# 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：
   1. 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，学习率设为0.1，迭代次数为500。

损失曲线迭代图：

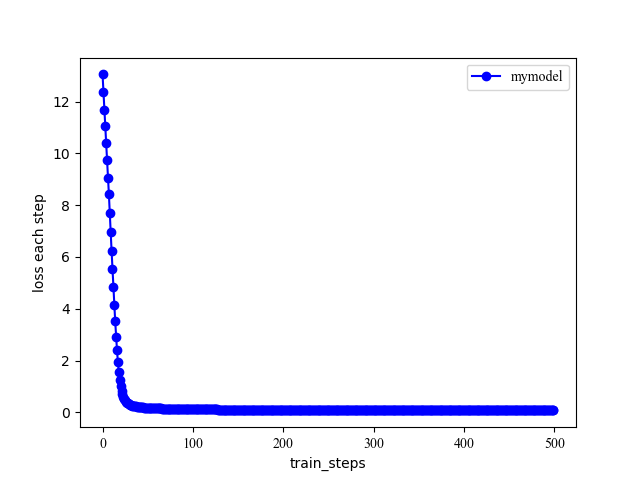
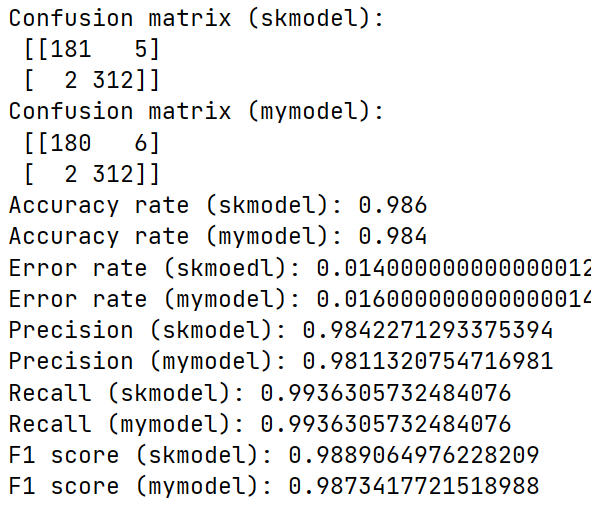


图1

程序运行结果图



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。