莆田学院新工科产业学院

课程名称：数据挖掘与机器学习

题 目：误差累积BP算法

学生姓名：inter

学 号：2019XXXXXXXX

专业班级：大数据19X

2023 年 12 月 9 日

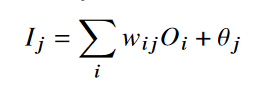
# 一．概述

BP算法是“误差反向传播”的简称，是**一种与最优化方法（如梯度下降法）结合使用的，用来训练人工神经网络的常见方法**。 该方法对网络中所有权重计算损失函数的梯度。 这个梯度会反馈给最优化方法，用来更新权值以最小化损失函数。

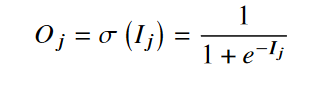
# 二．技术原理

**BP网络：用BP算法训练的多层前馈神经网络。**

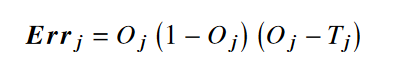
图为 BP 神经网络示意图，本文中 𝑛 为 1。本文使用的神经网络为三层，对于输入层的输入单元 𝑗 而言，其输出 𝑂 𝑗 就是它的输入 𝐼 𝑗 。对于隐藏层或者输出层的单元 𝑗 而言其输入为：



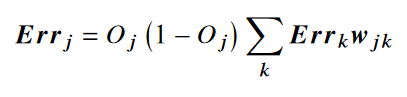
隐藏层或者输出层的单元 𝑗 的输出为：

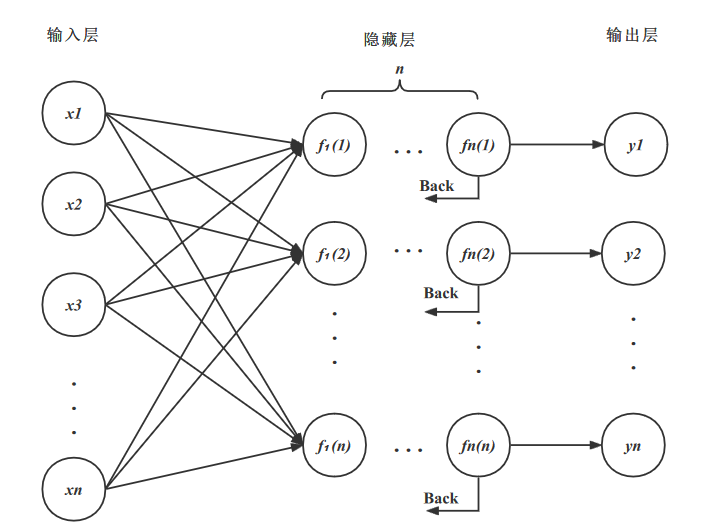


输出层的每个单元的误差为：



隐藏层的误差为：





**需通过学习确定的参数数目：(d+1)\*q+(q+1)\*l 。**

# 三.方案设计

### 一、算法的设计分为如下模块：

#### 1. sigmoid：

定义了一个激活函数：首先是一种非线性变化，讲输入值映射为0or1， 决定了后续神经元是否需要激活。 并且Sigmoid函数的导数可以在0到1之间平滑地变化。这种特性在反向传播算法中非常有用，因为它可以帮助调整神经网络的权重，减少误差，提高模型的准确性。

#### 2. sigmoid\_prime:

用来计算 Sigmoid 函数的导数。在神经网络中，当我们使用 Sigmoid 激活函数时，其导数（或者梯度）在反向传播（backpropagation）中是非常重要的。在反向传播算法中，需要计算损失函数对神经元输出值的导数，然后根据链式法则计算这些值对权重的导数。使用 Sigmoid 函数的导数，可以帮助在训练神经网络时有效地更新权重，以减小误差

#### 3. feed\_forward:

前向传输计算输出神经元的值

#### 4. MSGD:

小批量随机梯度下降算法，1.随机打乱训练集顺序2.划分为多个小批量训练集3.对每个小批量进行参数更新4.循环执行多个epochs

#### 5. updata\_WB\_by\_mini\_batch

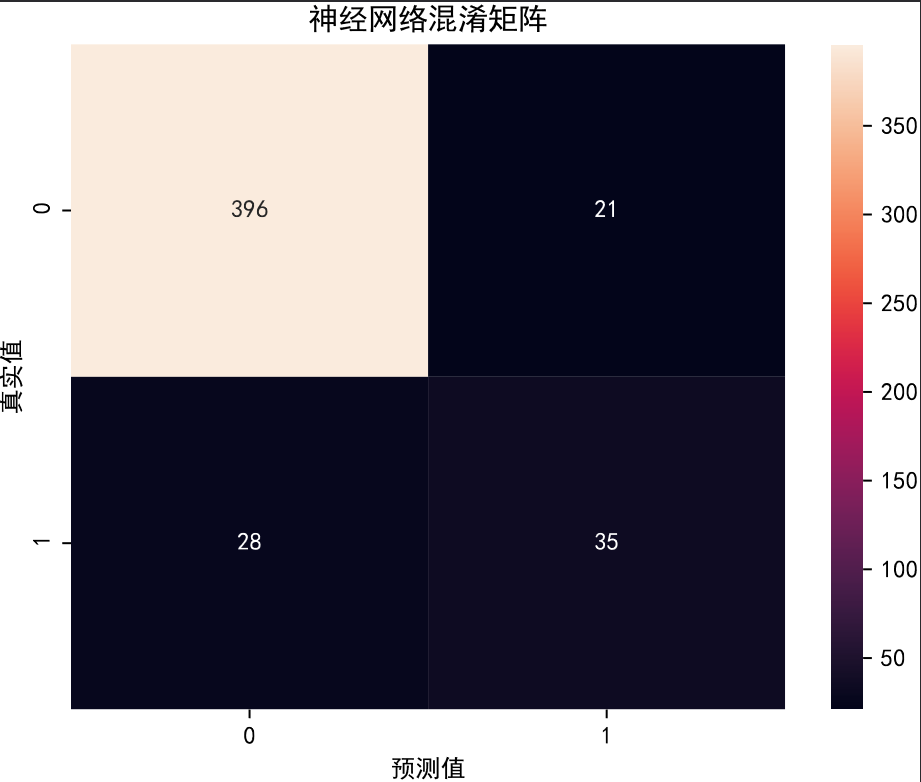
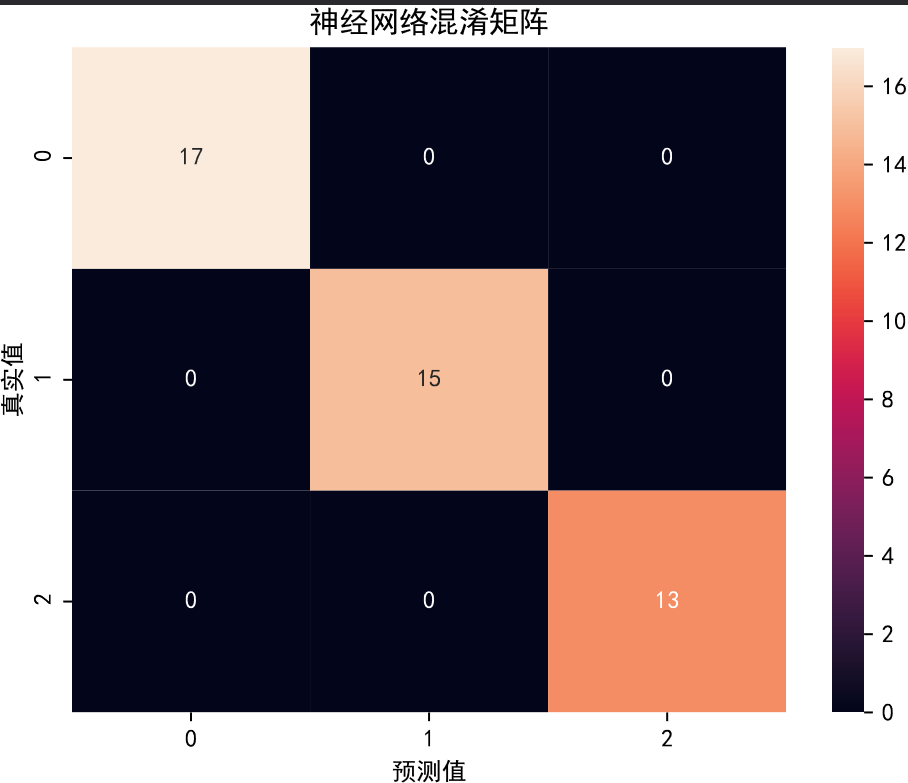
权重（weights）和偏置（biases）的更新：**1**. **初始化2. 对每个小样本计算偏导数3. 累加偏导数4. 更新权重和偏置**

#### 6.back\_propagation

反向传播（backpropagation）算法的实现，将前向传播存储在activations里，delta存储输出层的误差，使用该误差向前传播计算各层误差，计算各层权重和偏置的梯度（更新量），返回更新完的偏置和权重梯度。

### 二、评价指标结果好坏分析

采取sklearn库中函数，计算精确度，召回率，F1，准确率，并采取seaborn库绘制混淆矩阵如下。



四.改进过程

对鸢尾花数据集中的string指标进行编码，标准化，使得后面使用算法时不因为标准不同造成某指标的权重过大。

# 五.数据集

本次使用了两个数据集进行预测。分别是UCI公开数据集Iris和Wine，均为经典数据集。

IRIS：Iris数据集是常用的分类实验数据集，由Fisher, 1936收集整理。Iris也称鸢尾花卉数据集，是一类多重变量分析的数据集。数据集包含150个数据样本，分为3类，每类50个数据，每个数据包含4个属性。可通过花萼长度，花萼宽度，花瓣长度，花瓣宽度4个属性预测鸢尾花卉属于（Setosa，Versicolour，Virginica）三个种类中的哪一类。

Wine：Wine葡萄酒数据集是来自UCI上面的公开数据集，这些数据是对意大利同一地区种植的葡萄酒进行化学分析的结果，这些葡萄酒来自三个不同的品种。该分析确定了三种葡萄酒中每种葡萄酒中含有的13种成分的数量。从UCI数据库中得到的这个wine数据记录的是在意大利某一地区同一区域上三种不同品种的葡萄酒的化学成分。

# 六.小结

手动实现一个简单反向传播BP神经网络，对python的使用理解加深了，更了解了数据挖掘的实际实用意义。

# 七.附录（代码）

import numpy as np

import pandas as pd

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

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import classification\_report

import seaborn as sns

import matplotlib.pyplot as plt

import random

def sigmoid(x):

    '''

    激活函数

    '''

    return 1.0 / (1.0 + np.exp(-x))

def sigmoid\_prime(x):

    return sigmoid(x) \* (1 - sigmoid(x))

class BPClassification:

    def \_\_init\_\_(self, sizes):

        # 神经网络结构

        self.num\_layers = len(sizes)

        self.sizes = sizes

        # 初始化偏差，除输入层外， 其它每层每个节点都生成一个 biase 值（0-1）

        self.biases = [np.random.randn(n, 1) for n in sizes[1:]]

        # 随机生成每条神经元连接的 weight 值（0-1）

        self.weights = [np.random.randn(r, c)

                        for c, r in zip(sizes[:-1], sizes[1:])]

    def feed\_forward(self, a):

        '''

        前向传输计算输出神经元的值

        '''

        for b, w in zip(self.biases, self.weights):

            a = sigmoid(np.dot(w, a) + b)

        return a

    def MSGD(self, training\_data, epochs, mini\_batch\_size, eta, test\_data=None):

        '''

        小批量随机梯度下降法

        '''

        if test\_data: n\_test = len(test\_data)

        n = len(training\_data)

        for j in np.arange(epochs):

            # 随机打乱训练集顺序

            random.shuffle(training\_data)

            # 根据小样本大小划分子训练集集合

            mini\_batchs = [training\_data[k:k + mini\_batch\_size]

                           for k in np.arange(0, n, mini\_batch\_size)]

            # 利用每一个小样本训练集更新 w 和 b

            for mini\_batch in mini\_batchs:

                self.updata\_WB\_by\_mini\_batch(mini\_batch, eta)

            # 迭代一次后结果

            if test\_data:

                print("Epoch {0}: {1} / {2}".format(j, self.evaluate(test\_data), n\_test))

            else:

                print("Epoch {0}".format(j))

    def updata\_WB\_by\_mini\_batch(self, mini\_batch, eta):

        '''

        利用小样本训练集更新 w 和 b

        mini\_batch: 小样本训练集

        eta: 学习率

        '''

        # 创建存储迭代小样本得到的 b 和 w 偏导数空矩阵，大小与 biases 和 weights 一致，初始值为 0

        batch\_par\_b = [np.zeros(b.shape) for b in self.biases]

        batch\_par\_w = [np.zeros(w.shape) for w in self.weights]

        for x, y in mini\_batch:

            # 根据小样本中每个样本的输入 x, 输出 y, 计算 w 和 b 的偏导

            delta\_b, delta\_w = self.back\_propagation(x, y)

            # 累加偏导 delta\_b, delta\_w

            batch\_par\_b = [bb + dbb for bb, dbb in zip(batch\_par\_b, delta\_b)]

            batch\_par\_w = [bw + dbw for bw, dbw in zip(batch\_par\_w, delta\_w)]

        # 根据累加的偏导值 delta\_b, delta\_w 更新 b, w

        # 由于用了小样本，因此 eta 需除以小样本长度

        self.weights = [w - (eta / len(mini\_batch)) \* dw

                        for w, dw in zip(self.weights, batch\_par\_w)]

        self.biases = [b - (eta / len(mini\_batch)) \* db

                       for b, db in zip(self.biases, batch\_par\_b)]

    def back\_propagation(self, x, y):

        '''

        利用误差后向传播算法对每个样本求解其 w 和 b 的更新量

        x: 输入神经元，行向量

        y: 输出神经元，行向量

        '''

        delta\_b = [np.zeros(b.shape) for b in self.biases]

        delta\_w = [np.zeros(w.shape) for w in self.weights]

        # 前向传播，求得输出神经元的值

        a = x  # 神经元输出值

        # 存储每个神经元输出

        activations = [x]

        # 存储经过 sigmoid 函数计算的神经元的输入值，输入神经元除外

        zs = []

        for b, w in zip(self.biases, self.weights):

            z = np.dot(w, a) + b

            zs.append(z)

            a = sigmoid(z)  # 输出神经元

            activations.append(a)

        # 求解输出层δ

        delta = self.cost\_function(activations[-1], y) \* sigmoid\_prime(zs[-1])

        delta\_b[-1] = delta

        delta\_w[-1] = np.dot(delta, activations[-2].T)

        for lev in np.arange(2, self.num\_layers):

            # 从倒数第1层开始更新，因此需要采用-lev

            # 利用 lev + 1 层的 δ 计算 l 层的 δ

            z = zs[-lev]

            zp = sigmoid\_prime(z)

            delta = np.dot(self.weights[-lev + 1].T, delta) \* zp

            delta\_b[-lev] = delta

            delta\_w[-lev] = np.dot(delta, activations[-lev - 1].T)

        return (delta\_b, delta\_w)

    def evaluate(self, test\_data):

        test\_result = [(np.argmax(self.feed\_forward(x)), y)

                       for (x, y) in test\_data]

        return sum(int(x == y) for (x, y) in test\_result)

    def predict(self, test\_input):

        test\_result = [self.feed\_forward(x)

                       for x in test\_input]

        return test\_result

    def cost\_function(self, output\_a, y):

        '''

        损失函数

        '''

        return (output\_a - y)

    def accuracy(self,x\_test,y\_test):

        y\_predict = np.array([ np.argmax(i) for i in self.predict(x\_test)]).astype(y\_test.dtype)

        return np.sum(y\_test==y\_predict)/y\_test.shape[0]

    def macro\_precision(self,x\_test,y\_test):

        y\_unique = np.unique(y\_test)

        y\_predict = np.array([ np.argmax(i) for i in self.predict(x\_test)]).astype(y\_test.dtype)

        P = []

        for i in y\_unique:

            P.append(np.sum(y\_predict[y\_predict==y\_test]==i)/np.sum(y\_predict==i))

        return np.sum(P)/len(P)

    def macro\_recall(self,x\_test,y\_test):

        y\_unique = np.unique(y\_test)

        y\_predict = np.array([ np.argmax(i) for i in self.predict(x\_test)]).astype(y\_test.dtype)

        R = []

        for i in y\_unique:

            R.append(np.sum(y\_predict[y\_predict==y\_test]==i)/np.sum(y\_test==i))

        return np.sum(R) / len(R)

    def macro\_f1\_score(self,x\_test,y\_test):

        y\_unique = np.unique(y\_test)

        y\_predict = np.array([ np.argmax(i) for i in self.predict(x\_test)]).astype(y\_test.dtype)

        F = []

        for i in y\_unique:

            p = np.sum(y\_predict[y\_predict==y\_test]==i)/np.sum(y\_predict==i)

            r = np.sum(y\_predict[y\_predict==y\_test]==i)/np.sum(y\_test==i)

            F.append((2\*p\*r)/(p+r))

        return np.sum(F) / len(F)

    def precision(self,x\_test,y\_test):

        y\_predict = np.array([ np.argmax(i) for i in self.predict(x\_test)]).astype(y\_test.dtype)

        y\_unique = np.unique(y\_test)

        return np.sum(y\_predict[y\_predict==y\_test]==y\_unique[0])/np.sum(y\_predict==y\_unique[0]),np.sum(y\_predict[y\_predict==y\_test]==y\_unique[1])/np.sum(y\_predict==y\_unique[1])

    def recall(self,x\_test,y\_test):

        y\_predict = np.array([ np.argmax(i) for i in self.predict(x\_test)]).astype(y\_test.dtype)

        y\_unique = np.unique(y\_test)

        return np.sum(y\_predict[y\_predict==y\_test]==y\_unique[0])/np.sum(y\_test==y\_unique[0]),np.sum(y\_predict[y\_predict==y\_test]==y\_unique[1])/np.sum(y\_test==y\_unique[1])

    def f1\_score(self,x\_test,y\_test):

        precision = self.precision(x\_test,y\_test)

        recall = self.recall(x\_test,y\_test)

        return 2\*precision[0]\*recall[0]/(precision[0]+recall[0]),2\*precision[1]\*recall[1]/(precision[1]+recall[1])

    def report(self,x\_test,y\_test,labels):

        y\_predict = np.array([ np.argmax(i) for i in self.predict(x\_test)]).astype(y\_test.dtype)

        report = classification\_report(y\_test,y\_predict,labels=labels)

        return report

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

    # ///////////////////////////////////////////////// Iris

    data = pd.read\_csv('data/D2.csv').drop(['Unnamed: 0'],axis = 1)

    X = np.array(data.drop(columns=['Species']))

    y = np.array(data['Species'])

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=2)

    # ----------------- SVM

    print('Iris数据集下------------------------------------------------------')

    model\_svm = SVC(kernel='rbf', random\_state=42)

    model\_svm.fit(X\_train, y\_train)

    print('SVM下------------------------------------------')

    print('训练集下：------------------------------------')

    print(classification\_report(y\_train, model\_svm.predict(X\_train), labels=[0,1,2]))

    print('测试集下：------------------------------------')

    print(classification\_report(y\_test, model\_svm.predict(X\_test), labels=[0,1,2]))

    plt.rcParams['font.sans-serif'] = ['SimHei']

    plt.rcParams['axes.unicode\_minus'] = False

    f, ax = plt.subplots()

    C3 = confusion\_matrix(y\_test, model\_svm.predict(X\_test), labels=[0,1,2])

    sns.heatmap(C3, annot=True, ax=ax)  # 画热力图

    ax.set\_title('SVM混淆矩阵')  # 标题

    ax.set\_xlabel('预测值')  # x轴

    ax.set\_ylabel('真实值')  # y轴

    # plt.savefig('实验三图片/3\_5.pdf')

    plt.show()

    # ----------------- BP

    print('BP神经网络下------------------------------------------')

    train\_x = [i.reshape(-1,1) for i in X\_train]

    train\_y = [np.array([[1 if y==0 else 0],

                           [1 if y==1 else 0],

                           [1 if y==2 else 0]])

                           for y in y\_train]

    train\_data = [[x, y] for x, y in zip(train\_x, train\_y)]

    test\_x = [i.reshape(-1,1) for i in X\_test]

    test\_data = [[x, y] for x, y in zip(test\_x, y\_test)]

    bp = BPClassification([4, 15, 3])

    bp.MSGD(train\_data, 1000, 10, 0.5)

    print('训练集下：------------------------------------')

    print(bp.report(train\_x, y\_train, labels=[0, 1,2]))

    print('测试集下：------------------------------------')

    print('accuracy:', bp.accuracy(test\_x, y\_test))

    print('macro\_precision:', bp.macro\_precision(test\_x, y\_test))

    print('macro\_recall:', bp.macro\_recall(test\_x, y\_test))

    print('macro\_f1\_score:', bp.macro\_f1\_score(test\_x, y\_test))

    print('micro版本==accuracy:', bp.accuracy(test\_x, y\_test))

    print(bp.report(test\_x, y\_test, labels=[0, 1,2]))

    plt.rcParams['font.sans-serif'] = ['SimHei']

    plt.rcParams['axes.unicode\_minus'] = False

    f, ax = plt.subplots()

    C5 = confusion\_matrix(y\_test, np.array([ np.argmax(i) for i in bp.predict(test\_x)]).astype(y\_test.dtype), labels=[0,1,2])

    sns.heatmap(C5, annot=True, ax=ax)  # 画热力图

    ax.set\_title('神经网络混淆矩阵')  # 标题

    ax.set\_xlabel('预测值')  # x轴

    ax.set\_ylabel('真实值')  # y轴

    # plt.savefig('实验三图片/3\_7.pdf')

    plt.show()

    # ///////////////////////////////////////////////// Wine

    data1 = pd.read\_csv('data/D3.csv').drop(['Unnamed: 0'], axis=1)

    X1 = np.array(data1.drop(columns=['quality']))

    y1 = np.array(data1['quality'])

    X\_train1, X\_test1, y\_train1, y\_test1 = train\_test\_split(X1, y1, test\_size=0.3, random\_state=3)

    # ----------------- SVM

    print('Wine数据集下------------------------------------------------------')

    model\_svm1 = SVC(kernel='rbf', random\_state=42)

    model\_svm1.fit(X\_train1, y\_train1)

    print('SVM下------------------------------------------')

    print('训练集下：------------------------------------')

    print(classification\_report(y\_train1, model\_svm1.predict(X\_train1)))

    print('测试集下：------------------------------------')

    print(classification\_report(y\_test1, model\_svm1.predict(X\_test1)))

    plt.rcParams['font.sans-serif'] = ['SimHei']

    plt.rcParams['axes.unicode\_minus'] = False

    f, ax = plt.subplots()

    C4 = confusion\_matrix(y\_test1, model\_svm1.predict(X\_test1), labels=[0,1])

    sns.heatmap(C4, annot=True, fmt='.0f', ax=ax)  # 画热力图

    ax.set\_title('SVM混淆矩阵')  # 标题

    ax.set\_xlabel('预测值')  # x轴

    ax.set\_ylabel('真实值')  # y轴

    # plt.savefig('实验三图片/3\_6.pdf')

    plt.show()

    # ----------------- BP

    print('BP神经网络下------------------------------------------')

    train\_x1 = [i.reshape(-1,1) for i in X\_train1]

    train\_y1 = [np.array([[1 if y==0 else 0],

                           [1 if y==1 else 0]])

                           for y in y\_train1]

    train\_data1 = [[x, y] for x, y in zip(train\_x1, train\_y1)]

    test\_x1 = [i.reshape(-1,1) for i in X\_test1]

    test\_data1 = [[x, y] for x, y in zip(test\_x1, y\_test1)]

    bp1 = BPClassification([11, 15, 2])

    bp1.MSGD(train\_data1, 1000, 10, 0.5)

    print('训练集下：------------------------------------')

    print(bp1.report(train\_x1, y\_train1, labels=[0, 1]))

    print('测试集下：------------------------------------')

    print('accuracy:', bp1.accuracy(test\_x1, y\_test1))

    print('macro\_precision:', bp1.macro\_precision(test\_x1, y\_test1))

    print('macro\_recall:', bp1.macro\_recall(test\_x1, y\_test1))

    print('macro\_f1\_score:', bp1.macro\_f1\_score(test\_x1, y\_test1))

    print('micro版本==accuracy:', bp1.accuracy(test\_x1, y\_test1))

    print(bp1.report(test\_x1, y\_test1, labels=[0, 1]))

    plt.rcParams['font.sans-serif'] = ['SimHei']

    plt.rcParams['axes.unicode\_minus'] = False

    f, ax = plt.subplots()

    C6 = confusion\_matrix(y\_test1, np.array([ np.argmax(i) for i in bp1.predict(test\_x1)]).astype(y\_test1.dtype), labels=[0,1])

    sns.heatmap(C6, fmt='.0f',annot=True, ax=ax)  # 画热力图

    ax.set\_title('神经网络混淆矩阵')  # 标题

    ax.set\_xlabel('预测值')  # x轴

    ax.set\_ylabel('真实值')  # y轴

    # plt.savefig('实验三图片/3\_8.pdf')

    plt.show()

参考：

《机器学习》周志华

《统计学习方法》李航