**华中科技大学计算机科学与技术学院**

**机器学习报告**



专 业： 计算机科学与技术

班 级： 计算机1807班

学 号： U201814694

姓 名： 陈 帅

成 绩：

指导教师： 邹复好

**完成日期： 2020年 6月 20日**

# Project

## 一、实验题目：个人收入预测

## 二、实验要求

（1）从课程QQ群里下载数据集：题目4 个人收入预测.csv，数据有4000行59列，第一列为ID值，第59列为label（0或1），剩下57列为属性。要求根据属性判断此人年收入是否大于50K。

（2）将数据中前3000项作为训练集，后1000项作为测试集，使用logistic回归进行二分类，实现语言要求为Python；

（3）在使用梯度下降法时，调整学习率的固定值，有能力的同学可以学习并使用动态调整学习率的方法，探究不同学习率的选择对训练误差收敛速度的影响，绘制misclassification rate曲线进行比较并分析。

（4）实验时要求计算出准确率、要求画出训练和测试loss曲线、要求调整多个学习率和正则化参数后给出以上结果。

## 三、算法设计

（1）加载文件

函数使用open语句打开csv文件，并每次读取一行，将逗号“,”和结尾的“\n”去除，并取每行第2到第58个元素作为一个列表，最后加入一个元素1.0作为新增的维度以解决回归时的截距问题，再将前3000行元素以float类型的以列表的新式赋给x\_train，作为其的元素，后1000行则赋给x\_test；将前3000行的第59个元素以float类型赋给y\_train作为其元素，将后1000行的第59个元素赋给y\_test。并将这四个数据换成列表的新式作为返回值返回。

流程图如图3-1所示。

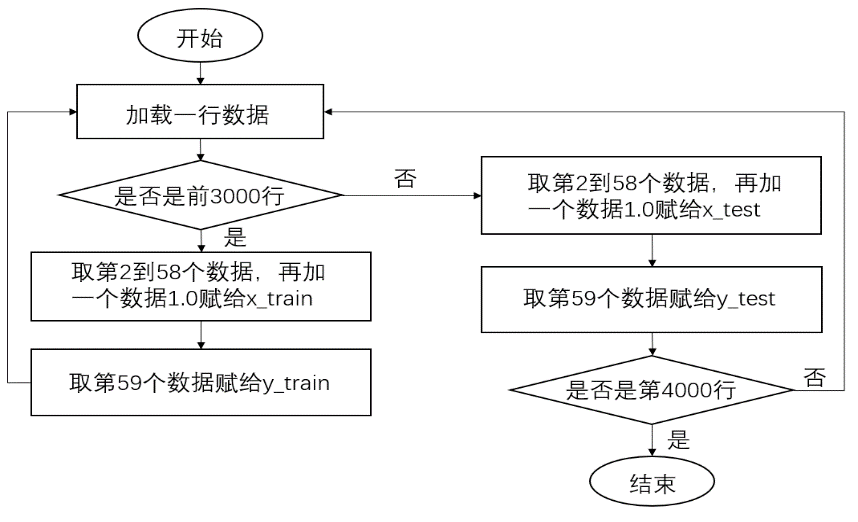


图 3-1 加载文件流程图

根据流程图，加载数据的函数代码如下：

def loadfile():

file = open('题目4 个人收入预测.csv')

x\_train = []#训练集数据

y\_train = []#训练集标签

x\_test = []#测试集数据

y\_test = []#测试集标签

for line in file:

line = line.strip().split(',')

if int(line[0]) <= 3000:#取前3000行作为训练集

line\_x = list(map(float, line[1:58]))

line\_x.append(1.0)#增加一个维度

x\_train.append(line\_x)

y\_train.append(float(line[58]))#取标签值

else:

line\_x = list(map(float, line[1:58]))

line\_x.append(1.0)

x\_test.append(line\_x)

y\_test.append(float(line[58]))

x\_train = np.mat(x\_train)

y\_train = np.mat(y\_train)

x\_test = np.mat(x\_test)

y\_test = np.mat(y\_test)

return x\_train,y\_train,x\_test,y\_test

（2）数据归一化和标准化

数据归一化通过最小值最大值归一化的方法，将数据映射到[0,1]，如公式（1）所示。

公式（1）

数据标准化使用的是z-score的方法，基于数据的均值和方差进行数据标准化，如公式（2）所示。

公式（2）

两种数据标准化的函数代码如下：

def z\_score\_normalization(x):

for i in range(np.shape(x)[1]):

std = np.std(x[:,i])#求方差

if std == 0.0 :

x[:,i] = 1

else:

x[:, i] = preprocessing.scale(x[:, i])#通过sklearn库进行归一化

return x#返回归一化后的数据集

def min\_max\_normalization(x):

for i in range(np.shape(x)[1]):

min = np.min(x[:,i])#最小值

max = np.max(x[:,i])#最大值

if max == min:

x[:, i] = 0.5

else:

x[:, i] = (x[:, i] - min) / (max - min)#最小值最大值归一化

return x#返回归一化后的结果

（3）sigmoid函数

函数的公式如公式（3）所示。

公式（3）

其中Z=，就是y的预测值，将Z映射到（0,1）之间。

（4）梯度下降法训练

梯度的求解公式为,其中m为训练集X的数据个数，题目要求为3000个，h为sigmoid函数。

梯度下降法的公式为，其中α是学习率，通过多次迭代，得到最终的W用于测试。

由于多次迭代使训练的结果过拟合，导致测试结果不理想，可以对数据进行L2正则化，即再原来的公式的基础上加上正则化项，最终结果如公式（4）所示，其中λ为正则化系数。

公式（4）

用梯度下降法对测试集进行训练的代码如下：

def train(learningrate,iteration,x\_train,y\_train):

m = np.shape(x\_train)[0]

w = np.zeros((58, 1))#给w初始化为0

loss = []

for k in range(iteration):

h = sigmoid(x\_train \* w) #sigmoid函数

grad=np.transpose(x\_train) \* (h - y\_train.T)/m#求梯度

L=0.1

w = (1-L\*learningrate/m)\*w - learningrate\*grad#梯度下降,L2正则化

loss.append(lossfunc(x\_train, y\_train, w))#损失函数

return w.T,loss

（5）求损失函数

经过推导，得损失函数为，通过此公式可以对当前w求损失函数，程序运行时，在迭代时计算并记录损失函数，以便观察损失函数的变化。

（6）使用动态学习率优化程序

使用指数衰减法动态调节参数，即a=a\*decay^(i/iteration)

其中decay是衰减率，iteration是总迭代次数，i是当前迭代次数，思路是先使用一个较大的学习率快速使结果接近最优解，随着迭代次数增加，学习率减小，使模型更加稳定。

（7）测试

利用训练所得的W与测试即数据进行训练，通过函数sigmoid（）求得结果，设置阈值为0.5，若求得结果大于等于0.5时，测得y为1，若求得结果小于0.5，则y值为0，再通过对比label的真实值y\_test，计算正确率。

测试函数代码如下

def test(x\_test,y\_test,w):

num\_right = 0#记录测试正确数

for i in range(np.shape(x\_test)[0]):

x = np.array(x\_test[i])#取一组数据

x = x[0]

h = sigmoid(x\*w.T)#通过sigmoid函数计算测试值

if int(h+0.5)/1 == y\_test[0,i]:#将测试值与实际值对比，并记录测试正确的此时

num\_right += 1

return num\_right/np.shape(x\_test)[0]#返回准确率

## 四、实验环境与平台

计算机系统：Windows10

代码编写软件：PyCharm Community Edition 2020.1 x64

Python版本：python3.8

## 五、程序实现

实现的代码如附录1 所示。

## 六、实验结果

（1）不同数据标准化的方法的实验结果

1）Z标准化：在学习率为0.5，迭代次数为100的情况下测试，准确率为0.923，损失函数如图6-1所示。

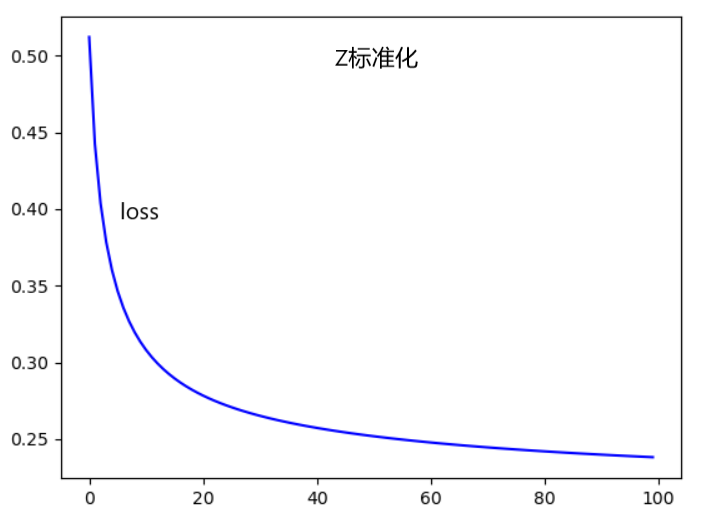


图 6-1 Z标准化收敛

由图可知，当迭代到100左右时便由很好的收敛效果

2）min-max标准化：在学习率为0.5，迭代次数为100的情况下测试，准确率为0.7，损失函数如图6-2所示。

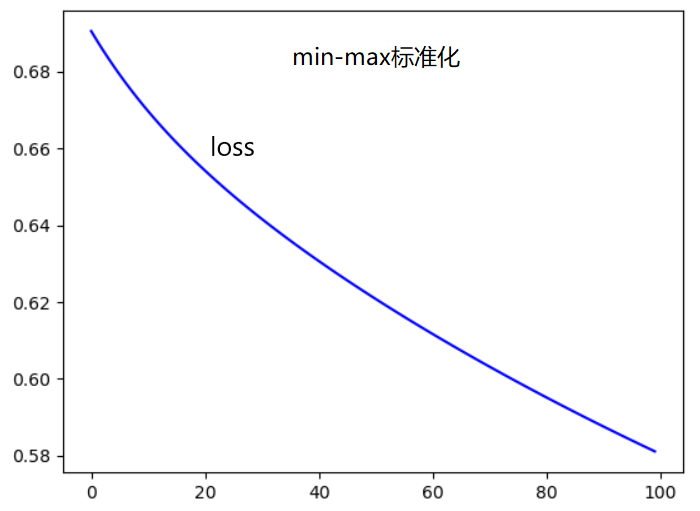


图 6-2 min-max标准化收敛

由图可知，使用min-max标准化，结果收敛过慢，所以尝试提升迭代次数到10000，得到的结果准确率为0.919，损失函数如图6-3所示。

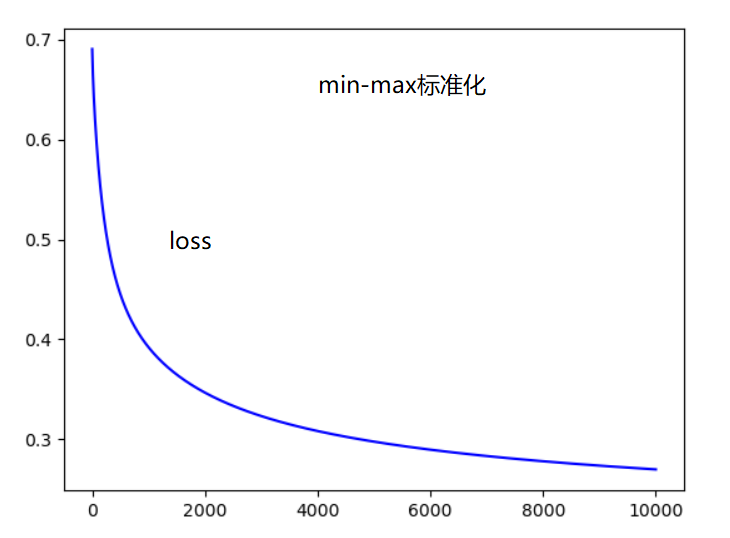


图 6-3 增加迭代次数

由此可知，使用Z标准化对数据进行处理可以提高训练的迭代效率，提高准确率。以下测试都是使用Z标准化对数据进行处理的。

两种标准化结果的准确率对比如图6-4所示。



图 6-4 两种标准化准确率结果

（2）不同迭代次数对实验准确率的影响

1）当学习率定为17时，调节不同迭代次数，得到准确率曲线如图6-5所示。

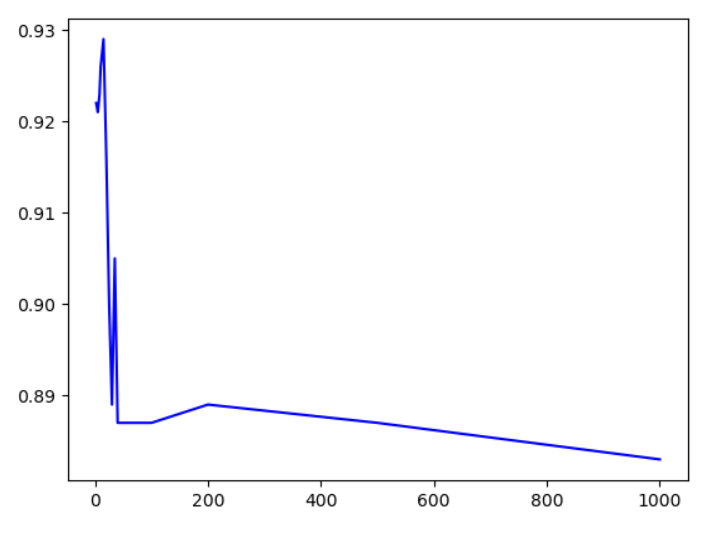


图 6-5 学习率为17

由图可知准确率不稳定，且迭代次数越高可能准确率反而越低，这是学习率过大引起的，调整学习率为0.1，将学习率调整到一个较小的数值后，准确率曲线如图6-6所示。

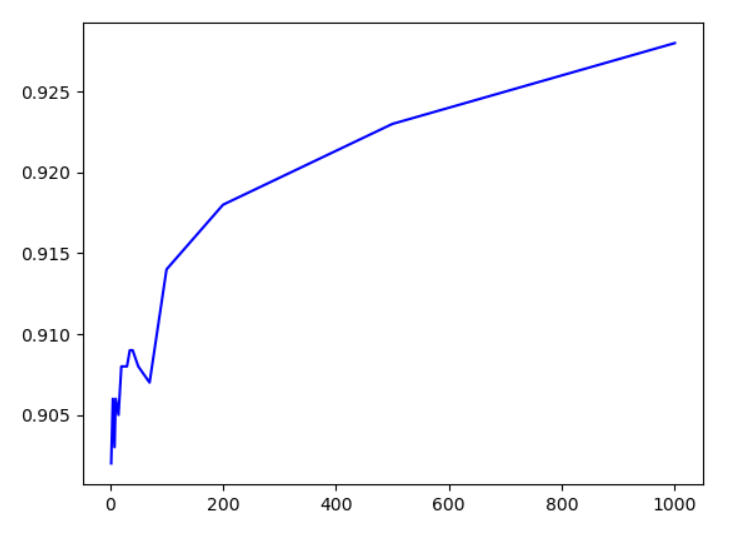


图 6-6 学习率为0.1

当学习率较小的时候，准确率会随着迭代次数的增加而增大。但是即便迭代次数达到1000的时候还没有达到一个比较稳定的值，这是由于学习率较小，收敛速度较慢。再将学习率调整为1，当学习率为一个比较合理的值时，结果如图6-7所示

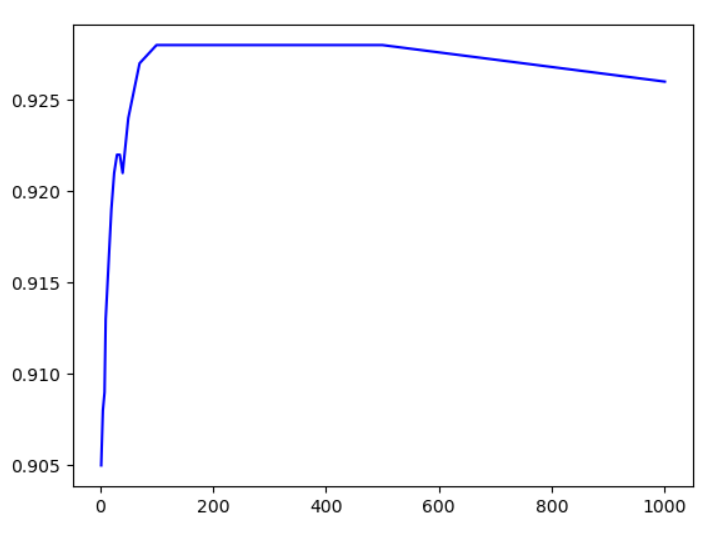


图 6-7 学习率为1

由图可知，学习率取1时，收敛效率快，迭代次数达到100时便可以得到较高的准确率，且准确率曲线趋于平缓。

（3）不同学习率对结果的影响

1）通过调整梯度下降法的学习率观察损失函数的收敛情况，结果如图6-8所示。

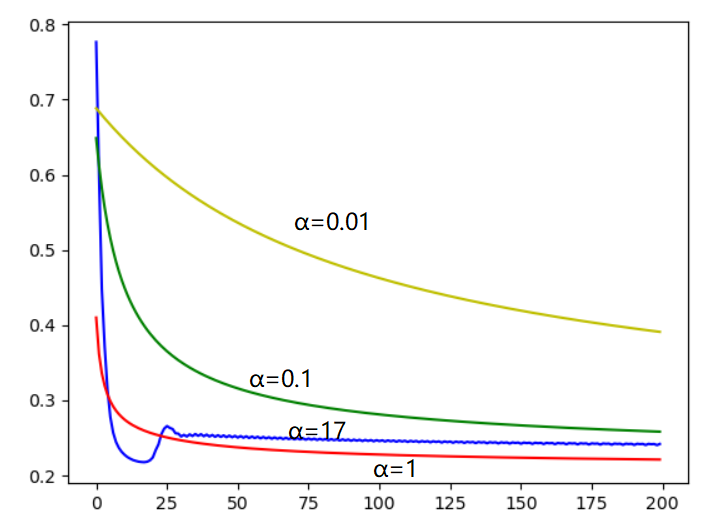


图 6-8 学习率对损失函数的影响

由损失函数曲线图可知，当学习率过小时，损失函数收敛速度慢，但当学习率过大时，损失函数反而出现震荡，导致收敛效果变差。

2）调整学习率，得到准确率曲线，如图6-9所示。

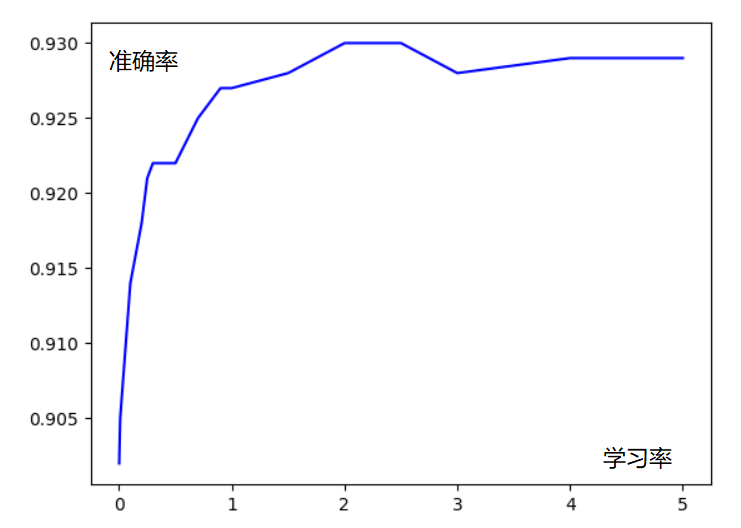


图 6-9 学习率对正确率的影响

由曲线图可知，当学习率在2到2.5附近时得最高值，达0.93。

（4）正则化对实验结果的影响

1）通过对数据加入正则化处理，在原来的基础上加入正则化项，使学习率为0.5，迭代次数为500，正则化系数为2，结果损失函数的收敛结果如图6-10所示。

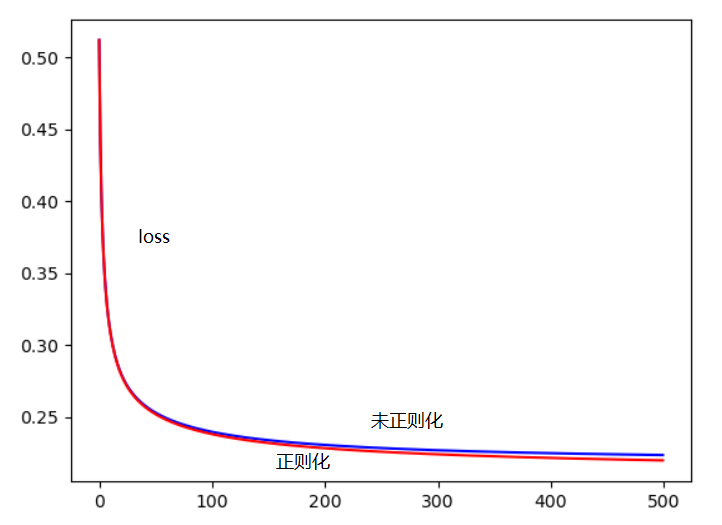


图 6-10 正则化对损失函数的影响

由图可以得知，使用正则化后，损失函数迭代收敛更快一些，且实验结果的准确率更高，正则化之前准确率为0.928，使用正则化后准确率可以达到0.930。

2）通过改变不同正则化系数，得到不同的准确率，根据准确率曲线如图6-11所示。

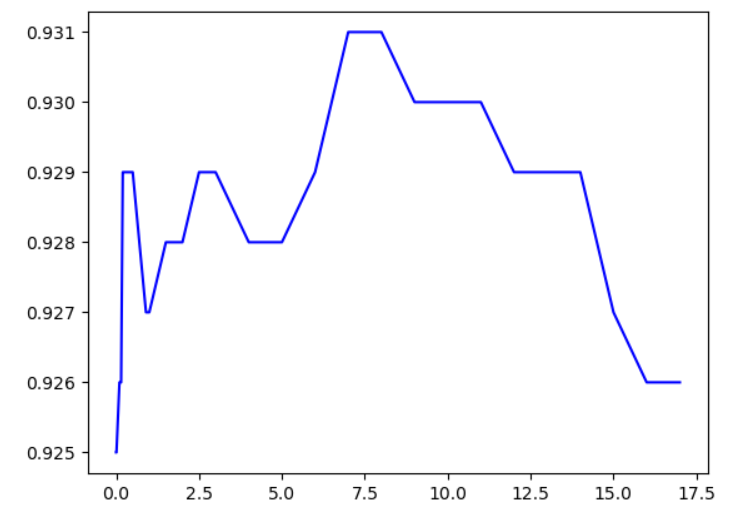


图 6-11 正则化对正确率的影响

根据准确率曲线可知，正则化系数过高或过低都会导致准确率结果不太理想，图示准确率高峰位置的正则化系数在7.5附近，准确率可以达到0.931。

（5）使用动态学习率

将学习率用指数衰减法优化，调整衰减率decay，得准确率曲线如图6-12所示。

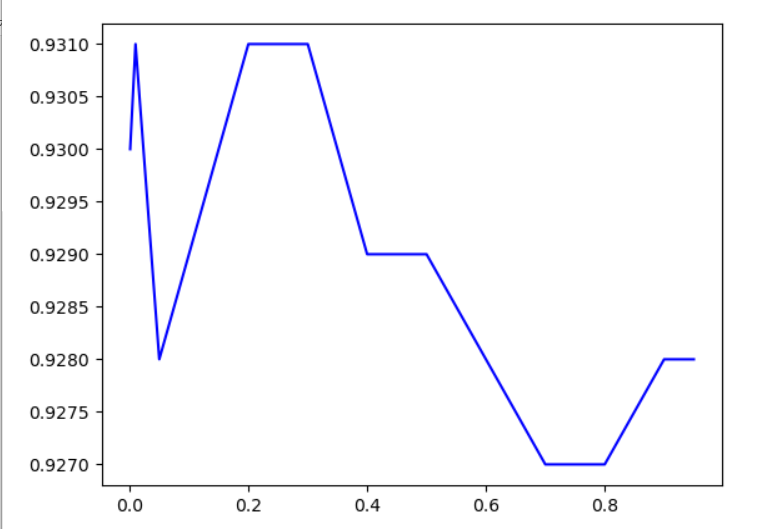


图 6-12 动态学习率的准确率曲线

对比没有优化的结果，如图6-13所示。

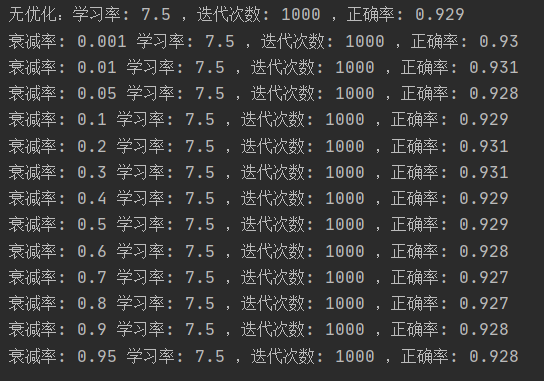


图 2- 13 未优化的结果对比

由曲线图可知，当衰减率取2.5附近时，准确率可以取到较高的结果。再对比没有使用动态学习率的情况可知，当取合适的学习率时，可以对提高准确率，使训练的结果更加稳定。

## 七、结果分析

（1）使用数据标准化对数据进行处理，可以消除个别数据特征指标对结果的影响，使梯度下降时的效率更高。本次实验使用的2中数据标准化的方法中，明显Z标准化后的结果准确率更高，损失函数收敛的效率更快。

（2）使用梯度下降法进行迭代时，需要根据学习率选择合适的迭代次数，当学习率小时，需要迭代多次才能的到一个比较精确的结果，当学习率较大时，迭代次数越多，反而出现准确率下降的情况。

（3）通过对数据正则化，可以避免多次迭代使得训练结果过拟合导致的结果不理想，正则化后损失函数收敛得更快且准确率更高。

（4）通过指数衰减法使学习率，可以使学习率在不断迭代过程中减小，使梯度下降过程中不会跳过最优解也可以保证收敛效率，这使训练的结果更稳定，结果的正确率也更高。

# 附录1 代码实现

import numpy as np

import matplotlib.pyplot as plt

from sklearn import preprocessing

#加载文件

def loadfile():

file = open('题目4 个人收入预测.csv')

x\_train = []#训练集数据

y\_train = []#训练集标签

x\_test = []#测试集数据

y\_test = []#测试集标签

for line in file:

line = line.strip().split(',')

if int(line[0]) <= 3000:#取前3000行作为训练集

line\_x = list(map(float, line[1:58]))

line\_x.append(1.0)#增加一个维度

x\_train.append(line\_x)

y\_train.append(float(line[58]))#取标签值

else:

line\_x = list(map(float, line[1:58]))

line\_x.append(1.0)

x\_test.append(line\_x)

y\_test.append(float(line[58]))

x\_train = np.mat(x\_train)

y\_train = np.mat(y\_train)

x\_test = np.mat(x\_test)

y\_test = np.mat(y\_test)

return x\_train,y\_train,x\_test,y\_test

#求损失函数的值

def lossfunc(x\_train,y\_train,w):

h=sigmoid(x\_train \* w)

y\_train = np.array(y\_train)[0]

loss = -(((y\_train \* np.array(np.log(h+0.000001).T))[0]+(1-y\_train)\*np.array(np.log(1-h+0.000001).T) )/np.shape(x\_train)[0])

return sum(loss[0])

def sigmoid(z):

h=1.0/(1+np.exp(-z))

return h

#梯度下降法进行训练

def train(learningrate,iteration,x\_train,y\_train):

m = np.shape(x\_train)[0]

w = np.zeros((58, 1))#给w初始化为0

loss = []

L=2

decay=0.25

for k in range(iteration):

learningrate=learningrate\*decay\*\*(k/iteration)

h = sigmoid(x\_train \* w) #sigmoid函数

grad=np.transpose(x\_train) \* (h - y\_train.T)/m#求梯度

w = (1-L\*learningrate/m)\*w - learningrate\*grad#梯度下降法公式,L2正则化

loss.append(lossfunc(x\_train, y\_train, w))#损失函数

return w.T,loss

#测试

def test(x\_test,y\_test,w):

num\_right = 0#记录测试正确数

for i in range(np.shape(x\_test)[0]):

x = np.array(x\_test[i])#取一组数据

x = x[0]

h = sigmoid(x\*w.T)#通过sigmoid函数计算测试值

if int(h+0.5)/1 == y\_test[0,i]:#将测试值与实际值对比，并记录测试正确的此时

num\_right += 1

return num\_right/np.shape(x\_test)[0]#返回准确率

#归一化处理

def z\_score\_normalization(x):

for i in range(np.shape(x)[1]):

std = np.std(x[:,i])#求方差

if std == 0.0 :

x[:,i] = 1

else:

x[:, i] = preprocessing.scale(x[:, i])#通过sklearn库进行归一化

return x#返回归一化后的数据集

def min\_max\_normalization(x):

for i in range(np.shape(x)[1]):

min = np.min(x[:,i])#最小值

max = np.max(x[:,i])#最大值

if max == min:

x[:, i] = 0.5

else:

x[:, i] = (x[:, i] - min) / (max - min)#最小值最大值归一化

return x#返回归一化后的结果

if \_\_name\_\_ == '\_\_main\_\_':

x\_train,y\_train,x\_test,y\_test = loadfile()

x\_train = z\_score\_normalization(x\_train)

x\_test = z\_score\_normalization(x\_test)

# x\_train = min\_max\_normalization(x\_train)

# x\_test = min\_max\_normalization(x\_test)

learningrate = 7.5

iteration = 1000

w, loss1 = train(learningrate, iteration, x\_train, y\_train)

rate = test(x\_test, y\_test, w)

print('学习率:',learningrate,'，迭代次数:',iteration,'，正确率:', rate)

learningrate = 1

w, loss2 = train(learningrate, iteration, x\_train, y\_train)

rate = test(x\_test, y\_test, w)

print('学习率:', learningrate, '，迭代次数:', iteration, '，正确率:', rate)

learningrate = 10

w, loss3 = train(learningrate, iteration, x\_train, y\_train)

rate = test(x\_test, y\_test, w)

print('学习率:', learningrate, '，迭代次数:', iteration, '，正确率:', rate)

learningrate = 5

w, loss4 = train(learningrate, iteration, x\_train, y\_train)

rate = test(x\_test, y\_test, w)

print('学习率:', learningrate, '，迭代次数:', iteration, '，正确率:', rate)

kl = [i for i in range(iteration)]

plt.plot(kl, loss1, 'b-')

plt.plot(kl, loss2, 'r-')

plt.plot(kl, loss3, 'g-')

plt.plot(kl, loss4, 'y-')

plt.show()

# 参考文献

[1] 李航 . 统计学习方法[m] 清华大学出版社，2012.3