第 1 章　实验名称:逻辑回归算法

1.1　实验目的:

1. 理解 pandas 进行数据清洗的基本方法,掌握 torch.utils.data.Dataset 和 DataLoader 的实现细节及其在深度学习数据中的应用。
   通过 pandas,可以实现数据的读取、缺失值处理、数据类型转换及异常值检测等清洗步骤;同时,Dataset 用于定义数据集结构,DataLoader 提供批量加载和数据 shuffle 功能,广泛应用于深度学习的数据预处理与输入。
2. 理解逻辑回归(Logistic Regression)的基本概念和原理,熟练掌握逻辑回归模型在二分类任务(如贷款违约预测)中的应用。
   逻辑回归基于 sigmoid 函数将线性回归输出映射到 (0,1) 区间,用于概率预测;其原理通过最大似然估计优化参数,在贷款违约预测中可根据特征(如收入、信用评分)区分违约与非违约。
3. 构建能够处理结构化数据的逻辑回归模型,并实现对数据样本进行准确的二分类预测。
   通过特征工程处理结构化数据(如表格形式的数值与类别特征),结合逻辑回归模型训练,利用交叉熵损失函数优化,确保对样本进行高效且准确的二分类预测。

1.2　实验原理

1.2.1　逻辑回归

回归(regression)是能为一个或多个自变量与因变量之间关系建模的一类方法。在自然科学和社会科学领域,回归经常用来表示输入和输出之间的关系。

在机器学习领域中的大多数任务通常都与预测(prediction)有关。当我们想预测一个数值时,就会涉及到回归问题。常见的例子包括:预测价格(房屋、股票等)、预测住院时间(针对住院病人等)、0-1 变量预测(预测是否贷款)。

为了解释线性回归,我们举一个实际的例子:需要根据贷款申请人的数据信息预测其是

否有违约的可能，以此判断是否通过此项贷款。为了开发一个能预测是否有违约可能的模型，我们需要收集一个真实的数据集。这个数据集包括了贷款申请人的各种数据信息。在机器学习的术语中，该数据集称为训练数据集（training data set）或训练集（training set）。每行数据（比如一个用户相对应的数据）称为样本（sample），也可以称为数据点（data point）或数据样本（data instance）。我们把试图预测的目标（比如是否违约）称为标签（label）或目标（target）。预测所依据的自变量（面积和房龄）称为特征（feature）。

通常，我们使用 n 来表示数据集中的样本数。对索引为 i 的样本，其输出表示为 $\mathbf{x}^{(i)} = [x_1^{(i)}, \ldots, x_n^{(i)}]$。

线性模型

线性假设是指目标可以表示为特征的加权和：

$$logit = \mathbf{x}^{(i)} \cdot \mathbf{w} + b$$

其中 $\mathbf{w} = [w_1^{(i)}, \ldots, w_n^{(i)}]$ 称为权重，权重决定了每个特征对我们预测值的影响。b 称为偏置（bias）、偏移量（offset）或截距（intercept）。偏置是指当所有特征都取值为 0 时，预测值应该为多少。给定一个数据集，我们的目标是寻找模型的权重 $\mathbf{w}$ 和偏置 b，使得根据模型做出的预测大体合理。

逻辑函数（Logistic，也称为 sigmoid, logit）

可以发现，线性假设输出值无界，所以我们需要把线性输出值映射到输出概率 $[0, 1]$ 之间，参考逻辑回归 PPT$_{\text{sigmoid 函数部分}}$ 的 sigmoid 函数部分。

损失函数

在我们开始考虑如何用模型拟合（fit）数据之前，我们需要确定一个拟合程度的度量。损失函数（loss function）能够量化目标的实际值与预测值之间的差距。通常我们会选择非负数作为损失，且数值越小表示损失越小，完美预测时的损失为 0。

在这个实验部分，我们选择最大对数最大似然估计，对于所有权重（参数组合），我们选择一组参数，使得预测似然性最大。

极大似然估计　在统计学中，极大似然估计（Maximum Likelihood Estimation）是用来估计模型参数的一种方法，就是利用已知样本的结果信息，反推出最有可能导致这样结果的模型参数值。简而言之，最大似然估计旨在找到能使已知数据最“自然”、最“合理”的模型参数。

拓展阅读：花书 5.5$_{\text{花书 5.5}}$。

对数最大似然估计 由于本次实验相当于二分类问题，模型预测一个数据点为正样本的概率为：

$$\hat{y}_i = \hat{P}_{positive}(i) = \frac{1}{1 + e^{-logit}}$$

预测其为负样本的概率为：

$$\hat{P}_{negative}(i) = \frac{e^{-logit}}{1 + e^{-logit}}$$

结合最大似然估计，我们希望样本的预测概率尽可能接近真实标签。对于二分类问题，交叉熵损失函数（Cross-Entropy Loss）通常用于衡量预测概率与真实标签之间的差异。交叉熵损失函数定义如下：

$$L(y, \hat{y}) = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$$

其中，y_i 是第 i 个样本的真实标签，$\hat{y}_i$ 是第 i 个样本预测为正样本的概率，N 是样本总数。

1.3 实验工具：

1. Anaconda 及 vscode
2. Python 3.9 以上，PyTorch（适配版本）。本实验主要涉及 torch 的基础操作和多维数组（来表示神经网络中的输入、输出、激活和参数矩阵等）的相关操作，请阅读以下 torch 介绍教程，并尝试运行一些简单的 torch 函数：
   (1) torch 常用函数及基础用法

3. Jupyter：轻松地将 Markdown 文本和可执行 Python 源代码组合在一个称为 NoteBook 的画布上。Visual Studio Code 支持本机并通过 Python 代码文件使用 Jupyter Notebook。
   (1) Jupyter on vscode 教程

1.4 数据集准备和说明

1.4.1 数据介绍

本次实验的数据来源于某信贷平台的贷款记录，以个人信贷为背景，旨在根据贷款申请人的数据信息预测其是否有违约的可能性，从而判断是否通过贷款申请。数据集包含以下信息：

- **输入特征**：共 47 种特征，涵盖申请人的多种信息，包括：

  - 数值类变量:42 种,其中:
    - 33 种为连续型数值变量(如贷款金额、贷款利率、年收入等);
    - 9 种为离散型数值变量(如工作年限计数等)。
  - 类别类特征:5 种,包括:
    - 贷款等级(grade,如 A、B、C、D);
    - 就业年限(如 <1 year, 1-3 year 等);
    - 时间类信息(格式为 yyyy-mm-dd)。
  - 匿名特征:15 列为申请人行为计数特征,具体含义未明确标注。
- **输出标签**:`isDefault` 特征,取值为 0 或 1,其中 1 表示贷款违约,0 表示未违约。
- **数据集划分**:共 1 万条数据,其中:
  - 训练集 8000 条(50% 为正样本,即违约贷款);
  - 验证集 1000 条;
  - 测试集 1000 条。

该数据集以表格形式存储,每一行代表一个贷款申请样本,每一列对应一种特征或标签,为后续逻辑回归模型的训练和评估提供了丰富的结构化数据。

1.4.2　数据清洗

为了将原始数据集转化为适合逻辑回归模型训练的结构化数据,我们设计并实现了一个数据清理流程,基于 Python 的 `pandas` 库。以下是数据清理的关键步骤及其实现方式:

数据初始化

在数据初始化阶段,我们从原始数据框中移除无关列并识别数值型特征:

- 使用 `df.select_dtypes(exclude=['object'])` 筛选出数值型特征列,存储在 `self.feature` 中。
- 删除 `id` 列,因其为唯一标识符,对预测无直接贡献。

日期特征处理

日期特征(如 `issueDate` 和 `earliesCreditLine`)需转换为数值形式:

- 对于 `issueDate`(格式为 yyyy-mm-dd):
  - 使用 `pd.to_datetime` 将其转换为日期时间格式。
  - 计算与最早日期的差值(单位:天),结果存储为整数。
- 对于 `earliesCreditLine`(格式为 MMM-YYYY,如 Jan-2020):
  - 转换为日期时间格式后,计算与最早日期的月份差值,公式为:

$$\text{月份差} = (\text{年} - \text{最早年}) \times 12 + (\text{月} - \text{最早月})$$

贷款等级处理

贷款等级（grade）为类别型变量，需转换为数值：

- 删除 subGrade 列，因其为次级分类，与 grade 冗余。
- 将 grade（如 A、B、C）映射为整数，方法为：

$$\text{grade\_int} = \text{ord}(\text{grade}[0]) - \text{ord}(\text{'A'}) + 1$$

例如，A 映射为 1，B 映射为 2，依此类推。

就业年限处理

就业年限（employmentLength）为文本型变量，需提取并转换为数值：

- 使用正则表达式匹配模式 (.*?)(?=\s*years)，提取年限部分。
- 转换规则：
  - "< 1 year" 转换为 1；
  - "10+ years" 转换为 12；
  - 其他（如"3 years"）转换为整数加 1（如 3+1=4）。

数据清理流水线

上述步骤整合为一个流水线函数 pipeline，最终将所有特征转换为浮点数类型（float），便于模型训练。

以下是完整的实现代码：

```
from typing import List
import pandas as pd

class data_utils:

    def __init__(self, df):
        self.df = df
        self.feature = df.select_dtypes(exclude=['object']).columns
        self.df = self.df.drop('id', axis=1)

    def drop_col(self, col:str):
        self.df = self.df.drop(col, axis=1)

    def date_init(self):
```

```
        self.df['issueDate'] = pd.to_datetime(self.df['issueDate'], format=
            '%Y-%m-%d')
        min_issue = self.df['issueDate'].min()
        self.df['issueDate'] = (self.df['issueDate'] - min_issue).dt.days

        self.df['earliesCreditLine'] = pd.to_datetime(self.df['
            earliesCreditLine'], format='%b-%Y')
        min_date = self.df['earliesCreditLine'].min()
        self.df['earliesCreditLine'] = (self.df['earliesCreditLine'].dt.
            year - min_date.year) * 12 + (self.df['earliesCreditLine'].dt.
            month - min_date.month)

    def grade(self):
        self.df = self.df.drop('subGrade', axis=1)
        def grade2int(x):
            if pd.isnull(x):
                return x
            else:
                return ord(x[0]) - ord('A') + 1
        self.df['grade'] = self.df['grade'].apply(grade2int)

    def employmentLength(self):
        import re
        def emp_length(x):
            pattern = r'(.*?)(?=\s*years)'
            if pd.isnull(x):
                return x
            if re.match(pattern, x):
                year = re.match(pattern, x).group(1)
                if year == '< 1':
                    return 1
                elif year == '10+':
                    return 12
                else:
                    return int(year)+1
        self.df['employmentLength'] = self.df['employmentLength'].apply(
```

```
            emp_length)

    def pipeline(self):
        self.date_init()
        self.grade()
        self.employmentLength()
        self.df = self.df.astype(float)
        return self.df
```

通过上述清理流程，原始数据被转换为纯数值型表格，确保逻辑回归模型能够高效处理。

1.5 实验步骤：

代码文件下载

逻辑回归实验代码，密码：2025ai

1.5.1 环境搭建

(1) 访问 Anaconda 清华镜像，如果已有 conda，跳过 conda 安装步骤。https://mirrors.tuna.tsinghua.edu.cn/anaconda/archive/，选择安装合适的版本（如 Windows 系统推荐最新的 Anaconda3-5.3.1-Windows_x86_64.exe）。建议安装路径不带中文。需要勾选添加到 PATH；

(2) 安装完成后打开命令行（Win+R，输入 cmd，打开命令行）输入：`conda -version` 如果返回版本号（如 conda 23.7.4），说明 Conda 已正确安装并在 PATH 中；

(3) 在命令行中输入 `conda create -n logic python=3.9 -y` 创建一个虚拟环境；

(4) 在命令行中输入 `conda activate logic` 激活环境，当命令行显示：

```
(logic) C:\Users\yourusername>
```

表明激活成功；

(5) 激活成功后分别在命令行中输入命令（以下只是参考 python 包环境，本实验并不对环境版本有十分严格要求，绝大多数较新版本包都能完成本实验。）：

```
pip install torch==2.6.0
pip install numpy==1.26.4
pip install tqdm
```

```
pip install scikit-learn
pip install jupyter
pip install pandas
```

(6) 安装完外部库后输入命令 `pip list` 可以确认上述外部库已成功安装到虚拟环境中;
(7) 访问 https://code.visualstudio.com/download 下载 VSCode。安装完成后,点击左侧扩展,安装 Python 相关插件,推荐插件包:Python Extension Pack;Jupyter.
(8) 新建一个目录,将所有脚本文件与数据文件放入其中,在 Vscode 中依次选择 file,open the folder 打开该文件夹;
(9) 按下 `Ctrl+Shift+R`,输入 `Python: Select Interpreter`,选择 logic 环境;
(10) 选择好环境后打开 logic.ipynb 脚本,VSCode 右下角应当显示:

```
3.9.21('logic':conda)
```

(11) 点击 Vscode 右上角处的三角形图标运行代码文件。

1.5.2　实验实现

本节按照实验流程逐步实现数据处理、数据集构建及逻辑回归模型的搭建,基于 Python 的 `pandas` 和 `PyTorch` 库完成。

数据处理

为了完成数据操作,我们需要存储和处理数据。主要任务包括:(1) 获取数据;(2) 将数据读入计算机后进行处理。若无存储数据的方法,获取数据将无意义。

预处理　在深度学习中,预处理原始数据是解决现实问题的第一步,而非直接使用准备好的张量格式数据。本实验使用 `pandas` 软件包进行数据分析,因其与张量兼容。以下为预处理步骤:

数据集读取　首先,使用 `pandas.read_csv` 读取训练集、验证集和测试集,并保留列名。同时,利用 `utils.py` 中定义的 `data_utils` 类对数据框进行预处理。代码如下:

```
import pandas as pd
# 数据集读取
# TODO
train =
val =
test =
```

```
7
8  # 数据集预处理
9  # TODO
10 train =
11 val =
12 test =
13
14 train.isnull().sum()
```

处理缺失值 通过 `train.isnull().sum()` 查看每列缺失值数量。为处理缺失值，常用方法包括插值法和删除法。本实验采用以下策略：

- 对于输出标签 `isDefault`，删除其值为 `NaN` 的行，因标签缺失无法用于训练。
- 对于其他输入特征的缺失值，使用该列均值填充。

实现代码如下：

```
1  # 过滤标签（`isDefault`列）为空的行
2  def filter_NA(data):
3      """
4      删除 `isDefault` 列为 NaN 的行
5      Args:
6          data (pd.DataFrame): 输入数据框
7      Returns:
8          pd.DataFrame: 过滤后的数据框
9      """
10     # TODO
11     pass
12 train, val, test = filter_NA(train), filter_NA(val), filter_NA(test)
13
14 # 填充对于其他input为空的行
15 def fill_mean(data):
16     """
17     用每列均值填充 NaN
18     Args:
19         data (pd.DataFrame): 输入数据框
20     Returns:
21         pd.DataFrame: 填充后的数据框
```

```
    """
    # TODO
    pass
train, val, test = fill_mean(train), fill_mean(val), fill_mean(test)
```

转换为张量格式　将数据划分为输入特征和标签,并转换为 PyTorch 张量格式:

```
train_input, train_label =
val_input, val_label =
test_input, test_label =
```

构建 PyTorch 数据集

PyTorch 提供了 torch.utils.data.Dataset 和 torch.utils.data.DataLoader 类,用于高效数据读取和管理。通过两者的组合,可生成数据迭代器,每次训练输出一个批次数据。

torch.utils.data.Dataset　自定义数据集类需继承 Dataset 并重载以下方法:

- __len__():返回数据集大小。
- __getitem__():实现按索引获取数据。

实现代码如下:

```
import torch
from torch.utils.data import Dataset
class MyDataset(Dataset):
    def __init__(self, *tensors):
        assert all(
            tensors[0].size(0) == tensor.size(0) for tensor in tensors
        ), "Size mismatch between tensors"
        self.tensor = tensors

    def __getitem__(self, index):
        # Return a tuple of tensors at the given index
        # Each tensor in self.tensor is sliced at the index position
        # For example, if we have input tensor and label tensor
        # This will return (input[index], label[index])
        # TODO
```

```
        pass

    def __len__(self):
        # Return the length of the dataset
        # TODO
        pass

train_dataset = MyDataset(train_input, train_label)
val_dataset = MyDataset(val_input, val_label)
test_dataset = MyDataset(test_input, test_label)
```

torch.utils.data.DataLoader DataLoader 将 Dataset 封装为迭代器，支持多进程、数据打乱等功能。主要参数包括：

- dataset：输入数据集对象。
- batch_size：每批次数据量。
- shuffle：是否打乱数据。

构建代码如下（批次大小设为 64，仅训练集打乱）：

```
from torch.utils.data import DataLoader
train_loader =
val_loader =
test_loader =
```

逻辑回归模型

逻辑回归模型基于 PyTorch 的 nn.Module 实现，包含权重正则化（L1 或 L2 范数）。代码如下：

```
import torch.nn as nn

class LogisticRegression(nn.Module):
    def __init__(self, input_dim, norm='l2'):
        super(LogisticRegression, self).__init__()
        assert norm in ['l1', 'l2']
        self.w = nn.Parameter(torch.randn(input_dim, 1))
        self.b = nn.Parameter(torch.randn(1))
```

```
9
10      def l1_norm(self):
11          # return l1 norm as a loss
12          # TODO
13          pass
14
15      def l2_norm(self):
16          # return l2 norm as a loss
17          # TODO
18          pass
19
20      def normnize(self):
21          # return l1 or l2 norm as a loss
22          if self.norm == 'l1':
23              return self.l1_norm()
24          else:
25              return self.l2_norm()
26
27      def sigmoid(self, x):
28          # return sigmoid of x
29          # TODO
30          pass
31
32      def forward(self, x):
33          # return logits
34          # TODO
35          pass
36
37  def criterion(y_pred, y_true):
38      # compute loss
39      # TODO
40      loss_fn = nn.BCELoss()
41      pass
```

表 1.1 超参数调整说明

超参数	典型调整范围	调整影响
学习率 (lr)	0.1 ∼ 1e-5	过大导致优化震荡甚至发散,过小收敛缓慢;
epoch	10 ∼ 100+	过少导致欠拟合,过多引发过拟合。
正则化权重 ($norm\_weight$)	0.001 ∼ 0.2	过大:限制模型学习能力,过小:正则化效果不明显。

1.5.3 测试验证

模型搭建后,参考超参数调整说明,尝试调整不同的超参数,训练逻辑回归模型。

1.5.4 代码和结果提交

请将 `logic.ipynb` 文件按照以下命名格式重命名:`学号 _ 姓名 _logic.zip`。提交至 BB 系统.

1.6 实验要求与评分细则

1. **环境搭建和数据集加载(3 分)**
   - 正确成功搭建实验环境。(0.5 分)
   - 加载并处理数据集,正确读取训练集(train)、验证集(val)和测试集(test)。(0.5 分)
   - 完成数据清洗,处理缺失值。(1 分)
   - 构建 `Dataset` 和 `DataLoader`。(1 分)
2. **模型实现(4 分)**
   - 正确实现模型类的初始化。(1 分)
   - 实现 `sigmoid` 激活函数。(1 分)
   - 实现前向传播 `forward` 函数。(1 分)
   - 正确实现 L1/L2 正则化。(1 分)
3. **训练与评估(3 分)**
   - 正确设置损失函数。(1 分)
   - 在测试集上进行最终评估。(1 分)
   - 在 logic.ipynb 最后新建单元格,选择 markdown 格式,反馈实验收获、难度,以及对逻辑回归实验的建议 (1 分)

完成所有的 TODO,成功运行 logic.ipynb 的结果示例:

```
Val Accuracy = 0.655
```