

**计算智能导论**

**（感知器实现二分类）**

授课老师：张梦璇

学院：人工智能学院

班级：1820013

专业：智能科学与技术

姓名：刘继垚

学号：18200100176

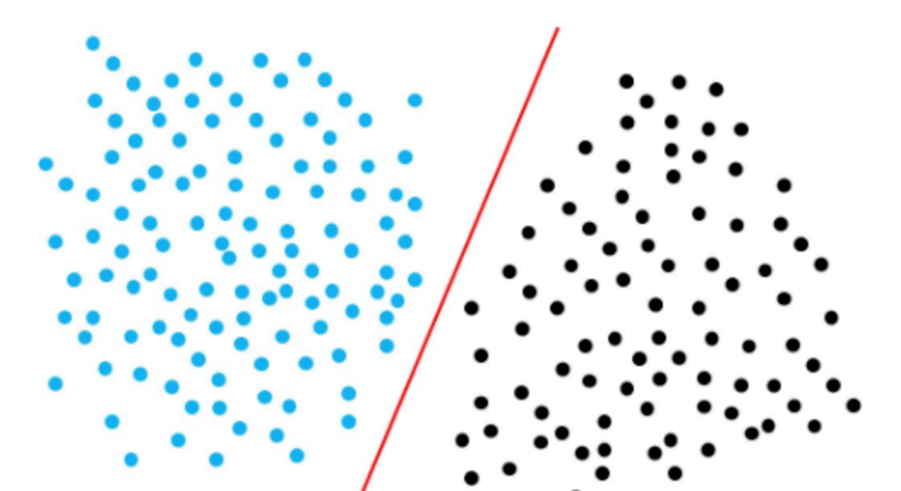
## 1背景介绍

感知机是1957年，由Rosenblatt提出会，是**神经网络和支持向量机**的基础。

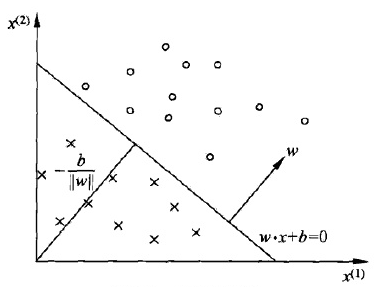
## 2．感知机的原理

感知机是二分类的线性模型，其输入是实例的特征向量，输出的是事例的类别，分别是+1和-1，属于判别模型。

假设训练数据集是线性可分的，感知机学习的目标是求得一个能够将训练数据集**正实例点和负实例点完全正确分开的分离超平面**。如果是非线性可分的数据，则最后无法获得超平面



**2.1 感知机模型**



感知机从输入空间到输出空间的模型如下：



* 1. **感知器的损失函数**

我们首先定义对于样本，如果，则记，如果，则。

这样取y的值有一个好处，就是方便定义损失函数。因为正确分类的样本满足，而错误分类的样本满足。我们损失函数的优化目标，**就是期望使误分类的所有样本，到超平面的距离之和最小**。**所以损失函数定义如下：**

****

其中M集合是误分类点的集合。

不考虑，就得到感知机模型的损失函数：



**2.3 感知器学习算法**

感知机学习算法是对上述损失函数进行极小化，求得ww和bb。但是用普通的基于所有样本的梯度和的均值的批量梯度下降法（BGD）是行不通的，原因在于我们的损失函数里面有限定，**只有误分类的M集合里面的样本才能参与损失函数的优化**。所以我们不能用最普通的批量梯度下降,只能采用随机梯度下降（SGD）。目标函数如下：



2.3.1 原始形式

输入：训练数据集，，学习率，

输出：；感知机模型

1. 赋初值 w0,b0
2. 选取数据点(xi,yi)
3. 判断该数据点是否为当前模型误分类点，即判断若yi(w⋅xi+b)<=0则更新

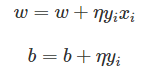




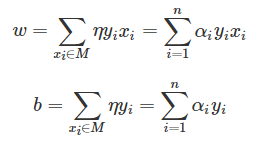
1. 转到2，直到训练集中没有误分类点。

2.3.2 对偶形式

由于w,b的梯度更新公式：



我们的w,b经过了n次修改后的，参数可以变化为下公式，其中：



这样我们就得出了感知机的对偶算法。

输入：训练数据集，，学习率，

输出：；感知机模型

其中

1. 赋初值 
2. 选取数据点(xi,yi)
3. 判断该数据点是否为当前模型误分类点，判断若则更新





转到2，直到训练集中没有误分类点。

为了减少计算量，我们可以预先计算式中的内积，得到Gram矩阵

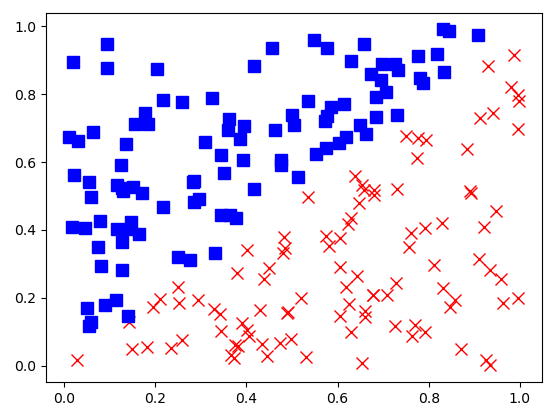


## 实验过程及结果及分析

本次实验使用200个散点集，以这条直线将散点集划分为两类。学习参数设置如下：

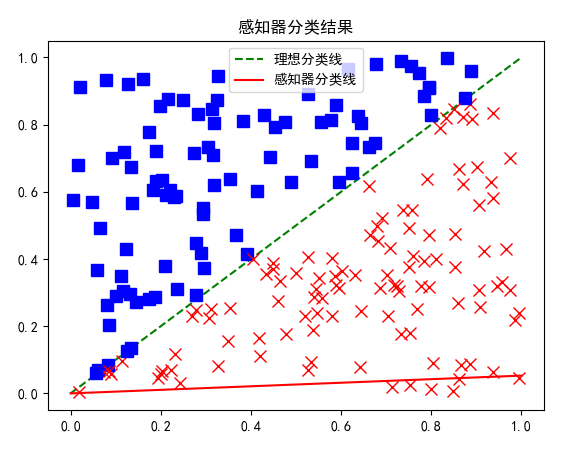
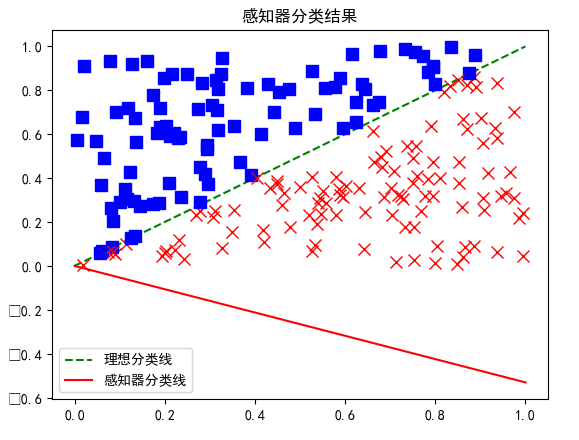
|  |  |
| --- | --- |
| 学习率 | 0.1 |
| 最大迭代次数 | 2000 |

散点集如下图所示：

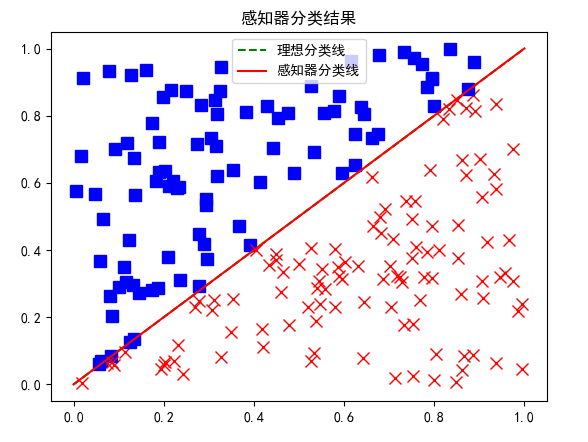
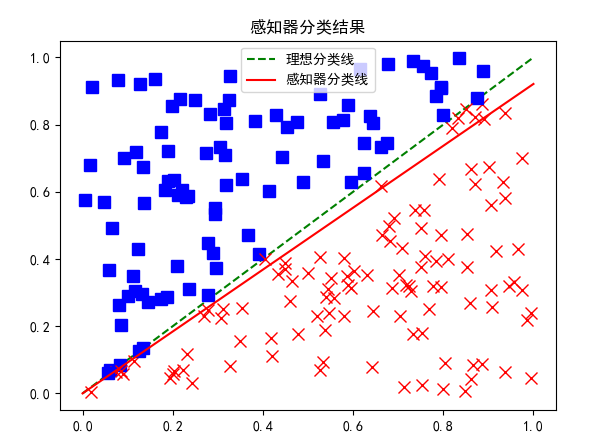


**3.1 实验结果**

采用上述参数训练感知器模型，我们记录了第1次，第10次，第100次，第200次迭代的结果，如下图：



迭代1次 迭代10次



迭代100次 迭代2000次

**3.2 实验结果分析**

可以看出，随着迭代次数的增加，感知器的分类线逐渐地向理想分类线靠近，最终当迭代次数为2000次的时候，几乎与理想分类线重合。可以认为我们训练的感知器二分类模型是有效的。

感知机算法是一个简单易懂的算法，自己编程实现也不太难。它是很多算法的鼻祖，比如**支持向量机算法，神经网络与深度学习**。因此虽然它现在已经不是一个在实践中广泛运用的算法，还是值得好好的去研究一下。感知机算法对偶形式为什么在实际运用中比原始形式快，也值得好好去体会。

## 四、实验代码（python）

**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
  
# 生成一条随机的直线  
x = np.linspace(0, 1, 2)  
# print(x)  
# 生成直线的方程: y = w \* x + b  
# w = np.random.rand()  
# b = np.random.rand()  
w\_real = np.array([1])  
b\_real = np.array([0])  
print(f'w={w\_real},b={b\_real}')  
# 创建一个函数,返回y值  
# def fn(x):  
# return w\*x+b  
fn = **lambda** x: w\_real \* x + b\_real  
# 通过可视化来创建直线  
  
# 通过直线把生成的100个点分成两个类别  
N = 200  
xn = np.random.rand(N, 2)  
# 存储每个样本的类别 [1,-1]  
yn = np.zeros([N, 1])  
# 通过之前的直线把样本分成两类  
  
**for** i **in** range(N):  
 **if** (fn(xn[i, 0]) >= xn[i, 1]):  
 # 当前的x[i]的点在直线的下方  
 yn[i] = -1  
 **else**:  
 yn[i] = 1  
  
  
  
# 对于给定的x,y值,感知器要寻找超平面  
**def perceptron**(xn, yn, max\_iter=2000, a=0.1, w=np.zeros(3)):  
 N = xn.shape[0]  
 # 函数里面在构造一个函数,对数据样本进行分类  
 # np.sign() 激活函数,可以把结果转化 1 或者 -1  
 # x ==> x[0] x[1]  
 f = **lambda** x: np.sign(w[0] \* 1 + w[1] \* x[0] + w[2] \* x[1])  
 # 循环反向传播,如果预测值与标准值不等则修改权重和偏置  
 W=[]  
 **for** iter **in** range(max\_iter):  
 # 随机获取一个样本作为测试样本  
 i = np.random.randint(N)  
 # yn[i] 是样本的目标值, xn[i,:] 第i个样本的特征值 f(xn[i,:]) 预测值  
 **if** (yn[i] != f(xn[i, :])):  
 # 更新权重与偏置 权重原值 + 目标值 \* 输入值 \* 学习率  
 w[0] = w[0] + yn[i] \* 1 \* a  
 w[1] = w[1] + yn[i] \* xn[i,0] \* a  
 w[2] = w[2] + yn[i] \* xn[i,1] \* a  
 # print(w)  
 **if** iter==1 **or** iter==10 **or** iter==100 **or** iter==1999:  
 # print(iter)  
 W.append(w)  
 print(w)  
 show(w)  
 # return W  
  
  
**def show**(w):  
 plt.plot(x, w\_real\*x+b\_real, 'g--',label='理想分类线')  
  
 **for** i **in** range(N):  
 **if** (fn(xn[i, 0]) >= xn[i, 1]):  
 # 当前的x[i]的点在直线的下方  
 yn[i] = -1  
 plt.plot(xn[i, 0], xn[i, 1], 'rx', markersize=8)  
 **else**:  
 yn[i] = 1  
 plt.plot(xn[i, 0], xn[i, 1], 'bs', markersize=8)  
  
 bnew = -w[0] / w[2]  
 anew = -w[1] / w[2]  
 # 通过a与b生成预测分类线函数  
 # y = lambda x:anew \* x + bnew  
 plt.rcParams['font.sans-serif'] = ['SimHei']  
 plt.plot(x,x\*anew+bnew,'r',label="感知器分类线")  
 plt.legend()  
 plt.title('感知器分类结果')  
 plt.show()  
  
perceptron(xn,yn,max\_iter=2000)  
# W = perceptron(xn,yn,max\_iter=2000)  
# print(W)  
# 通过w生成预测分类线函数的a,b的值  
# bnew = -w[0] / w[2]  
# anew = -w[1] / w[2]  
# # 通过a与b生成预测分类线函数  
# y = lambda x:anew \* x + bnew  
# plt.rcParams['font.sans-serif'] = ['SimHei']  
# plt.plot(x,y(x),'r',label="感知器分类线")  
# plt.legend()  
# plt.title('感知器分类结果')  
# plt.show()  
  
#  
# for w in W:  
# show(w)