《模式识别与机器学习 A》实验报告

实验题目：逻辑回归

班级：2203601

学号：2022113416

姓名：刘子康

实验报告内容

一、实验目的

- 理解逻辑回归模型的原理，掌握逻辑回归模型的参数估计算法；
- 利用逻辑回归模型实现二分类算法，并生成数据集或利用实际数据加以验证。

二、实验内容

实现两种损失函数的参数估计（1.无惩罚项；2.加入对参数的惩罚），可以采用梯度下降、共轭梯度或者牛顿法等。

- 手动生成两个类别数据（可以用高斯分布），验证逻辑回归模型算法。考察特征分布在满足和不满足朴素贝叶斯假设时，分别会得到什么样的结果。
- 在 UCI 网站上找一些实际数据加以测试，验证逻辑回归模型在不同领域的广泛用处，例如广告预测。

三、实验环境

- 操作系统：Windows 11
- 编程语言：Python 3.10
- 第三方库：Numpy 1.23.4，Matplotlib 3.8.2
- IDE：Pycharm 2022 社区版

四、实验过程、结果及分析

4.1 实验原理

逻辑回归（Logistic Regression）是一种常用于二分类问题的机器学习算法。它基于线性回归模型的思想，但通过引入一个 Sigmoid 函数（也称为逻辑函数），将输出限制在 0 到 1 之间，从而进行类别概率预测，达到分类效果。

逻辑回归的核心思想是建立一个线性模型来进行分类。对于一个输入特征向量 $X = [x_1, x_2, \dots, x_n]$，线性回归的预测结果可以表示为：$y = w_0 + w_1 \cdot x_1 + w_2 \cdot x_2 + \dots + w_n \cdot x_n$，其中，$w_0$是偏置项 b，$(w_1, w_2, \dots, w_n)$是权重（系数向量）w，这个公式表示了特征和目标之间的线性关系。在线性回归中，预测值 y 可以是任意实数，而在分类问题中，需要将预测值转换为一个概率值，表示样本属于某个类别的概率，这个转换是逻辑回归的关键步骤，通过一个非线性激活函数 Sigmoid 实现。

Sigmoid 函数的公式为 $sigmoid(z) = \frac{1}{1+e^{-z}}$，其中$z = w_0 + w_1 \cdot x_1 + w_2 \cdot x_2 + \dots + w_n \cdot x_n$，即线性回归的输出。Sigmoid 函数的值域在(0, 1)之间，当 z 趋近于正无穷时输出趋近于 1，当 z 趋近于负无穷时输出趋近于 0，适合用于表示概率。此时逻辑回归预测某样本属于类别 1 的后验概率为$P(y = 1|X) = sigmoid(w^T \cdot X) = \frac{1}{1+e^{-w^T \cdot X}}$。

损失函数选择交叉熵函数，并使用梯度下降法进行优化。

4.2 实验过程

4.2.1 生成二分类数据集

使用 sklearn 库生成并划分两个数据集，其特征分布分别满足和不满足朴素贝叶斯假设，并为两个数据集添加高斯噪声。

使用 datasets 模块的 make_classification 函数生成 1000 个样本（默认类别数为 2），为简化模型、方便绘图，设置特征数量为 2、每个类别的簇数为 1。对于特征分布满足朴素贝叶斯假设的数据集，设置信息特征数量为 2、冗余特征数量为 0；对于特征分布不满足朴素贝叶斯假设的数据集，设置信息特征数量为 1、冗余特征数量为 1，其中冗余特征由信息特征线性组合得到。

使用 model_selection 模块的 train_test_split 函数随机将数据集 75%划分为训练集，25%划分为测试集。

```python
# 生成二分类数据，flag表示是否满足朴素贝叶斯假设
单元测试 | 注释生成 | 代码解释 | 缺陷检测
def generate_data(n_samples=1000, flag=True):
    global random_seed
    if flag:
        X, y = make_classification(n_samples=n_samples, n_features=2, n_informative=2, n_redundant=0, n_clusters_per_class=1, random_state=random_seed)
        X[:, 0], X[:, 1] = X[:, 0] + np.random.normal(0, 0.36, n_samples), X[:, 1] + np.random.normal(0, 0.36, n_samples)
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=random_seed)
    else:
        X, y = make_classification(n_samples=n_samples, n_features=2, n_informative=1, n_redundant=1, n_clusters_per_class=1, random_state=random_seed)
        X[:, 0], X[:, 1] = X[:, 0] + np.random.normal(0, 0.64, n_samples), X[:, 1] + np.random.normal(0, 0.64, n_samples)
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=random_seed)
    return X_train, X_test, y_train, y_test
```

4.2.2 加载实际数据

UCI 数据集是一个常用的机器学习标准测试数据集，选择其中的 Diabetes 数据集进行模型测试。Diabetes 数据集用于糖尿病领域的研究和预测，其包含 442 个样本、8 个特征和 2 个输出类别，可用于二分类算法的测试。

从 Github 上下载 Diabetes 数据集的表格格式 diabetes.csv，放到代码文件同一目录下，使用 Pandas 库加载并转化为 Numpy 数组格式，之后划分数据集。

```python
    ''' 任务2 使用sklearn的diabetes数据集，构建并测试逻辑回归模型 '''
    # 加载并划分数据集
    diabetes = pd.read_csv('diabetes.csv')
    X = diabetes.drop('Outcome', axis=1).to_numpy()
    y = diabetes['Outcome'].to_numpy()
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=random_seed)
```

4.2.3 定义 Sigmoid 函数、损失函数和梯度计算函数

Sigmoid 函数输入为$w^T \cdot X$，输出为预测值。

```python
# Sigmoid 激活函数
单元测试 | 注释生成 | 代码解释 | 缺陷检测
def sigmoid(z):
    return 1 / (1 + np.exp(-z))
```

损失函数 ce_loss()，输入为真实值 y_true、预测值 y_pred、权重 w，lamda 为惩罚项系数（默认 0.1），flag 代表是否加入惩罚项（默认 False）。

```python
# 交叉熵损失函数Cross-Entropy，flag表示是否加入惩罚项
单元测试 | 注释生成 | 代码解释 | 缺陷检测
def ce_loss(y_true, y_pred, w, lamda=0.1, flag=False):
    epsilon = 1e-8  # 防止log(0)问题
    y_pred = np.clip(y_pred, epsilon, 1 - epsilon)
    if flag:
        return -np.mean(y_true * np.log(y_pred) + (1 - y_true) * np.log(1 - y_pred)) + lamda * np.sum(w ** 2)
    return -np.mean(y_true * np.log(y_pred) + (1 - y_true) * np.log(1 - y_pred))
```

梯度计算函数 gradient()，输入为特征矩阵 X、真实值 y_true、预测值 y_pred、权重 w，lamda 为惩罚项系数（默认 0.1），flag 代表是否加入惩罚项（默认 False）。每次迭代过程的下降梯度由损失函数对权重 w 和偏置 b 求偏导得到。

```python
# 梯度计算函数，flag表示是否加入惩罚项
单元测试 | 注释生成 | 代码解释 | 缺陷检测
def gradient(X, y_true, y_pred, w, lamda=0.1, flag=False):
    m = y_true.shape[0] # 样本量
    y = y_true.reshape(-1, 1)
    if flag:
        dw = (1/m) * np.dot(X.T, (y_pred - y)) + lamda * w  # 权重的梯度
        db = (1/m) * np.sum(y_pred - y)                     # 偏置的梯度
        return dw, db
    dw = (1/m) * np.dot(X.T, (y_pred - y))  # 权重的梯度
    db = (1/m) * np.sum(y_pred - y)         # 偏置的梯度
    return dw, db
```

4.2.4 训练过程

设置迭代次数 epochs（默认 1000）和学习率 lr（默认 0.01），初始化权重 w 和偏置 b，然后开始训练。在每次迭代中，特征矩阵 X 与权重 w 相乘再加上偏置 b，并通过 Sigmoid 函数的到预测值 y_pred，之后通过 ce_loss()函数和 gradient()函数计算当前 loss 值和下降梯度，更新参数并进入下一次迭代。

可以选择有惩罚项或无惩罚项，若选择有惩罚项，则在计算 loss 和梯度时加入正则化项，loss 为交叉熵加上$\lambda * \|w\|_2^2$，其中 λ 为惩罚项系数，$\|w\|_2^2$为权重 w 的 L2 范数的平方；权重梯度为损失函数对权重 w 的偏导数加上$\lambda * w$，偏置梯度不变。

```python
# 逻辑回归模型训练，flag表示是否加入惩罚项
单元测试 | 注释生成 | 代码解释 | 缺陷检测
def train_logistic_regression(X, y, lr=0.01, epochs=1000, flag=False):
    m, n = X.shape  # m: 样本量, n: 特征数
    # 初始化参数
    w = np.zeros((n, 1))
    b = 0

    # 训练过程
    for epoch in range(epochs):
        # 计算预测值
        z = np.dot(X, w) + b
        y_pred = sigmoid(z)

        # 计算损失和梯度
        loss_value = ce_loss(y, y_pred, w, flag)
        dw, db = gradient(X, y, y_pred, w, flag)

        # 更新参数
        w -= lr * dw
        b -= lr * db

        # 每迭代1/10打印一次loss
        if epoch % (epochs // 10) == 0:
            print(f"Epoch {epoch}, Loss: {loss_value}")

    return w, b
```

4.2.5 输出结果并绘图

predict()函数使用得到的权重 w 和偏置 b 对测试集进行预测，并返回预测值。accuracy()函数比较真实值和预测值，计算准确率。

```python
# 预测函数
单元测试 | 注释生成 | 代码解释 | 缺陷检测
def predict(X, w, b):
    z = np.dot(X, w) + b
    y_pred = sigmoid(z)
    return (y_pred >= 0.5).astype(int).reshape(-1)

# 准确率
单元测试 | 注释生成 | 代码解释 | 缺陷检测
def accuracy(y_true, y_pred):
    return np.mean(y_true == y_pred)
```

对于任务 1，输出测试准确率，并使用 matplotlib 库进行绘图，将数据集两个类别可视化，并分别绘制未加入惩罚项和加入惩罚项后的分界曲线。

```python
        # 结果预测
        y_pred = predict(X_test, w, b)

        # 计算准确率
        acc = accuracy(y_test, y_pred)
        print(f"Test Accuracy: {acc:.3f}")
        print("=" * 50)

        weights.append(w)
        bias.append(b)

    #绘图
    x_class = np.linspace(np.min(X_train[:, 0]), np.max(X_train[:, 0]), 100)
    y_class = [(-w[0] * x_class - b) / w[1] for w, b in zip(weights, bias)]

    plt.scatter(X_train[y_train == 0, 0], X_train[y_train == 0, 1], c= 'red', s=10, label='category_1')
    plt.scatter(X_train[y_train == 1, 0], X_train[y_train == 1, 1], c= 'blue', s=10, label='category_2')
    plt.plot(x_class, y_class[0], color='green', label='classification_line_noL2')
    plt.plot(x_class, y_class[1], color='orange', label='classification_line_L2')
    plt.xlabel('x1')
    plt.ylabel('x2')
    plt.legend()
    plt.show()
```

对于任务 2，由于使用的糖尿病数据集特征数较多，不便于绘图，故只输出测试准确率。

```python
        # 结果预测
        y_pred = predict(X_test, w, b)

        # 计算准确率
        acc = accuracy(y_test, y_pred)
        print(f"Test Accuracy: {acc:.3f}")
```

4.3 实验结果及分析

4.3.1 任务 1

- 特征分布满足朴素贝叶斯假设
  - 无惩罚项

    训练过程 loss 值变化及测试准确率如下：

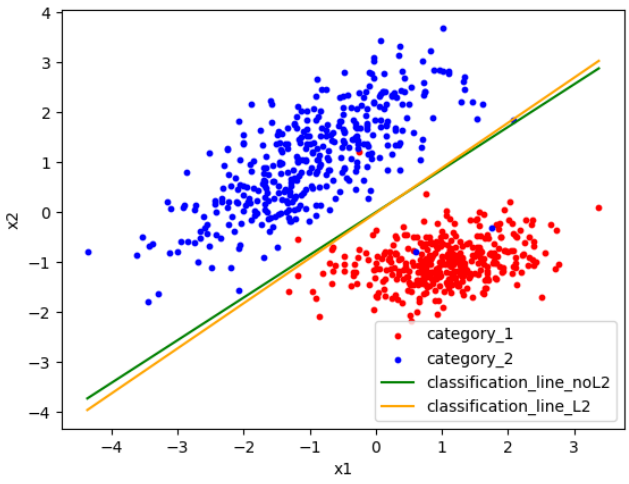
```
Epoch 0, Loss: 0.6931471805599453
Epoch 100, Loss: 0.778595553732291
Epoch 200, Loss: 0.905638072311607
Epoch 300, Loss: 1.0202719359322026
Epoch 400, Loss: 1.11898567147714
Epoch 500, Loss: 1.204446026223081
Epoch 600, Loss: 1.2794231679785584
Epoch 700, Loss: 1.3460777160327544
Epoch 800, Loss: 1.406023429622188
Epoch 900, Loss: 1.4604690758293812
Test Accuracy: 0.988
```

  - 有惩罚项

    训练过程 loss 值变化及测试准确率如下：

```
Epoch 0, Loss: 0.6931471805599453
Epoch 100, Loss: 0.8469594572500091
Epoch 200, Loss: 0.9757629571603608
Epoch 300, Loss: 1.0351073586497148
Epoch 400, Loss: 1.0600167400082372
Epoch 500, Loss: 1.0701960126661723
Epoch 600, Loss: 1.0743184178201521
Epoch 700, Loss: 1.0759829818778657
Epoch 800, Loss: 1.0766546276576432
Epoch 900, Loss: 1.0769256274614514
Test Accuracy: 0.988
```

    总体数据绘图如下：



- 特征分布不满足朴素贝叶斯假设
  - 无惩罚项

    训练过程 loss 值变化及测试准确率如下：

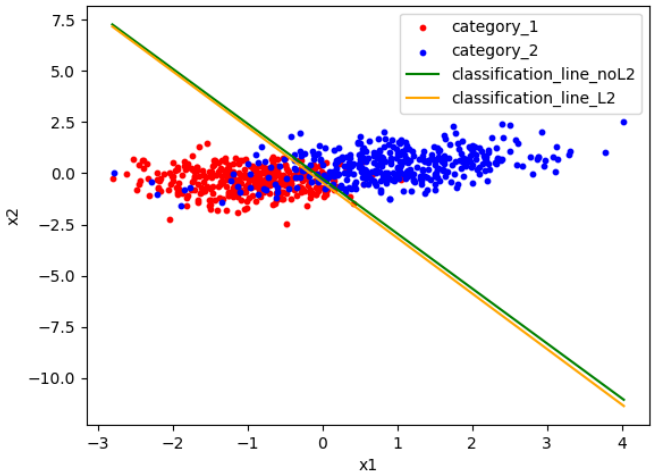
```
Epoch 0, Loss: 0.6931471805599453
Epoch 100, Loss: 0.7312734176395584
Epoch 200, Loss: 0.7955436874310301
Epoch 300, Loss: 0.8581422694565193
Epoch 400, Loss: 0.9143465639149667
Epoch 500, Loss: 0.964109928529127
Epoch 600, Loss: 1.0082526531827687
Epoch 700, Loss: 1.0476429097645388
Epoch 800, Loss: 1.0830280755175508
Epoch 900, Loss: 1.1150194510651208
Test Accuracy: 0.928
```

  - 有惩罚项

    训练过程 loss 值变化及测试准确率如下：

```
Epoch 0, Loss: 0.6931471805599453
Epoch 100, Loss: 0.7435714471747937
Epoch 200, Loss: 0.8201157689975855
Epoch 300, Loss: 0.8864075324822753
Epoch 400, Loss: 0.9387001558075947
Epoch 500, Loss: 0.9789255021871659
Epoch 600, Loss: 1.0096383089829255
Epoch 700, Loss: 1.0330496055291056
Epoch 800, Loss: 1.050905002923069
Epoch 900, Loss: 1.0645428866750082
Test Accuracy: 0.928
```

    总体数据绘图如下：



4.3.2 任务 2

- 无惩罚项

训练过程 loss 值变化及测试准确率如下：

```
Epoch 1000, Loss: 7.0895501712644196
Epoch 2000, Loss: 6.601079839051725
Epoch 3000, Loss: 11.819103988097497
Epoch 4000, Loss: 6.621413639206468
Epoch 5000, Loss: 6.405804471244826
Epoch 6000, Loss: 8.798034817307206
Epoch 7000, Loss: 6.990682492791736
Epoch 8000, Loss: 6.638109159689238
Epoch 9000, Loss: 9.460161499527663
Test Accuracy: 0.693
```

- 有惩罚项

训练过程 loss 值变化及测试准确率如下：

```
Epoch 1000, Loss: 9.323064680311866
Epoch 2000, Loss: 14.318743214794747
Epoch 3000, Loss: 9.199264198168382
Epoch 4000, Loss: 14.200663364389548
Epoch 5000, Loss: 9.74723345144794
Epoch 6000, Loss: 8.433337217745162
Epoch 7000, Loss: 14.381182891442668
Epoch 8000, Loss: 14.089148806777786
Epoch 9000, Loss: 8.925527741921037
Test Accuracy: 0.646
```

五、实验总体结论

逻辑回归是一种广义的线性回归分析模型，属于机器学习中的监督学习，主要用于解决二分类问题。其关键点在于 Sigmoid 函数，将计算值映射到(0, 1)之间，可以表示类别预测概率。逻辑回归具有简单高效、鲁棒性强、输出概率应用广泛的优点，但也存在不适用于非线性分布的数据、假设特征独立、需改进才能用于多分类问题的缺点。

使用交叉熵作为损失函数，梯度下降法作为优化方法，实现了逻辑回归模型在手动生成的二分类数据集和 UCI 中的糖尿病数据集上的训练和预测，并在测试集上达到了较好的分类效果，但也存在一些误分类点，可能与数据本身特点和模型较为简单有关，后续可使用其他分类模型或优化算法来提高性能。

在不满足朴素贝叶斯假设的数据集上，由于特征之间不是相互独立，导致有效特征变少，逻辑回归模型性能略有下降。

手动生成的二分类数据集，分类任务较为简单，迭代次数较多，模型出现过拟合现象，在加入惩罚项后过拟合现象得到一定程度的抑制。

六、完整实验代码

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_classification
```

```
6.
7.
8.  random_seed = 43
9.  np.random.seed(random_seed)
10.
11. # 生成二分类数据，flag 表示是否满足朴素贝叶斯假设
12. def generate_data(n_samples=1000, flag=True):
13.     global random_seed
14.     if flag:
15.         X, y = make_classification(n_samples=n_samples, n_features=2, n_informative=2, n_r
    edundant=0, n_clusters_per_class=1, random_state=random_seed)
16.         X[:, 0], X[:, 1] = X[:, 0] + np.random.normal(0, 0.36, n_samples), X[:, 1] + np.ra
    ndom.normal(0, 0.36, n_samples)
17.         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_s
    tate=random_seed)
18.     else:
19.         X, y = make_classification(n_samples=n_samples, n_features=2, n_informative=1, n_r
    edundant=1, n_clusters_per_class=1, random_state=random_seed)
20.         X[:, 0], X[:, 1] = X[:, 0] + np.random.normal(0, 0.64, n_samples), X[:, 1] + np.ra
    ndom.normal(0, 0.64, n_samples)
21.         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_s
    tate=random_seed)
22.     return X_train, X_test, y_train, y_test
23.
24. # Sigmoid 激活函数
25. def sigmoid(z):
26.     return 1 / (1 + np.exp(-z))
27.
28. # 交叉熵损失函数Cross-Entropy，flag 表示是否加入惩罚项
29. def ce_loss(y_true, y_pred, w, lamda=0.5, flag=False):
30.     epsilon = 1e-8  # 防止 log(0)问题
31.     y_pred = np.clip(y_pred, epsilon, 1 - epsilon)
32.     if flag:
33.         return -
    np.mean(y_true * np.log(y_pred) + (1 - y_true) * np.log(1 - y_pred)) + lamda * np.sum(w **
     2)
34.     return -np.mean(y_true * np.log(y_pred) + (1 - y_true) * np.log(1 - y_pred))
35.
36. # 梯度计算函数，flag 表示是否加入惩罚项
37. def gradient(X, y_true, y_pred, w, lamda=0.5, flag=False):
38.     m = y_true.shape[0] # 样本量
39.     y = y_true.reshape(-1, 1)
40.     if flag:
41.         dw = (1/m) * np.dot(X.T, (y_pred - y)) + lamda * w  # 权重的梯度
```

```
        db = (1/m) * np.sum(y_pred - y)                    # 偏置的梯度
        return dw, db
    dw = (1/m) * np.dot(X.T, (y_pred - y))  # 权重的梯度
    db = (1/m) * np.sum(y_pred - y)         # 偏置的梯度
    return dw, db

# 逻辑回归模型训练，flag 表示是否加入惩罚项
def train_logistic_regression(X, y, lr=0.01, epochs=1000, lamda=0.5, flag=False):
    m, n = X.shape  # m: 样本量, n: 特征数
    # 初始化参数
    w = np.zeros((n, 1))
    b = 0

    # 训练过程
    for epoch in range(epochs):
        # 计算预测值
        z = np.dot(X, w) + b
        y_pred = sigmoid(z)

        # 计算损失和梯度
        loss_value = ce_loss(y, y_pred, w, lamda=lamda, flag=flag)
        dw, db = gradient(X, y, y_pred, w, lamda=lamda, flag=flag)

        # 更新参数
        w -= lr * dw
        b -= lr * db

        # 每迭代 1/10 打印一次 loss
        if epoch % (epochs // 10) == 0:
            print(f"Epoch {epoch}, Loss: {loss_value}")

    return w, b

# 预测函数
def predict(X, w, b):
    z = np.dot(X, w) + b
    y_pred = sigmoid(z)
    return (y_pred >= 0.5).astype(int).reshape(-1)

# 准确率
def accuracy(y_true, y_pred):
    return np.mean(y_true == y_pred)

# 主程序
```

```python
if __name__ == "__main__":
    is_Bayes = True
    is_punish = True

    ''' 任务 1 手动生成数据集，构建并测试逻辑回归模型 '''
    X_train, X_test, y_train, y_test = generate_data(flag=is_Bayes)

    weights, bias = [], []
    for flag in [False, True]:
        # 逻辑回归模型训练
        w, b = train_logistic_regression(X_train, y_train, flag=flag)

        # 结果预测
        y_pred = predict(X_test, w, b)

        # 计算准确率
        acc = accuracy(y_test, y_pred)
        print(f"Test Accuracy: {acc:.3f}")
        print("=" * 50)

        weights.append(w)
        bias.append(b)

    #绘图
    x_class = np.linspace(np.min(X_train[:, 0]), np.max(X_train[:, 0]), 100)
    y_class = [(-w[0] * x_class - b) / w[1] for w, b in zip(weights, bias)]

    plt.scatter(X_train[y_train == 0, 0], X_train[y_train == 0, 1], c= 'red', s=10, label='category_1')
    plt.scatter(X_train[y_train == 1, 0], X_train[y_train == 1, 1], c= 'blue', s=10, label='category_2')
    plt.plot(x_class, y_class[0], color='green', label='classification_line_noL2')
    plt.plot(x_class, y_class[1], color='orange', label='classification_line_L2')
    plt.xlabel('x1')
    plt.ylabel('x2')
    plt.legend()
    plt.show()


    ''' 任务 2 使用 UCI 的 Diabetes 数据集，构建并测试逻辑回归模型 '''
    # 加载并划分数据集
    diabetes = pd.read_csv('diabetes.csv')
    X = diabetes.drop('Outcome', axis=1).to_numpy()
    y = diabetes['Outcome'].to_numpy()
```

```
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=random_seed)

    # 逻辑回归模型训练
    w, b = train_logistic_regression(X_train, y_train, lr=0.05, epochs=10000, lamda=0.1, flag=is_punish)

    # 结果预测
    y_pred = predict(X_test, w, b)

    # 计算准确率
    acc = accuracy(y_test, y_pred)
    print(f"Test Accuracy: {acc:.3f}")
```

七、参考文献

[1] 刘远超. 深度学习基础: 高等教育出版社, 2023.

[2] 阿库塔姆. Logistic 回归（逻辑回归）及 python 代码实现[EB/OL].CSDN 博客, 2023-07-03 [2024-11-22]. https://blog.csdn.net/weixin_50744311/article/details/131523136.

[3] 思绪无限.UCI 数据集整理（附论文常用数据集）[EB/OL].CSDN 博客, 2022-09-27[2024-11-22]. https://blog.csdn.net/qq_32892383/article/details/82225663.