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
计科三班	计算机科学与技术	21307185	张礼贤

一、实验题目

- 神经网络实现逻辑回归
- 在数据集data1.txt上完成大学生录取预测分类训练：
  - 设计合适的网络结构，选择合适的损失函数
  - 利用训练集完成网络训练
  - 画出数据可视化图、loss曲线图，计算模型收敛后的分类准确率

二、实验内容

1.算法原理

- 利用神经网络实现逻辑回归进行二分类问题的算法原理如下：
  - 数据预处理：首先，对输入数据进行预处理，包括特征缩放、数据归一化等操作，以确保数据的范围和分布合适。
  - 初始化参数：初始化神经网络的参数，包括权重矩阵和偏置向量。
  - 前向传播：通过前向传播计算，将输入特征数据传递到神经网络中，计算每个神经元的加权和，并经过激活函数（如Sigmoid函数）得到输出值。

- 计算损失函数：根据实际标签值和神经网络的输出值，计算损失函数，常用的损失函数是交叉熵损失函数。
- 反向传播：通过反向传播算法，根据损失函数的梯度，计算参数的梯度值，并更新权重矩阵和偏置向量。这个过程利用梯度下降法来最小化损失函数，使得模型的预测结果更接近实际标签值。
- 重复迭代：重复进行步骤3-5，直到达到设定的迭代次数或收敛条件。
- 预测：通过训练得到的模型，在新的输入数据上进行预测，即根据输入特征计算输出值，并通过阈值进行分类判断。

以上就是利用神经网络实现逻辑回归进行二分类问题的基本算法原理。通过不断地调整权重和偏置，使得模型能够根据输入特征对样本进行准确的分类预测。

2.关键代码及注释

```
import numpy as np
import matplotlib.pyplot as plt
from timeit import default_timer as timer

class Model:
    def __init__(self, input_size):
        #self.weight = np.random.rand(input_size, 1) #初始化权重矩阵
        self.weight = np.zeros((input_size,1))
        self.bias = 0   #初始化偏置量
        self.loss = []  #初始化损失列表

    def sigmoid(self, x):
        """激活函数"""
        return 1 / (1 + np.exp(-x))

    def forward(self, X):
        """前向传播,计算输出值"""
        temp = np.dot(X, self.weight) + self.bias    #初始输入与权重矩阵进行点积，加上偏置
        output = self.sigmoid(temp) #根据上一步获得的值计算激活函数后的值
        return output

    def backward(self, x, y, output, learning_rate):
        """反向传播算法"""
        delta = (y - output) * output * (1 - output)     #求偏导的公式
        dw = learning_rate * delta * x.reshape(-1, 1)    #利用矩阵乘法求出dw偏导值
        db = learning_rate * delta  #更新偏置
        loss = (y - output)**2  #获取损失函数值
        return dw, db, loss

    def train(self, X, Y, num_iterations, learning_rate):
        """训练函数"""
        for i in range(num_iterations): #更新迭代每一轮
            sum = 0
            for j in range(len(X)):
                x = X[j]
                y = Y[j]
                output = self.forward(x)
                dw, db, loss = self.backward(x, y, output, learning_rate)
```

```
                self.weight += dw
                self.bias += db
                sum += loss
            self.loss.append(sum)    #将每一轮的损失函数值加入loss列表

    def predict(self, X):
        """根据训练的结果,预测测试集"""
        result = self.forward(X)
        predictions = (result >= 0.5).astype(int)    #如果大于等于0.5则预测
为1
        return predictions

    def plot_decision_boundary(self, X, Y, std, mean):
        """将两科成绩和分界线可视化"""
        new_std = np.std(X, axis=0)
        new_mean = np.mean(X, axis=0)
        X = X * std + mean
        admitted = X[Y == 1]
        not_admitted = X[Y == 0]
        plt.scatter(admitted[:, 0], admitted[:, 1], color='green',
label='Admitted')
        plt.scatter(not_admitted[:, 0], not_admitted[:, 1], color='red',
label='Not Admitted')
        # 绘制分界线
        weight_inverse = self.weight * new_std / np.std(X, axis=0)
        bias_inverse = self.bias - np.dot(np.mean(X, axis=0),
weight_inverse)   #逆归一化
        x_min, x_max = X[:, 0].min() - 10, X[:, 0].max()
        y_min, y_max = X[0, :].min() - 10, X[0, :].max() + 30
        weights = self.weight.reshape(-1)
        boundary_x = np.array([x_min, x_max])
        boundary_y_inverse = -(weight_inverse[0] * boundary_x +
bias_inverse) / weight_inverse[1]
        plt.plot(boundary_x, boundary_y_inverse, color='blue',
label='Decision Boundary')
        plt.ylim(y_min,y_max)
        plt.xlabel('Exam 1 Score')
        plt.ylabel('Exam 2 Score')
        plt.legend()
        plt.show()

    def plot_show_loss(self,num_iterations):
        """loss函数可视化"""
```

```
        plt.plot(range(1, num_iterations + 1), self.loss)
        plt.xlabel('Iterations')
        plt.ylabel('Loss')
        plt.title('Training Loss')
        plt.show()

def read_file(filename):
    """读取并分离输入文件中的信息"""
    data = np.loadtxt(filename, delimiter=',',dtype=float)
    res = [int(data[i][2]) for i in range(len(data))]
    X = [[data[i][0] , data[i][1]] for i in range(len(data))]
    return np.array(X), np.array(res)

def normalize(X):
    norms = np.linalg.norm(X, axis=1, keepdims=True)
    X_normalized = X / norms
    return X_normalized

# 准备数据
X,Y = read_file('data1.txt')
std = np.std(X, axis=0)
mean = np.mean(X, axis=0)
# 数据归一化
X = (X - np.mean(X, axis=0)) / np.std(X, axis=0)
# 创建逻辑回归模型并进行训练
num_iterations = 1000
input_size = X.shape[1]
model = Model(input_size)
tic = timer()
model.train(X, Y, num_iterations, learning_rate = 0.01)
toc = timer()
# 进行预测
test_data = X
test_data = (test_data - np.mean(X, axis=0)) / np.std(X, axis=0)
predictions = model.predict(test_data)
count = 0
for i in range(len(predictions)):
    pre = predictions[i][0]
    if(pre == Y[i]) : count += 1

print(f"The accuracy is: {count/len(X):.4f}")
print(f'The time spent is: {(toc - tic):.4f} secs')
model.plot_decision_boundary(X, Y, std, mean)
```

```
model.plot_show_loss(num_iterations)
```

3.创新点 & 优化

1. 使用了归一化操作，统一了数据集的样本特征.
    - 去除量纲影响：不同特征的取值范围可能不同，如果直接使用原始数据进行分析，具有较大取值范围的特征可能会对模型的训练产生更大的影响。通过数据归一化，可以消除不同特征之间的量纲影响，使得它们在相同的尺度上进行比较和权衡。
    - 提高模型收敛速度：如果特征具有较大的取值范围，可能会导致算法收敛速度较慢。数据归一化可以加快模型的收敛速度，提高训练效率。
2. 合理利用了矩阵乘法操作，提高了代码的可读性，并且简化了代码和运算。
   例如此处：

```
dw = learning_rate * delta * x.reshape(-1, 1)    #利用矩阵乘法求出dw偏导值
```

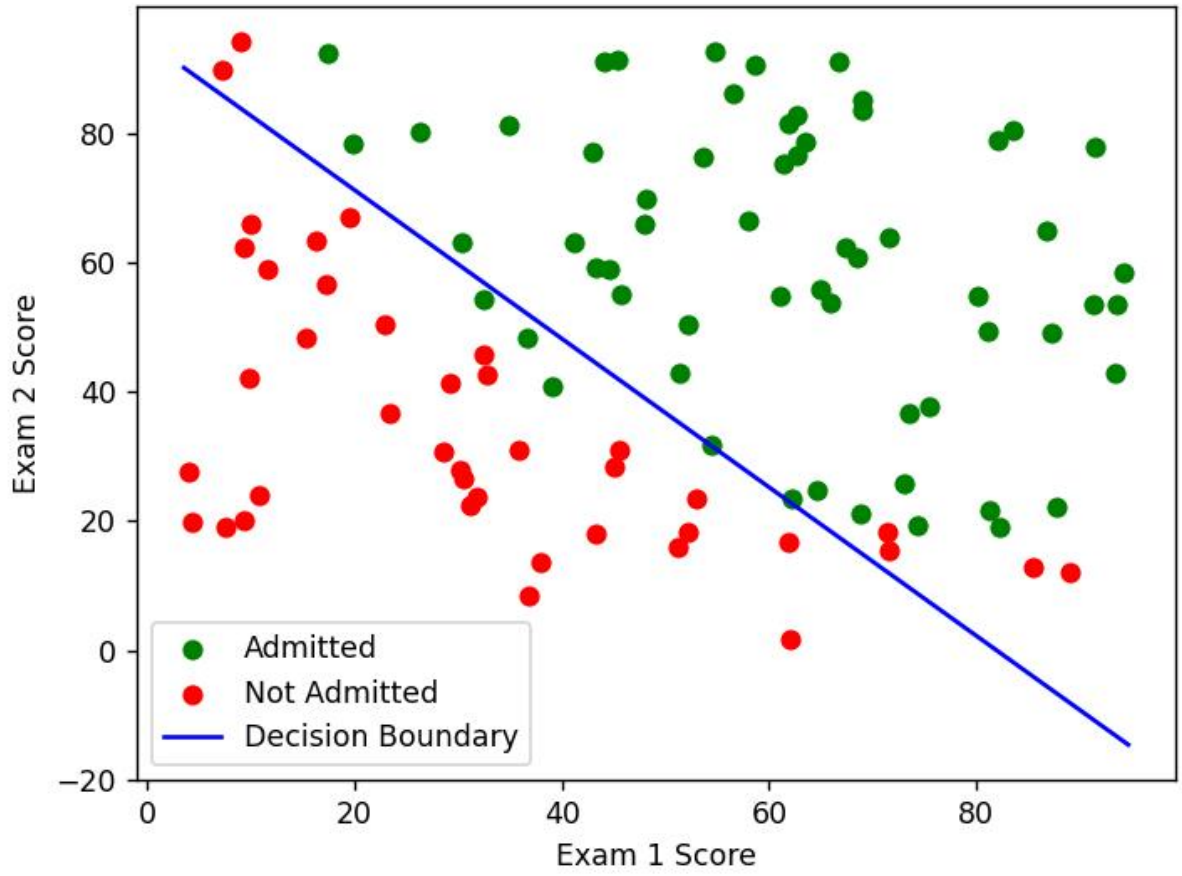
三、实验结果与分析

实验结果展示

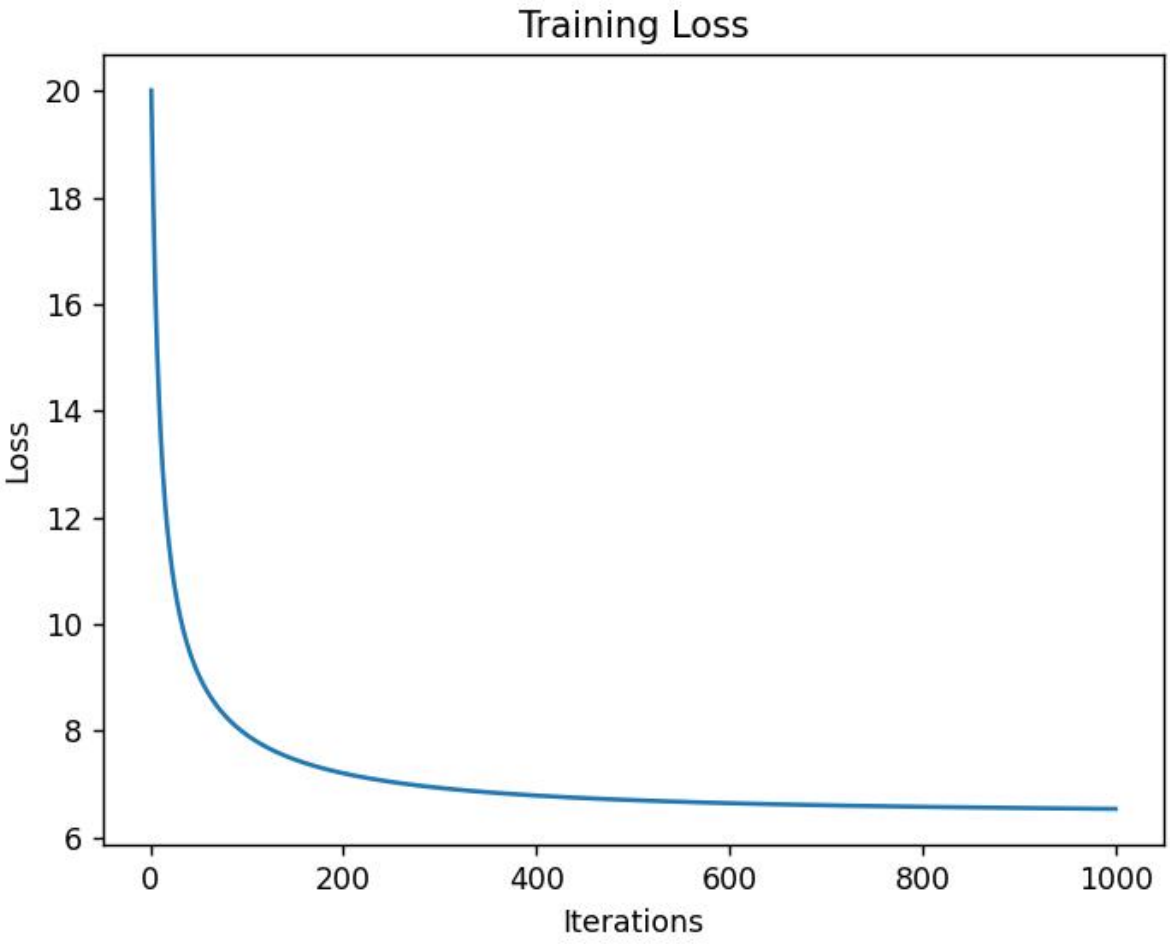
1. 准确率与运行时间

```
PS D:\桌面\net_work> python -u "d:\桌面\net_work\admitted.py"
The accuracy is: 0.9100
The time spent is: 2.0169 secs
```

2. 数据可视化图


Exam 2 Score
Exam 1 Score
Admitted
Not Admitted
Decision Boundary
80
60
40
20
0
−20
0
20
40
60
80

3. loss曲线图



实验结果分析

- 通过上述实验结果，可以看到，经过训练后的神经网络对于数据集有着较好的预测精准度，并且在较快的时间内收敛。
- 而针对loss曲线图，这里loss函数选取的是平方误差损失，我们可以发现呈现先陡峭后缓慢的趋势下降，证明梯度下降存在良好的效果。