

**计算机与信息 学院实验报告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 实验课程： | Python数据分析 | | | | |
| 实验编号： | 实验十一 | | | | |
| 实验名称： | 感知机模型学习双输入或（or）函数 | | | | |
| 实验人员： | 学号 | 18111207248 | | | |
| 姓名 | 吴钰 | | | |
| 班级 | 18级计算机科学与技术（创新班） | | | |
| 实验日期： | 2020.12.10 | | | | |
| 实验室： | 2060402 | | | | |
|  |  | | | | |
| 实验评价： |  | | | | |
| 实验成绩： | |  | 评价日期： |  |
|  | 指导教师： | |  | | |

# 实验目的

编程程序，构建感知机模型，学习双输入或（or）函数，提交实验报告

# 二、实验要求

## 完成实验后请填写实验报告并上交。要求在实验报告中必须记录实验中遇到的问题及其问题解决方案。

# 三、实验内容

编程程序，构建感知机模型，学习双输入或（or）函数，提交实验报告

代码：

# -\*- coding: utf-8 -\*-

"""

Created on Wed Dec 30 10:16:33 2020

@author: i

"""

# 创建感知机类

# 属性是激活函数，学习率，权重，偏置项

# 方法是

# 1.初始化：传入激活函数，学习率，权重个数；初始化权重，偏置项为0

# 2.预测：传入一个样本，输出一个预测值

# 3.更新权重和偏置项：传入预测值，计算新的权重和偏置项

# 4.打印：打印权重，偏置项

# 初始化感知机

# :param w\_num: 权重个数

# :param rate: 学习率

# :param activator: 激活函数

class perceptron(object):

def \_\_init\_\_(self, w\_num, rate, activator):

self.rate = rate #传入学习率

self.activator = activator #传入激活函数

self.bias = 0.0 #偏置项初始化为0

self.weights = [0.0 for x in range(w\_num)] #权重初始化为0

# 预测一个样本

# :param sample: 被预测的一个样本

# :return: 通过激活函数后的预测值

def predict(self, sample):

sum = 0

for feature, weight in zip(sample, self.weights): #实现wx

sum += feature \* weight

sum += self.bias #实现wx+b

y\_predict = self.activator(sum) #激活

return y\_predict #返回预测值

def update(self, sample, label):

"""

用一个样本来更新权重和偏置项

:param sample: 一个样本

:param label: 它的标签

:return:

"""

output = self.predict(sample) #得到该样本的预测值

delta = label - output #预测值和真实值的差异

self.bias += delta \* self.rate #更新偏置项

self.weights = [w + self.rate \* delta \* x for x, w in zip(sample, self.weights)] #更新权重

def print(self):

print("weight\t:%s\n bias\t:%f\n" % (self.weights, self.bias)) #打印偏置项和权重

# 创建迭代训练感知机函数

# 输入：指定感知机，多个样本点，对应的标签，指定迭代次数

# 输出：训练好的感知机，主要是权重，偏置项

def activator(x):

"""

创建激活函数-阶跃函数

:param x: 自变量

:return: 若x为正数，则返回值是1；若x为负数，则返回值是0

"""

return 1 if x > 0 else 0

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

samples = [[1, 1], [0, 0], [1, 0], [0, 1]] #多个样本

labels = [1, 0, 0, 0] #标签

p = perceptron(2, 0.1, activator) #创建一个感知机，每个样本有2个特征，学习率是0.1,激活函数是activator

for i in range(10): #迭代10次

for sample, label in zip(samples, labels): #每次迭代中，所有样本均参与

p.update(sample, label) #每次用一个样本更新权重和偏置项

p.print() #打印10次迭代后的权重和偏置项

#预测

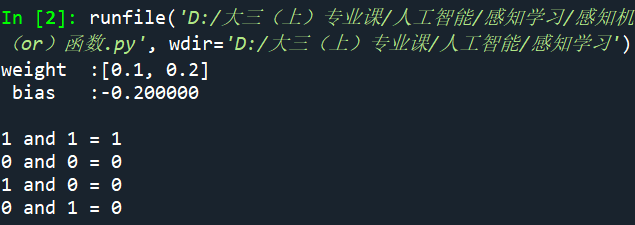
print('1 and 1 = %d' % p.predict([1, 1]))

print('0 and 0 = %d' % p.predict([0, 0]))

print('1 and 0 = %d' % p.predict([1, 0]))

print('0 and 1 = %d' % p.predict([0, 1]))

运行截图：



# 四、实验心得

通过这次实验，让我深入地理解感知机的原理和结构。感知机的思想很简单，比如我们在一个平台上有很多的男孩女孩，感知机的模型就是尝试找到一条直线，能够把所有的男孩和女孩隔离开。放到三维空间或者更高维的空间，感知机的模型就是尝试找到一个超平面，能够把所有的二元类别隔离开。如果我们找不到这么一条直线的话怎么办？找不到的话那就意味着类别线性不可分，也就意味着感知机模型不适合你的数据的分类。使用感知机一个最大的前提，就是数据是线性可分的。这严重限制了感知机的使用场景。它的分类竞争对手在面对不可分的情况时，比如支持向量机可以通过核技巧来让数据在高维可分，神经网络可以通过激活函数和增加隐藏层来让数据可分。感知机的基本概念：

线性可分数据集 ->超平面->确定w，b系数->根据损失函数来调整w，b。

感知机的算法表示方式（原始方法和对偶方法）

感知机的误份次数K 满足不等式。