中山大学计算机学院

人工智能

本科生实验报告

课程名称：Artificial Intelligence

学号	22336259	姓名	谢宇桐

一、 实验题目

购房预测分类任务

1. 算法原理

本次实验要求我们使用两种算法分别预测：逻辑回归算法和多层感知机。

（1）逻辑回归算法

Logistic regression （逻辑回归）是一种非线性回归模型，是当前业界比较常用的机器学习方法，用于估计某种事物的可能性。

给定一个特征向量 x 和参数向量 θ，逻辑回归模型的预测函数可以表示为：

$$p(y=1|x;\theta)=\sigma(\theta^T x)$$

其中 δ 是 Sigmoid 函数：

$$\sigma(z)=\frac{1}{1+e^{-z}}$$

函数的定义域(-inf,+inf),值域为（0，1），这也就是一个概率区间，大于 0.5 的属于 1 分类，小于 0.5 的属于 0 分类。

预测往往会有误差，也就需要一个损失函数计算并调整损失，使损失最小化，准确度最大化。

（2）多层感知机

感知机是一种基本的线性分类器，可以用于逻辑回归任务。其接收多个输入信号，并产生一个输出信号。

多层感知机（Multilayer Perceptron，简称 MLP），是一种基于前馈神经网络（Feedforward Neural Network）的深度学习模型，由多个神经元层组成，其中每个神经元层与前一层全连接。多层感知机可以用于解决分类、回归和聚类等各种机器学习问题。

感知机构造为输入层，输出层，隐含层。MLP 所有的参数就是各个层之间的连接权重以及偏置，包括 W1、b1、W2、b2。每个神经元的输出是其输入的加权和，通过一个非线性激活函数进行变换。常用的激活函数包括 Sigmoid、ReLU 等。

2. 关键代码展示（可选）

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# 预处理各项数据
data = pd.read_csv("data.csv")
X = data.iloc[:, :-1]  # 去除最后一列作为特征
Y = data.iloc[:, -1]  # 最后一列作为标签
X_min = X.min()  # 计算特征最小值
X_max = X.max()  # 计算特征最大值
X = (X - X_min) / (X_max - X_min)  # 特征归一化
```

逻辑回归算法：

```python
# 逻辑回归类
1 个用法
class logic:
    def __init__(self, input_dim, lv=0.1):
        np.random.seed(0) # 设置随机数种子，保证结果可复现
        self.weights = np.random.randn(input_dim, 1) * np.sqrt(2. / input_dim)
        self.bias = np.zeros((1, 1))
        self.lv = lv

    # Sigmoid激活函数
    1 个用法
    def sigmoid(self, z):
        return 1 / (1 + np.exp(-z))

    # 前向传播过程
    3 个用法
    def forward(self, x):
        return self.sigmoid(np.dot(x, self.weights) + self.bias)
```

```python
    # 计算损失
    1 个用法
    @staticmethod
    def loss(y_pre, y):
        return -np.mean(y * np.log(y_pre) + (1 - y) * np.log(1 - y_pre))

    # 反向传播，更新参数
    1 个用法
    def back(self, x, y):
        y_pre = self.forward(x)  # 计算预测值
        dz = y_pre - y  # 计算梯度
        d_w = np.dot(x.T, dz) / x.shape[0]  # 计算权重的梯度
        d_bias = np.sum(dz) / x.shape[0]  # 计算偏置的梯度
        # 更新参数
        self.weights -= self.lv * d_w
        self.bias -= self.lv * d_bias
```

```python
    # 训练模型
    1 个用法
    def train(self, X, Y, epochs, size):
        losses = []
        best_loss = float('inf')
        best_e = 0
        best_w = None
        best_bias = None

        for epoch in range(epochs):
            cur_loss = 0
            for i in range(0, X.shape[0], size):
                x_batch = X.iloc[i : i + size].values
                y_batch = Y.iloc[i : i + size].values.reshape(-1, 1)

                y_pre = self.forward(x_batch)
                # 计算损失
                cur_loss = self.loss(y_pre, y_batch)
                # 反向传播
                self.back(x_batch, y_batch)
```

```
                losses.append(cur_loss)
                if cur_loss < best_loss:
                    best_loss = cur_loss
                    best_e = epoch + 1
                    best_w = self.weights.copy()
                    best_bias = self.bias.copy()

                print(f"Epoch: {epoch + 1}, Loss: {cur_loss}")

            # 使用最佳参数
            self.weights = best_w
            self.bias = best_bias

            return losses, best_e, best_loss

    def predict(self, X):
        return (self.forward(X) > 0.5).astype(np.int32)
```

```
# 模型训练
input_dim = X.shape[1]  # 获取特征的维度
model = logic(input_dim)  # 创建逻辑回归模型实例
epochs = 3000  # 设置训练的迭代次数
size = 32  # 设置每次迭代的批量大小

losses, best_e, best_loss = model.train(X, Y, epochs, size)

# 进行预测并计算准确率
results = model.predict(X)
accuracy = (results == np.array(Y).reshape(-1, 1)).astype(np.int32).mean()
print(f"最终准确率：{accuracy * 100:.2f}%，最佳loss：{best_loss}")

# 提取权重和偏置用于绘制决策边界
w = model.weights.flatten()
b = model.bias.flatten()
```

MLP:

```python
class MLP:
    def __init__(self, layers, lv=0.1):
        np.random.seed(0)
        self.layers = layers # 层数
        self.lv = lv  # 学习率
        self.weights = []  # 权重
        self.bias = []  # 偏置
        for i in range(len(layers) - 1):
            weight = np.random.randn(layers[i], layers[i + 1]) * np.sqrt(2. / layers[i])  # 初始化权重
            self.weights.append(weight)
            bias = np.zeros((1, layers[i + 1]))  # 初始化偏置
            self.bias.append(bias)

    # Sigmoid激活函数
    1 个用法
    def sigmoid(self, z):
        return 1 / (1 + np.exp(-z))
```

```python
    # ReLu激活函数
    1 个用法
    def relu(self, z):
        return np.maximum(0, z)

    # 前向传播过程
    2 个用法
    def forward(self, x):
        y = x
        self.cache = {'A0': x}  # 缓存每一层的输出结果
        for i in range(len(self.weights) - 1):
            z = np.dot(y, self.weights[i]) + self.bias[i]
            y = self.relu(z)
            self.cache[f'Z{i + 1}'] = z
            self.cache[f'A{i + 1}'] = y
        z = np.dot(y, self.weights[-1]) + self.bias[-1]
        y = self.sigmoid(z)
        self.cache[f'Z{len(self.weights)}'] = z
        self.cache[f'A{len(self.weights)}'] = y
        return y
```

```python
    # 计算损失
    1 个用法
    def loss(self, y_pred, y):
        return np.mean((y_pred - y) ** 2)

    # 反向传播，更新参数
    1 个用法
    def back(self, x, y):
        m = x.shape[0]  # 样本数
        output = self.cache[f'A{len(self.weights)}']  # 输出层输出
        errors = [None] * len(self.weights)

        errors[-1] = (output - y) * (output * (1 - output))  # 计算输出层的误差

        for i in reversed(range(len(self.weights) - 1)):
            errors[i] = np.dot(errors[i + 1], self.weights[i + 1].T) * (self.cache[f'A{i + 1}'] > 0)  # 计算隐藏层的误差

        for i in range(len(self.weights)):
            grad_weight = np.dot(self.cache[f'A{i}'].T, errors[i]) / m  # 计算权重梯度
            grad_bias = np.sum(errors[i], axis=0, keepdims=True) / m  # 计算偏置梯度
            self.weights[i] -= self.lv * grad_weight  # 更新权重
            self.bias[i] -= self.lv * grad_bias  # 更新偏置
```

```python
# 模型训练
layers = [2, 16, 1]  # 输入层，一个隐藏层，一个输出层
model = MLP(layers)  # 初始化模型
epochs = 2000
batch_size = 32
losses = []  # 记录损失值
best_loss = float('inf')  # 初始最佳损失
best_e = 0  # 最佳Epoch
best_w = []  # 保存每层的最佳权重
best_bias = []  # 保存每层的最佳偏置
```

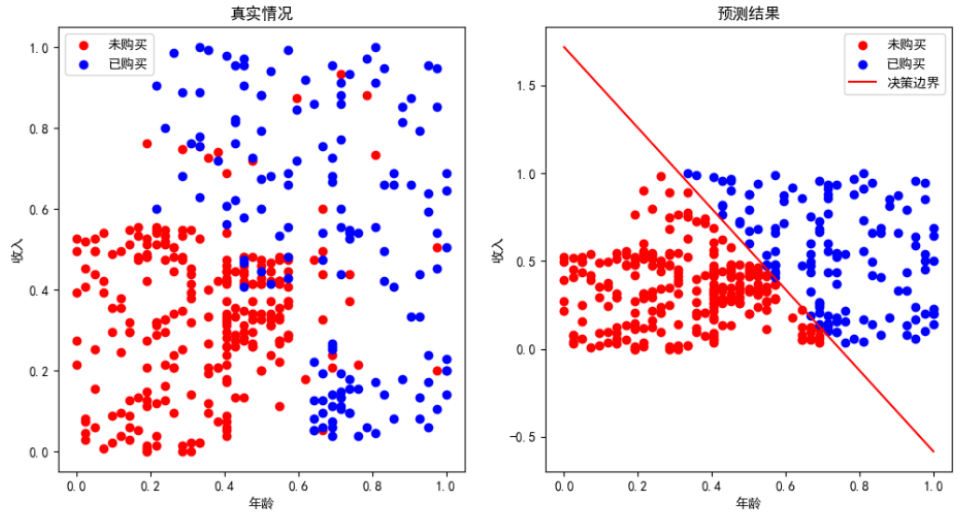
```python
for epoch in range(epochs):
    for i in range(0, X.shape[0], batch_size):
        x_batch = X.iloc[i: i + batch_size].values
        y_batch = Y.iloc[i: i + batch_size].values.reshape(-1, 1)

        y_pre = model.forward(x_batch)  # 前向传播
        cur_loss = model.loss(y_pre, y_batch)  # 计算当前损失
        model.back(x_batch, y_batch)  # 反向传播更新参数

    losses.append(cur_loss)
    if cur_loss < best_loss:
        best_loss = cur_loss
        best_e = epoch + 1
        best_w = [w.copy() for w in model.weights]  # 深复制每层权重
        best_bias = [b.copy() for b in model.bias]  # 深复制每层偏置
```

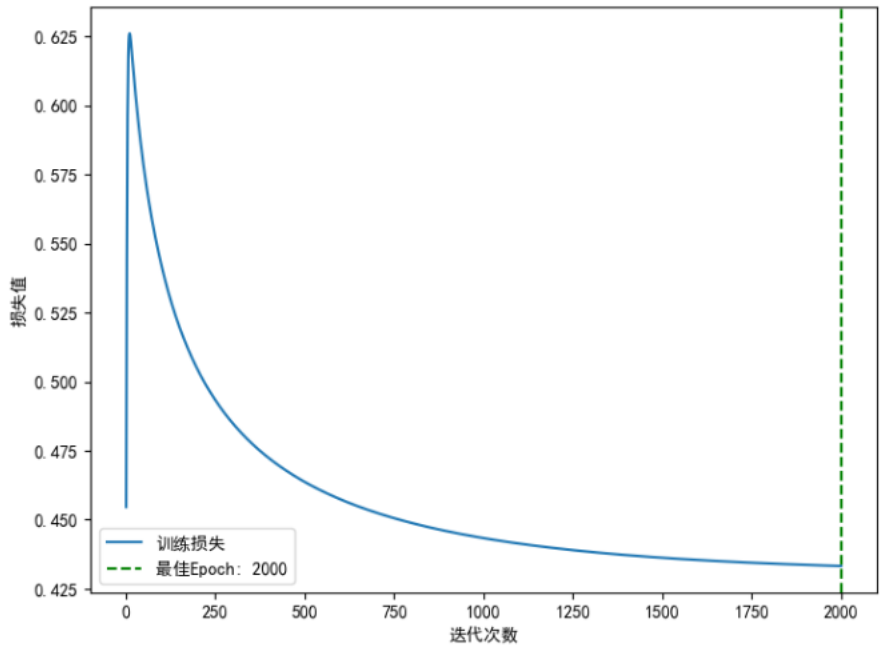
```python
# 使用最佳参数进行预测
model.weights = best_w
model.bias = best_bias

results = (model.forward(X) > 0.5).astype(np.int32)
accuracy = (results == np.array(Y).reshape(-1, 1)).astype(np.int32).mean()
print(f"最终准确率：{accuracy * 100:.2f}%，最佳loss：{best_loss}")

# 提取第一层的权重和偏置，因为输入层直接连接输出层
w = model.weights[0].flatten()
b = model.bias[0].flatten()
```

二、 实验结果及分析

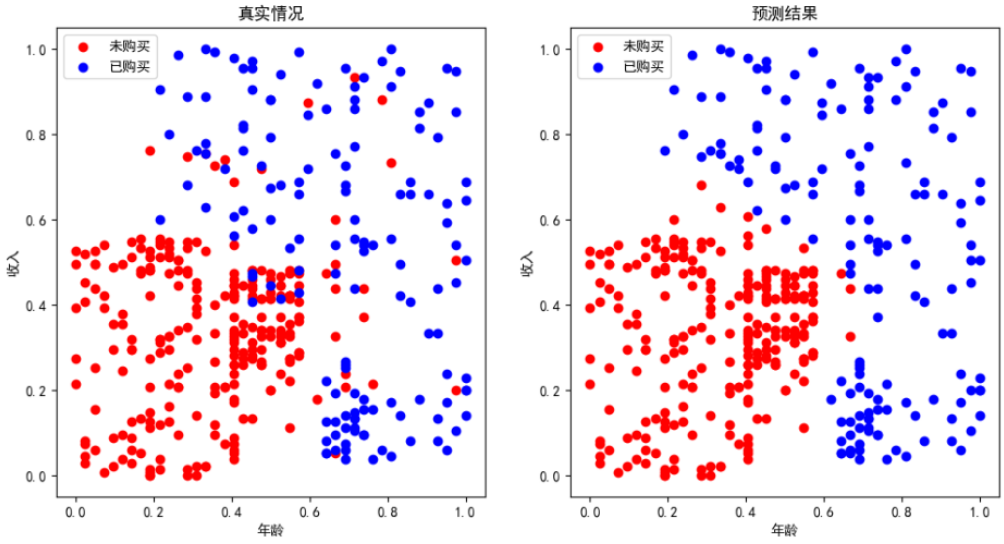
1. 实验结果展示示例（可图可表可文字，尽量可视化）

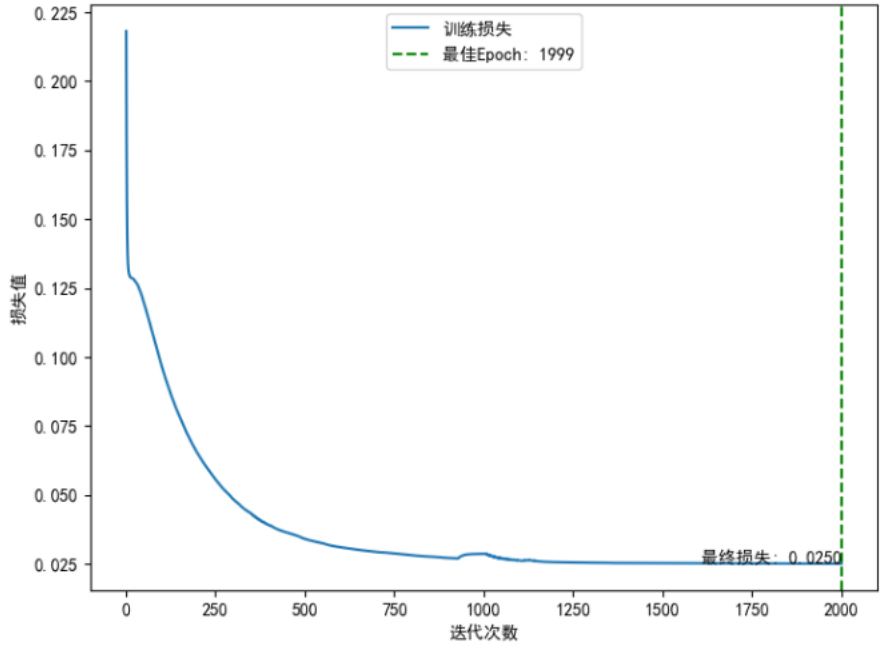
逻辑回归算法：




0.625
0.600
0.575
0.550
0.525
0.500
0.475
0.450
0.425
损失值
训练损失
最佳Epoch：2000
0
250
500
750
1000
1250
1500
1750
2000
迭代次数

MLP:


真实情况
未购买
已购买
收入
年龄
预测结果
未购买
已购买
收入
年龄



3. 评测指标展示及分析（机器学习实验必须有此项，其它可分析运行时间等）

逻辑回归算法：

由图得，最终计算后的准确率为 86%，最佳 loss 随迭代次数 epochs 增加而减少，

```
Epoch: 2998, Loss: 0.4312833674816657
Epoch: 2999, Loss: 0.43128254316182335
Epoch: 3000, Loss: 0.43128172009829885
最终准确率：86.00%，最佳loss：0.43128172009829885

进程已结束，退出代码为 0
```

增加每次迭代批量大小 size 为 64，会提升准确率。

```
Epoch: 2999, Loss: 0.3406817394638997
Epoch: 3000, Loss: 0.34067699150372927
最终准确率：86.25%，最佳loss：0.34067699150372927

进程已结束，退出代码为 0
```

图片显示的结果也与预期一致。

MLP：

由图得，最终计算后的准确率为 91%，最佳 loss 随迭代次数 epochs 增加而减少，

```
D:\homework\AI\第8周实验\.venv\Scripts\python.exe D:\homework\AI\第8周实验\MLP.py
最终准确率：91.00%，最佳loss：0.025034059644781323

进程已结束，退出代码为 0
```

与逻辑回归算法不同的是，倍数提升 size 并不会使最终准确率提高，反而会下降。当 size=128 时：

```
D:\homework\AI\第8周实验\.venv\Scripts\python.exe D:\homework\AI\第8周实验\MLP.py
最终准确率：90.25%，最佳loss：0.01566978895720945

进程已结束，退出代码为 0
```

目前最佳的是当 size=64 时：

```
D:\homework\AI\第8周实验\.venv\Scripts\python.exe D:\homework\AI\第8周实验\MLP.py
最终准确率：91.00%，最佳loss：0.018405560367149347

进程已结束，退出代码为 0
```

根据显示的生成损失曲线也可以看出，模型损失逐渐减小，最后趋于稳定，表明模型逐渐学习且得到较为稳定的结果，与逻辑回归算法效果类似。

三、 参考资料

通俗易懂--逻辑回归算法讲解(算法+案例)
多层感知机 Multilayer Perceptron | MLP