mat(classLabels).transpose()
    #返回dataMatrix的大小, 其中m,n分别为行数和列数
    #也就是m个数据量,即样本数, n个特征
    m, n = np.shape(dataMatrix)
    #设置步长,也就是学习率
    alpha = 0.001
    #设置最大迭代次数
    maxCycles = 500
    #生成一个长度和特征数相同的矩阵， 此处n为3 -> [[1],[1],[1]]
    #weights 代表回归系数， 此处的 ones((n,1)) 创建一个长度和特征数相同的矩阵， 其中的数全部都是 1
    weights = np.ones((n,1))
    for k in range(maxCycles):
        #m*n 的矩阵 * n*1 的矩阵 = m*1的矩阵
        #这个乘法的结果就是通过公式计算得到的理论值(此时还不是label,因为label是0或者1,通过sigmoid函数后,h就是理论label)
        h = sigmoid(dataMatrix*weights)
        #求出错误矩阵
        #labelmat是实际值
        error = (labelMat-h)
        #关于梯度上升中梯度的求解,以及为何矩阵乘积是下面的形式,有详细参考.
        weights = weights + alpha * dataMatrix.transpose() * error
    return np.array(weights)




"""
函数说明: 随机梯度上升法

Parameters:
    dataMatrix - 数据集(是一个2维NumPy数组， 每列分别代表每个不同的特征， 每行则代表每个训练样本)(需输入np.ndarray)
    classLabels - 数据标签(是类别标签， 它是一个 1*100 的行向量. 为了便于矩阵计算，
                  需要将该行向量转换为列向量， 做法是将原向量转置， 再将它赋值给 labelMat
                  (array_like)
Returns:
```

```
    weights- 返回权重(即最优的参数)(np.ndarray)
"""

def stocGradAscent0(dataMatrix, classLabels):
    #返回dataMatrix的大小, 其中m,n分别为行数和列数
    #也就是m个数据量,即样本数, n个特征
    m,n = np.shape(dataMatrix)
    #初始化alpha的值
    alpha = 0.01
    #函数ones创建一个全1的数组
    #初始化长度为n的数组,  元素全部为 1
    weights = np.ones(n)
    #遍历所有样本
    for i in range(m):
        #sum(dataMatrix[i]*weights)为了求 f(x)的值,   f(x)=a1*x1+b2*x2+..+nn*xn,
        #此处求出的 h 是一个具体的数值,  而不是一个矩阵
        h = sigmoid(sum(dataMatrix[i] * weights))
        #计算真实类别与预测类别之间的差值,  然后按照该差值调整回归系数
        error = classLabels[i] - h
        weights = weights + alpha * error*dataMatrix[i]
    return weights




"""
函数说明: 随机梯度上升法(改进)

Parameters:
    dataMatrix - 数据集(是一个2维NumPy数组,  每列分别代表每个不同的特征,  每行则代表每个训
练样本)(需输入np.ndarray)
    classLabels - 数据标签(是类别标签,  它是一个 1*100 的行向量.  为了便于矩阵计算,
                    需要将该行向量转换为列向量,  做法是将原向量转置,  再将它赋值给
labelMat
                    (array_like)
    numIter - 迭代次数(整数)
Returns:
    weights- 返回权重(即最优的参数)(np.ndarray)
"""

def stocGradAscent1(dataMatrix, classLabels, numIter = 150):
    # 返回dataMatrix的大小, 其中m,n分别为行数和列数
    # 也就是m个数据量,即样本数, n个特征
    m, n = np.shape(dataMatrix)
    # 函数ones创建一个全1的数组
    # 初始化长度为n的数组,  元素全部为 1
    weights = np.ones(n)
    #随机梯度, 循环150次, 观察是否收敛
    for j in range(numIter):
        dataIndex = list(range(m))
        for i in range(m):
            #i和j的不断增大,  导致alpha的值不断减少,  但是不为0
            alpha = 4 / (1+j+i) + 0.0001
            #随机产生一个 0～len()之间的一个整数值
            randIndex = int(random.uniform(0,len(dataIndex)))
            # sum(dataMatrix[i]*weights)为了求 f(x)的值,
f(x)=a1*x1+b2*x2+..+nn*xn,
            # 此处求出的 h 是一个具体的数值,  而不是一个矩阵
```

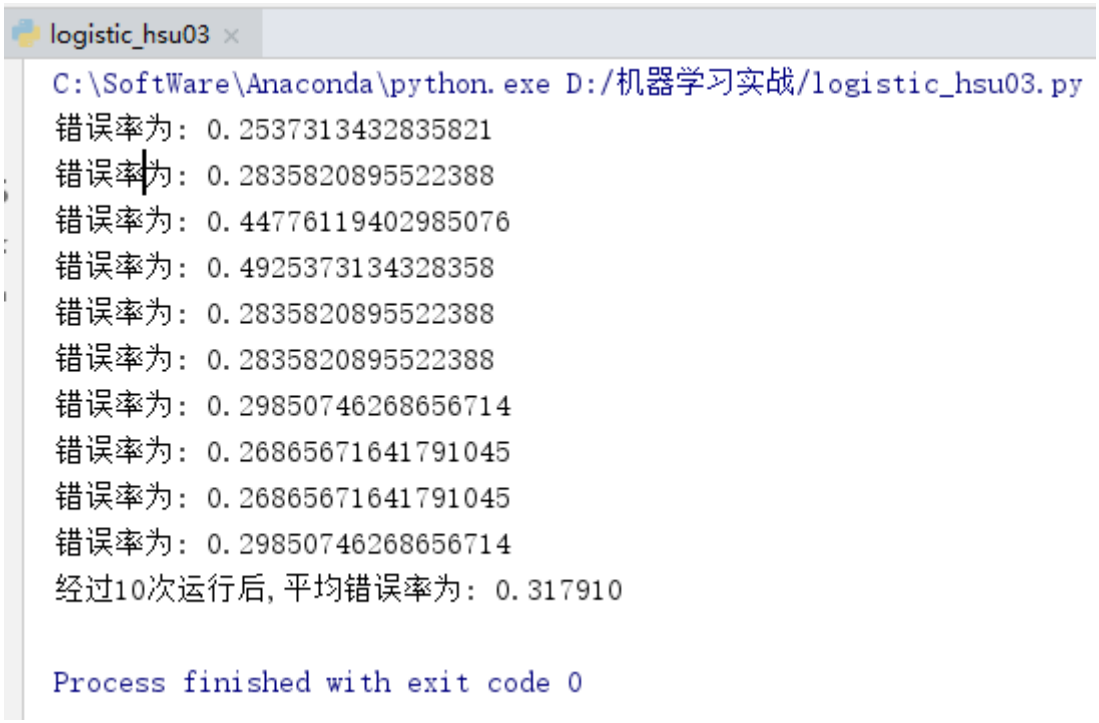
```
            h = sigmoid(sum(dataMatrix[dataIndex[randIndex]]*weights))
            error = classLabels[dataIndex[randIndex]] - h
            weights = weights + alpha * error * dataMatrix[dataIndex[randIndex]]
            del(dataIndex[randIndex])
    return weights


"""
函数说明: 最终的分类函数,根据回归系数和特征向量来计算sigmoid的值, 大于0.5返回1,否则返回0

Parameters:
    inX - 特征向量,features (np.ndarray)
    weights - 根据梯度下降或者随机梯度下降, 计算的回归系数(np.ndarray)
Returns:
    返回0或者1
"""
def classifyVector(inX, weights):
    prob = sigmoid(sum(inX*weights))
    if prob > 0.5:
        return 1
    else:
        return 0



"""
函数说明: 打开测试数据集合训练数据集, 并对数据进行格式化处理

Parameters:
    无
Returns:
    errorRate - 分类错误率
"""
def colicTest():
    frTrain = open('horseColicTraining.txt')
    frTest = open('horseColicTest.txt')
    trainingSet = []; trainingLabels = []
    for line in frTrain.readlines():
        currLine = line.strip().split('\t')
        lineArr = []
        for i in range(21):
            lineArr.append(float(currLine[i]))
        trainingSet.append(lineArr)
        trainingLabels.append(float(currLine[21]))
    #使用改进后的随机梯度下降算法求得在此数据集上的最佳回归系数 trainWeights
    trainWeights1 = gradAscend(trainingSet,trainingLabels)
    #trainWeights2 = stocGradAscent1(np.array(trainingSet), trainingLabels,500)
    errorCount = 0
    numTestVec = 0
    #读取测试数据集进行测试,  计算分类错误的样本条数和最终的错误率
    for line in frTest.readlines():
        #每一行作为一个样本,样本数加一
        numTestVec += 1
        currLine = line.strip().split('\t')
        lineArr = []
        for i in range(21):
            lineArr.append(float(currLine[i]))
        #trainWeights1[:,0]可以将(21,1)转换成(21,)
```

```
        if int(classifyVector(np.array(lineArr), trainWeights1[:,0])) !=
int(currLine[21]):
        #if int(classifyVector(np.array(lineArr),trainWeights2)) !=
int(currLine[21]):
            errorCount += 1
    errorRate = float(errorCount) / numTestVec
    print('错误率为: '+ str(errorRate))
    return errorRate

"""
函数说明:调用 colicTest() 10次并求结果的平均值

Parameters:
    无
returns:
    无
"""
def multiTest():
    numTests = 10
    errorSum = 0.0
    for k in range(numTests):
        errorSum += colicTest()
    print ("经过%d次运行后,平均错误率为: %f" % (numTests, errorSum/float(numTests))
)

if __name__ == '__main__':
    multiTest()
```

使用gradAscend梯度上升, 运行的结果为:

```
C:\SoftWare\Anaconda\python.exe D:/机器学习实战/logistic_hsu03.py
错误率为: 0.2835820895522388
错误率为: 0.2835820895522388
错误率为: 0.2835820895522388
错误率为: 0.2835820895522388
错误率为: 0.2835820895522388
错误率为: 0.2835820895522388
错误率为: 0.2835820895522388
错误率为: 0.2835820895522388
错误率为: 0.2835820895522388
错误率为: 0.2835820895522388
经过10次运行后, 平均错误率为: 0.283582
```

使用stocGradAscent1改进的随机梯度上升, 运行的结果为:



这一章书上的东西终于弄完了，下一篇继续使用sklearn来看看如何进行本章的logistic分类.