Logistic 回归实验

⇒ **主要思想：**根据现有的数据对分类边界线建立回归公式，以此进行分类。

1.优点：

计算代价不高，易于理解和实现

2.缺点：

容易欠拟合，分类精度不高

3.适用数据类型：

数值型和标称型数据

注：logistic 回归又称 logistic 回归分析，主要在流行病学中应用较多，比较常用的情形是探索某疾病的危险因素，根据危险因素预测某疾病发生的概率，等等。例如，想探讨胃癌发生的危险因素，可以选择两组人群，一组是胃癌组，一组是非胃癌组，两组人群肯定有不同的体征和生活方式等。**这里的因变量就是是否胃癌，即“是”或“否”，为两分类变量**，自变量就可以包括很多了，例如年龄、性别、饮食习惯、幽门螺杆菌感染等。自变量既可以是连续的，也可以是分类的。通过 logistic 回归分析，**就可以大致了解到底哪些因素是胃癌的危险因素**。

【4.实验测试】

==**实验数据：**机器学习测试数据、iris 数据、马疝气病数据集

1）代码：

a：logRegress(核心)

```
#coding:utf-8
from numpy import *
import matplotlib.pyplot as plt

'''
    Logistic 回归实验版-2018.04.02
        @auto Alan.hcy
'''


#testSet 数据集
def loadDataSet():
    '''
    数据读取-'testSet.txt':100*3
    其中 100*2 位 dataMat
    其余的为标签 labelsMat
    :return: dataMat, labelsMat
    '''
    fr = open('testSet.txt')
    dataMat = []; labelsMat = []
    for line in fr.readlines():
        curlines = line.strip().split()
        dataMat.append([1.0, float(curlines[0]), float(curlines[1])])
```

```
        labelsMat.append([int(curlines[2])])
    return dataMat, labelsMat

def sigmoid(inZ):
    '''
    定义 sigmoid 函数-singmoid(Z)=1/(1+exp(-Z))
        Z=w0*x0+w1*x1+...+wn*xn
    :param inZ:
    :return: sig
    '''
    sig = 1/(1 + exp(-inZ))
    return sig


def gradAscent(dataMat, labelsMat):
    '''
    step01:梯度上升优化算法-对整个数据集遍历
    根据 w = w + alpha*f(x),f(x)=WX 求解
    :param dataMat:
    :param labelsMat:
    :return: weighs
    '''
    dataMatrix = mat(dataMat)
    labelsMatrix = mat(labelsMat).transpose()
    alpha = 0.001
    iters = 500
    n = shape(dataMatrix)[1]
    weighs = ones((n,1)) #n 行 1 列,n=3
    #迭代回归系数
    for i in range(iters):
        h = sigmoid(dataMatrix * weighs)
        error = labelsMatrix - h
        weighs = weighs + alpha * dataMatrix.transpose() * error
    return weighs

def plotBestfit(weighs):
    '''
    #画出最佳拟合曲线（根据 weighs）
    :return: 模型图-决策边界
    '''
    dataMat, labelsMat = loadDataSet()
    #m = shape(dataMat)[0]
    dataMatArry = array(dataMat)
    #注意：若 labelsMat 不是 array 格式，遍历就会出错
```

```
    labelsMat = array(labelsMat)
    m = shape(dataMatArry)[0]
    xdata1 = []; ydata1 = []
    xdata2 = []; ydata2 = []
    #遍历数据
    for i in range(m):
        if labelsMat[i] == 0:
            xdata1.append(dataMatArry[i, 1]); ydata1.append(dataMatArry[i, 2])
            #print xdata1
        else:
            xdata2.append(dataMatArry[i, 1]); ydata2.append(dataMatArry[i, 2])
            #print xdata2
    #定义画布
    ax = plt.figure().add_subplot(111)
    ax.scatter(xdata1, ydata1, s=36, c='red', marker ='s')
    ax.scatter(xdata2, ydata2, s=36, c='green')
    x = arange(-3.0, 3.0, 0.1)
    y = (-weighs[0] - weighs[1] * x) / weighs[2]
    ax.plot(x, y)
    plt.xlabel('x1'); plt.ylabel('x2')
    plt.title('LogRegress')
    plt.show()

def stogradAscent01(dataMatrix, classLabels):
    '''
    step02:随机梯度上升优化算法
    --与之前的算法比较：不用每次遍历整个数据集来更新回归
                        系数，而是一次只仅用一个样本点来更新
    :return: weighs
    '''
    m, n = shape(dataMatrix)
    dataMatrix = array(dataMatrix)
    alpha = 0.01
    weights = ones(n) #1*3
    #遍历每一行数据
    for i in range(m):
        h = sigmoid(sum(dataMatrix[i]*weights)) #1*3 * 3*1
        #计算误差
        error = classLabels[i] - h
        #更新权重值
        weights = weights + alpha * error * dataMatrix[i]
    return weights

def stogradAscent02(dataMatrix, classbelMat,indexIter=150):
```

```
    '''
    step03:改进的随机梯度上升算法
        #-与之前的比较：1）增加了对 alpha 进行迭代更新；
                        #2）将“对样本进行逐个遍历来更新回归系数”改为随机抽取样本进行更新
                        #3）同时增加对以上过程的迭代次数：150 次
    #:return: weighs
    '''
    dataArray = array(dataMatrix)
    labelsArry = array(classbelMat)
    m, n = shape(dataArray)
    weighs = ones(n)
    for iters in range(indexIter):
        dataIndex = range(m)
        for i in range(m):
            #alpha 迭代更新
            alpha = 4/(1.0 + i + iters) + 0.0001
            #初始化索引- 整型
            randIndex = int(random.uniform(0, len(dataIndex)))
            h = sigmoid(sum(dataArray[randIndex] * weighs))
            error = labelsArry[randIndex] - h
            weighs = weighs + alpha * error * dataArray[randIndex]
            del(dataIndex[randIndex])
    return weighs

#疝气病症的病马数据集
def classifier(inX, weights):
    '''
    该分类器原理:(0 和 1 的类标签)
        利用 sigmoid 函数进行计算，大于 0.5 的就看成 1，反之为 0；
    :param inX: 采用“疝气病症的病马数据集”—这里 inX 是测试集数据
    :param weights: 训练集学习后的 weights
    :return: 0 或 1
    '''
    if sigmoid(sum(inX * weights)) > 0.5:
        return 1.0
    else:
        return 0.0

def verTestResults():
    '''
    验证和测试结果：horseColicTest.txt 与 horseColicTraining.txt
        step01: 先进行读取样本数据-马疝气病数据集
        step02: 利用样本数据集，得到训练集的 TrainSet 与 TrainLabels；
```

```
                再通过 stogradAscent02（dataMatrix, classLabels, 500）函数计算
weights;
        step03: 利用上述 weights 和测试集的 TestSet 测试，从而与测试集的标签比
较，得到正确率
    :return: trainweigths,errorRate
    '''
    frTrain = open('horseColicTraining.txt')
    frTest = open('horseColicTest.txt')
    trainSet =[]; trainLabels =[]
    #训练集
    for line in frTrain.readlines():
        curlines = line.strip().split()
        lineArr = []
        for i in range(21):
            lineArr.append(float(curlines[i]))
        trainSet.append(lineArr)
        trainLabels.append(float(curlines[21]))
    #return trainSet, trainLabels
    trainweigths = stogradAscent02(array(trainSet), trainLabels, 500)
    #plotBestfit(trainweigths)
    #return trainLabels
    #测试集
    errorCount = 0.0; numTestLab = 0.0
    for lines in frTest.readlines():
        numTestLab += 1.0
        curlines = lines.strip().split()
        linesArr =[]
        for i in range(21):
            linesArr.append(float(curlines[i])) #必须得和其他数据保持 float
型，否则会报错
        #判断测试集的每一行经过分类器获得的预测标签是否与真实标签一致
        if int(classifier(array(linesArr), trainweigths)) != int(curlines[21]):
            errorCount +=1.0
    print 'numTestLab:',numTestLab
    print 'errorCount:',errorCount
    errorRate = (errorCount/numTestLab)
    return trainweigths,errorRate

def mutiTest():
    '''
    求 10 次错误率的平均值
    :return:
    '''
    numErrorRate = 0.0; aveErrorRate = 0.0
```

```
    for i in range(10):
        trainweigths, errorRate = verTestResults()
        numErrorRate += errorRate
        aveErrorRate =  float(numErrorRate)/10
    return aveErrorRate
```

b)：执行代码-plotResult

```
    #coding:utf-8
import matplotlib.pyplot as plt
from numpy import *
import logRegress


'''
    1.机器学习测试数据
'''
#数据读取-'testSet.txt':100*3
datas, labels = logRegress.loadDataSet()

#梯度上升算法
#weights1 = logRegress.gradAscent(datas, labels)
#print weights1
#logRegress.plotBestfit(weights1)

#随机梯度上升算法
weights2 = logRegress.stogradAscent01(datas, labels)
print 'weights2:',weights2
#logRegress.plotBestfit(weights2)

#改进的随机梯度上升算法
weights3 = logRegress.stogradAscent02(datas, labels)
print 'weights3:',weights3
#logRegress.plotBestfit(weights3)

'''
    2. 马疝气病数据集：366*22
'''
tranweights, errorRate = logRegress.verTestResults()
print 'tranweights:',tranweights
print 'errorRate:',errorRate

aveErrorRate = logRegress.mutiTest()
print 'aveErrorRate:',aveErrorRate
#logRegress.plotBestfit(tranweights) #多维空间
```

2）实验结果

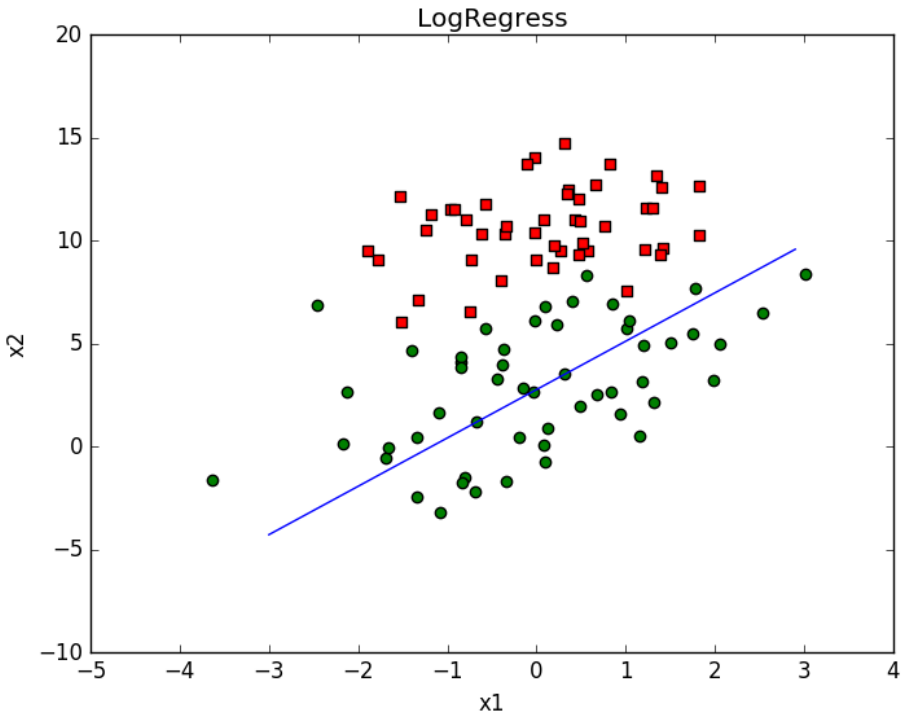
a. 机器学习测试数据--二维（100*3）

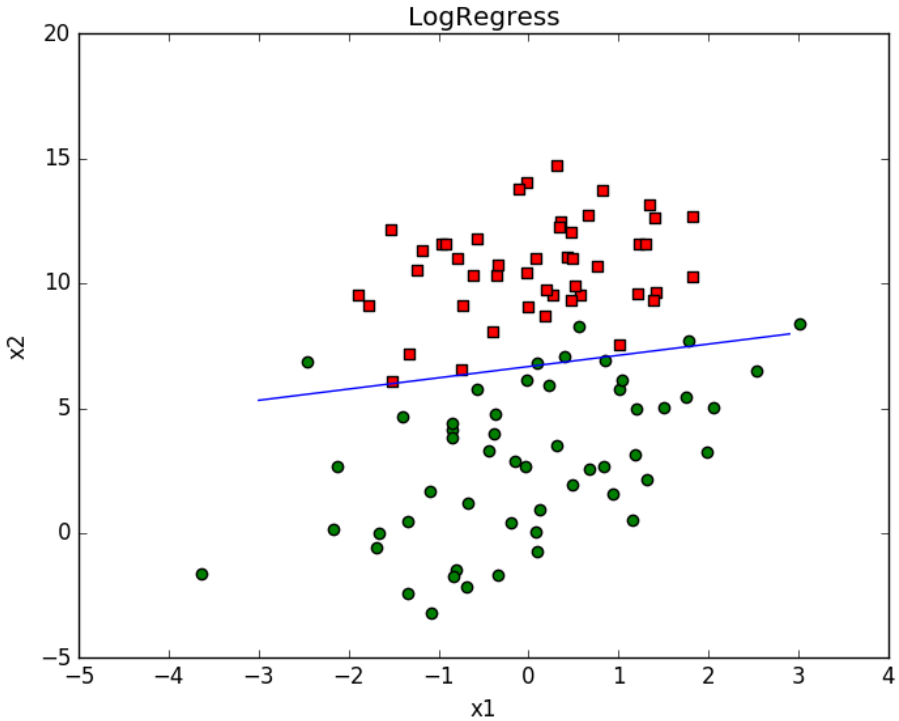
===随机梯度上升算法回归系数===

weights2: [1.01702007 0.85914348 -0.36579921]

===改进的随机梯度上升算法回归系数===

weights3: [13.962819   0.94306107 -2.09331999]





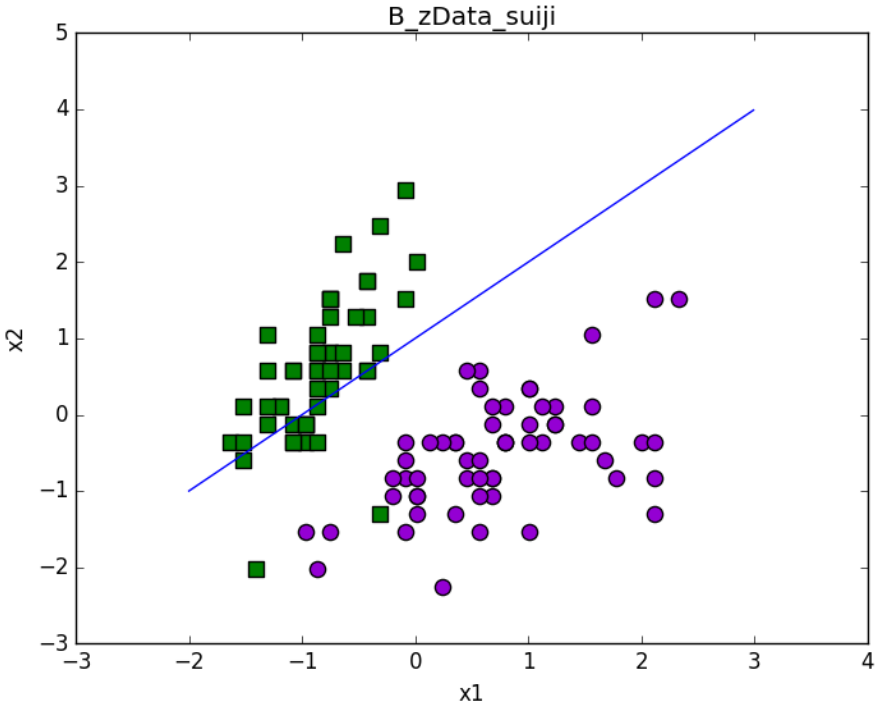
b.iris 数据集--二维（110*3）

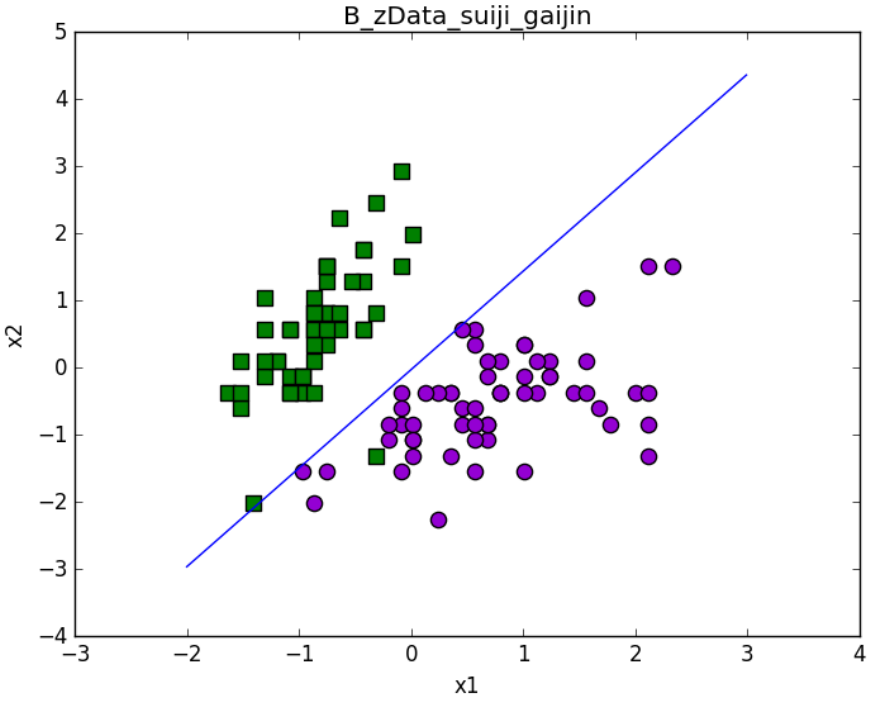
===随机梯度上升算法回归系数===

weighs [-0.002118 0.06423442 -0.03176099]

===改进的随机梯度上升算法回归系数===

weighs_new01 [-3.06965478 5.72389 -9.5540524]

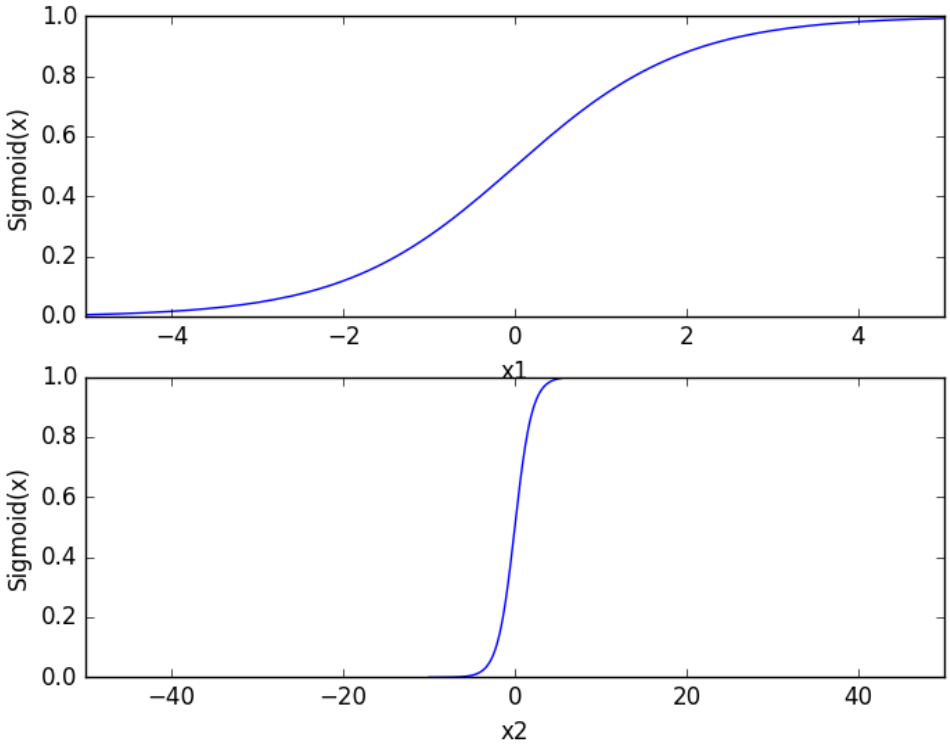




c. 马疝气病数据集-多维（366*22）

```
=========马疝气病数据集=========
===训练集回归系数与测试集分类错误率===
tranweights: [  15.95225279   -2.56528914    1.69110137   -1.9509334
0.73505079
  -5.34569483    5.65709293   -6.78123818   -6.95330715   -9.5454885
  21.20618926  -24.76183978   27.4054603    12.97742962  -11.14111687
   3.61530515   -3.70442148    0.08746094    0.18109077   -4.75746514
  -3.11115268]
errorRate: 0.388059701493
===测试集每次分类的错误率 1===
numTestLab: 67.0
errorCount: 28.0
===测试集每次分类的错误率 2===
numTestLab: 67.0
errorCount: 22.0
===测试集每次分类的错误率 3===
numTestLab: 67.0
errorCount: 23.0
===测试集每次分类的错误率 4===
numTestLab: 67.0
errorCount: 21.0
===测试集每次分类的错误率 5===
numTestLab: 67.0
errorCount: 21.0
===测试集每次分类的错误率 6===
numTestLab: 67.0
errorCount: 17.0
===测试集每次分类的错误率 7===
numTestLab: 67.0
errorCount: 26.0
===测试集每次分类的错误率 8===
numTestLab: 67.0
errorCount: 24.0
===测试集每次分类的错误率 9===
numTestLab: 67.0
errorCount: 27.0
===测试集每次分类的错误率 10===
numTestLab: 67.0
errorCount: 24.0
===测试集 10 次分类的错误率与平均错误率===
➔aveErrorRate: 0.34776119403
```

附：sigmoid 函数：



以 sigmoid(x)=0.5 为界线，大于它为 1，小于则为 0（分类）