《机器学习实战》学习笔记（五）：Logistic 回归

原创 我是管小亮 2019-09-01 10:41:14 2296 收藏 15 版权

分类专栏：Machine Learning♡ 文章标签：机器学习 机器学习实战 学习笔记 逻辑回归 Logistic 回归

欢迎关注WX公众号：【程序员管小亮】

【机器学习】《机器学习实战》读书笔记及代码 总目录

- https://blog.csdn.net/TeFuirnever/article/details/99701256

GitHub代码地址：

- https://github.com/TeFuirnever/Machine-Learning-in-Action

目录

本章内容

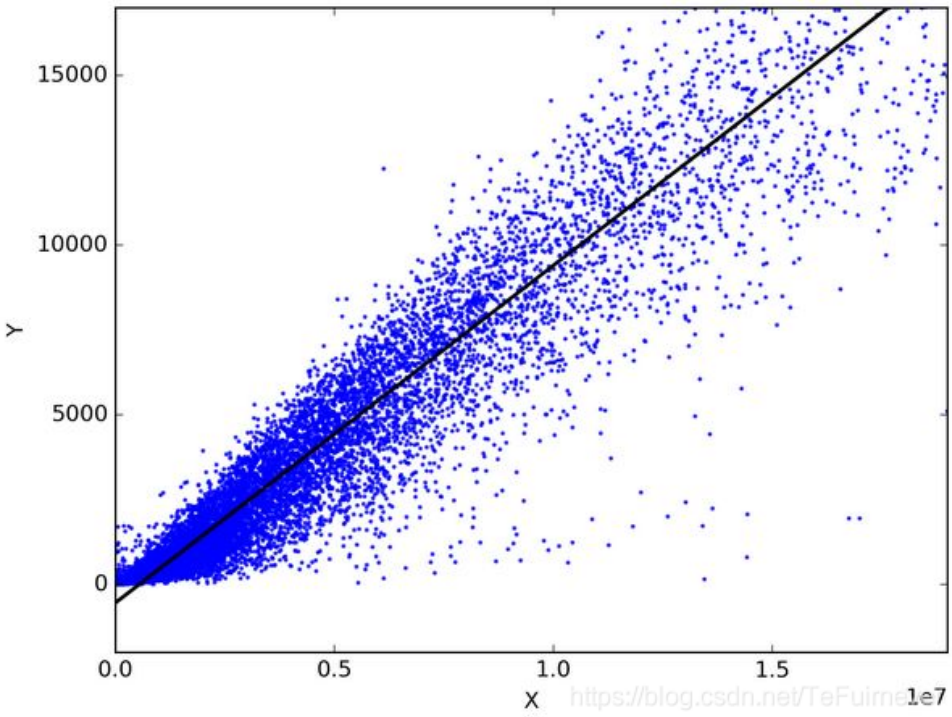
- Sigmoid函数和Logistic回归分类器
- 最优化理论初步
- 梯度下降最优化算法
- 数据中的缺失项处理

这会是激动人心的一章，因为会首次接触到 **最优化算法**。仔细想想就会发现，其实日常生活中遇到过很多最优化问题，比如如何在最短时间内从A点到达B点？如何投入最少工作量却获得最大的效益？如何设计发动机使得油耗最少而功率最大？可见，最优化的作用十分强大。接下来会介绍几个最优化算法，并利用它们训练出一个非线性函数用于分类。

- 《机器学习》周志华西瓜书学习笔记（三）：线性模型
- 《机器学习》周志华西瓜书习题参考答案：第3章 - 线性模型

1、Logistic 回归

假设现在有一些数据点，用一条直线对这些点进行拟合（该线称为 **最佳拟合直线**），这个拟合过程就称作 **回归**。



利用 **Logistic回归** 进行分类的主要思想是：**根据现有数据对分类边界线建立回归公式，以此进行分类。**这里的“**回归**”一词源于最佳拟合，表示要找到 **最佳拟合参数集**，其背后的数学分析将在下一部分介绍。训练分类器时的做法就是寻找最佳拟合参数，使用的是最优化算法。

> Logistic回归的一般过程
> (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。

Logistic回归的因变量可以是二分类的，也可以是多分类的，但是实际中最为常用的就是二分类的Logistic回归。它利用的是Sigmoid函数阈值在[0,1]这个特性。Logistic回归进行分类的主要思想是：根据现有数据对分类边界线建立回归公式，以此进行分类。其实，Logistic本质上是一个基于条件概率的判别模型(Discriminative Model)。

> Logistic回归
> 优点：计算代价不高，易于理解和实现。

> 缺点：容易欠拟合，分类精度可能不高。
> 适用数据类型：数值型和标称型数据。

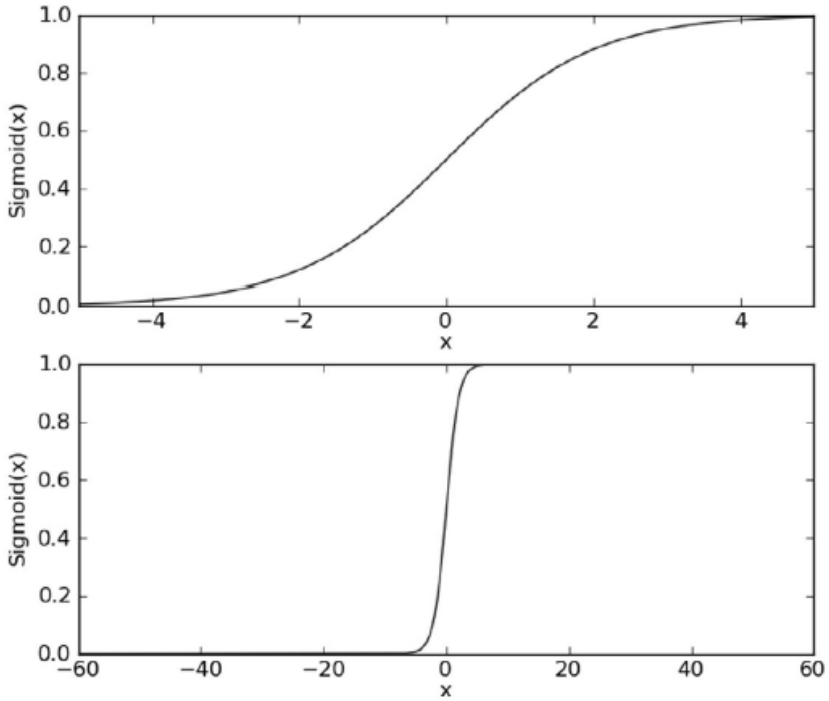
2、基于Logistic 回归和Sigmoid 函数的分类

为什么利用的是Sigmoid函数呢?

首先我们想要的函数应该是，能接受所有的输入然后预测出类别。例如，在两个类的情况下，上述函数输出0或1。或许你之前接触过具有这种性质的函数，该函数称为 **海维塞德阶跃函数（Heaviside step function）**，或者直接称为 **单位阶跃函数**。然而，海维塞德阶跃函数的问题在于：该函数在跳跃点上从0瞬间跳跃到1，这个瞬间跳跃过程有时很难处理。

幸好，另一个函数也有类似的性质（可以输出0或者1），且数学上更易处理，这就是Sigmoid函数。Sigmoid函数具体的计算公式如下：

$$\sigma(z) = \frac{1}{1+e^{-z}}$$



两种坐标尺度下的Sigmoid函数图。上图的横坐标为–5到5，这时的曲线变化较为平滑；下图横坐标的尺度足够大，可以看到，在x = 0点处Sigmoid函数看起来很像阶跃函数

如图中上图给出了Sigmoid函数在不同坐标尺度下的两条曲线图。当x为0时，Sigmoid函数值为0.5。随着x的增大，对应的Sigmoid值将逼近于1；而随着x的减小，Sigmoid值将逼近于0。如果横坐标刻度足够大（图中下图），Sigmoid函数看起来很像一个阶跃函数。

因此，为了实现Logistic回归分类器，我们可以在每个特征上都乘以一个回归系数，然后把所有的结果值相加，将这个总和代入Sigmoid函数中，进而得到一个范围在0~1之间的数值。任何大于0.5的数据被分入1类，小于0.5即被归入0类。所以，Logistic回归也可以被看成是一种概率估计。确定了分类器的函数形式之后，那么现在的问题变成了：最佳回归系数是多少？ 如何确定它们的大小?

3、梯度上升算法

Sigmoid函数的输入记为z，由下面公式得出：

$$z = w_0 x_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$$

如果采用向量的写法，上述公式可以写成 $z = w^T x$，它表示将这两个数值向量对应元素相乘然后全部加起来即得到z值。其中的向量x是分类器的输入数据，向量w也就是我们要找到的最佳参数（系数），从而使得分类器尽可能地精确。为了寻找该最佳参数，需要用到最优化理论的一些知识。本文使用梯度上升算法进行求解。

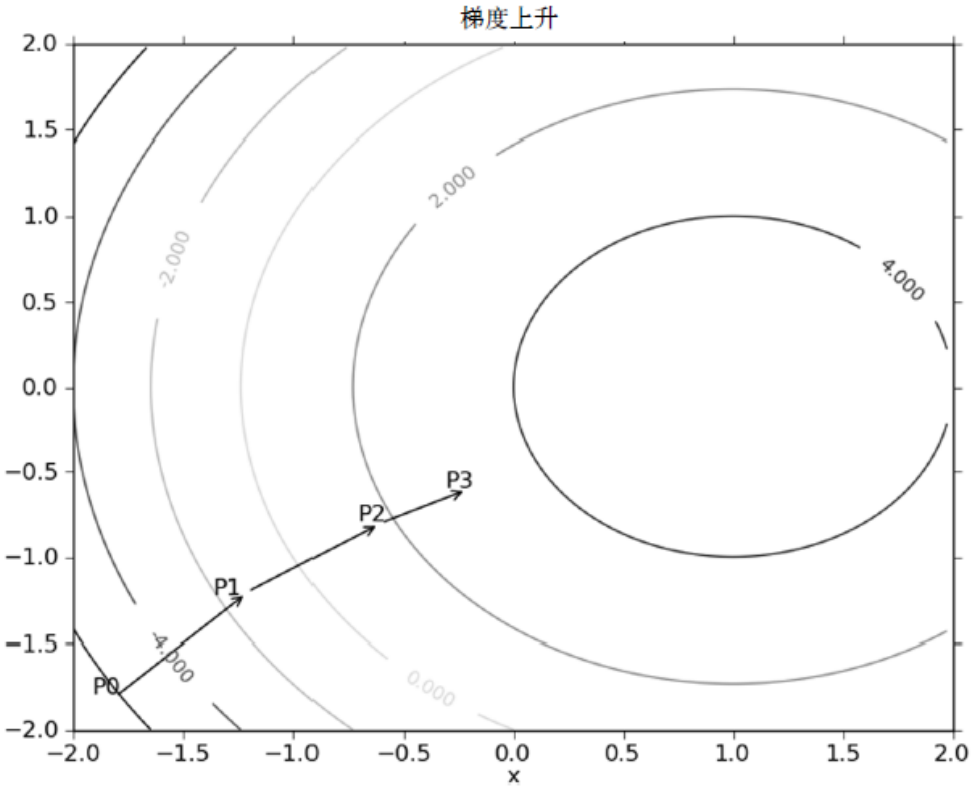
那么什么是梯度上升算法？梯度上升法基于的思想是：要找到某函数的最大值，最好的方法是沿着该函数的梯度方向探寻。如果梯度记为∇，则函数f(x,y)的梯度由下式表示：

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

这是机器学习中最易造成混淆的一个地方，但在数学上并不难，需要做的只是牢记这些符号的意义。这个梯度

$$\frac{\partial f(x,y)}{\partial x} \qquad \frac{\partial f(x,y)}{\partial y}$$

意味着要沿x的方向移动 ，沿y的方向移动 。其中，函数f (x,y)必须要在待计算的点上有定义并且可微。一个具体的函数例子见下图：



梯度上升算法到达每个点后都会重新估计移动的方向。从P0开始，计算完该点的梯度，函数就根据梯度移动到下一点P1。在P1点，梯度再次被重新计算，并沿新的梯度方向移动到P2。如此循环迭代，直到满足停止条件。迭代的过程中，梯度算子总是保证我们能选取到最佳的移动方向

上图中的梯度上升算法沿梯度方向移动了一步。可以看到，梯度算子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算法达到某个可以允许的误差范围。

> 梯度下降算法
> 你最经常听到的应该是梯度下降算法，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。

4、基于最优化方法的最佳回归系数确定

1）查看数据的分布情况

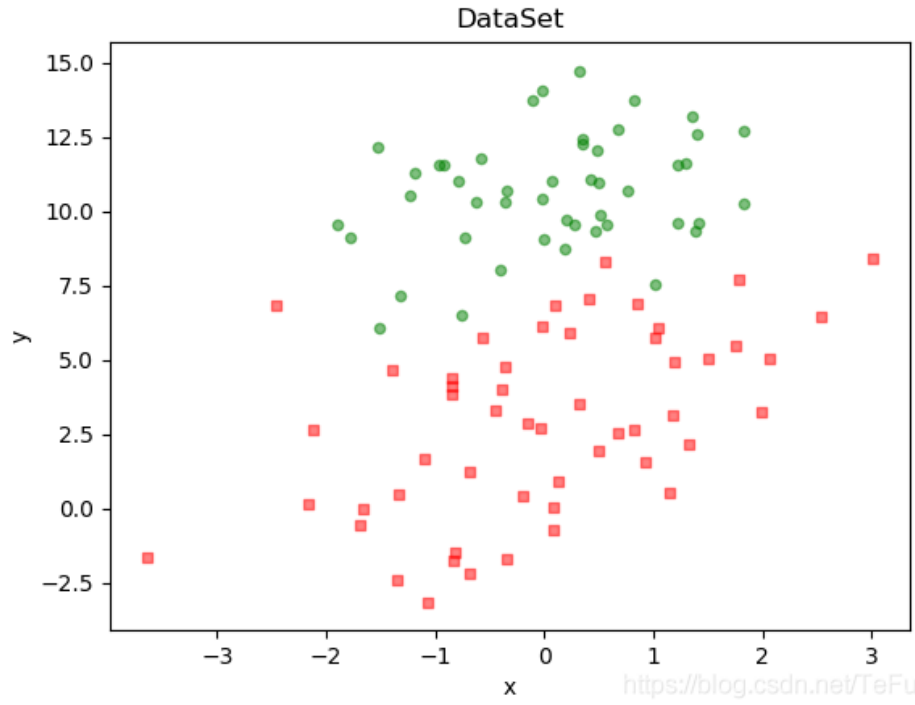
这是一个简单的没什么实际含义的数据集，先看一些具体的数据是怎么样的：

1	-0.017612	14.053064	0
2	-1.395634	4.662541	1
3	-0.752157	6.538620	0
4	-1.322371	7.152853	0
5	0.423363	11.054677	0
6	0.406704	7.067335	1
7	0.667394	12.741452	0
8	-2.460150	6.866805	1
9	0.569411	9.548755	0
10	-0.026632	10.427743	0
11	0.850433	6.920334	1
12	1.347183	13.175500	0
13	1.176813	3.167020	1
14	-1.781871	9.097953	0
15	-0.566606	5.749003	1
16	0.931635	1.589505	1
17	-0.024205	6.151823	1
18	-0.036453	2.690988	1
19	-0.196949	0.444165	1
20	1.014459	5.754399	1
21	1.985298	3.230619	1
22	-1.693453	-0.557540	1

这个数据有两维特征，因此可以在一个二维平面上展示。首先将第一列数据(X1)看作x轴上的值，然后第二列数据(X2)看作y轴上的值，最后把最后一列数据即为分类标签。根据标签的不同，对这些点进行分类。

```
import matplotlib.pyplot as plt
import numpy as np

"""
Parameters:
    无
Returns:
    dataMat - 数据列表
    labelMat - 标签列表
"""
# 函数说明:加载数据
def loadDataSet():
    dataMat = []                                                #创建数据列表
    labelMat = []                                               #创建标签列表
    fr = open('testSet.txt')                                    #打开文件
    for line in fr.readlines():                                 #逐行读取
        lineArr = line.strip().split()                          #去回车，放入列表
        dataMat.append([1.0, float(lineArr[0]), float(lineArr[1])])  #添加数据
        labelMat.append(int(lineArr[2]))                        #添加标签
    fr.close()                                                  #关闭文件

```

```
    return dataMat, labelMat                                    #返回

# 函数说明:绘制数据集
def plotDataSet():
    dataMat, labelMat = loadDataSet()                           #加载数据集
    dataArr = np.array(dataMat)                                 #转换成numpy的array数组
    n = np.shape(dataMat)[0]                                    #数据个数
    xcord1 = []; ycord1 = []                                    #正样本
    xcord2 = []; ycord2 = []                                    #负样本
    for i in range(n):                                          #根据数据集标签进行分类
        if int(labelMat[i]) == 1:
            xcord1.append(dataArr[i,1]); ycord1.append(dataArr[i,2])    #1为正样本
        else:
            xcord2.append(dataArr[i,1]); ycord2.append(dataArr[i,2])    #0为负样本
    fig = plt.figure()
    ax = fig.add_subplot(111)                                   #添加subplot
    ax.scatter(xcord1, ycord1, s = 20, c = 'red', marker = 's',alpha=.5)#绘制正样本
    ax.scatter(xcord2, ycord2, s = 20, c = 'green',alpha=.5)    #绘制负样本
    plt.title('DataSet')                                        #绘制title
    plt.xlabel('x'); plt.ylabel('y')                            #绘制label
    plt.show()                                                  #显示


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



从上图可以看出我们采用的数据的分布情况。

2）训练算法：使用梯度上升找到最佳参数

数据中有100个样本点，每个点包含两个数值型特征：X1和X2。在此数据集上，通过使用梯度上升法找到最佳回归系数，也就是拟合出Logistic回归模型的最佳参数。

梯度上升法的伪代码如下：

```
每个回归系数初始化为1
重复R次：
        计算整个数据集的梯度
        使用alpha × gradient更新回归系数的向量
        返回回归系数
```

具体实现代码如下：

```python
import numpy as np

'''
Parameters:
    无
Returns:
    dataMat - 数据列表
    labelMat - 标签列表
'''
# 函数说明:加载数据
def loadDataSet():
    dataMat = []                                                #创建数据列表
    labelMat = []                                               #创建标签列表
    fr = open('testSet.txt')                                    #打开文件
    for line in fr.readlines():                                 #逐行读取
        lineArr = line.strip().split()                          #去回车，放入列表
        dataMat.append([1.0, float(lineArr[0]), float(lineArr[1])]) #添加数据
        labelMat.append(int(lineArr[2]))                        #添加标签
    fr.close()                                                  #关闭文件
    return dataMat, labelMat                                    #返回

'''
Parameters:
    inX - 数据
Returns:
    sigmoid函数
'''
# 函数说明:sigmoid函数
def sigmoid(inX):
    return 1.0 / (1 + np.exp(-inX))

'''
Parameters:
    dataMatIn - 数据集
    classLabels - 数据标签
Returns:
'''
# 函数说明:梯度上升算法
def gradAscent(dataMatIn, classLabels):
    dataMatrix = np.mat(dataMatIn)                              #转换成numpy的mat
    labelMat = np.mat(classLabels).transpose()                  #转换成numpy的mat,并进行转置
    m, n = np.shape(dataMatrix)                                 #返回dataMatrix的大小。m为行数,n为列数
    alpha = 0.001                                               #移动步长,也就是学习速率,控制更新的幅度
    maxCycles = 500                                             #最大迭代次数
    weights = np.ones((n,1))
    for k in range(maxCycles):
        h = sigmoid(dataMatrix * weights)                       #梯度上升矢量化公式
        error = labelMat - h
        weights = weights + alpha * dataMatrix.transpose() * error
    return weights.getA()                                       #将矩阵转换为数组，返回权重数组


```

```
if __name__ == '__main__':
    dataMat, labelMat = loadDataSet()
    print(gradAscent(dataMat, labelMat))
```

```
[[ 4.12414349]
 [ 0.48007329]
 [-0.6168482 ]]
```

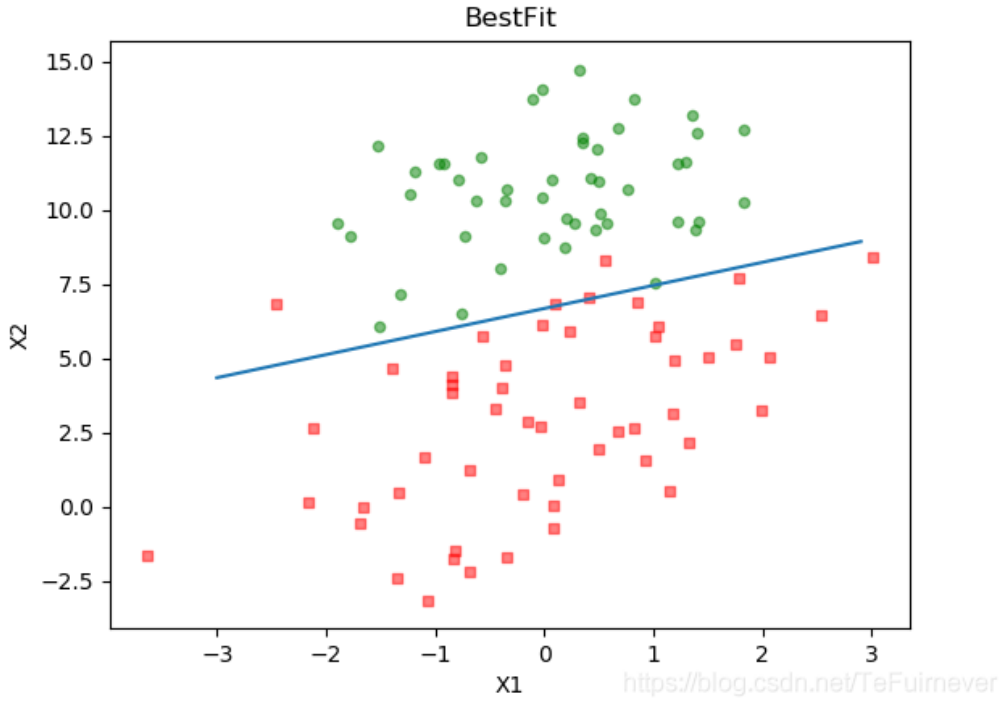
假设Sigmoid函数的输入记为 z，那么 $z = w_0 x_0 + w_1 x_1 + w_2 x_2$，即可将数据分割开。其中，$x_0$ 为全是1的向量，x_1 为数据集的第一列数据，x_2 为数据集的第二列数据。另 $z = 0$，则 $0 = w_0 + w_1 x_1 + w_2 x_2$。横坐标为 x_1，纵坐标为 x_2。这个方程未知的参数为 w_0， w_1， w_2，也就是我们需要求的回归系数(最优参数)。已经求解出的就是回归系数 $[w_0, w_1, w_2]$，通过系数就可以确定不同类别数据之间的分割线，画出决策边界。

3）分析数据：画出决策边界

已经解出了一组回归系数。现在开始绘制这个分隔线：

```
import matplotlib.pyplot as plt
import numpy as np

'''
Parameters:
    无
Returns:
    dataMat - 数据列表
    labelMat - 标签列表
'''
# 函数说明:加载数据
def loadDataSet():
    dataMat = []                                                #创建数据列表
    labelMat = []                                               #创建标签列表
    fr = open('testSet.txt')                                    #打开文件
    for line in fr.readlines():                                 #逐行读取
        lineArr = line.strip().split()                          #去回车，放入列表
        dataMat.append([1.0, float(lineArr[0]), float(lineArr[1])]) #添加数据
        labelMat.append(int(lineArr[2]))                        #添加标签
    fr.close()                                                  #关闭文件
    return dataMat, labelMat                                    #返回

'''
Parameters:
    inX - 数据
Returns:
    sigmoid函数
'''
# 函数说明:sigmoid函数
def sigmoid(inX):
    return 1.0 / (1 + np.exp(-inX))

'''
Parameters:
    dataMatIn - 数据集
    classLabels - 数据标签
Returns:
    weights.getA() - 求得的权重数组(最优参数)
'''
# 函数说明:梯度上升算法
def gradAscent(dataMatIn, classLabels):
```

```
    dataMatrix = np.mat(dataMatIn)                                  #转换成numpy的mat
    labelMat = np.mat(classLabels).transpose()                      #转换成numpy的mat,并进行转置
    m, n = np.shape(dataMatrix)                                     #返回dataMatrix的大小。m为行数,n为列数
    alpha = 0.001                                                   #移动步长,也就是学习速率,控制更新的幅度
    maxCycles = 500                                                 #最大迭代次数
    weights = np.ones((n,1))
    for k in range(maxCycles):
        h = sigmoid(dataMatrix * weights)                           #梯度上升矢量化公式
        error = labelMat - h
        weights = weights + alpha * dataMatrix.transpose() * error
    return weights.getA()                                           #将矩阵转换为数组，返回权重数组

'''
Parameters:
    weights - 权重参数数组
Returns:
    无
'''
# 函数说明:绘制数据集
def plotBestFit(weights):
    dataMat, labelMat = loadDataSet()                               #加载数据集
    dataArr = np.array(dataMat)                                     #转换成numpy的array数组
    n = np.shape(dataMat)[0]                                        #数据个数
    xcord1 = []; ycord1 = []                                        #正样本
    xcord2 = []; ycord2 = []                                        #负样本
    for i in range(n):                                              #根据数据集标签进行分类
        if int(labelMat[i]) == 1:
            xcord1.append(dataArr[i,1]); ycord1.append(dataArr[i,2])    #1为正样本
        else:
            xcord2.append(dataArr[i,1]); ycord2.append(dataArr[i,2])    #0为负样本
    fig = plt.figure()
    ax = fig.add_subplot(111)                                       #添加subplot
    ax.scatter(xcord1, ycord1, s = 20, c = 'red', marker = 's',alpha=.5)#绘制正样本
    ax.scatter(xcord2, ycord2, s = 20, c = 'green',alpha=.5)        #绘制负样本
    x = np.arange(-3.0, 3.0, 0.1)
    y = (-weights[0] - weights[1] * x) / weights[2]
    ax.plot(x, y)
    plt.title('BestFit')                                            #绘制title
    plt.xlabel('X1'); plt.ylabel('X2')                              #绘制label
    plt.show()


if __name__ == '__main__':
    dataMat, labelMat = loadDataSet()
    weights = gradAscent(dataMat, labelMat)
    plotBestFit(weights)
```



这个分类结果相当不错，从上图可以看出，只分错了几个点而已。但是，尽管例子简单切数据集很小，但是这个方法却需要大量的计算(300次乘法)。因此还是需要对算法稍作改进，从而减少计算量，使其可以应用于大数据集上。

4）训练算法：随机梯度上升

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”。

随机梯度上升算法可以写成如下的伪代码：

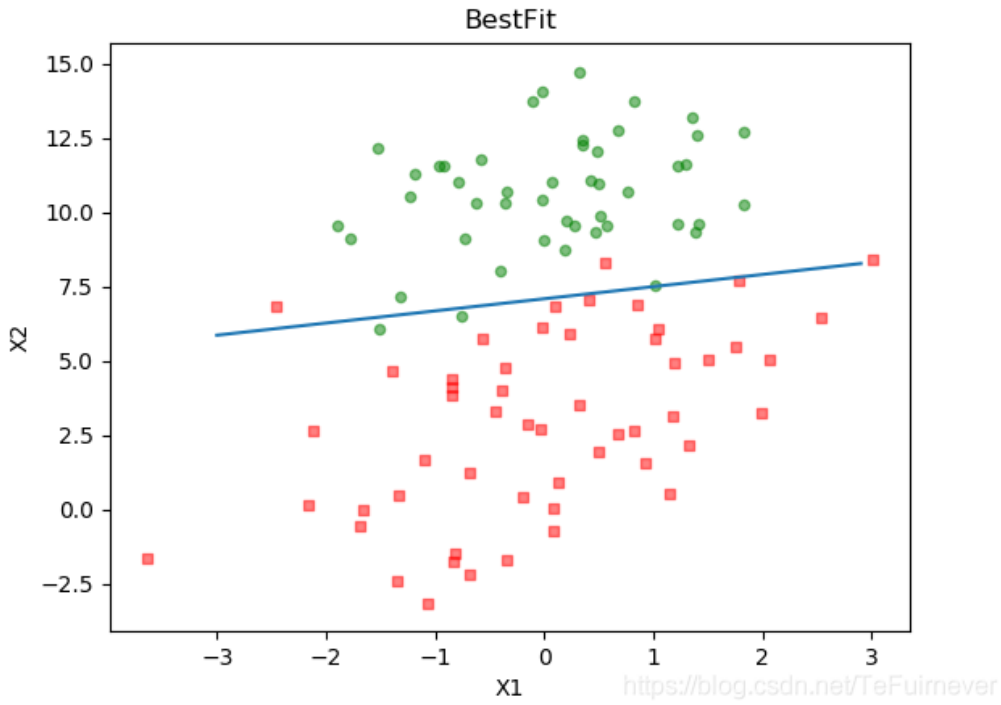
```
所有回归系数初始化为1
对数据集中每个样本
        计算该样本的梯度
        使用alpha × gradient更新回归系数值
返回回归系数值
```

改进算法还增加了一个迭代次数作为第3个参数。如果该参数没有给定的话，算法将默认迭代150次。如果给定，那么算法将按照新的参数值进行迭代。代码如下：

```
from matplotlib.font_manager import FontProperties
import matplotlib.pyplot as plt
import numpy as np
import random

'''
Parameters:
    无
Returns:
    dataMat - 数据列表
    labelMat - 标签列表
```

```
'''
# 函数说明:加载数据
def loadDataSet():
    dataMat = []                                                    #创建数据列表
    labelMat = []                                                   #创建标签列表
    fr = open('testSet.txt')                                        #打开文件
    for line in fr.readlines():                                     #逐行读取
        lineArr = line.strip().split()                              #去回车，放入列表
        dataMat.append([1.0, float(lineArr[0]), float(lineArr[1])]) #添加数据
        labelMat.append(int(lineArr[2]))                            #添加标签
    fr.close()                                                      #关闭文件
    return dataMat, labelMat                                        #返回

'''
Parameters:
    inX - 数据
Returns:
    sigmoid函数
'''
# 函数说明:sigmoid函数
def sigmoid(inX):
    return 1.0 / (1 + np.exp(-inX))

'''
Parameters:
    weights - 权重参数数组
Returns:
    无
'''
# 函数说明:绘制数据集
def plotBestFit(weights):
    dataMat, labelMat = loadDataSet()                               #加载数据集
    dataArr = np.array(dataMat)                                     #转换成numpy的array数组
    n = np.shape(dataMat)[0]                                        #数据个数
    xcord1 = []; ycord1 = []                                        #正样本
    xcord2 = []; ycord2 = []                                        #负样本
    for i in range(n):                                              #根据数据集标签进行分类
        if int(labelMat[i]) == 1:
            xcord1.append(dataArr[i,1]); ycord1.append(dataArr[i,2])    #1为正样本
        else:
            xcord2.append(dataArr[i,1]); ycord2.append(dataArr[i,2])    #0为负样本
    fig = plt.figure()
    ax = fig.add_subplot(111)                                       #添加subplot
    ax.scatter(xcord1, ycord1, s = 20, c = 'red', marker = 's',alpha=.5)#绘制正样本
    ax.scatter(xcord2, ycord2, s = 20, c = 'green',alpha=.5)        #绘制负样本
    x = np.arange(-3.0, 3.0, 0.1)
    y = (-weights[0] - weights[1] * x) / weights[2]
    ax.plot(x, y)
    plt.title('BestFit')                                            #绘制title
    plt.xlabel('X1'); plt.ylabel('X2')                              #绘制label
    plt.show()

'''
Parameters:
    dataMatrix - 数据数组
    classLabels - 数据标签
    numIter - 迭代次数
Returns:
    weights - 求得的回归系数数组(最优参数)
'''
# 函数说明:改进的随机梯度上升算法
def stocGradAscent1(dataMatrix, classLabels, numIter=150):

```

```
    m,n = np.shape(dataMatrix)                                      #返回dataMatrix的大小。m为行数,n为列数
    weights = np.ones(n)                                            #参数初始化
    for j in range(numIter):
        dataIndex = list(range(m))
        for i in range(m):
            alpha = 4/(1.0+j+i)+0.01                                #降低alpha的大小，每次减小1/(j+i)。
            randIndex = int(random.uniform(0,len(dataIndex)))       #随机选取样本
            h = sigmoid(sum(dataMatrix[randIndex]*weights))         #选择随机选取的一个样本，计算h
            error = classLabels[randIndex] - h                      #计算误差
            weights = weights + alpha * error * dataMatrix[randIndex]#更新回归系数
            del(dataIndex[randIndex])                               #删除已经使用的样本
    return weights                                                  #返回


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



因为存在偶然性，所以进行了多次试验，发现改进之前的运行时间大概是 `Running time: 0.019952099999999997 Seconds` ，改进之后的运行时间大概是 `Running time: 0.09281479999999999 Seconds` ，虽然运行时间增加了，但是运算复杂度降低了，小数据集可能看不出来。

5）回归系数与迭代次数的关系

> 转自 https://blog.csdn.net/c406495762/article/details/77851973

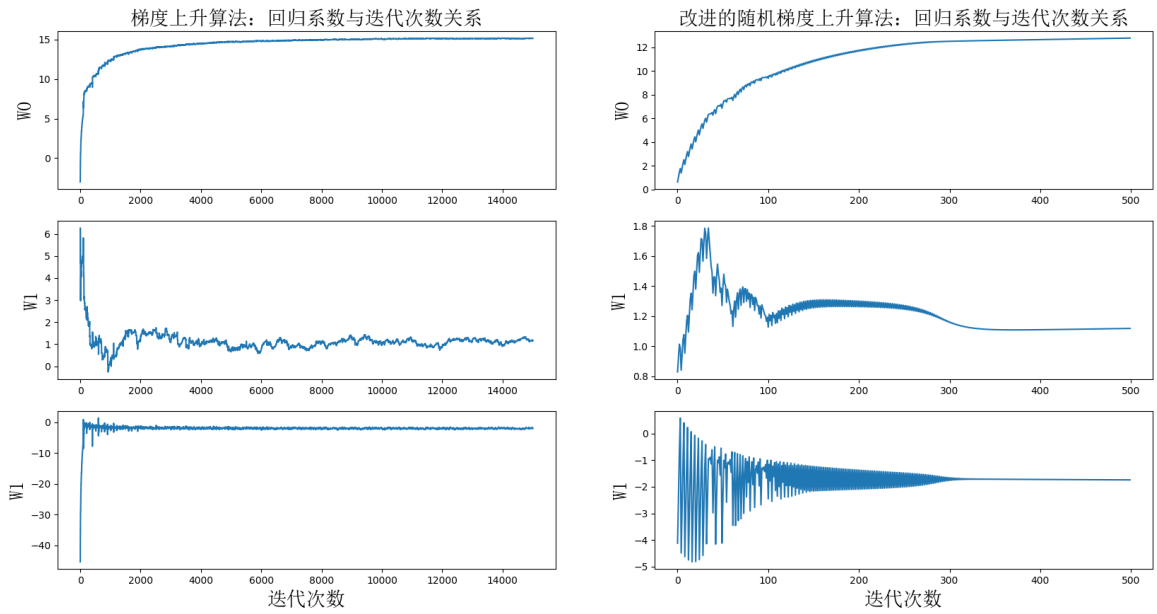
可以看到分类效果也是不错的。不过，从这个分类结果中，不好看出迭代次数和回归系数的关系，也就不能直观的看到每个回归方法的收敛情况。因此，编写程序，绘制出回归系数和迭代次数的关系曲线：

```
from matplotlib.font_manager import FontProperties
import matplotlib.pyplot as plt
```

```python
import numpy as np
import random

'''
Parameters:
    无
Returns:
    dataMat - 数据列表
    labelMat - 标签列表
'''
# 函数说明:加载数据
def loadDataSet():
    dataMat = []                                                #创建数据列表
    labelMat = []                                               #创建标签列表
    fr = open('testSet.txt')                                    #打开文件
    for line in fr.readlines():                                 #逐行读取
        lineArr = line.strip().split()                          #去回车，放入列表
        dataMat.append([1.0, float(lineArr[0]), float(lineArr[1])]) #添加数据
        labelMat.append(int(lineArr[2]))                        #添加标签
    fr.close()                                                  #关闭文件
    return dataMat, labelMat                                    #返回

'''
Parameters:
    inX - 数据
Returns:
    sigmoid函数
'''
# 函数说明:sigmoid函数
def sigmoid(inX):
    return 1.0 / (1 + np.exp(-inX))

'''
Parameters:
    dataMatIn - 数据集
    classLabels - 数据标签
Returns:
'''
# 函数说明:梯度上升算法
def gradAscent(dataMatIn, classLabels):
    dataMatrix = np.mat(dataMatIn)                              #转换成numpy的mat
    labelMat = np.mat(classLabels).transpose()                  #转换成numpy的mat,并进行转置
    m, n = np.shape(dataMatrix)                                 #返回dataMatrix的大小。m为行数,n为列数
    alpha = 0.001                                               #移动步长,也就是学习速率,控制更新的幅度
    maxCycles = 500                                             #最大迭代次数
    weights = np.ones((n,1))
    for k in range(maxCycles):
        h = sigmoid(dataMatrix * weights)                       #梯度上升矢量化公式
        error = labelMat - h
        weights = weights + alpha * dataMatrix.transpose() * error
    return weights.getA()                                       #将矩阵转换为数组，返回权重数组

'''
Parameters:
    dataMatrix - 数据数组
    classLabels - 数据标签
    numIter - 迭代次数
Returns:
    weights - 求得的回归系数数组(最优参数)
'''
# 函数说明:改进的随机梯度上升算法
def stocGradAscent1(dataMatrix, classLabels, numIter=150):
    m,n = np.shape(dataMatrix)                                  #返回dataMatrix的大小。m为行数,n为列
    weights = np.ones(n)                                        #参数初始化
```

```
    for j in range(numIter):
        dataIndex = list(range(m))
        for i in range(m):
            alpha = 4/(1.0+j+i)+0.01                                    #降低alpha的大小，每次减小1/(j+i)。
            randIndex = int(random.uniform(0,len(dataIndex)))           #随机选取样本
            h = sigmoid(sum(dataMatrix[randIndex]*weights))             #选择随机选取的一个样本，计算h
            error = classLabels[randIndex] - h                          #计算误差
            weights = weights + alpha * error * dataMatrix[randIndex]#更新回归系数
            del(dataIndex[randIndex])                                   #删除已经使用的样本
    return weights                                                      #返回

'''
Parameters:
    weights_array1 - 回归系数数组1
    weights_array2 - 回归系数数组2
Returns:
    无
'''
# 函数说明:绘制回归系数与迭代次数的关系
def plotWeights(weights_array1,weights_array2):
    #设置汉字格式
    font = FontProperties(fname=r"c:\windows\fonts\simsun.ttc", size=14)
    #将fig画布分隔成1行1列,不共享x轴和y轴,fig画布的大小为(13,8)
    #当nrow=3,nclos=2时,代表fig画布被分为六个区域,axs[0][0]表示第一行第一列
    fig, axs = plt.subplots(nrows=3, ncols=2,sharex=False, sharey=False, figsize=(20,10))
    x1 = np.arange(0, len(weights_array1), 1)
    #绘制w0与迭代次数的关系
    axs[0][0].plot(x1,weights_array1[:,0])
    axs0_title_text = axs[0][0].set_title(u'梯度上升算法：回归系数与迭代次数关系',FontProperties=font)
    axs0_ylabel_text = axs[0][0].set_ylabel(u'W0',FontProperties=font)
    plt.setp(axs0_title_text, size=20, weight='bold', color='black')
    plt.setp(axs0_ylabel_text, size=20, weight='bold', color='black')
    #绘制w1与迭代次数的关系
    axs[1][0].plot(x1,weights_array1[:,1])
    axs1_ylabel_text = axs[1][0].set_ylabel(u'W1',FontProperties=font)
    plt.setp(axs1_ylabel_text, size=20, weight='bold', color='black')
    #绘制w2与迭代次数的关系
    axs[2][0].plot(x1,weights_array1[:,2])
    axs2_xlabel_text = axs[2][0].set_xlabel(u'迭代次数',FontProperties=font)
    axs2_ylabel_text = axs[2][0].set_ylabel(u'W1',FontProperties=font)
    plt.setp(axs2_xlabel_text, size=20, weight='bold', color='black')
    plt.setp(axs2_ylabel_text, size=20, weight='bold', color='black')



    x2 = np.arange(0, len(weights_array2), 1)
    #绘制w0与迭代次数的关系
    axs[0][1].plot(x2,weights_array2[:,0])
    axs0_title_text = axs[0][1].set_title(u'改进的随机梯度上升算法：回归系数与迭代次数关系',FontProperties=f
    axs0_ylabel_text = axs[0][1].set_ylabel(u'W0',FontProperties=font)
    plt.setp(axs0_title_text, size=20, weight='bold', color='black')
    plt.setp(axs0_ylabel_text, size=20, weight='bold', color='black')
    #绘制w1与迭代次数的关系
    axs[1][1].plot(x2,weights_array2[:,1])
    axs1_ylabel_text = axs[1][1].set_ylabel(u'W1',FontProperties=font)
    plt.setp(axs1_ylabel_text, size=20, weight='bold', color='black')
    #绘制w2与迭代次数的关系
    axs[2][1].plot(x2,weights_array2[:,2])
    axs2_xlabel_text = axs[2][1].set_xlabel(u'迭代次数',FontProperties=font)
    axs2_ylabel_text = axs[2][1].set_ylabel(u'W1',FontProperties=font)
    plt.setp(axs2_xlabel_text, size=20, weight='bold', color='black')
```

```
    plt.setp(axs2_ylabel_text, size=20, weight='bold', color='black')

    plt.show()


if __name__ == '__main__':
    dataMat, labelMat = loadDataSet()
    weights1,weights_array1 = stocGradAscent1(np.array(dataMat), labelMat)

    weights2,weights_array2 = gradAscent(dataMat, labelMat)
    plotWeights(weights_array1, weights_array2)
```



由于改进的随机梯度上升算法，随机选取样本点，所以每次的运行结果是不同的。但是大体趋势是一样的。改进的随机梯度上升算法收敛效果更好。为什么这么说呢?

让我们分析一下。一共有100个样本点，改进的随机梯度上升算法迭代次数为150。而上图显示15000次迭代次数的原因是，使用一次样本就更新一下回归系数。因此，迭代150次，相当于更新回归系数150*100=15000次。简而言之，迭代150次，更新1.5万次回归参数。从上图左侧的改进随机梯度上升算法回归效果中可以看出，其实在更新2000次回归系数的时候，已经收敛了。相当于遍历整个数据集20次的时候，回归系数已收敛。训练已完成。

再让我们看看上图右侧的梯度上升算法回归效果，梯度上升算法每次更新回归系数都要遍历整个数据集。从图中可以看出，当迭代次数为300多次的时候，回归系数才收敛。凑个整，就当它在遍历整个数据集300次的时候已经收敛好了。

没有对比就没有伤害，改进的随机梯度上升算法，在遍历数据集的第20次开始收敛。而梯度上升算法，在遍历数据集的第300次才开始收敛。想像一下，大量数据的情况下，谁效果更好呢？不言而喻，但是程序的时间我们也做了对比，没有免费的午餐。

5、示例：从疝气病症预测病马的死亡率

本节将使用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。我并非育马专家，从一些文献中了解到，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。

> 示例：使用Logistic回归估计马疝病的死亡率
> (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。

另外需要说明的是，除了部分指标主观和难以测量外，该数据还存在一个问题，数据集中有30%的值是缺失的。下面将首先介绍如何处理数据集中的数据缺失问题，然后再利用Logistic回归和随机梯度上升算法来预测病马的生死。

1）准备数据：处理数据中的缺失值

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。那么，数据缺失究竟带来了什么问题？假设有100个样本和20个特征，这些数据都是机器收集回来的。若机器上的某个传感器损坏导致一个特征无效时该怎么办？此时是否要扔掉整个数据？这种情况下，另外19个特征怎么办？它们是否还可用？答案是肯定的。因为有时候数据相当昂贵，扔掉和重新获取都是不可取的，所以必须采用一些方法来解决这个问题。

下面给出了一些可选的做法：

- 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。

预处理数据做两件事：

- 如果测试集中一条数据的特征值已经确实，那么我们选择实数0来替换所有缺失值，因为本文使用Logistic回归。因此这样做不会影响回归系数的值。sigmoid(0)=0.5，即它对结果的预测不具有任何倾向性。
- 如果测试集中一条数据的类别标签已经缺失，那么我们将该类别数据丢弃，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。

2）测试算法：用Logistic 回归进行分类

在使用Sklearn构建Logistic回归分类器之前，我们先用自己写的改进的随机梯度上升算法进行预测，先热热身。使用Logistic
回归方法进行分类并不需要做很多工作，所需做的只是把测试集上每个特征向量乘以最优化方法得来的回归系数，再将该乘积结果求和，最后输入到Sigmoid函数中即可。如果对应的Sigmoid值大于0.5就预测类别标签为1，否则为0。

```
import numpy as np
import random

'''
Parameters:
    inX - 数据
Returns:

```

```
    sigmoid函数
'''
# 函数说明:sigmoid函数
def sigmoid(inX):
    return 1.0 / (1 + np.exp(-inX))

'''
Parameters:
    dataMatrix - 数据数组
    classLabels - 数据标签
    numIter - 迭代次数
Returns:
    weights - 求得的回归系数数组(最优参数)
'''
# 函数说明:改进的随机梯度上升算法
def stocGradAscent1(dataMatrix, classLabels, numIter=150):
    m,n = np.shape(dataMatrix)                                          #返回dataMatrix的大小。m为行数,n为
    weights = np.ones(n)                                                #参数初始化
    #存储每次更新的回归系数
    for j in range(numIter):
        dataIndex = list(range(m))
        for i in range(m):
            alpha = 4/(1.0+j+i)+0.01                                    #降低alpha的大小，每次减小1/(j+i)。
            randIndex = int(random.uniform(0,len(dataIndex)))           #随机选取样本
            h = sigmoid(sum(dataMatrix[randIndex]*weights))             #选择随机选取的一个样本，计算h
            error = classLabels[randIndex] - h                          #计算误差
            weights = weights + alpha * error * dataMatrix[randIndex] #更新回归系数
            del(dataIndex[randIndex])                                   #删除已经使用的样本
    return weights                                                      #返回

# 函数说明:使用Python写的Logistic分类器做预测
def colicTest():
    frTrain = open('horseColicTraining.txt')                                        #打开训练集
    frTest = open('horseColicTest.txt')                                             #打开测试集
    trainingSet = []; trainingLabels = []
    for line in frTrain.readlines():
        currLine = line.strip().split('\t')
        lineArr = []
        for i in range(len(currLine)-1):
            lineArr.append(float(currLine[i]))
        trainingSet.append(lineArr)
        trainingLabels.append(float(currLine[-1]))
    trainWeights = stocGradAscent1(np.array(trainingSet), trainingLabels, 500)      #使用改进的随即上升
    errorCount = 0; numTestVec = 0.0
    for line in frTest.readlines():
        numTestVec += 1.0
        currLine = line.strip().split('\t')
        lineArr =[]
        for i in range(len(currLine)-1):
            lineArr.append(float(currLine[i]))
        if int(classifyVector(np.array(lineArr), trainWeights))!= int(currLine[-1]):
            errorCount += 1
    errorRate = (float(errorCount)/numTestVec) * 100                                #错误率计算
    print("测试集错误率为: %.2f%%" % errorRate)

'''
Parameters:
    inX - 特征向量
    weights - 回归系数
Returns:
    分类结果
'''
```

```
# 函数说明:分类函数
def classifyVector(inX, weights):
    prob = sigmoid(sum(inX*weights))
    if prob > 0.5: return 1.0
    else: return 0.0


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

```
测试集错误率为: 32.84%
```

错误率还是蛮高的，并且每次运行的错误率也是不同的，错误率高的时候可能达到40%多。为什么会这样？首先，因为数据集本身有30%的数据缺失，这个是不能避免的。另一个主要原因是，我们使用的是改进的随机梯度上升算法，因为数据集本身就很小，就几百的数据量。用改进的随机梯度上升算法显然不合适。让我们再试试梯度上升算法，看看它的效果如何？

```
import numpy as np
import random

'''
Parameters:
    inX - 数据
Returns:
    sigmoid函数
'''
# 函数说明:sigmoid函数
def sigmoid(inX):
    return 1.0 / (1 + np.exp(-inX))

'''
Parameters:
    dataMatIn - 数据集
    classLabels - 数据标签
Returns:
    weights.getA() - 求得的权重数组(最优参数)
'''
# 函数说明:梯度上升算法
def gradAscent(dataMatIn, classLabels):
    dataMatrix = np.mat(dataMatIn)                                      #转换成numpy的mat
    labelMat = np.mat(classLabels).transpose()                          #转换成numpy的mat,并进行转置
    m, n = np.shape(dataMatrix)                                         #返回dataMatrix的大小。m为行数,n为列数。
    alpha = 0.01                                                        #移动步长,也就是学习速率,控制更新的幅度。
    maxCycles = 500                                                     #最大迭代次数
    weights = np.ones((n,1))
    for k in range(maxCycles):
        h = sigmoid(dataMatrix * weights)                               #梯度上升矢量化公式
        error = labelMat - h
        weights = weights + alpha * dataMatrix.transpose() * error
    return weights.getA()                                               #将矩阵转换为数组，并返回

# 函数说明:使用Python写的Logistic分类器做预测
def colicTest():
    frTrain = open('horseColicTraining.txt')                                        #打开训练集
    frTest = open('horseColicTest.txt')                                             #打开测试集
    trainingSet = []; trainingLabels = []
```

```
    for line in frTrain.readlines():
        currLine = line.strip().split('\t')
        lineArr = []
        for i in range(len(currLine)-1):
            lineArr.append(float(currLine[i]))
        trainingSet.append(lineArr)
        trainingLabels.append(float(currLine[-1]))
    trainWeights = stocGradAscent1(np.array(trainingSet), trainingLabels, 500)        #使用改进的随即上升
    errorCount = 0; numTestVec = 0.0
    for line in frTest.readlines():
        numTestVec += 1.0
        currLine = line.strip().split('\t')
        lineArr =[]
        for i in range(len(currLine)-1):
            lineArr.append(float(currLine[i]))
        if int(classifyVector(np.array(lineArr), trainWeights))!= int(currLine[-1]):
            errorCount += 1
    errorRate = (float(errorCount)/numTestVec) * 100                                  #错误率计算
    print("测试集错误率为: %.2f%%" % errorRate)

'''
Parameters:
    inX - 特征向量
    weights - 回归系数
Returns:
    分类结果
'''
# 函数说明:分类函数
def classifyVector(inX, weights):
    prob = sigmoid(sum(inX*weights))
    if prob > 0.5: return 1.0
    else: return 0.0


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

```
测试集错误率为: 28.36%
```

可以看到错误率下降了。总结一下，可以得到以下结论：

- 当数据集较小时，使用梯度上升算法
- 当数据集较大时，使用改进的随机梯度上升算法

对应到Sklearn中，我们就可以根据数据情况选择优化算法，比如：

- 数据较小的时候，我们使用liblinear
- 数据较大时，我们使用sag和saga

6、Sklearn构建Logistic回归分类器

```
from sklearn.linear_model import LogisticRegression

# 函数说明:使用Sklearn构建Logistic回归分类器
def colicSklearn():
```

```
    frTrain = open('horseColicTraining.txt')                #打开训练集
    frTest = open('horseColicTest.txt')                     #打开测试集
    trainingSet = []; trainingLabels = []
    testSet = []; testLabels = []
    for line in frTrain.readlines():
        currLine = line.strip().split('\t')
        lineArr = []
        for i in range(len(currLine)-1):
            lineArr.append(float(currLine[i]))
        trainingSet.append(lineArr)
        trainingLabels.append(float(currLine[-1]))
    for line in frTest.readlines():
        currLine = line.strip().split('\t')
        lineArr =[]
        for i in range(len(currLine)-1):
            lineArr.append(float(currLine[i]))
        testSet.append(lineArr)
        testLabels.append(float(currLine[-1]))
    classifier = LogisticRegression(solver='liblinear',max_iter=10).fit(trainingSet, trainingLabels)
    test_accurcy = classifier.score(testSet, testLabels) * 100
    print('正确率:%f%%' % test_accurcy)


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

```
正确率:73.134328%
```

更改 `LogisticRegression` 函数的 `solver` 参数为 `sag`，也就是随机平均梯度下降法。

```
C:\Users\Administrat\Anaconda3\lib\site-packages\sklearn\linear_model\sag.py:326:
  "the coef_ did not converge", ConvergenceWarning)
正确率:74.626866%

Process finished with exit code 0
```

发现有警告了，提示算法并未收敛，增加迭代次数为10000，再进行尝试。

```
正确率:73.134328%
```

可以看到前面的结论的正确性。

对应到Sklearn中，我们就可以根据数据情况选择优化算法，比如：

- 数据较小的时候，我们使用liblinear
- 数据较大时，我们使用sag和saga

7、sklearn.linear_model.LogisticRegression

sklearn.linear_model.LogisticRegression是一个很好的模型，决策树算法就是通过它实现的，详细的看这个博客——sklearn.linear_model.LogisticRegression()函数解析（最清晰的解释）

8、总结

Logistic回归的目的是寻找一个非线性函数Sigmoid的最佳拟合参数，求解过程可以由最优化算法来完成。在最优化算法中，最常用的就是 **梯度上升算法**，而 **梯度上升算法** 又可以简化为 **随机梯度上升算法。**

随机梯度上升算法与梯度上升算法的效果相当，但占用更少的计算资源。此外，随机梯度上升是一个在线算法，它可以在新数据到来时就完成参数更新，而不需要重新读取整个数据集来进行批处理运算。

机器学习的一个重要问题就是如何处理缺失数据。这个问题没有标准答案，取决于实际应用中的需求。现有一些解决方案，每种方案都各有优缺点。下一章将介绍与Logistic回归类似的另一种分类算法：支持向量机，它被认为是目前最好的现成的算法之一。