**使用python实现对数几率模型**

班级：计算机1602班 姓名：詹璧瑞 学号：1611640206

1. **题目：**

使用python实现对数几率回归模型

1. **问题描述：**

使用鸢尾花数据集，用对数几率回归模型实现对鸢尾花的二分类。也就是用python编程，训练出一个模型，输入鸢尾花的四个属性值就可以预测鸢尾花的类别。

1. **数据集描述：**

鸢尾花的数据集一共包含150个实例，有三类，每一类有50个实例，因为题目要实现的是二分类问题，所以选其中两个类，也就是选择100个实例。每个类有4个属性：萼片长度、萼片宽度、花瓣长度、花瓣宽度。这里选择的是前两个类： Iris-Setosa、Iris-Versicolour

1. **实现方法：**

要训练出能对鸢尾花进行二分类的模型，分为以下几个四个主要步骤：导入数据集、用梯度下降法求得损失函数的最优解、计算模型的正确率、输出损失函数的值随训练次数的变化曲线图。

1、导入数据集：

编写函数pre\_Data()，将网页上的数据复制到文本文件中，与源代码放在同一个目录下。文本文件中有五列数据，前四列为属性值，最后一列是用字符串表示的鸢尾花分类。每一行的数据是一个样例，共有100行，即100个样例。每一行的数据之间用“，”逗号分隔。

使用pandas库的read\_table()函数读取文本文件里的数据，读出来的数据是DataFrame类型，先将最后一列的字符串们根据类别分别转换为整数1和0，然后，将前四列、最后一列转换为ndarray类型，分别赋值给X和y，。因为模型最终还要加上一个bias，所以为X数据集加上一列全为1的列向量。另外，准备好总的数据集之后，准备分测试集与训练集。

在这里，我使用了sklearn库的train\_test\_split()方法分测试集和数据集。给train\_test\_split()传入数据源X，y，传入测试集的大小，将参数stratify设为y，就可以根据y数据集里各类的比例来公平地选出数据，在这里，测试集和训练集里两个类的比例都是1：1。得出X，y的训练集和测试集之后，将它们转换为矩阵，方便后面进行矩阵运算。

以上就实现了从文本文件中导入训练集和测试集的操作。

2、梯度下降法求最优解：

在logistic\_regression()函数里，调用处理数据集的函数pre\_Data()，得到训练集和测试集。接着再使用numpy库的rand()函数，随机生成5\*1维的weight参数。另外，还要将y转置为列向量。提前设置好了步长alpha和迭代次数max\_iter，准备好所有数据后就准备开始计算。

用for循环，循环max\_iter次，按照公式：来计算使得损失函数达到最小值参数，每迭代一次，就将算得的第i个的值传入函数cost\_function()中，计算第i次训练时损失函数的值。

迭代完max\_iter次后，logistic\_regression()函数返回使损失函数达到最优解的的值。

3、计算正确率：

编写accuracy\_count()函数，调用pre\_Data()函数得到测试集的数据，对y进行转置。再调用logistic\_regression()函数，得到训练好的模型。使用训练出来的参数和X的测试集，根据公式：y=计算出预测值。

用for循环比较预测值和测试集的值，两者的差求绝对值，若差的绝对值大于1e-1（0.1），就认为预测结果与测试集不符合，计数器count加一。最后输出的正确率为1-count/测试集样例总数。

4、输出损失函数变化曲线图：

在函数Plot()里，x轴为一个从1到迭代次数max\_iter的集合，y轴的值就是每次计算出的损失函数的值。

1. **实验结果图：**

|  |  |  |
| --- | --- | --- |
|  | **迭代次数：100** | **迭代次数：500** |
| 测试集：70%  训练集：30% |  |  |
| 测试集：70%  训练集：30% |  |  |
| 测试集：90%  训练集：10% |  |  |

1. **实验结果分析：**

本次实验用到的梯度，损失函数都参考了课本的公式，但是输出的结果有些错误，损失函数的数量级较大，而且函数振荡得比较厉害，还有，随着测试集数量的减少，损失函数的振荡也相应减弱。正确率每次都是100%，但后来调参发现，迭代次数为10步以下的时候，正确率从0.5逐渐变为0.57，0.58，10步之后就是100%了，这些结果出现的原因都还没有找出来。

1. **源代码：**

**import numpy as np**

**import pandas as pd**

**from sklearn.model\_selection import train\_test\_split**

**import matplotlib.pyplot as plt**

**y=list();**

**max\_iter=100**

**alpha=0.01 ##设置步长和迭代次数**

**my\_testsize=0.1 ##输入测试集的比例：my\_testsize**

**def pre\_Data(): ##从文本文件导入数据**

**getdata=pd.read\_table(r'C:\A Script Library\files\irisdata.txt',header=None,sep=',')**

**dataset=getdata.replace(['Iris-setosa','Iris-versicolor'],[1,0]) ##把两个用字符串表示的类别名替换为整数1，0**

**X=np.array(dataset.ix[:,:3])**

**X\_add1=np.ones(X.shape[0])**

**X=np.c\_[X,X\_add1] ##给每个特征向量后面加上1**

**y=np.array(dataset[4]) ##分割数据集，得到特征值X和标签y**

**X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=my\_testsize,random\_state=0,stratify=y)**

**return np.mat(X\_train),np.mat(X\_test),np.mat(y\_train),np.mat(y\_test) ##分别返回测试集和训练集的属性集、标签集**

**def gradient(y\_train,X\_train,weight): ##计算损失函数的梯度**

**error=y\_train-np.exp(X\_train\*weight)/(1+np.exp(X\_train\*weight))**

**return (-1)\*X\_train.T\*error**

**def logistic\_regression(): ##对率回归**

**X\_train,X\_test,y\_train,y\_test=pre\_Data()**

**y\_train=y\_train.T**

**weight=np.random.rand(5,1) ##随机在[0,1)范围内生成 W值和 bias值, ##设置步长与迭代次数**

**for iter in range(max\_iter):**

**grad=gradient(y\_train,X\_train,weight)**

**weight=weight-alpha\*grad**

**cost\_function(weight,y\_train,X\_train)**

**return weight**

**def cost\_function(weight,y\_train,X\_train): ##计算损失函数的值**

**cost\_value=-1\*y\_train.T\*X\_train\*weight+np.log(1+np.exp(X\_train\*weight))**

**value=cost\_value[0,0]**

**y.append(value)**

**def accuracy\_count(): ##计算分类正确率**

**X\_train,X\_test,y\_train,y\_test=pre\_Data()**

**y\_test=y\_test.T**

**weight=logistic\_regression()**

**y\_pred= 1/(1+np.exp(-X\_test\*weight)) ##计算预测值**

**y\_pred=np.mat(y\_pred)**

**count=0**

**for i in range(y\_test.shape[0]-1):**

**if np.absolute(y\_pred[i]-y\_test[i])>=1e-1:**

**count=count+1**

**return (1-(count/y\_test.shape[0]))**

**def Plot():**

**x=np.linspace(1,max\_iter,max\_iter)**

**plt.figure(figsize=(8,6))**

**plt.plot(x,y,color='red',linewidth=2)**

**plt.xlabel("training\_times")**

**plt.ylabel("loss function")**

**plt.show()**