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	22336326	姓名	朱禹溪

一、 实验题目

分类

购房预测分类任务

`data.csv`数据集包含三列共400条数据，其中第一列`Age`表示用户年龄，第二列`EstimatedSalary`表示用户估计的薪水，第三列`Purchased`表示用户是否购房。请根据用户的年龄以及估计的薪水，利用逻辑回归算法和感知机算法预测用户是否购房，并画出数据可视化图、loss曲线图，计算模型收敛后的分类准确率。

提示

1. 最后提交的代码只需包含性能最好的实现方法和参数设置. 只需提交一个代码文件, 请不要提交其他文件.
2. 本次作业可以使用`numpy`库、`matplotlib`库以及python标准库.
3. 数据集可以在Github上下载，或者如下所示。

二、 实验内容

1. 算法原理

逻辑回归算法原理：

逻辑回归是一种广义的线性回归模型，用于解决二分类问题。它的基本思想是使用逻辑函数（sigmoid 函数）将线性回归的预测值映射到 0 和 1 之间，从而得到分类的概率。

线性回归：首先，模型会执行一个线性变换，计算输入特征 X 与权重向量 θ 的点积，得到一个线性预测值 z = Xθ。

逻辑函数（Sigmoid）：然后，使用 sigmoid 函数将线性预测值 z 映射到概率值 p，p = 1 / (1 + exp(-z))。这个概率值表示了样本属于正类的可能性。

损失函数：逻辑回归使用二元交叉熵损失函数（Binary Cross-Entropy Loss）来衡量预测概率与真实标签之间的差异。损失函数定义为 L = -1/N * Σ(y_i * log(p_i) + (1 - y_i) * log(1 - p_i))，其中 N 是样本数量，y_i 是真实标签，p_i 是预测概率。

优化算法：通过梯度下降算法来最小化损失函数，更新权重向量 θ。梯度下降通过计算损失函数关于权重 θ 的梯度，并按负梯度方向更新 θ，从而逐步减小损失函数的值。

感知机算法原理：

感知机是一种二分类的线性分类模型，它的输入是实例的特征向量，输出是实例的类别，分别是+1 和-1。感知机模型的假设空间是定义在特征空间中的所有线性分类模型或线性分类器的集合。

线性变换：与逻辑回归类似，感知机首先计算输入特征 X 与权重向量 w 的点积，并加上偏置项 b，得到激活值 z = Xw + b。

激活函数：感知机使用单位阶跃函数作为激活函数，将激活值 z 映射到类别标签。具体来说，如果 $z \geq 0$，则输出类别为+1；否则输出类别为-1。但在代码中，为了数值稳定性，通常使用符号函数的近似形式，如 np.where 函数。

损失函数（fit）：感知机的损失函数不是直接定义在输出上的，而是定义在误分类点上。

感知机学习的策略是极小化误分类点到超平面 S 的总距离。因此，损失函数可以视为误分类点的数量或误分类点到超平面的距离之和。

优化算法（fit）：感知机使用随机梯度下降算法来更新权重向量 w 和偏置项 b。在每次迭代中，随机选择一个误分类点，计算其对应的梯度，并按负梯度方向更新 w 和 b。

2. 关键代码展示（可选）

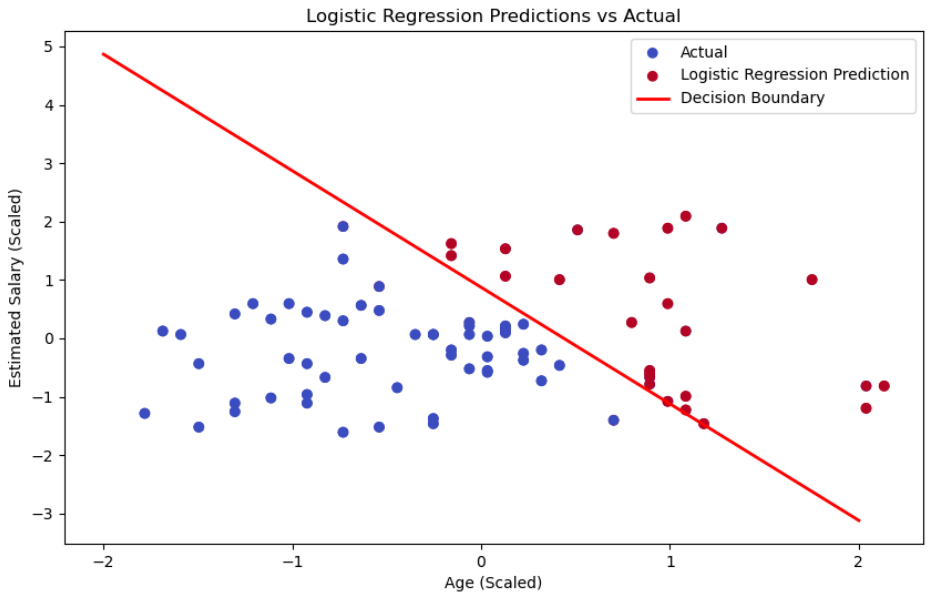
```python
import numpy as np
import matplotlib.pyplot as plt
class LogisticRegressionModel:
    def __init__(self, X_train, y_train, X_test, y_test, lr=0.01, epochs=10000):
        self.X_train = X_train
        self.y_train = y_train
        self.X_test = X_test
        self.y_test = y_test
        self.lr = lr
        self.epochs = epochs
        self.theta = np.zeros(X_train.shape[1])
        self.loss_history = []
    def sigmoid(self, z):#激活函数
        return 1 / (1 + np.exp(-z))
    def binary_cross_entropy_loss(self, y_true, y_pred):# 二元交叉熵损失函数
        return -np.mean(y_true * np.log(y_pred) + (1 - y_true) * np.log(1 - y_pred))
    def fit(self):#训练模型
        for _ in range(self.epochs):
            z = np.dot(self.X_train, self.theta)
            h = self.sigmoid(z)
            gradient = np.dot(self.X_train.T, (h - self.y_train)) / self.y_train.size
            self.theta -= self.lr * gradient
            loss = self.binary_cross_entropy_loss(self.y_train, h)
            self.loss_history.append(loss)
    def predict(self):
        return np.round(self.sigmoid(np.dot(self.X_test, self.theta)))
    def calculate_accuracy(self, predictions):
        return np.mean(predictions == self.y_test)
```

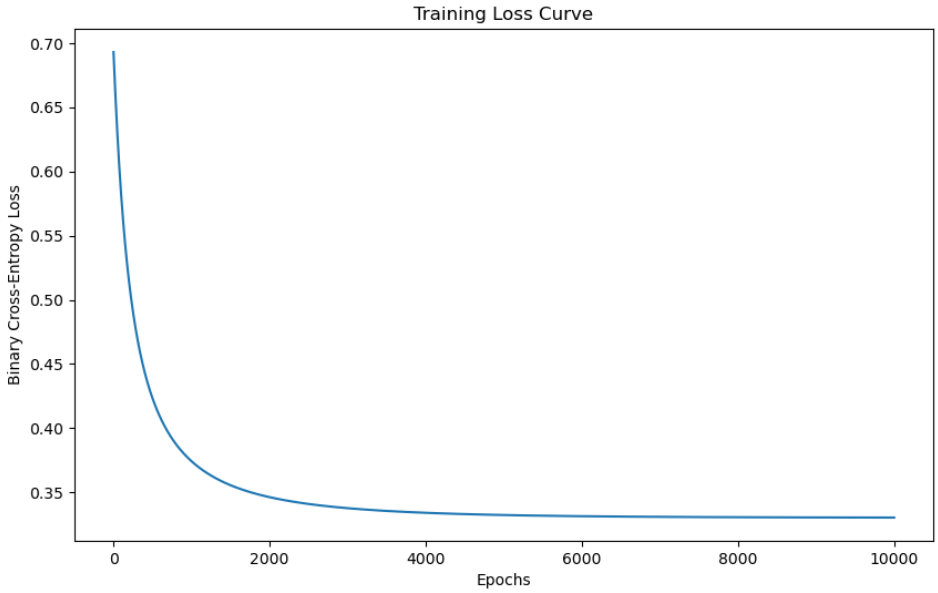
```python
class PerceptronModel:
    def __init__(self, X_train, y_train, X_test, y_test, learning_rate=0.01, epochs=1000, lambda_param=0.01):
        """
        初始化感知机模型。

        参数:
        X_train: 训练集特征数据
        y_train: 训练集标签数据
        X_test: 测试集特征数据
        y_test: 测试集标签数据
        learning_rate: 学习率,默认为0.01
        epochs: 迭代次数,默认为1000
        lambda_param: 正则化参数,默认为0.01
        """
        self.X_train = X_train
        self.y_train = y_train
        self.X_test = X_test
        self.y_test = y_test
        self.learning_rate = learning_rate
        self.epochs = epochs
        self.weights = np.zeros(X_train.shape[1])
        self.bias = 0
        self.loss_history = []
        self.lambda_param = lambda_param  # 正则化参数
```

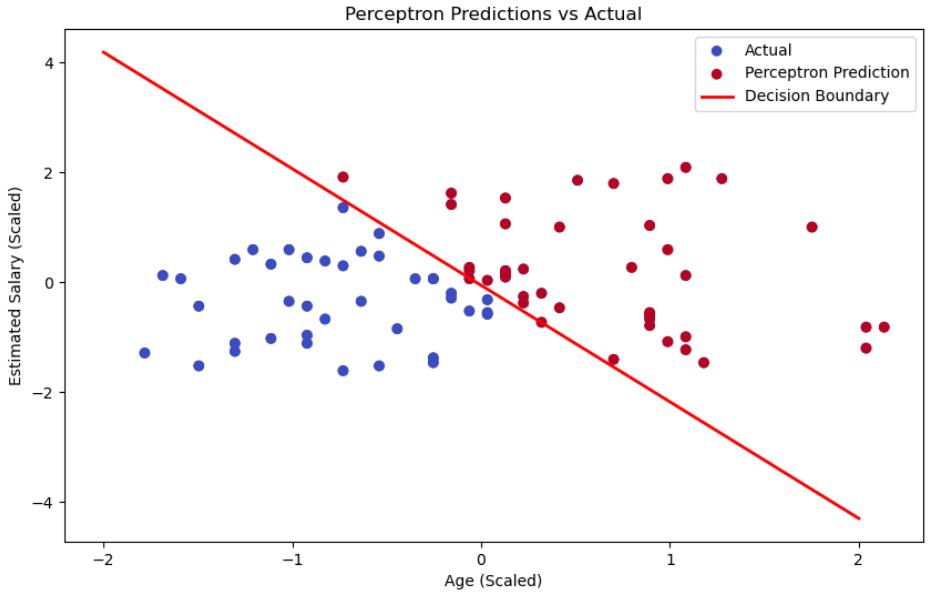
```python
    def fit(self):
        for _ in range(self.epochs):
            activation = np.dot(self.X_train, self.weights) + self.bias
            y_pred = np.where(activation >= 0, 1, 0)
            self.weights += self.learning_rate * (np.dot(self.X_train.T, (self.y_train - y_pred)) - self.lambda_param * self.weights)  # 添加L2正则化项
            self.bias += self.learning_rate * np.sum(self.y_train - y_pred)
            loss = np.mean(np.abs(self.y_train - y_pred)) + self.lambda_param / 2 * np.sum(self.weights ** 2)  # 添加L2正则化项
            self.loss_history.append(loss)

    def predict(self):
        return np.where(np.dot(self.X_test, self.weights) + self.bias >= 0, 1, 0)

    def calculate_accuracy(self, predictions):
        return np.mean(predictions == self.y_test)
```

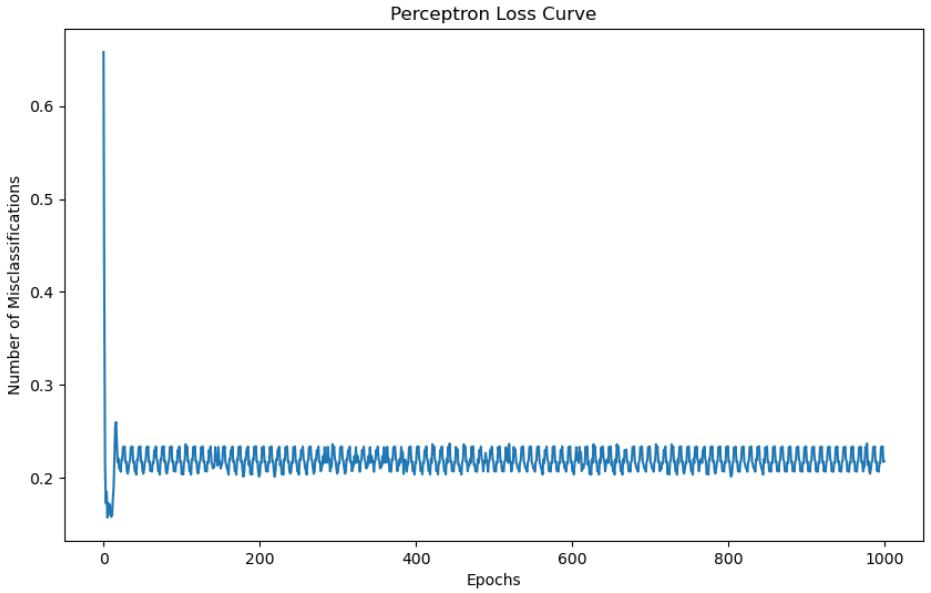
三、 实验结果及分析

1. 实验结果展示示例（可图可表可文字，尽量可视化）




Training Loss Curve
Binary Cross-Entropy Loss
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0.35
0
2000
4000
6000
8000
10000
Epochs


Perceptron Predictions vs Actual
Actual
Perceptron Prediction
Decision Boundary
Estimated Salary (Scaled)
4
2
0
−2
−4
−2
−1
0
1
2
Age (Scaled)




```
Logistic Regression Accuracy: 0.8625
Perceptron Accuracy: 0.775
```

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）

逻辑回归模型的评估准确率最高达到了 86.25%，感知机模型的评估准确率最高达到了 77.5%，感知机模型的结果图象不太理想，考虑到 PerceptronModel 类中只是实现了一个单层感知机模型，因而只能处理线性可分的数据，如果增加一个或多个隐藏层，可以处理更复杂的数据分布。另外对于损失函数，PerceptronModel 类使用的是误分类点的数量作为损失函数，如果优化成交叉熵损失函数并增加激活函数，则可能会有更加理想的结果。

对于两种算法学习率决定了模型参数在每次迭代时的更新幅度。如果学习率设置得过高，模型可能会在最优解附近震荡，无法收敛；如果学习率设置得过低，模型可能需要更多的迭代次数才能收敛，或者可能会陷入局部最优解。而迭代次数决定了模型训练的次数。如果迭代次数过少，模型可能无法收敛，即模型可能无法找到最小化损失函数的参数值；如果迭代次数过多，模型可能会过拟合，即模型在训练数据上的表现很好，但在未见过的数据上的表现较差。以上两种算法的默认值已经是我找到比较好的结果了，后续还需要进一部改进优化 。

四、 参考资料

PPT