

中山大学计算机学院人工智能本科生实验报告（2022学年春季学期）

课程名称：Artificial Intelligence

教学班级	专业（方向）	学号	姓名
2班	计算机科学与技术	21307174	刘俊杰

一、实验题目

Week 13 神经网络

二、实验内容

实验内容

实验任务

- 在给定数据集完成k-means算法聚类，完成以下内容数据读取:文本形式读入行，split后转浮点数: 或使用numpy pandas 等库直接读取
- 设计合适的k以及距离度量函数并实现聚类算法k自选，度量函数自选 (如: 欧氏距离)
- 画出聚类后的数据可视化图,可使用matplotlib

1. 算法原理

(1)人工神经网络：

模拟大脑进行计算

组成:

- 节点(也称单元)对应于神经元,连接对应于突触
- 计算:节点间传送的数值信号对应于神经元之间的化学信号节占对数值信号进行修改对应干神经元激活率

(2)激活函数用以模拟神经元的激活

对于“正确的”输入，单元应该是“激活的”:输出接近1。对于“错误的”输入，单元应该是“静息的”:输出接近0

(3)损失函数

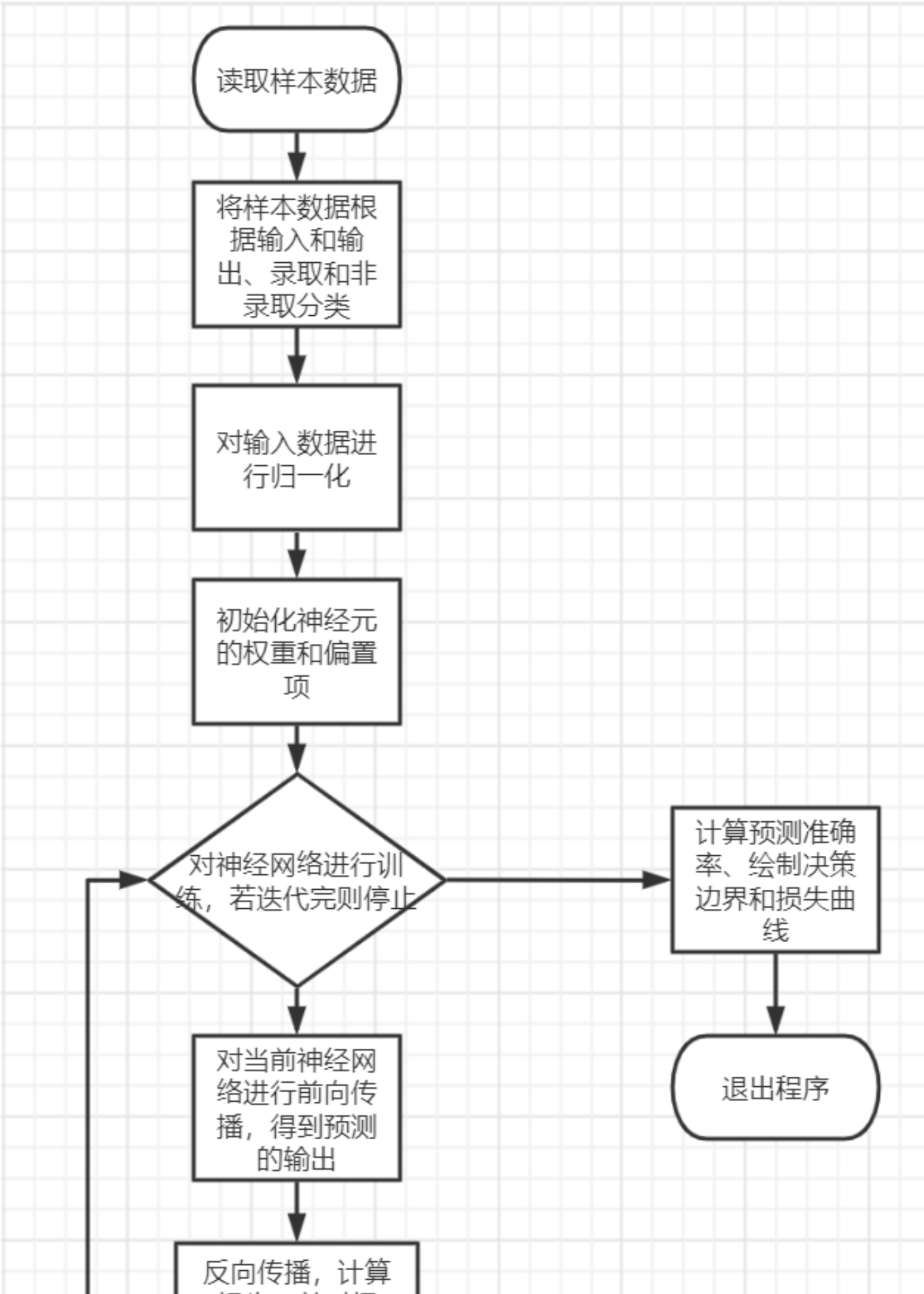
损失函数 $L(x,y,y)$ 定义为当正确的答案是 $f(a)=y$ ，预测 $h(a)=y$ 的效用损失量

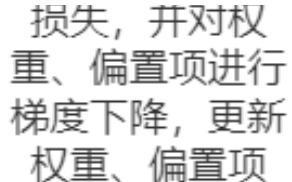
(4)反向传播

1. 利用前向传播求出误差E,
2. 求出误差E对权重W的偏导数,
3. 利用，权重更新公式 更新权重W，其中 α 是学习率
4. 继续反向传播，更新更接近输入层的权重W，直到更新所有的权重W，

5. 循环1,2,3,4过程，不断更新权重 W ，降低误差 E ，最终得到训练好的神经网络（即适合的权重 W ）

2.流程图





损失, 并对权重、偏置项进行梯度下降, 更新权重、偏置项

3.关键代码展示(代码细节写在了注释中)

针对data1构建了无隐藏层的神经网络

(1)定义神经元(单元)的类

```
class Layer():#定义神经元的类
    def __init__(self,input_size):
        self.bias=0#偏置项
        self.input=[]#输入
        self.output=0#输出(针对data1, 设置每个神经元只有一个输出)
        self.weight=np.random.rand(input_size)#权重
    def layer_function(self,input):#该神经元的激活函数
        x=np.dot(self.weight,np.array(input))+self.bias#输入乘以权重加偏置
        self.input=[i for i in input]#记录该神经元的输入
        return sigmoid(x)
```

(2)sigmoid函数

```
def sigmoid(x):#sigmoid函数
    return 1 / (1 + np.exp(-x))
```

(3)前向传播

```
def forward(input,layers):#前向传播(这里针对data1,设计前一层的所有神经元的输出是后一层
    每一个神经元的输入)
    layers_output=[]#记入每一层神经元的输出
    for i in range(len(layers)):
        layers_output.append([])
        for j in range(len(layers[i])):
            if i==0:#第一层的输入是数据
                layers[i][j].output=layers[i][j].layer_function(input)
                layers_output[-1].append(layers[i][j].output)
            else:#其他层的输入是前一层所有神经元的输出
                layers[i][j].output=layers[i][j].layer_function(layers_output[i-
    1])
                layers_output[-1].append(layers[i][j].output)
```

```
output=layers[-1][-1].output
return output#返回最后的输出
```

(4)反向传播

```
def backward(input, layers, correct_output): #反向传播
    d = [[] for i in range(len(layers))] #记录每一层的损失导数
    loss = 0 #记录损失值
    old_weight = [[] for i in range(len(layers))] #记录未更新的每一层每一个神经元的权重
    for i in range(len(layers)):
        index = len(layers) - i - 1 #从后向前传播
        for j in range(len(layers[index])):
            old_weight[index].append([w for w in layers[index][j].weight])
            output = layers[index][j].output #输出
            if i == 0: #最后一层即输出层
                loss += (correct_output - output) ** 2 #计算损失
                d_out = output * (1 - output) * (correct_output - output) #计算损失函数的导数
                d[index].append(d_out)
            else: #隐藏层
                #计算损失函数的导数
                d_out = output * (1 - output)
                tmp_sum = 0
                for k in range(len(layers[index + 1])):
                    tmp_sum += old_weight[index + 1][k][j] * d[index + 1][k]
                d_out *= tmp_sum
                d[index].append(d_out)
            #对每权重进行更新
            for k in range(len(layers[index][j].weight)):
                layers[index][j].weight[k] += d_out * learning_rate * layers[index]
        [j].input[k]
        layers[index][j].bias += d_out * learning_rate
    return loss
```

(5)训练函数

```
def train(times, input, correct_output, layers): #训练函数
    loss = [0 for i in range(times)] #每次训练的损失
    while(times > 0): #times训练次数
        for i in range(len(input)):
            forward(input[i], layers) #前向传播
            loss[iter - times] += backward(input[i], layers, correct_output[i]) #反向传播
        times -= 1
    return loss
```

(6)对样本数据进行归一化

```
def one(x):#归一化
    o=[]#记录数据样本的最大值和最小值，后面画图需要返回归一化
    for j in range(len(x[0])):
        min_x=100000000
        max_x=-100000000
        for i in range(len(x)):
            if max_x<x[i][j]:
                max_x=x[i][j]
            if min_x>x[i][j]:
                min_x=x[i][j]
        for i in range(len(x)):
            x[i][j]=2*(x[i][j]-min_x)/(max_x-min_x)-1#归一化
        o.append([max_x,min_x])
    return x,o
```

(7)逆归一化计算决策边界的y值

```
def find_y(x1,theta,o):#逆归一化计算决策边界的y值
    return [((-theta[0]-theta[1]*((x_1-o[0][1])*2/(o[0][0]-o[0][1])-1))/theta[2]+1)/2*(o[1][0]-o[1][1])+o[1][1] for x_1 in x1]
```

(8)读取样本数据

```
def read_file(f,k):#读取文本信息
    input=[]#输入
    output=[]#输出
    accept=[]#录取的样本数据
    refuse=[]#拒绝的样本数据
    for line in f:
        input.append([])
        line=line.strip().split(",")
        for i in range(1,k):
            input[-1].append(float(line[0])**i)
            input[-1].append(float(line[1])**i)
        output.append(int(line[2]))
    for i in range(len(input)):
        if output[i]:
            accept.append([input[i][0],input[i][1]])
        else:
            refuse.append([input[i][0],input[i][1]])
    return input,output,accept,refuse
```

(9)计算预测准确率

```
def calculate_accuracy(layers,input,output):#计算预测准确率
    c=0
    for i in range(len(input)):
        pred=forward(input[i],layers)
        if pred>=0.5:
            pred=1
        else :
            pred=0
        if pred==output[i]:
            c+=1
    print("accuracy",":",c/len(input))
```

(10)画出决策边界

```
def draw_boundary(L,accept,refuse,o):#画出决策边界
    w=[L.bias]#w存放输出层的偏置和权重
    for i in L.weight:
        w.append(i)
    x = np.array([0, 100])
    y=find_y(x,w,o)#逆归一化计算决策边界的y值
    plt.plot(x,y,color='r',label="decision boundary")
    for i in range(len(accept)):
        plt.scatter(accept[i][0],accept[i][1],c="g")
    for i in range(len(refuse)):
        plt.scatter(refuse[i][0],refuse[i][1],c="b")
    plt.show()
```

(10)画出损失曲线

```
def draw_loss(loss):#画出损失曲线
    plt.plot([i for i in range(1,iter+1)],loss,color="r",label="loss")
    plt.show()
```

(11)神经网络预测

```
def nn(k):#神经网络预测
    f = open(r"ai\E10\data1.txt",'r')
    input,output,accept,refuse=read_file(f,k)#读取样本数据
    input,o=one(input)#对样本数据进行归一
    L=Layer(len(input[0]))#针对data1只需要一层输出层和输入层，这是输出层，输入层直接输入
    layers=[[L]]#每一层的神经元
    loss=train(iter,input,output,layers)#训练
    calculate_accuracy(layers,input,output)#计算预测准确率
```

```
draw_boundary(L,accept,refuse,o)#画出决策边界  
draw_loss(loss)#画出损失曲线
```

4.创新点&优化

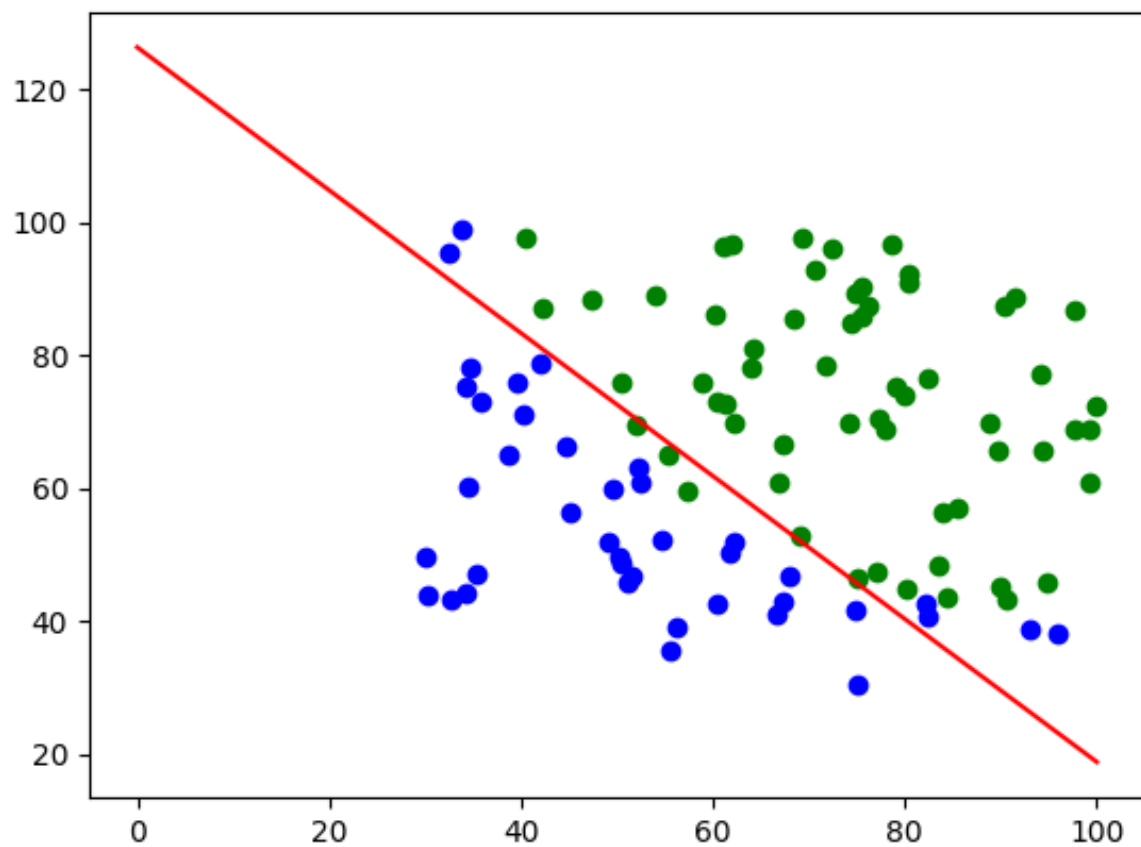
无

三、实验结果及分析

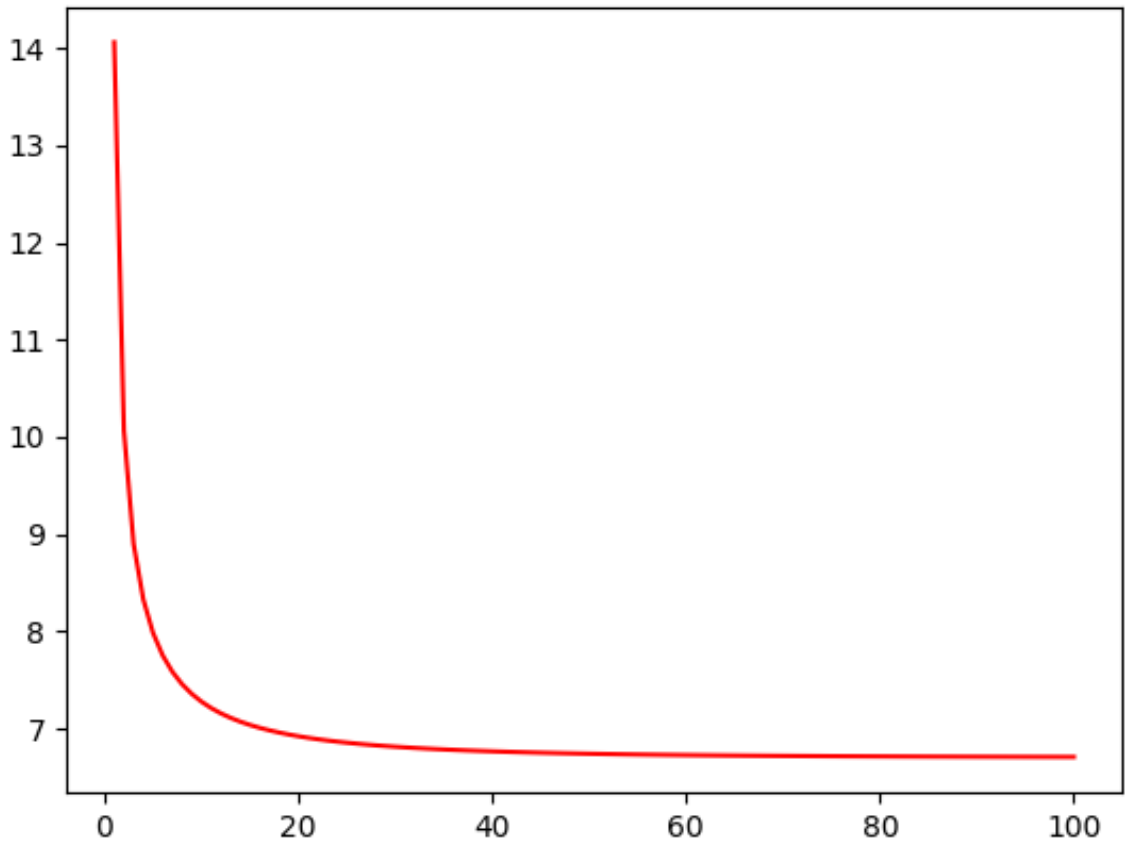
1. 实验结果展示示例(实验结果放入result文件夹中)

准确率:91%

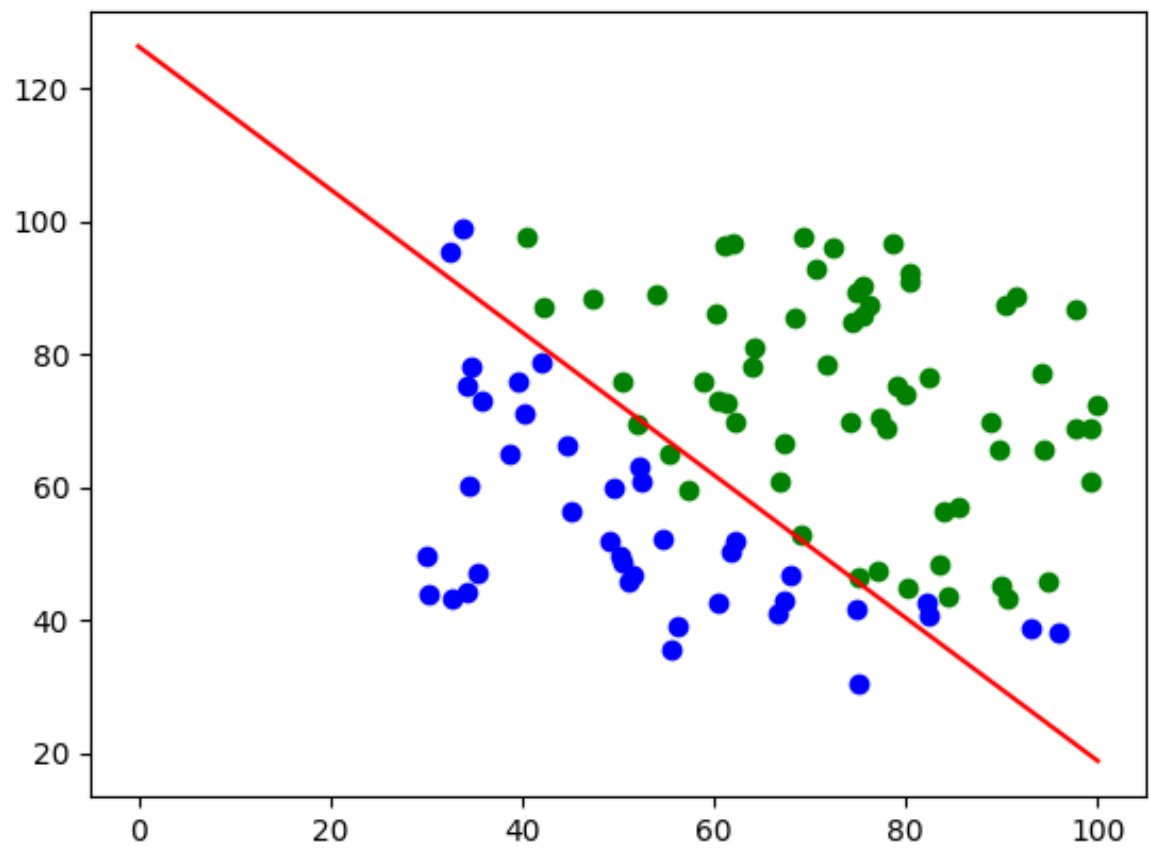
决策边界:



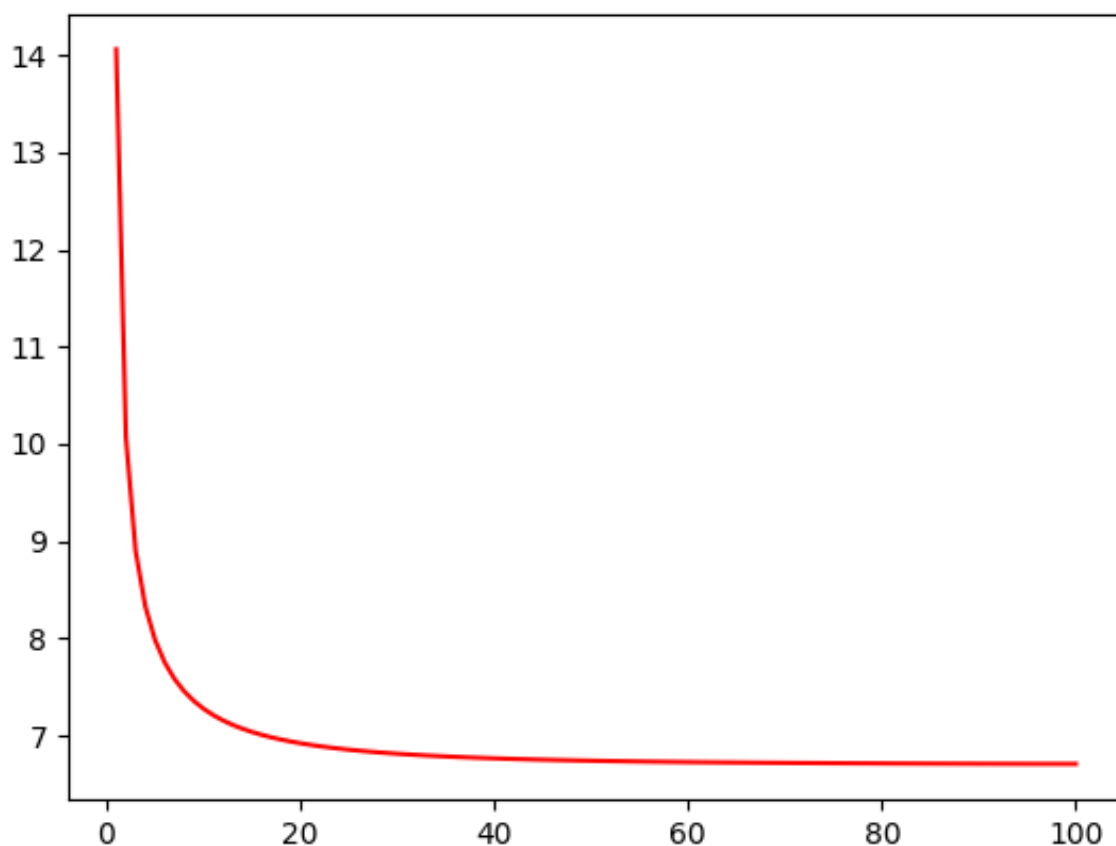
loss曲线



2. 评测指标展示及分析



可以看到，数据集被决策分界线明显分为了两个部分但还存在着一些误差，预测的准确率也达到了91%



在刚开始的前20次迭代中,loss损失函数下降得很快,不过在之后, Loss曲线逐渐趋于平缓,但仍由保持下降的趋势。

这是因为一开始的权重和偏置项是随机产生的,神经网络预测的输出会和正确的有比较大的误差。所以初期的反向传播了对权重的更新变化也比较大,这就导致了刚开始的损失函数下降较快。

随着迭代次数的逐渐增多,神经网络的输出也逐渐逼近了准确的输出,误差减小,反向传播对权重的更新变小,也就导致了损失函数的变化逐渐趋于平缓。

四、参考资料

实验课PPT