**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： U201814737

姓 名： 王祺霖

成 绩：

指导教师： 邹复好

**完成日期： 2020年 6 月 24 日**

# PROJECT

## 一、实验题目：个人收入预测

## 二、实验要求

（1）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；

（2）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。

## 三、算法设计

（一）思路分析

这是一个典型的二分类问题，结合课上所学内容，决定采用Logistic回归算法。

与线性回归用于预测不同，Logistic回归则常用于分类(通常是二分类问题)。Logistic回归实质上就是在普通的线性回归后面加上了一个sigmoid函数，把线性回归预测到的数值压缩成为一个概率，进而实现二分类。

在损失函数方面，Logistic回归并没有使用传统的欧式距离来度量误差，而使用了交叉熵(用于衡量两个概率分布之间的相似程度)。

1. 数据预处理

本次project的数据相对简单纯净，在数据预处理方面并不需要花太多精力。

首先是空值处理(尽管没看到空值，但为了以防万一，还是做一下)，所有空值用0填充。

接着就是把数据范围尽量scale到同一个数量级上，观察数据后发现，多数数据值为0，非0值也都在1附近，只有倒数第二列和倒数第三列数据值较大，可以将这两列分别除上每列的平均值，把数值范围拉到1附近。

由于并没有给出这57个属性具体是什么属性，因此无法对数据进行进一步的挖掘应用。

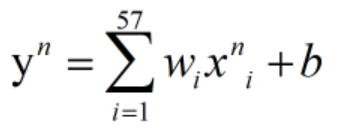
上述操作完成后，将表格的第2列至58列取出为x(shape为4000×57)，将最后一列取出做label y(shape为4000X1)。进一步划分训练集和验证集，分别取x、y中前3000个样本为训练集x\_test(shape为3000×57)，y\_test(shape为3000×1)，后500个样本为验证集x\_val(shape为1000×57)，y\_val(shape为1000×1)。

数据预处理到此结束。

### 建立模型

### 1. 线性回归

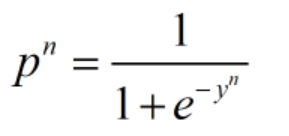
　　先对数据做线性回归，得出每个样本对应的回归值。下式为对第n个样本IMG_256的回归，回归结果为IMG_257。



y\_pre = weights.dot(x\_val[j, :]) + bias

### 2. sigmoid函数压缩回归值

　　之后将回归结果送进sigmoid函数，得到概率值。

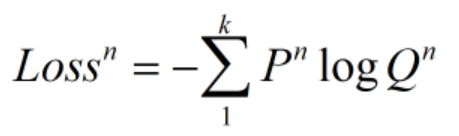


sig = 1 / (1 + np.exp(-y\_pre))

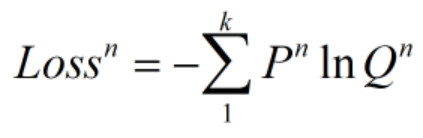
### **3. 误差反向传播**

　　众所周知，不管线性回归还是Logistic回归，其关键和核心就在于通过误差的反向传播来更新参数，进而使模型不断优化。因此，损失函数的确定及对各参数的求导就成了重中之重。在分类问题中，模型一般针对各类别输出一个概率分布，因此常用交叉熵作为损失函数。交叉熵可用于衡量两个概率分布之间的相似、统一程度，两个概率分布越相似、越统一，则交叉熵越小；反之，两概率分布之间差异越大、越混乱，则交叉熵越大。

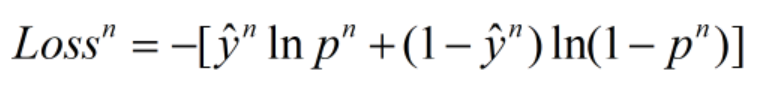
　　下式表示k分类问题的交叉熵，P为label，是一个概率分布，常用one\_hot编码。例如针对3分类问题而言，若样本属于第一类，则P为(1,0,0)，若属于第二类，则P为(0,1,0)，若属于第三类，则为(0,0,1)。即所属的类概率值为1，其他类概率值为0。Q为模型得出的概率分布，可以是(0.1,0.8,0.1)等。



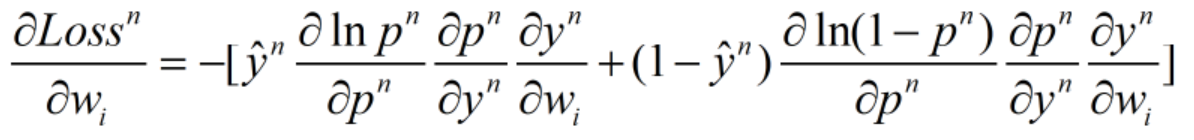
　　在实际应用中，为求导方便，常使用以e为底的对数。



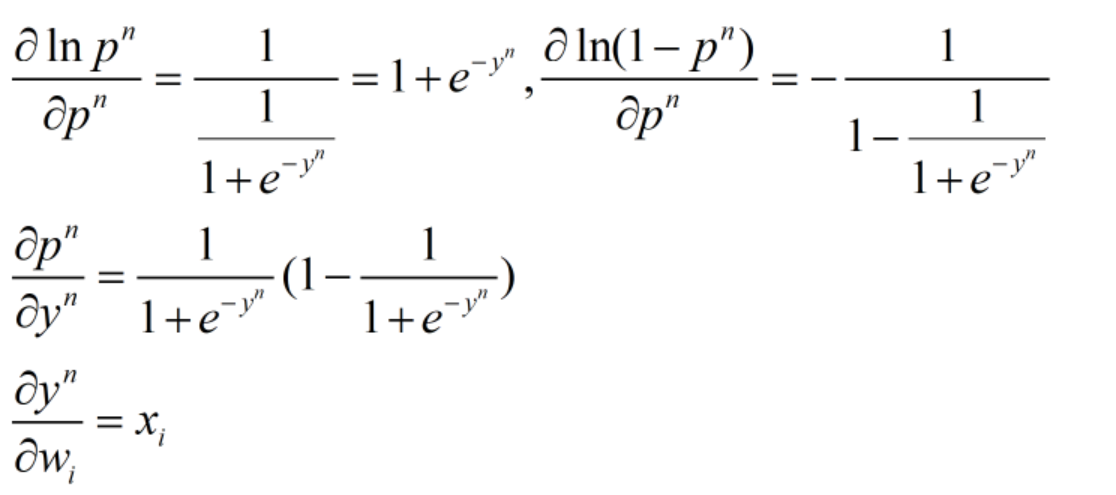
　　针对本次作业而言，虽然模型只输出了一个概率值p，但由于处理的是二分类问题，因此可以很快求出另一概率值为1-p，即可视为模型输出的概率分布为Q(p，1-p)。将本次的label视为概率分布P(y,1-y)，即Winner(label为1)的概率分布为(1,0)，分类为Losser(label为0)的概率分布为(0,1)。



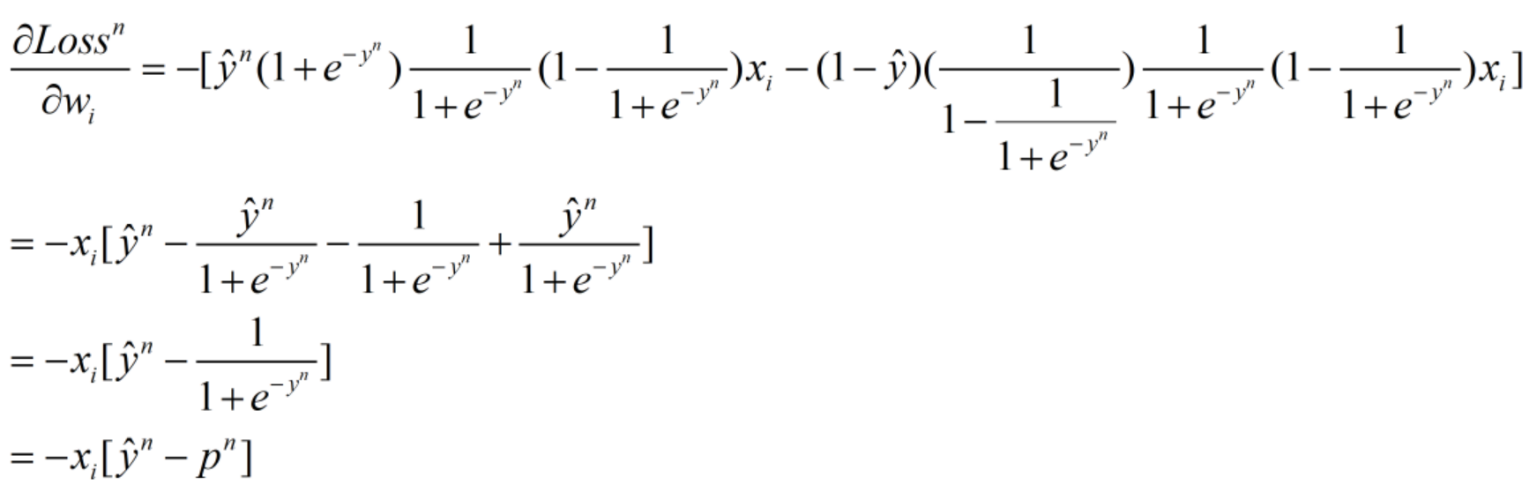
　　损失函数对权重w求偏导，可得：



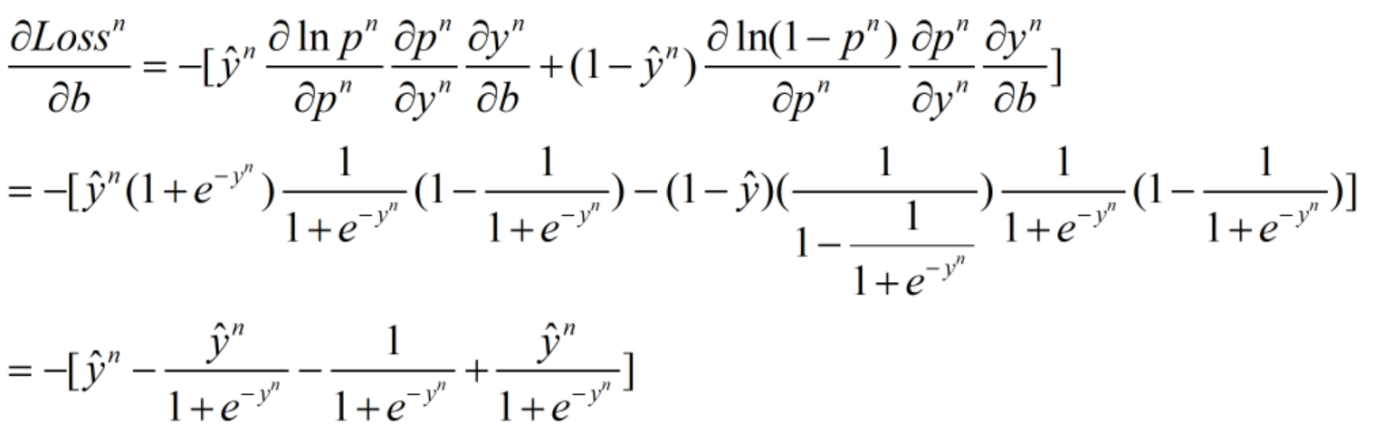
　　因为：



　　所以有：



 　　同理，损失函数对偏置b求偏导，可得：





### 4. 参数更新

　　求出梯度后，再拿原参数减去梯度与学习率的乘积，即可实现参数的更新。

## 四、实验环境与平台

VS2019 Python3.7（64-bit）

## 五、程序实现

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

# 更新参数，训练模型

def train(x\_train, y\_train, epoch):

num = x\_train.shape[0]

dim = x\_train.shape[1]

bias = 0 # 偏置值初始化

weights = np.ones(dim) # 权重初始化

learning\_rate = 1 # 初始学习率

reg\_rate = 0.001 # 正则项系数

bg2\_sum = 0 # 用于存放偏置值的梯度平方和

wg2\_sum = np.zeros(dim) # 用于存放权重的梯度平方和

losslist = {}

mislist = {}

for i in range(epoch):

b\_g = 0

w\_g = np.zeros(dim)

# 在所有数据上计算梯度，梯度计算时针对损失函数求导

for j in range(num):

y\_pre = weights.dot(x\_train[j, :]) + bias

sig = 1 / (1 + np.exp(-y\_pre))

b\_g += (-1) \* (y\_train[j] - sig)

for k in range(dim):

w\_g[k] += (-1) \* (y\_train[j] - sig) \* x\_train[j, k] + 2 \* reg\_rate \* weights[k]

b\_g /= num

w\_g /= num

# adagrad

bg2\_sum += b\_g \*\* 2

wg2\_sum += w\_g \*\* 2

# 更新权重和偏置

bias -= learning\_rate / bg2\_sum \*\* 0.5 \* b\_g

weights -= learning\_rate / wg2\_sum \*\* 0.5 \* w\_g

# 每训练3轮，输出一次在训练集上的正确率

loss = 0

acc = 0

result = np.zeros(num)

for j in range(num):

y\_pre = weights.dot(x\_train[j, :]) + bias

sig = 1 / (1 + np.exp(-y\_pre))

if sig >= 0.5:

result[j] = 1

else:

result[j] = 0

if result[j] == y\_train[j]:

acc += 1.0

loss += (-1) \* (y\_train[j] \* np.log(sig + 1e-10) + (1 - y\_train[j]) \* np.log(1 - sig + 1e-10))

losslist[i] = loss / num

mislist[i] = 1 - acc / num

if i % 3 == 0:

print('after {} epochs, the loss on train data is:'.format(i), loss / num)

print('after {} epochs, the acc on train data is:'.format(i), acc / num)

y = [losslist[0], losslist[1], losslist[2], losslist[3], losslist[4], losslist[5], losslist[6], losslist[7], losslist[8], losslist[9], losslist[10], losslist[11], losslist[12], losslist[13], losslist[14], losslist[15], losslist[16], losslist[17], losslist[18], losslist[19], losslist[20], losslist[21], losslist[22], losslist[23], losslist[24], losslist[25], losslist[26], losslist[27], losslist[28], losslist[29]]

x = np.linspace(0, 29, 30)

z = [mislist[0], mislist[1], mislist[2], mislist[3], mislist[4], mislist[5], mislist[6], mislist[7], mislist[8], mislist[9], mislist[10], mislist[11], mislist[12], mislist[13], mislist[14], mislist[15], mislist[16], mislist[17], mislist[18], mislist[19], mislist[20], mislist[21], mislist[22], mislist[23], mislist[24], mislist[25], mislist[26], mislist[27], mislist[28], mislist[29]]

plt.plot(x, y, ls="-", lw=2, label="train loss")

plt.plot(x, z, ls="-", lw=2, label="misclassification rate")

plt.legend()

plt.show()

return weights, bias

# 验证模型效果

def validate(x\_val, y\_val, weights, bias):

num = 1000

loss = 0

acc = 0

result = np.zeros(num)

for j in range(num):

y\_pre = weights.dot(x\_val[j, :]) + bias

sig = 1 / (1 + np.exp(-y\_pre))

if sig >= 0.5:

result[j] = 1

else:

result[j] = 0

if result[j] == y\_val[j]:

acc += 1.0

loss += (-1) \* (y\_val[j] \* np.log(sig + 1e-10) + (1 - y\_val[j]) \* np.log(1 - sig + 1e-10))

print('the test loss is:' , loss / num)

return acc / num

def main():

# 从csv中读取有用的信息

df = pd.read\_csv('spam\_train.csv')

# 空值填0

df = df.fillna(0)

# (4000, 59)

array = np.array(df)

# (4000, 57)

x = array[:, 1:-1]

# scale

x[:, -1] /= np.mean(x[:, -1])

x[:, -2] /= np.mean(x[:, -2])

# (4000, )

y = array[:, -1]

# 划分训练集与验证集

x\_train, x\_val = x[0:2999, :], x[2999:3999, :]

y\_train, y\_val = y[0:2999], y[2999:3999]

epoch = 30 # 训练轮数

# 开始训练

w, b = train(x\_train, y\_train, epoch)

# 在验证集上看效果

acc = validate(x\_val, y\_val, w, b)

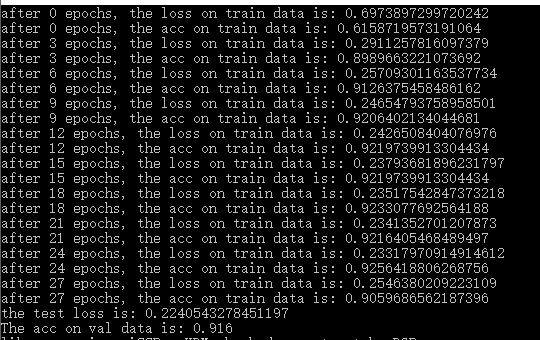
print('The acc on val data is:', acc)

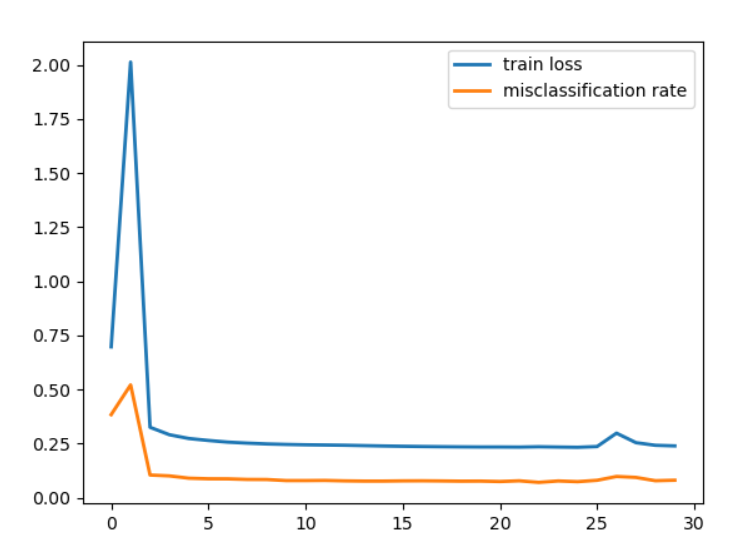
if \_\_name\_\_ == '\_\_main\_\_':

main()

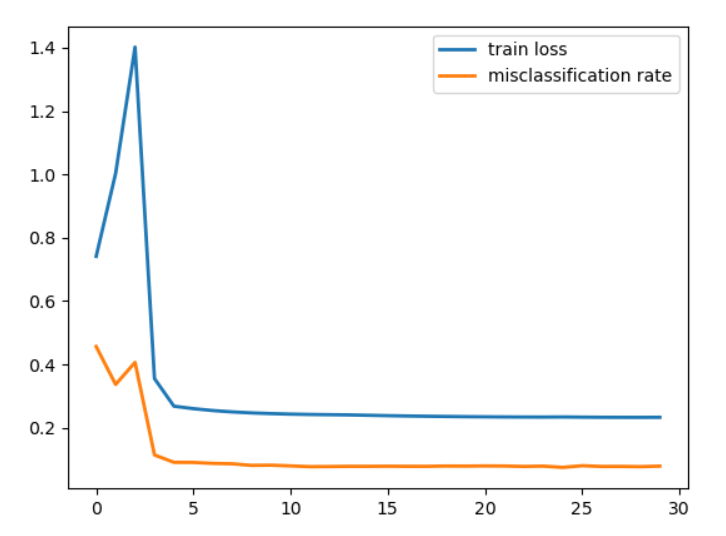
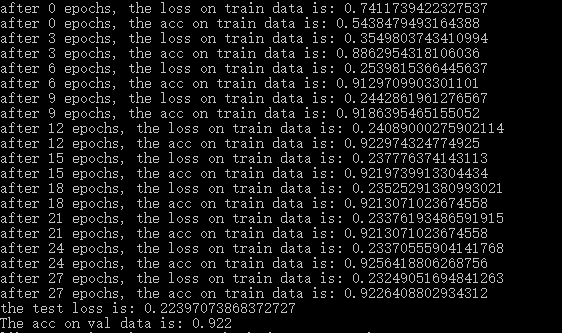
## 六、实验结果

1、learning\_rate = 1 reg\_rate = 0.001 时：

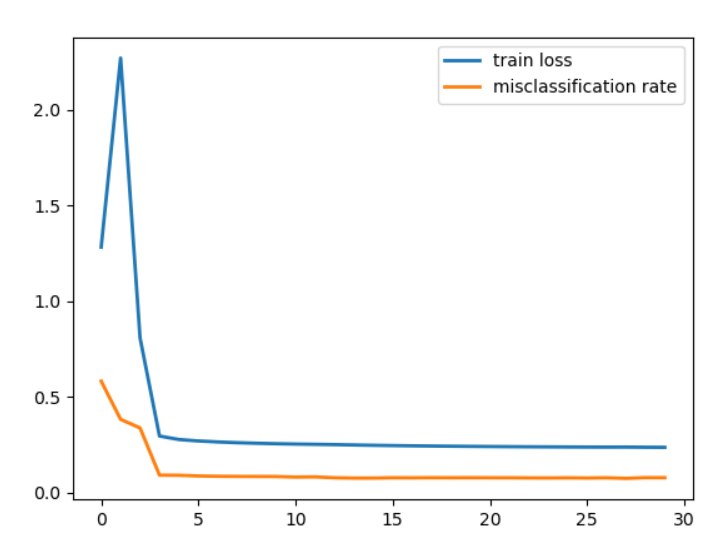
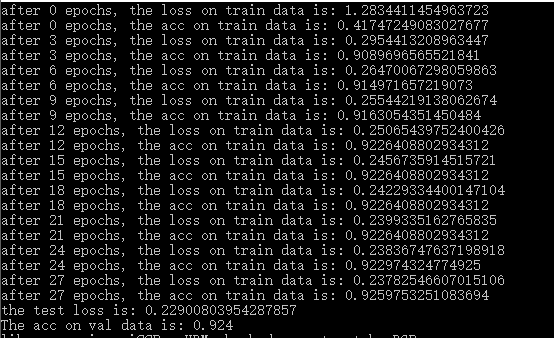




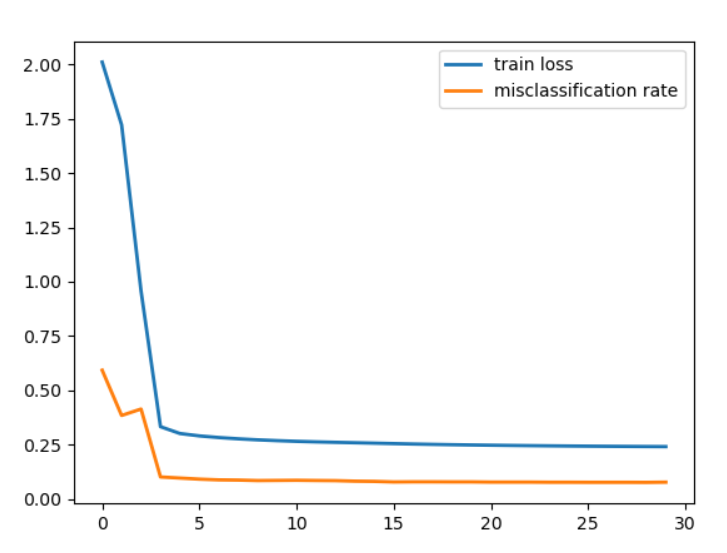
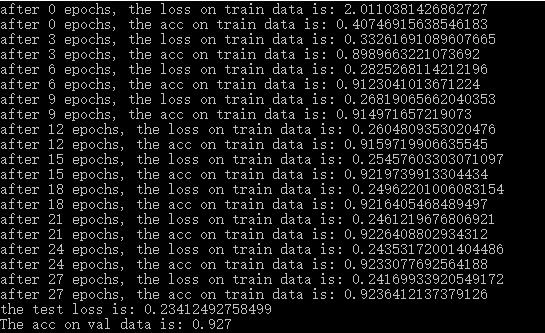
2、 learning\_rate = 0.9 reg\_rate = 0.001 时：



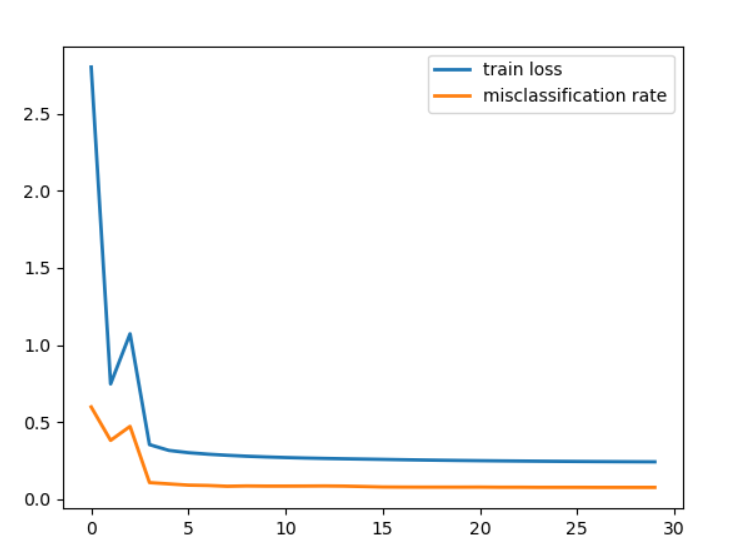
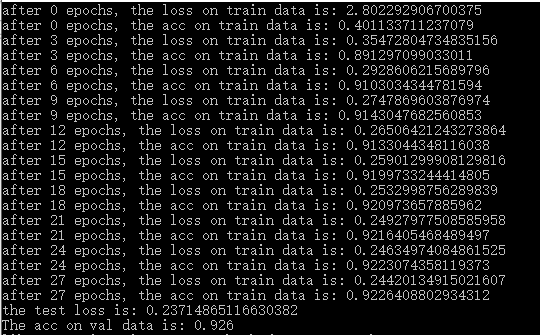
1. learning\_rate = 0.8 reg\_rate = 0.001 时：



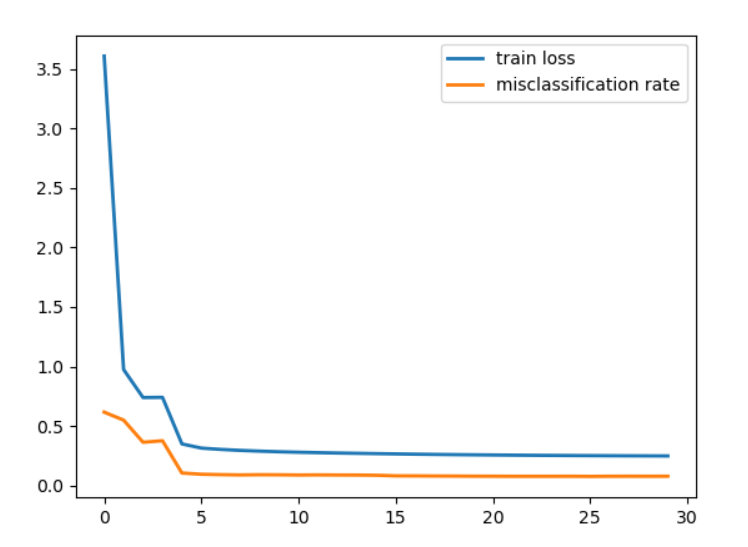
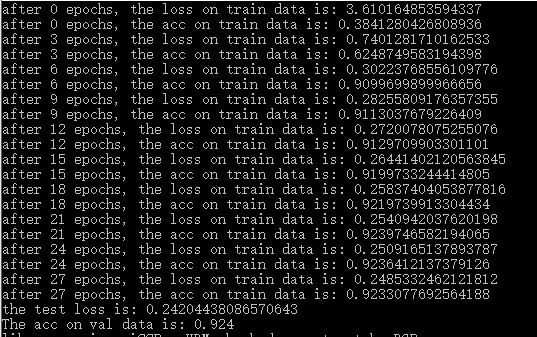
1. learning\_rate = 0.7 reg\_rate = 0.001 时：



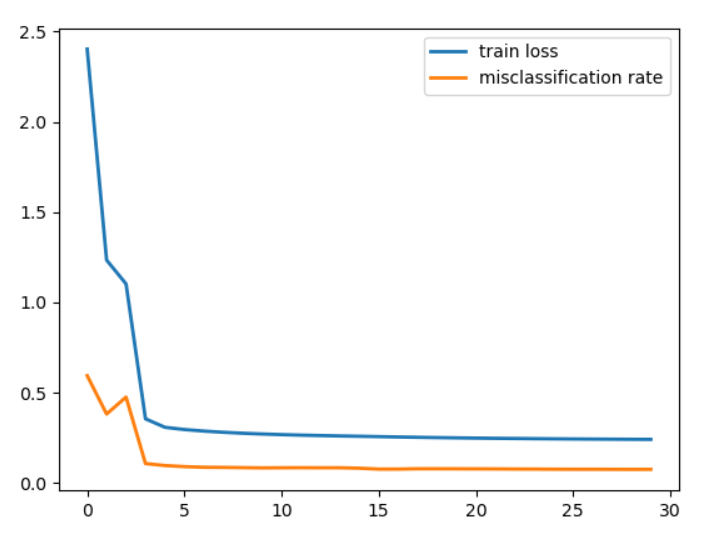
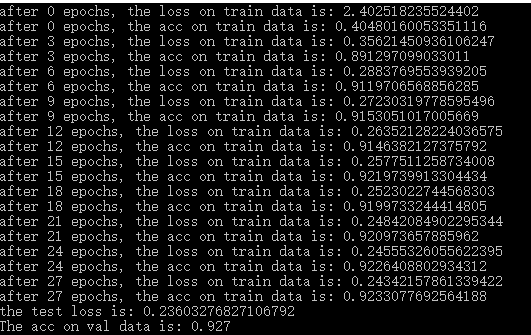
1. learning\_rate = 0.6 reg\_rate = 0.001 时：



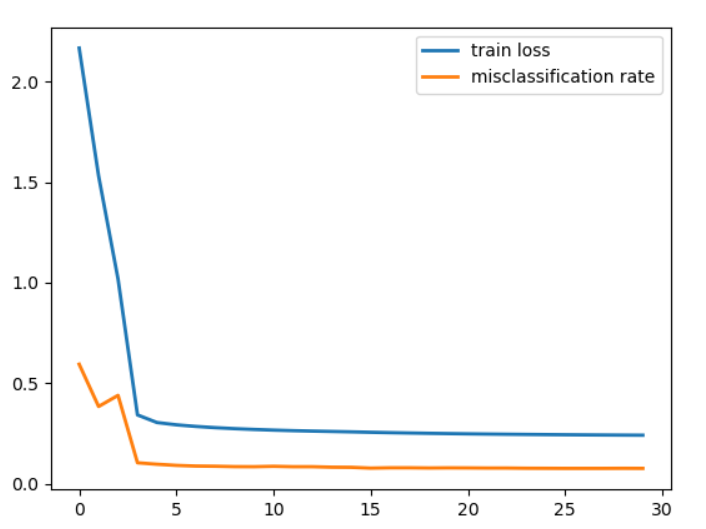
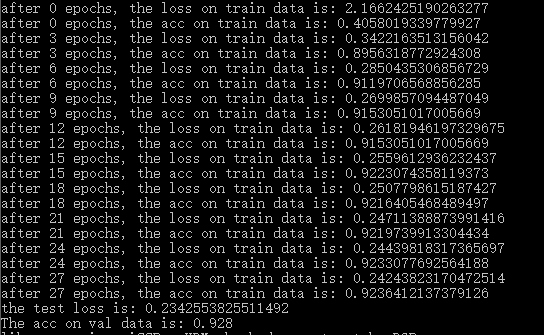
1. learning\_rate = 0.5 reg\_rate = 0.001 时：



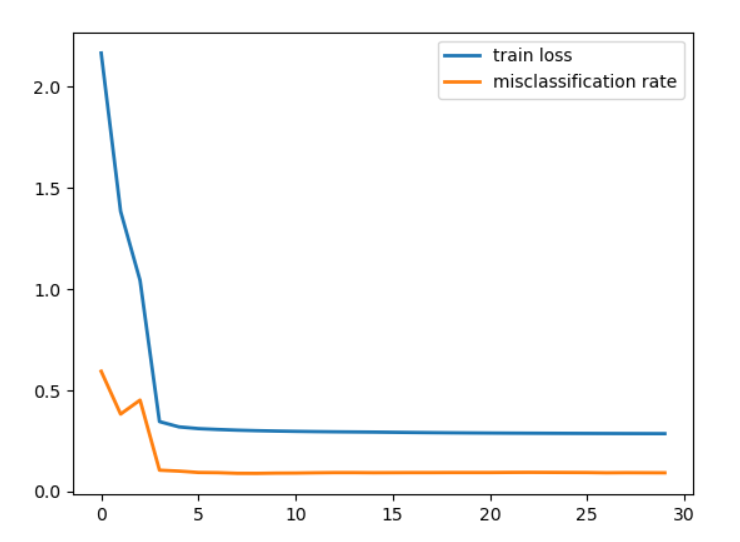
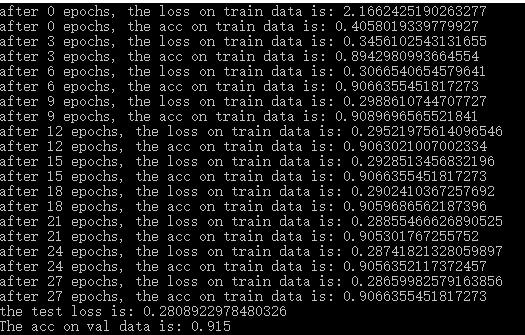
1. learning\_rate = 0.65 reg\_rate = 0.001 时：



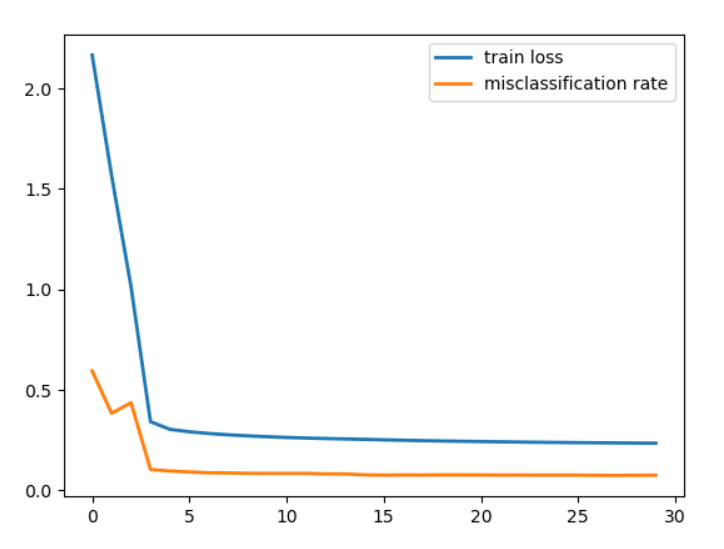
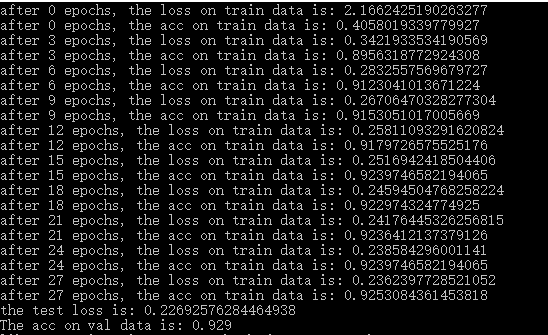
1. learning\_rate = 0.68 reg\_rate = 0.001 时：



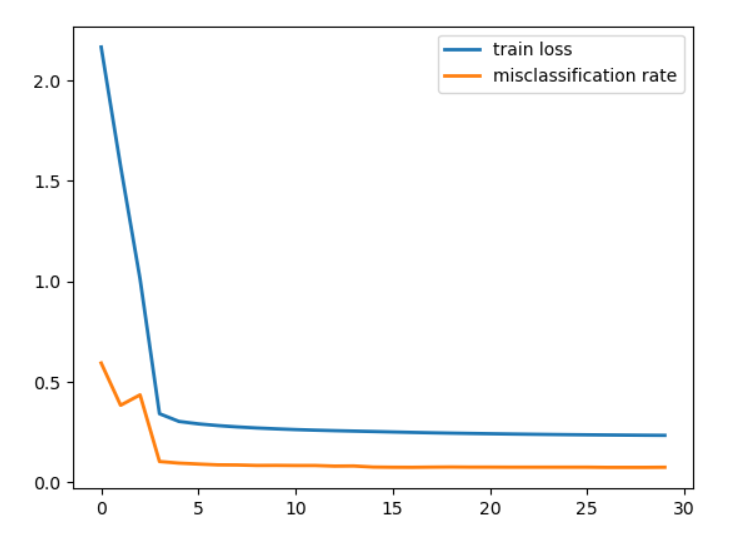
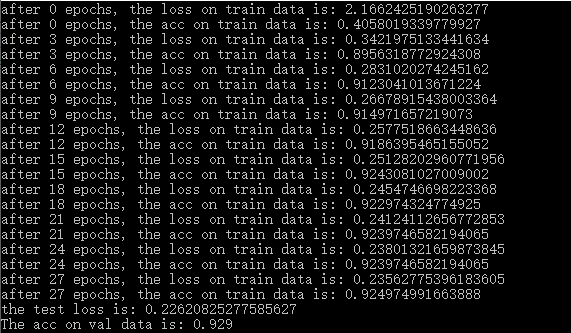
1. learning\_rate = 0.68 reg\_rate = 0.01 时：



1. learning\_rate = 0.68 reg\_rate = 0.0001 时：



1. learning\_rate = 0.68 reg\_rate = 0.00001 时：



## 七、结果分析

由于进行的是30次训练，最佳学习率相对较大，在0.68左右。当正则化参数在0.00001时，测试正确率可以到达92.9%，测试loss为22.62%。

如果适当增加训练集数量，可以在一定程度上提高正确率，降低loss。

由于时间不太充裕，所以并没有通过动态调整学习率的方法来求出准确的最佳学习率是多少。在暑假有空的时候可以进行深入探究。同时在之后的学习中，我也会继续探究在开始几次训练中loss和错误率反弹的原因。

# 参考文献

[1]XXXX

[2]XXXXXX

[3]XXXXX