湖南大學

HUNAN UNIVERSITY

人工智能实验四报告
逻辑回归

目录

一、实验目的

本实验要求应用逻辑回归模型解决实际问题，通过实验加深对逻辑回归原理的理解。建议使用python编程实现，并在Mindspore框架下实现。

二、实验描述

逻辑回归是一种广泛使用的分类算法，它用于预测一个二元因变量（响应变量）的概率。逻辑回归可以用于预测和因果关系分析。

基本思想：

1.损失函数：逻辑回归通过最小化损失函数（通常是交叉熵损失）来寻找最佳拟合模型。损失函数衡量了模型预测概率与实际类别之间的差异。

2.参数估计：通过最小化损失函数，可以估计逻辑回归模型的参数（系数）。这些参数表示了自变量对因变量对数几率的影响程度。

3.模型评估：使用诸如准确率、精确率、召回率、F1分数或接收者操作特征曲线（ROC曲线）等指标来评估模型的性能和预测能力。

4.预测：一旦模型训练完成，就可以使用它来预测新数据的因变量值，即预测属于某个类别的概率。

5.概率输出：逻辑回归的输出是一个介于0和1之间的概率值，表示属于正类（如患病或不患病）的概率。通常，通过设定一个阈值（如0.5），将概率转换为类别标签。

6.特征重要性：逻辑回归模型的系数可以用来评估各个特征对预测结果的影响，系数的绝对值越大，表示该特征对分类结果的影响越大。

7.正则化：为了防止过拟合，逻辑回归可以引入正则化项（如L1或L2正则化）到损失函数中。正则化参数控制了模型复杂度和拟合程度之间的权衡。

数据集处理：

使用 numpy 库的 loadtxt 方法来读取文件。

三、实验及结果分析

（1）开发语言及运行环境；

与实验一相同，不再赘述。

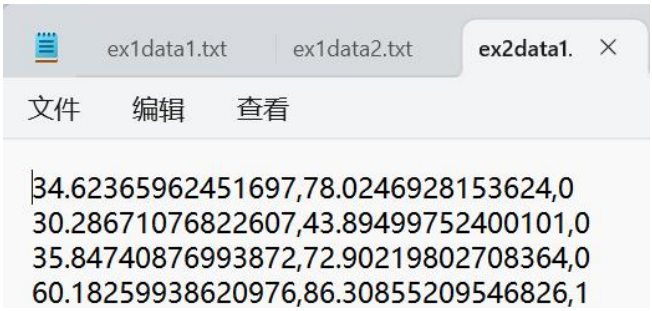
（2）实验的具体步骤；

题目：

应用逻辑回归模型预测某学生能否被大学录取。假设你是某大学的系主任，你想根据两次考试的结果决定每个申请者的录取机会。现有以往申请者的历史数据，可以此作为训练集建立逻辑回归模型进行预测。

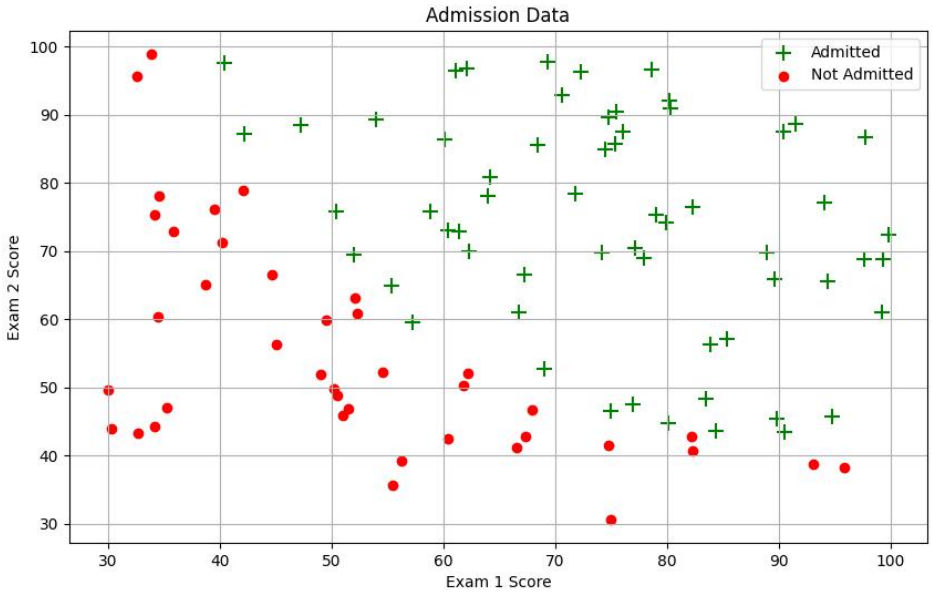
数据集：

文件ex2data1.txt为该实验的数据集，第一列、第二列分别表示申请者两次考试的成绩，第三列表示录取结果（1表示录取，0表示不录取）。



步骤与要求：

1）请导入数据并进行数据可视化，观察数据分布特征。（建议用python的matplotlib）



分类边界应该类似一条斜率为负的直线。

在后续计算时使用了特征标准化（main中调用）。

2）将逻辑回归参数初始化为0，然后计算代价函数（cost function）并求

出初始值。

```python
def gradient_descent(X, y, learning_rate=0.01, num_iters=1000):
    """梯度下降优化"""
    m, n = X.shape
    theta = np.zeros(n)
    cost_history = []  # 记录cost历史

    initial_cost, _ = compute_cost(X, y, theta)
    print(f"初始代价函数值: {initial_cost:.4f}")
    cost_history.append(initial_cost)

    print("开始训练模型:")
    for i in range(num_iters):
        cost, grad = compute_cost(X, y, theta)
        theta = theta - learning_rate * grad
        cost_history.append(cost)

        if i % 100 == 0:
            print(f"Iteration {i}: Cost = {cost:.4f}")

    return theta, cost_history
```

```python
def sigmoid(z):
    """Sigmoid函数"""
    return 1.0 / (1.0 + np.exp(-z))


2 usages
def compute_cost(X, y, theta):
    """计算代价函数值和梯度"""
    m = len(y)
    h = sigmoid(np.dot(X, theta))
    eps = 1e-15
    h = np.clip(h, eps, 1 - eps)
    J = (-1 / m) * np.sum(y * np.log(h) + (1 - y) * np.log(1 - h))
    grad = (1 / m) * np.dot(X.T, (h - y))
    return J, grad
```

compute_cost 这里使用的交叉熵损失对损失函数进行评估，使用sigmoid函数计算（初始的）预测值。（后面训练结束后计算预测值使用的是predict函数，在这个基础上点积了参数theta）

gradient_descent 是梯度下降优化函数，每一百次打印当前一次损失函数值。初始化参数theta为0，然后在过程中更新。

整个流程训练是：

训练过程：通过梯度下降找到最优的 theta 值

预测过程：使用这个训练好的 theta，代入到 sigmoid(np.dot(X, theta))中进行预测

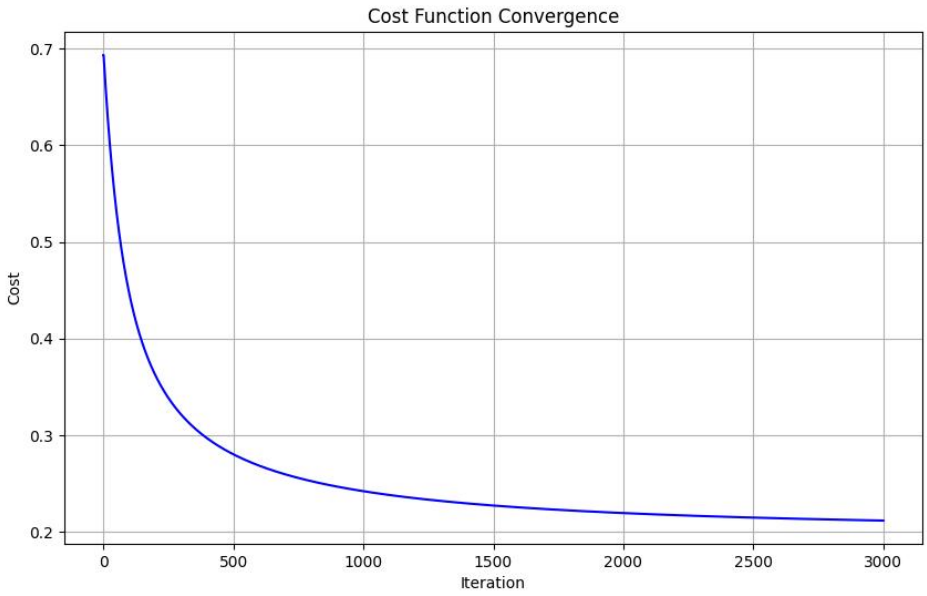
3）选择一种优化方法求解逻辑回归参数。

同上问，这里使用的是梯度下降算法。

```
136     # 4. 训练模型
137     theta, cost_history = gradient_descent(X_b, y, learning_rate=0.03, num_iters=3000)
```

参数我使用的0.03,3000，经过测试，在我的代码中这个参数效果会比较好。

cost函数：



4）某学生两次考试成绩分别为45、85，预测其被录取的概率。

```
143     # 5. 预测新学生
144     x_test = np.array([45, 85])
145     x_test_norm = (x_test - mu) / sigma
146     x_test_b = np.r_[1, x_test_norm]
147     prob = predict(x_test_b, theta)
148     print(f"\n考试成绩为[45, 85]的学生被录取的概率为: {prob * 100:.2f}%")
```

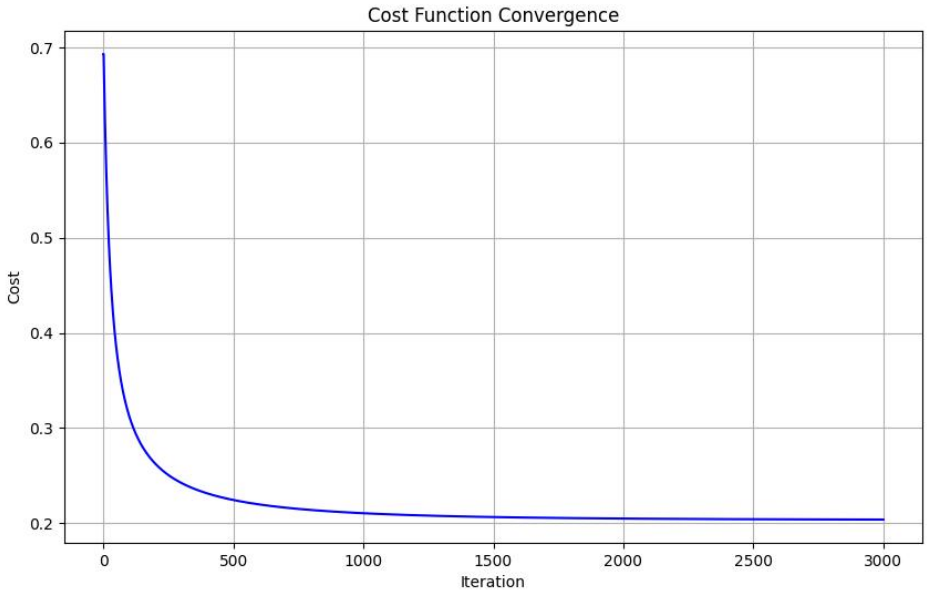
```
def predict(X, theta):
118     """预测函数"""
119     return sigmoid(np.dot(X, theta))
```

概率为：

```
考试成绩为[45, 85]的学生被录取的概率为: 69.94%
```

我发现换参数对概率的影响还挺大的，如果换成0.1,3000.

那么有：

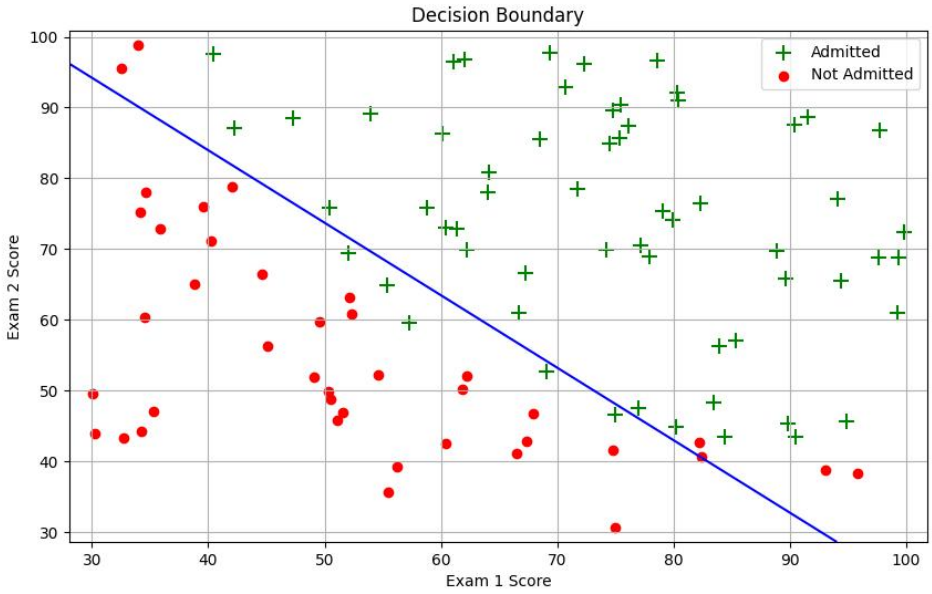



```
考试成绩为[45, 85]的学生被录取的概率为: 76.01%
```

最后cost会收敛在0.2035左右。

5）画出分类边界。

```
1 usage
def plot_decision_boundary(theta, X, y, mu, sigma):
    """绘制决策边界"""
    plot_data(X, y, 'Decision Boundary')

    # 计算决策边界点
    x1_min, x1_max = X[:, 0].min() - 2, X[:, 0].max() + 2
    x2_min, x2_max = X[:, 1].min() - 2, X[:, 1].max() + 2

    # 生成网格点
    xx1, xx2 = np.meshgrid(np.linspace(x1_min, x1_max, 100),
                           np.linspace(x2_min, x2_max, 100))

    # 将网格点标准化
    grid_points = np.c_[xx1.ravel(), xx2.ravel()]
    grid_points_norm = (grid_points - mu) / sigma

    # 添加截距项并计算预测值
    grid_points_norm = np.c_[np.ones(grid_points_norm.shape[0]), grid_points_norm]
    predictions = sigmoid(np.dot(grid_points_norm, theta))

    # 重塑预测值并绘制等高线
```



添加截距项并计算预测值：在网格点前添加一列全1，以包含截距项，然后计算这些点的预测概率。

（3）根据实验数据集，按实验要求给出相应的结果（截图）并对实验结果进行简要分析。

计算逻辑回归模型在给定数据集上的分类准确率：模型正确预测的样本数占总样本数的比例。

其余实验结果前面都有解释。

四、心得

在画分类边界那里，一开始应该是算的 prediction 有问题，画得一直有点问题，后面改掉了。

五、程序文件名清单

main.py：源代码

main.py.pdf：源代码 pdf 版

六、附录

代码如下。

main.py

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


def load_data(path):
    """加载数据"""
    data = np.loadtxt(path, delimiter=',')
    X = data[:, [0, 1]]
    y = data[:, 2]
    return X, y


def feature_normalize(X):
    """特征标准化"""
    mu = np.mean(X, axis=0)
    sigma = np.std(X, axis=0)
    X_norm = (X - mu) / sigma
    return X_norm, mu, sigma


def plot_data(X, y, title=''):
    """绘制数据点"""
    plt.figure(figsize=(10, 6))
    pos = y == 1
    neg = y == 0
    plt.scatter(X[pos, 0], X[pos, 1], c='green', marker='+',
                label='Admitted', s=100)
    plt.scatter(X[neg, 0], X[neg, 1], c='red', marker='o',
                label='Not Admitted')
    plt.xlabel('Exam 1 Score')
    plt.ylabel('Exam 2 Score')
    plt.legend()
    if title:
        plt.title(title)
    plt.grid(True)


def sigmoid(z):
    """Sigmoid函数"""
    return 1.0 / (1.0 + np.exp(-z))


def compute_cost(X, y, theta):
    """计算代价函数值和梯度"""
    m = len(y)
    h = sigmoid(np.dot(X, theta))
    eps = 1e-15
    h = np.clip(h, eps, 1 - eps)
    J = (-1 / m) * np.sum(y * np.log(h) + (1 - y) * np.log(1 - h))
    grad = (1 / m) * np.dot(X.T, (h - y))
    return J, grad


def gradient_descent(X, y, learning_rate=0.01, num_iters=1000):
    """梯度下降优化"""
    m, n = X.shape
    theta = np.zeros(n)
    cost_history = []  # 记录cost历史

    initial_cost, _ = compute_cost(X, y, theta)
    print(f"初始代价函数值: {initial_cost:.4f}")
    cost_history.append(initial_cost)

    print("开始训练模型:")
    for i in range(num_iters):
        cost, grad = compute_cost(X, y, theta)
```

```
67          theta = theta - learning_rate * grad
68          cost_history.append(cost)
69
70          if i % 100 == 0:
71              print(f"Iteration {i}: Cost = {cost:.4f}")
72
73      return theta, cost_history
74
75
76  def plot_cost_history(cost_history):
77      """绘制代价函数的收敛曲线"""
78      plt.figure(figsize=(10, 6))
79      plt.plot(cost_history, 'b-')
80      plt.xlabel('Iteration')
81      plt.ylabel('Cost')
82      plt.title('Cost Function Convergence')
83      plt.grid(True)
84      plt.show()
85
86
87  def plot_decision_boundary(theta, X, y, mu, sigma):
88      """绘制决策边界"""
89      plot_data(X, y, 'Decision Boundary')
90
91      # 计算决策边界点
92      x1_min, x1_max = X[:, 0].min() - 2, X[:, 0].max() + 2
93      x2_min, x2_max = X[:, 1].min() - 2, X[:, 1].max() + 2
94
95      # 生成网格点
96      xx1, xx2 = np.meshgrid(np.linspace(x1_min, x1_max, 100),
97                             np.linspace(x2_min, x2_max, 100))
98
99      # 将网格点标准化
100     grid_points = np.c_[xx1.ravel(), xx2.ravel()]
101     grid_points_norm = (grid_points - mu) / sigma
102
103     # 添加截距项并计算预测值
104     grid_points_norm = np.c_[np.ones(grid_points_norm.shape[0]), grid_points_norm]
105     predictions = sigmoid(np.dot(grid_points_norm, theta))
106
107     # 重塑预测值并绘制等高线
108     predictions = predictions.reshape(xx1.shape)
109     plt.contour(xx1, xx2, predictions, levels=[0.5], colors='blue', label='Decision Boundary')
110
111     plt.xlim(x1_min, x1_max)
112     plt.ylim(x2_min, x2_max)
113     plt.legend()
114     plt.show()
115
116
117 def predict(X, theta):
118     """预测函数"""
119     return sigmoid(np.dot(X, theta))
120
121
122 # 主程序
123 if __name__ == "__main__":
124     # 1. 加载数据
125     X, y = load_data('ex2data1.txt')
126     print("数据分布可视化：")
127     plot_data(X, y, 'Admission Data')
128     plt.show()
129
130     # 2. 特征标准化
131     X_norm, mu, sigma = feature_normalize(X)
132
133     # 3. 添加截距项
134     X_b = np.c_[np.ones((X_norm.shape[0], 1)), X_norm]
135
```

```python
    # 4. 训练模型
    theta, cost_history = gradient_descent(X_b, y, learning_rate=0.1, num_iters=5000)

    # 4.1 绘制代价函数收敛曲线
    print("\n代价函数收敛过程：")
    plot_cost_history(cost_history)

    # 5. 预测新学生
    x_test = np.array([45, 85])
    x_test_norm = (x_test - mu) / sigma
    x_test_b = np.r_[1, x_test_norm]
    prob = predict(x_test_b, theta)
    print(f"\n考试成绩为[45, 85]的学生被录取的概率为：{prob * 100:.2f}%")

    # 6. 绘制决策边界
    print("决策边界可视化：")
    plot_decision_boundary(theta, X, y, mu, sigma)

    # 7. 打印模型准确率
    predictions = predict(X_b, theta) >= 0.5
    accuracy = np.mean(predictions == y)
    print(f"\n模型准确率：{accuracy * 100:.2f}%")
```