实验三：逻辑回归

姓名： 孙武周 学号：2021113501

- 实验目的

理解和掌握逻辑回归模型基本原理和方法，学会使用逻辑回归模型对分类问题进行建模和预测，掌握分类问题上模型评估方法。

- 实验内容

编程实现逻辑回归模型，在给定数据集上，绘制损失函数曲线图。使用混淆矩阵、错误率、精度、查全率、查准率、F1 指标评估逻辑回归模型性能表现。

- 实验环境

python

numpy

matplotlib

- 实验代码(关键代码、中文注释、必要说明，源代码随实验报告一同提交)

（1）必要说明：

a) 使用 numpy 编写的模型和 sklearn 中的模型分别训练，然后对比两种模型混淆矩阵，错误率，精度，查全率，查准率，F1 指标

（2）关键代码：

```
import csv
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from numpy import mat, ravel
from sklearn.metrics import confusion_matrix, accuracy_score, recall_score, precision_score, f1_score

#sigmoid 函数
def sigmoid(z):
    return (1/(1+np.exp(-z))).reshape((-1,1))

#模型预测函数
def prediction(theta,x):
    #将大于 0.5 的变成 1，小于 0.5 的变成 0
```

```
    y=np.where(sigmoid(np.dot(x,theta))>0.5,1,0)
    return y

#模型损失以及梯度计算函数
def costFunction(theta,x,y):
    m=len(y)
    #模型分类数据
    h=sigmoid(np.dot(x, theta))
    #将预测计算中不合格数据矫正
    one_index,zero_index=np.argwhere(h>=1),np.argwhere(h<=0)
    h[one_index]=1-1e-10
    h[zero_index] = 1e-10
    #损失值
    loss=(-1/m)*np.sum(y*np.log(h)+(1-y)*np.log(1-h))
    #梯度
    grad=(1/m)*np.dot(x.T,(h-y))
    return loss,grad

#数据标准化函数
def data_score_stdlize(data):
    m,n=data.shape
    tempdata=data.copy()
    #对每一中属性的所有数据进行标准化
    for i in range(n):
        mu=np.mean(tempdata[:,i])
        sigma=np.std(tempdata[:,i])
        tempdata[:,i]=(tempdata[:,i]-mu)/sigma
    return tempdata


#读取数据函数，返回属性和标签
def read_data(filename):
    data=[]
    with open(filename) as csvfile:
        csvreader = csv.reader(csvfile)
        #跳过第一行
        header = next(csvreader)
        data = [row for row in csvreader]
    data = np.array(data).astype(float)
    return data[:,:-1],data[:,-1]

#读取数据以及数据标准化
train_filename='experiment_03_training_set.csv'
test_filename='experiment_03_testing_set.csv'
x_train,y_train=read_data(train_filename)
x_test,y_test=read_data(train_filename)
y_train=y_train.reshape((-1,1))
```

```
y_test=y_test.reshape((-1,1))
x_train_std=data_score_stdlize(x_train)
x_test_std=data_score_stdlize(x_test)

#改变训练集维度，为 x 加一维
x_train_std=np.concatenate((np.ones((x_train_std.shape[0],1)),
                            x_train_std),axis=1)
x_test_std=np.concatenate((np.ones((x_test_std.shape[0],1)),
                           x_test_std),axis=1)
train_num=x_train_std.shape[0]
train_feature=x_train_std.shape[1]


#初始化系数矩阵,系数权重为 1
theta=np.ones((train_feature,1))

#设置超参数
alpha=0.1
train_steps=500

#开始使用梯度训练模型
loss=[]
index=np.arange(0,train_steps,1)
for i in range(train_steps):
    per_loss,grad=costFunction(theta,x_train_std,y_train)
    theta=theta-alpha*grad#模型更新
    loss.append(per_loss)

#定义 sklearn 中的模型并训练
skmodel=LogisticRegression()
skmodel.fit(x_train_std,ravel(y_train))

#分别计算自己的模型和 sklearn 中模型的预测值
y_pred1=skmodel.predict(x_test_std)
y_pred2=prediction(theta,x_test_std)

# 计算混淆矩阵
cm_pred1 = confusion_matrix(y_test,y_pred1)
cm_pred2 = confusion_matrix(y_test,y_pred2)

# 计算错误率
acc_pred1 =  accuracy_score(y_test, y_pred1)
acc_pred2 =  accuracy_score(y_test, y_pred2)

# 计算错误率
err_pred1 = 1 - accuracy_score(y_test, y_pred1)
err_pred2 = 1 - accuracy_score(y_test, y_pred2)
```

```
# 计算精度
prec_pred1 = precision_score(y_test, y_pred1)
prec_pred2 = precision_score(y_test, y_pred2)

# 计算查全率
rec_pred1 = recall_score(y_test, y_pred1)
rec_pred2= recall_score(y_test, y_pred2)


#计算 F1 分数
f1_pred1 = f1_score(y_test, y_pred1)
f1_pred2 = f1_score(y_test, y_pred2)

#输出结果
print("Confusion matrix (skmodel):\n", cm_pred1)
print("Confusion matrix (mymodel):\n", cm_pred2)

print("Accuracy rate (skmodel):", acc_pred1)
print("Accuracy rate (mymodel):", acc_pred2)

print("Error rate (skmoedl):", err_pred1)
print("Error rate (mymodel):", err_pred2)

print("Precision (skmodel):", prec_pred1)
print("Precision (mymodel):", prec_pred2)

print("Recall (skmodel):", rec_pred1)
print("Recall (mymodel):", rec_pred2)

print("F1 score (skmodel):", f1_pred1)
print("F1 score (mymodel):", f1_pred2)

#绘图
plt.plot(index,loss,c='blue',marker='o',linestyle='-',label='mymodel')
font={'family':'Times New Roman','weight':'normal','size':10}
plt.xticks(fontproperties='Times New Roman',fontsize=10)
plt.xticks(fontproperties='Times New Roman',fontsize=10)
plt.xlabel(u'train_steps')
plt.ylabel(u'loss each step')
plt.legend(loc=1,prop=font)
plt.show()
```

- 结果分析（列表、绘图对结果分析）

初始权值设为$\boldsymbol{w} = [1,1,\ldots,1]$，学习率设为 0.1，迭代次数为 500。
损失曲线迭代图：

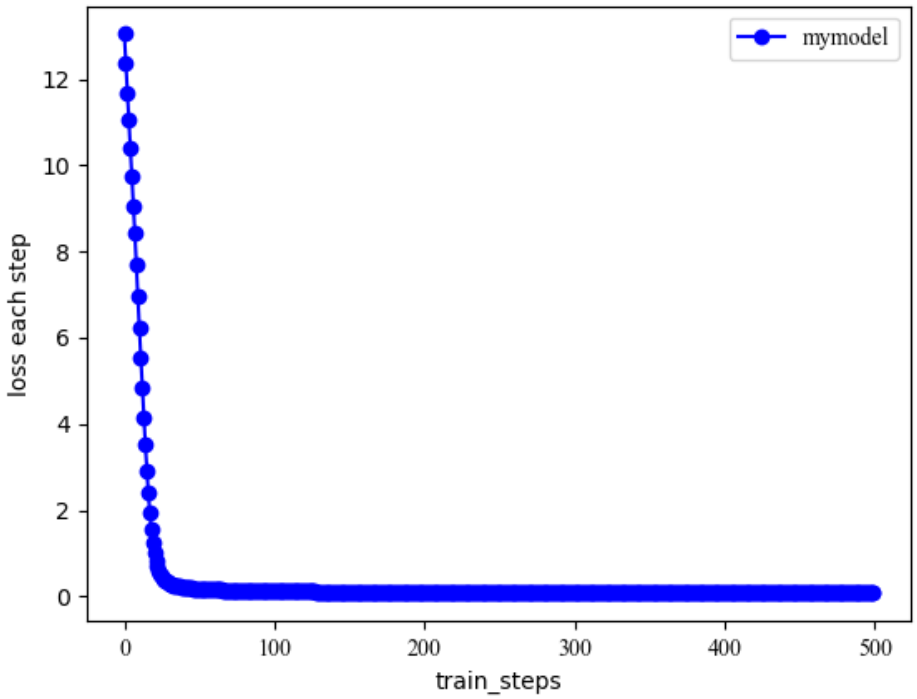

图 1

程序运行结果图

```
Confusion matrix (skmodel):
 [[181   5]
 [  2 312]]
Confusion matrix (mymodel):
 [[180   6]
 [  2 312]]
Accuracy rate (skmodel): 0.986
Accuracy rate (mymodel): 0.984
Error rate (skmoedl): 0.014000000000000012
Error rate (mymodel): 0.016000000000000014
Precision (skmodel): 0.9842271293375394
Precision (mymodel): 0.9811320754716981
Recall (skmodel): 0.9936305732484076
Recall (mymodel): 0.9936305732484076
F1 score (skmodel): 0.9889064976228209
F1 score (mymodel): 0.9873417721518988
```

混淆矩阵：

mymodel：

真实情况	预测结果	
	正例	反例
正例	312	2
反例	5	181

skmodel：

真实情况	预测结果	
	正例	反例
正例	312	2
反例	6	180

评价指标：

mymodel：

指标	数值
错误率（error rate）	0.016000000000000014
精度（accuracy）	0.984
查准率（precision）	0.9811320754716981
查全率（recall）	0.9936305732484076
F1	0.9873417721518988

skmodel：

指标	数值
错误率（error rate）	0.014000000000000012
精度（accuracy）	0.986
查准率（precision）	0.9842271293375394
查全率（recall）	0.9936305732484076
F1	0.9889064976228209

结果分析：

（1） 图 1 的损失函数曲线符合预期，模型能够很快收敛。

（2） 使用 numpy 编写的模型与 sklearn 的模型在各种评价指标下相差不大，模型在测试集上分类精度能够达到 98%，能够很好的完成所给数据集分类任务。