基于鲸鱼优化 LSSVM 和残差网络的信用风险评估模型

摘要

本文是对信用风险识别问题的研究，在德国和澳大利亚信用数据集基础上，先进行了**相关性分析**、**PCA** 降维与 **CDA** 增补得到增强数据集，再使用**鲸鱼优化算法**改进的 **LSSVM 算法**得到个人信用评分模型，然后搭建了**残差网络** SimpleNet 进行预测并同传统机器学习方法进行了对比，最后使用基于**模拟退火**的**非线性规划**得到信用等级划分。

对于问题一, 我们首先对数据进行了探索性分析，意识到其中存在离群点与相关性，并计算了变量之间的皮尔逊与斯皮尔曼**相关系数**, 找到了 **14 个**相互之间相关性较小的特征。然后使用球状检验与 KMO 检验的方式判断各个变量是否各自独立。最后，用**主成分分析**的方式挑选出三个可以代表总体的特征值，实现了数据降维，最终分类结果见**图 1**。

对于问题二, 我们首先引入**反事实数据增强（CDA）**, 解决了问题一中数据集标签比例不平衡的问题；并使用**最小二乘支持向量机（LSSVM）**，以此挖掘评价指标与违约风险之间的非线性关系，经过实验，简单使用 LSSVM，可以达到 **77.41%** 的准确率；再此基础上，我们将启发式算法——**鲸鱼优化算法（WOA）**与 LSSVM 相组合，进行超参数调优，其准确率变化见**图 8(b)**，最终到达 **84.35%** 的准确率，并将最优超参数的 LSSVM 最后的输出值作为我们的信用评分模型。

对于问题三, 我们为了解决传统算法面对不良数据集表现下滑严重的问题，提出了**残差神经网络** SimpleNet，其架构图见**图 11**，具体由**自编码器（AE）**，**残差网络模块（ResNet）**和**多层感知机（MLP）**组成，经过多次消融实验，与题目给的出的所有模型加 Xgboost 在四个评价准则上进行了对比，在德国信用数据集上的表现见**表 4**，澳大利亚信用数据集上的表现见**表 5**。

对于问题四, 我们以信用等级之间失信率差异应该较大为抓手，构造出**目标函数**。以同一信用等级之间属性相似作为构造**约束条件**的关键，并使用 **minimize 函数**求解非线性规划的解。在此基础上，我们提出了使用**模拟退火**并根据目标函数来对 **k-means 聚类**进行优化的方式来求解最优划分，最终将信用水平划分成了 5 个信用等级, 划分结果见图 15

最后，我们在公开数据集 **kaggle-home-credit-credit-risk-model-stability** 上测试了我们的模型，并取得了 **82.3%** 的准确率，证明我们的模型具有较好的泛用性与鲁棒性。

关键字：信用风险识别　主成分分析　启发式算法　LSSVM　非线性规划

一、 问题重述

1.1 问题背景

在经济交融发展中，金融行业对个体借贷的信用风险识别准确与否，直接关系到贷款机构的资产健康与国际市场的经济质量。通过分析发现，信用风险评价研究大致可以划分为“信用风险评价指标筛选 → 信用风险得分测算 → 信用风险等级划分”三个步骤，需要克服多重共线性、可解释性不足等问题。

1.2 问题要求

问题 1 个体的信用风险往往与多种指标相关，但是高维数据会带来评价指标信息冗余等问题，如何对德国信用数据集进行筛选, 来提升信用风险评价准确性及可解释性?

问题 2 现今主流的信用风险评估方法存在线性关系不强、对违约样本识别不足等问题. 如何建立一个和是的信用评分模型，揭示评价指标与个体违约风险之间的联系，求解德国信用数据集中的得分并给出理由。

问题 3 智能 AI 算法的出现为信用风险准确识别带来了新的机遇。请自建信用评分模型，与决策树、K 最近邻、随机森林、支持向量机等多种现有分类模型进行对比分析效果。

问题 4 按照信用等级越高、信用风险越低的等级划分标准，构建非线性规划模型在德国信用数据集上划分个体信用等级并解释理由。

二、 问题分析

2.1 问题一分析

对于问题一，题目要求我们解决由于评价指标过多带来的信息冗余问题，选用合适的筛选模型，以期达到提升后续评价准确性和可解释性的目标。对此，我们采取以下措施：

- 对已有数据进行了**探索性分析**；
- 使用**皮尔逊与斯皮尔曼相关系数**分析了指标间的相关性，删去部分重复指标进行解耦；
- 运用**主成分分析法（PCA）**对解耦后的指标进行降维，使用 **Johnsen-lindenstrauss 定理**将各个指标汇总成 3 个大类别指标

2.2 问题二分析

对于问题二，题目要求我们建立一个合适的信用评分模型，克服数据不均衡和传统线性加权法的问题。对此，我们采取以下措施：

- **数据不均衡问题**
  Dheeraj Rajagopal 等人为了减少不均衡数据对模型的影响，提出了 **CDA**（Counterfactual Data Augmentation）的数据增补方法，本文采取了相似的做法进行补全，最后将德国数据集中 1000 个数据，0.3 的违约比，增补至 1400 个数据，0.5 的违约比，构建我们自己的数据集；
- **关联非线性问题**
  我们选用 **LSSVM 算法**来寻找评价指标与个体违约风险之间的联系，**LSSVM 算法**建立在统计学中学习理论、VC 维理论和最小结构风险理论之上，可以有效的发掘指标与评估目标之间的非线性关系；

为了更进一步提升模型的鲁棒性，本文使用**鲸鱼优化算法（WOA）**对 LSSVM 进行超参数调优，最后的模型准确率分析见**图 10(b)**;

2.3 问题三分析

对于问题三，题目要求我们分别在德国与澳大利亚数据集上实现 KNN、RF 等传统机器学习方法，具体结果见**表 4**、**表 5**。但是，传统方法可能不完全适用于信用风险评估任务，所以我们自建了网络模型并取得了更好的效果，其中具有以下模块：

- **自编码器**
  自编码器具有降维压缩功能，广泛的应用于生成式模型中，比 PCA 有更好的对非线性数据的适应能力，本模型将该模块作为第一层，在保留更多用户信息的情况下解决高维冗余问题；
- **ResNet**
  何凯明等人提出的残差神经网络是深度学习的经典网络模型，解决了模型随着深度加深而造成的梯度消失问题，能更好的使用有限的数据训练出更优；
- **多层感知机**
  该模块构成了我们自建网络的主体，主要作用是提取信息并输出结果；

2.4 问题四分析

对于问题四，题目要求我们使用非线性规划的方式对信用等级进行分类，信用等级越高，信用风险越低。对于不同的信用等级，我们可以用这个信用等级内失信人员的占比来刻画信用风险。使用非线性规划解决问题，其关键在于确定目标函数以及约束条件。由于要进行分类，那么样本中每个元素所在的类别应该作为目标函数的变量进行迭代。根据这些变量构造目标函数来评价分类结果的好坏。而且，在同一个类别中的元素，其属性的欧几里得距离不能太大，我们将这一性质非线性规划约束条件。构造出一个非线性规划问题并求解。

由于这种方法的目标函数的变量数量过于庞大，我们可以尝试一种非传统的非线性规划的方式优化算法。可以先使用 **k-means** 算法先将元素分成若干类，再计算目标函数并根据结果进行退火并随机化 k-means 的初始 k 个聚类中心。这种方式虽然没有了传统非线性规划的约束条件，但是生成出来的 k 个聚类相较于第一个方法更加合理也更加方便，可以节省时间。

三、 模型假设

为简化问题，本文做出以下假设：

- **完全随机抽样假设:**
  随机抽样的样本具有更好的代表性，能提升模型的泛化能力与推广能力；
- **同质性假设:**
  在分类问题种，确保同一类别内的样本在特征上足够相似，有利于建立准确的分类模型。
- **数据独立性假设:**
  对于已知的德国信用数据集与澳大利亚信用数据集，我们无法知道抽样个体先前的信用记录，同样，我们也无法知道不同样本之间的人际关系，所有我们做出数据独立性检验以更好的使用已有数据。

四、 符号说明

- **此处给出了部分的符号声明，其余符号在使用前均会定义。**

Symbol	Definition
D_k	WOA 算法中不同个体的欧氏距离
$X^*(t)$	t 时刻的最优个体
ω	LSSVM 中的决策函数的权重向量
$\varphi(x)$	特征映射的应用结果
$LABEL_j$	第 j 个样本的标签属性
r	集合中守信人数与总人数的比值

五、 问题一的模型的建立和求解

对于问题一，我们先使用对德国信用数据集进行了探索性分析，并依据其相关系数删去了部分强相关指标 **(X2，X4，X8，X10，X13，X16，X18)**，如此就能假定剩下的评价指标在高维也是近似正交的；对于选择出来的 14 个指标，我们利用 **PCA** 进行降维，最终取前三个主成分作为最终的降维结果，根据原数据集对其的贡献度，分为 **“经济状况类”**，**“个人信息类”**，**“社会关系类”**，最终结果如图 1 所示：

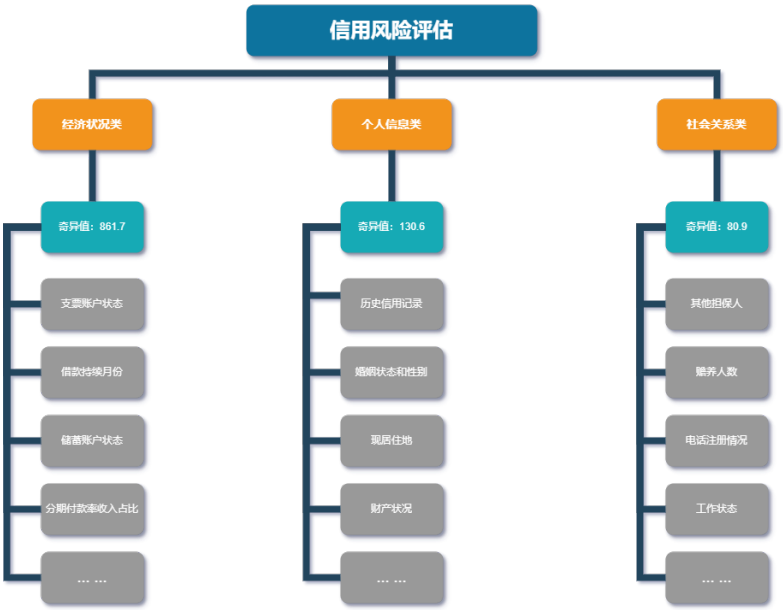


图 1　指标归类

5.1 理论基础

- **皮尔逊与斯皮尔曼相关系数**
  皮尔逊相关系数是用于衡量两个连续性随机变量间的相关系数，当两个变量 X、Y 都是正态连续变量，且两者之间呈线性关系时，则可以用 Pearson 来计算相关系数。取值范围 [-1,1]。计算公式如下：

$$\rho(x,y)=\frac{\operatorname{cov}(x,y)}{\sigma(x)\cdot\sigma(y)}=\frac{E[(x-\mu_x)(y-\mu_y)]}{\sigma(x)\cdot\sigma(y)} \tag{1}$$

  斯皮尔曼相关系数是秩相关系数，根据原始数据的等级排序进行求解，也称为等级变量之间的皮尔逊相关系数，它利用单调方程评价两个统计变量的相关性，通过对两个变量 X、Y 做等级变换，然后按 Pearson 相关性分析的方法计算的相关性。计算

公式如下：

$$\rho(x,y)=\frac{\sum_i(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum_i(x_i-\bar{x})^2\sum_i(y_i-\bar{y})^2}} \tag{2}$$

在实际应用中，变量间的连结是无关紧要的，于是可以通过简单的步骤计算 ρ 被观测的两个变量的等级的差值。计算公式如下：

$$\rho(x,y)=1-\frac{6\sum d_i^2}{n(n^2-1)} \tag{3}$$

- **主成分分析法（PCA)**
主成分分析法通过线性变换将原始数据投影到一个由各主成分组成的新坐标系中，这些主成分是原始数据中方差最大的方向。目标是找到一组正交的新变量（主成分），使得数据在这些新变量上的方差最大。原始指标数据的标准化采集 p 维随机向量 $x=(x_1,x_2,...,x_p)^T$ 和 n 个样品 $x_i=(x_{i1},x_{i2},...,x_{ip})^T$，n > p，构造样本阵，对样本阵元进行如下标准化变换，从而得到标准阵 Z：

$$Z_{ij}=\frac{x_{ij}-\bar{x}_j}{s_j}, i=1,2,...,n; j=1,2,...,p \tag{4}$$

$$\bar{x}_j=\frac{\sum_{i=1}^n x_{ij}}{n}, s_j^2=\frac{\sum_{i=1}^n(x_{ij}-\bar{x}_j)^2}{n-1} \tag{5}$$

对标准化阵 Z，求相关系数矩阵：

$$R=[r_{ij}]_p\, xp=\frac{Z^TZ}{n-1} \tag{6}$$

$$r_{ij}=\frac{\sum z_{kj}\cdot z_{kj}}{n-1}, i,j=1,2,...,p \tag{7}$$

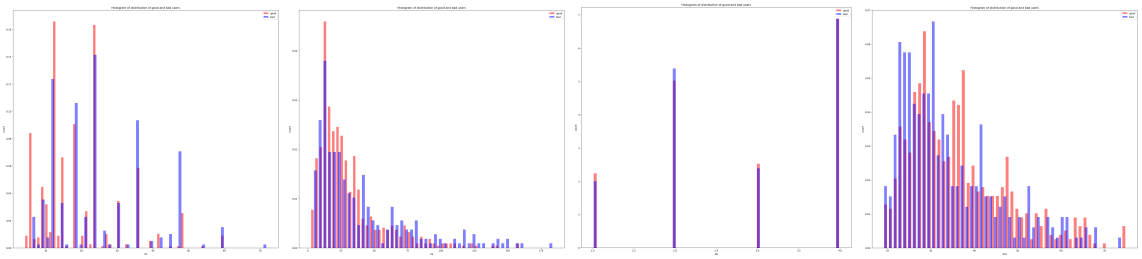
解样本相关矩阵 R 的特征方程 $|R-\lambda I_p|=0$ 得 p 个特征根，确定主成分，按 $\frac{\sum_{j=1}^m\lambda_j}{\sum_{j=1}^p\lambda_j}\geq 0.85$ 确定 m 值，使信息的利用率达 85% 以上，对每个 λ_j, j=1,2,...,m, 解方程组 Rb = λ_jb 得单位特征向量 b_j^o
将标准化后的指标变量转换为主成分

$$U_{ij}=z_i^Tb_j^o, j=1,2,...,m \tag{8}$$

U_1 称为第一主成分,U_2 称为第二主成分,⋯,U_p 称为第 p 主成分。
最后对 m 个主成分进行加权求和，即得最终评价值，权数为每个主成分的方差贡献率。

表 1　德国信用数据集组成

X1	X2	X3	X4
(定性) 现有支票账户的状态	(数字) 借款持续月份	(定性) 历史信用记录	(定性) 借款目的
X5	**X6**	**X7**	**X8**
(数字) 额度	(定性) 储蓄账户状态	(定性) 当前就业状态	(数字) 分期付款率收入占比
X9	**X10**	**X11**	**X12**
(定性) 婚姻状态和性别	(定性) 其他担保人	(数字) 现居住地	(定性) 财产状况
X13	**X14**	**X15**	**X16**
(数字) 年龄	(定性) 其他分期情况	(定性) 房产状态	(数字) 现有信用卡数量
X17	**X18**	**X19**	**X20**
(定性) 工作状态	(数字) 赡养人数	(定性) 电话注册情况	(定性) 是否有国外经历



(a) X1　**(b) X3**　**(c) X5**　**(d) X6**

图 2　连续变量

5.2 模型求解

5.2.1 信用数据集组成

通过查阅相关资料，得到德国信用数据集的属性名称及其对应的含义如表 1 所示。

5.2.2 数据预处理与探索性分析

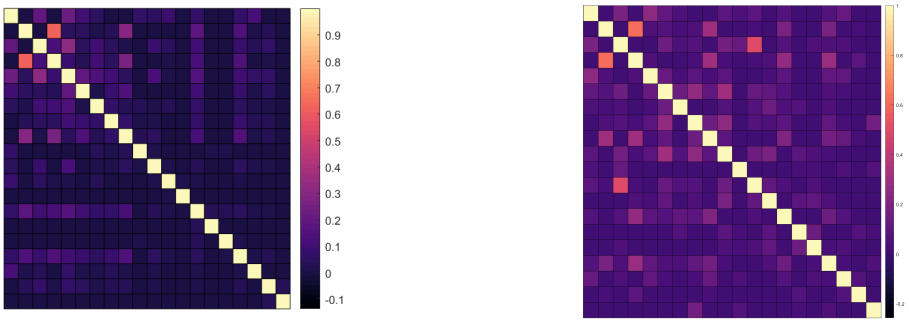
在数据集成和数据清洗的过程中，我们进行探索性数据分析，通过计算统计量、数据可视化等方法，对 24 个特征进行了可视化处理，处理部分结果如下图所示（全部结果详见附件）:

在对数据集的处理中，红蓝柱形表示的是各个值对应的好坏样本的数量，折线表示坏用户的占比。

通过对数据集的分析，我们可以得出如下结论：在以 X1（现有支票账户的状态）、X2（借款持续月份）等为代表的特征中，好坏用户占比通过变量来看是有一定的分辨性，而以 X21、X24 为代表的特征中，好坏用户占比通过变量来看分辨性较小，不具有较好的预测效果，故将该类特征删去。

5.2.3 评价指标相关性分析

高维数据容易出现冗余的一大原因，是其很容易造成多个指标相关性过大导致评价重复，所以，我们利用 SPSS 软件对已有数据集的 20 个评价指标进行了相关性分析，分析结果如图 4 所示：对于部分数据（如 X2——借款持续月份、X4——借款目的）无论是**皮尔逊**相关系数分析还是**斯皮尔曼**相关系数分析都体现出了较大的相关性，



(a) 皮尔逊相关系数　　(b) 斯皮尔曼相关系数

图 3　相关系数热力图

故考虑仅保留其中一个指标，这样可以造保证信息损失较少的情况下，做到对已有数据集的去耦与减少冗余，经过分析，我们将 **(X2，X4，X8，X10，X13，X16，X18)** 去除，留下 14 个剩余指标进行进一步分析。

5.2.4 主成分分析

球状检验，即检验总体变量的相关矩阵是否是单位阵（相关系数矩阵对角线的所有元素均为 1, 所有非对角线上的元素均为零），从而得出各个变量是否各自独立。我们的检验假设是：H0：相关系数矩阵为单位阵（即变量不相关）,H1：相关系数矩阵不是单位阵（即变量间有相关关系）。经计算，得出的 P-value 值约为为 2.132e-204，小于我们预先设定的的显著性水平 (0.05), 故我们拒绝原假设 H0，H1 成立，即变量之间有相关关系，说明我们可使用因子分析法做下一步分析。
KMO 检验, 通过比较样本相关系数与样本偏相关系数，它用于检验样本是否适于作主成分分析。通过对我们的数据集进行 KMO 检验，得出结果如下：

$$\begin{bmatrix} 0.68420557 & 0.686915 & 0.69178035 & \dots & 0.63505767 & 0.50356082 & 0.50307623 \end{bmatrix}$$

由输出结果可以看出，通过涮选之后的数据集各特征的 KOM 统计量均大于 0.5，故变量间的相关性较强，偏相关性越弱，因子分析的效果较好。
该数据集通过了球状检验和 KMO 检验，故可以通过主成分分析法进行进一步探究。
绘制散点图和折线图如下：
绘制热力图如下:

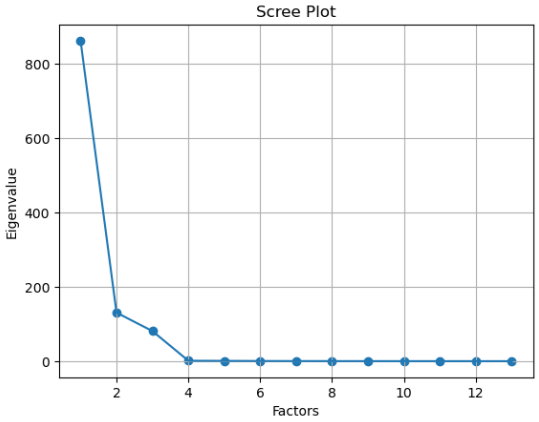


图 4　主成分分析散点图和折线图

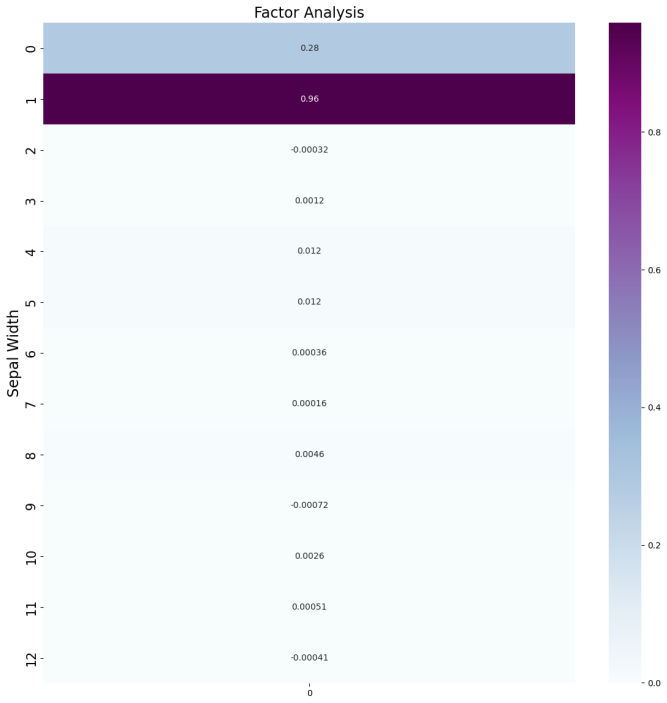


图 5　主成分分析热力图

5.3 结果分析

通过对数据探索性分析，我们从原先 25 个特征中涮选到了 14 个相互之间相关性较小的特征，再通过主成分分析法，我们在最大化保留样本间方差的情况下实现了降维。通过对结果分析，我们可以用 3 个特征值取代样本原来的 14 维特征。最终得到的结果如图 1 所示。

六、 问题二的模型的建立和求解

对于问题二，我们采用了 **LSSVM** 来寻找评价指标与个体违约风险之间的联系，并使用鲸鱼优化算法对其超参数进行优化，最终达到了 **84.35%** 的准确率，并有效的降低了两类不同错误分布不平衡的问题，其流程图如下：

6.1 理论基础

6.1.1 鲸鱼优化算法（WOA）

鲸鱼优化算法是模仿鲸鱼捕食行为的群体智能优化算法，主要分为包围、发泡网、捕食三个步骤。对该方法进行数学模拟，可以得到下列公式：

- **包围猎物**
  在搜索阶段，由于最优位置是未知的，所以将当前之最佳候选解的位置当作目标，其余点则在迭代中向此处靠近，该过程的计算公式如下：

$$D = |CX^*(t) - X(t)| \tag{9}$$

$$X(t+1) = X^*(t) - AD \tag{10}$$

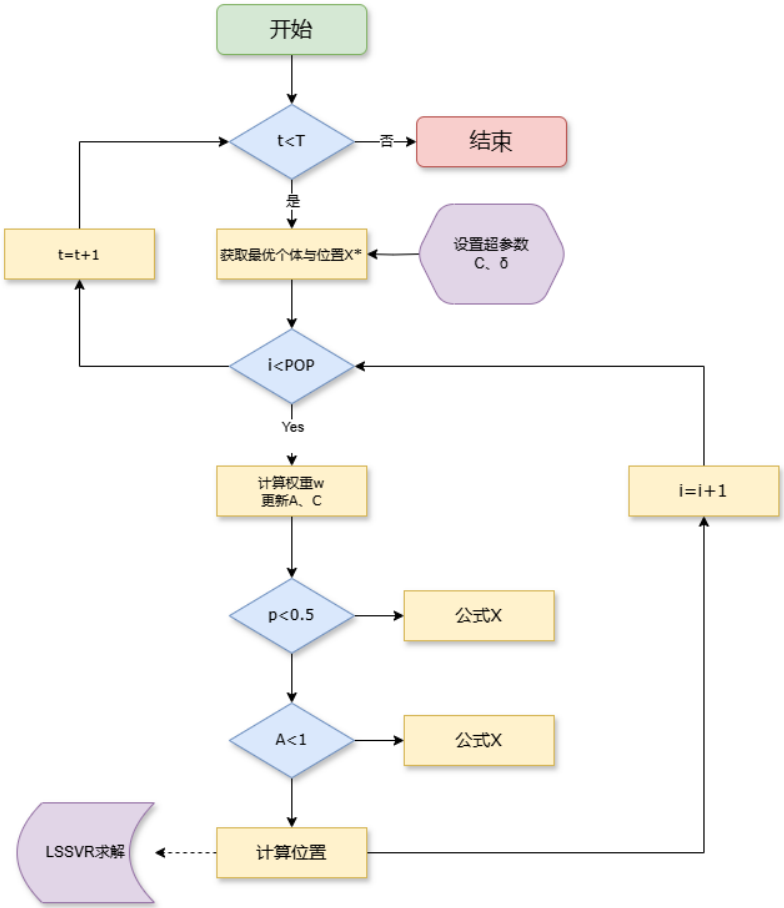


图 6　采用 WOA 优化的 LSSVM 流程图

其中：t 为当前迭代次数；$X^*(t)$ 为最好位置；$X(t)$ 为鲸鱼位置；A，C 为系数，其计算方式如下：

$$\begin{aligned} A &= 2ar_1 - a \\ C &= 2r_2 \\ a &= 2 - \frac{2t}{T} \end{aligned} \tag{11}$$

- **发泡网**

这个部分模拟了鲸鱼利用发泡网螺旋游向猎物的过程，具体表现为个体 $X(t)$ 有 P 的概率收缩包围圈，或者螺旋游向猎物：

$$D_k = |X^*(t) - X(t)| \tag{12}$$

$$X(t+1) = X^*(t) + D_k e^{bl} \cos 2\pi l \tag{13}$$

$$X(t+1) = \begin{cases} X^*(t) - A \cdot D & (p < 0.5) \\ X^*(t) + D_k \cdot e^{bl} \cdot \cos 2\pi l & (p \geq 0.5) \end{cases} \tag{14}$$

- **搜索捕食**

为了避免最优个体不更新，每个个体还有可能随机选择一个其他个体而不是最优个体来更新下个位置，具体公式为：

$$D_r = |C \cdot X_{rand} - X(t)| \tag{15}$$

$$X(t+1) = X_{rand} - A \cdot D_r \tag{16}$$

其中，A 是在随着时间逐步缩小范围中游走的随机变量，这符合当“捕猎”到达后期时，每个个体会更加专注于正确的“猎物”。

6.1.2 LSSVM

最小二乘支持向量机（LSSVM）是一种基于统计理论的改进的支持向量机，它能简单的将二次优化问题转化为线性方程组求解，其准确程度依赖于其超参数的选择，故在本模型中将其与 WOA 相融合。

基于结构化风险最小化原则，评估问题可以被描述为优化问题：

$$\begin{aligned} &\min J(\boldsymbol{w}, e)=\min \left(\frac{1}{2}\|\boldsymbol{w}\|^{2}+\frac{1}{2} \gamma \sum_{i=1}^{N} e_{i}^{2}\right) \\ &\text { s.t. } \quad y_{i}=\langle \boldsymbol{w}, \phi\left(\boldsymbol{x}_{i}\right)\rangle+b+e_{i} \quad i=1,2, \ldots, N \\ &\quad \gamma>0 \end{aligned} \tag{17}$$

构造相应的拉格朗日函数：

$$L_{\mathrm{LSSVM}}=\frac{1}{2}\|\boldsymbol{w}\|^{2}+\frac{1}{2} \gamma \sum_{i=1}^{N} e_{i}^{2}-\sum_{i=1}^{N} \alpha_{i}\left\{\langle \boldsymbol{w}, \phi\left(\boldsymbol{x}_{i}\right)\rangle+b+e_{i}-y_{i}\right\} \tag{18}$$

求解后可得化简的 LSSVM 模型：

$$\begin{aligned} y(\boldsymbol{x}) &=\langle \boldsymbol{w}, \phi(\boldsymbol{x})\rangle+b \\ &=\sum_{i=1}^{n} \alpha_{i} \phi\left(\boldsymbol{x}_{i}\right) \cdot \phi(\boldsymbol{x})+b \\ &=\sum_{i=1}^{n} \alpha_{i} K\left(\boldsymbol{x}_{i}, \boldsymbol{x}\right)+b \end{aligned} \tag{19}$$

本题中核函数选择径向基函数（RBF）。

6.2 模型建立与求解

6.2.1 数据不均衡

由前文对德国信用数据集的探索性分析可得，其失信用户与正常用户之比见下表：

表 2　德国信用数据集组成

失信用户（人）	正常用户（人）
300	700

这是符合实际的，但是却会为建立的模型带来信息不足而产生的偏见，即更容易把失信人员判断为正常人员。

所以，我们选择采用 **CDA（反事实数据增强）**的方法，对数据集进行补全，即通过计算出各个属性的均值，并创建反事实的新数据以平衡类别分布，公式与补全后的数据集如下：

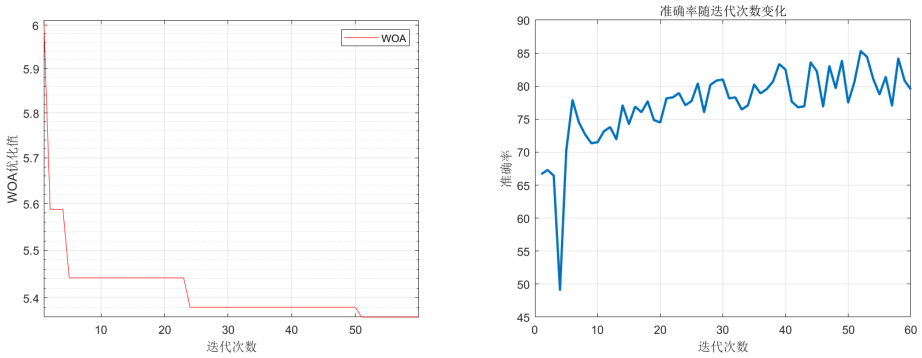
$$\begin{cases} X_{i}^{new}=2\overline{X_{i}}-X_{i}^{ori} \\ Label_{new}=\overline{Label_{ori}} \end{cases} \tag{20}$$

表 3　补全信用数据集组成

失信用户（人）	正常用户（人）
700	700

6.2.2 基于鲸鱼优化的 LSSVM 模型建立

LSSVM 由于能简单的将二次优化问题转化为线性方程组求解，很适合探究个人信用与各项指标之间的非线性关系。同时，常规 LSSVM 算法的求解依赖于超参数 A 和 θ 的选择，所以在这里引入启发式算法——**鲸鱼优化算法（WOA）**，将 LSSVM 算法的最终结果与实际标签之间的差别作为优化目标，以超参数 A 和 θ 为因变量，构建了基于鲸鱼优化的 LSSVM 模型，图 9 是算法的训练过程与结果：



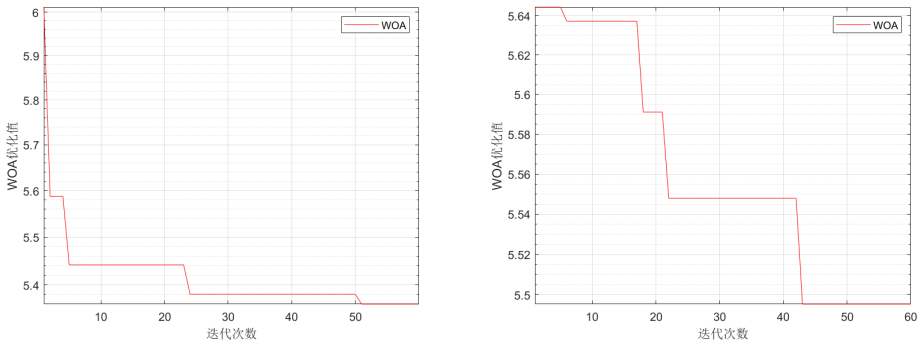
(a) WOA 结果图　　**(b) 准确率变化**

图 7　基于鲸鱼优化的 LSSVM 模型

最终，在德国信用数据集上，本模型达到了 84.35% 的准确率，比单纯使用 LSSVM 模型的 77.41% 高出了近 20% 的准确率，模型有显著优势。

6.2.3 模型在不同数据集上的表现

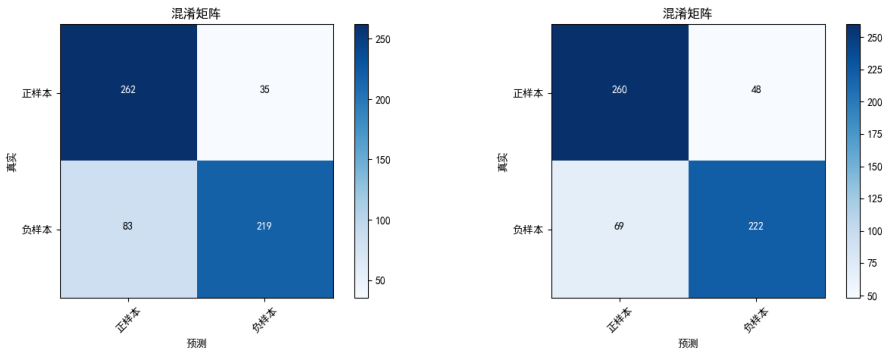
- **降维与未降维**



(a) 降维结果　　**(b) 未降维结果**

图 8　数据集降维对比试验图

如图 10，可以看到模型在降维数据上的训练效果，在不改变最终优化值的情况下，比未降维数据更快的收敛，这表明，我们的降维方法在不损失已有信息的同时，有效的提取克服了指标间相关性带来的数据冗余问题。



(a) 原始数据混淆矩阵 **(b) 补全数据混淆矩阵**

图 9　数据集补全对比试验图

- **补全与未补全**
  如图 11，我们基于 CDA 的数据补全法在相同测试集上的测试结果略微优于原始数据，并且在第二类错误数量上显著的少于原始数据集，可以认为此种数据处理方法可以显著的减少由于数据不平衡带来的不均衡评估。

6.3 求解结果

综上，对于问题 3：

- **我们对使用 CDA 法对数据集进行了进一步处理，克服了信用评估时存在的偏见；**
- **我们建立了基于鲸鱼优化算法的 LSSVM 算法，并最终在原始德国信用数据集上达到了 84.35% 的准确率，显著优于传统算法与简单 LSSVM 算法；**

七、 问题三的模型的建立和求解

对于问题三，我们使用**自编码器（autoencoder）**、**残差网络（resnet）**搭建了如图 13 所示的深度神经网络 SimpleNet，在德国和澳大利亚信用数据集上，分别达到 **76.31%** 和 **73.14%** 的准确率，并在不平衡数据集上展现了很好的鲁棒性，优于 DT、SVM 等传统算法。

7.1 理论基础

- **决策树（DT）算法**
  决策树是一种基于树状结构的机器学习算法，用于分类和回归任务。它通过一系列简单的问题或条件，逐步将数据集划分到不同的类别或值。每个内部节点表示一个特征/属性，每个分支代表一个可能的特征值，而每个叶节点表示一个类别或值。
- **K 最近邻（KNN）分类算法**
  K 近邻算法，即是给定一个训练数据集，对新的输入实例，在训练数据集中找到与该实例最邻近的 K 个实例（也就是上面所说的 K 个邻居），这 K 个实例的多数属于某个类，就把该输入实例分类到这个类中；
- **随机森林（RF）算法**
  随机森林算法基于决策树的集成思想，其中每个决策树由随机抽样的训练样本构建而成。在构建每个决策树时，随机森林会随机选择特征子集进行训练，以增加模型的多样性。随机森林在构建决策树时还会引入随机性，通过限制每个决策树的最大深度或叶子节点的最小样本数，防止模型过拟合；
- **支持向量机（SVM）算法**

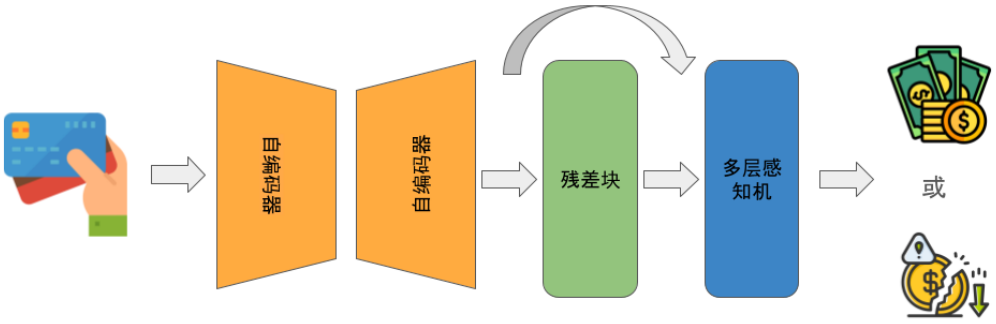


图 10　SimpleNet 结构图

支持向量机（SVM）是一类按监督学习方式对数据进行二元分类的广义线性分类器，其决策边界是对学习样本求解的最大边距超平面。SVM 使用铰链损失函数计算经验风险并在求解系统中加入了正则化项以优化结构风险，是一个具有稀疏性和稳健性的分类器；

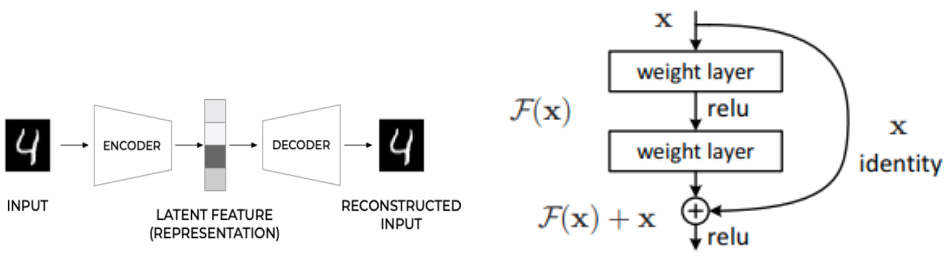
- **XGBoost 算法**
  XGBoost 是一种基于梯度提升的决策树算法。梯度提升是一种集成学习方法，它通过迭代地添加新的弱学习器（通常是决策树）来优化目标函数。XGBoost 在每次迭代中，都会尝试找到一个能够最小化损失函数的决策树，从而不断提升模型的性能；

7.2 模型建立与求解

7.2.1 我们的模型：SimpleNet

为了适应原始数据集指标冗余、分布不均的特点，我们选择深度学习的方法，来克服传统机器学习在此方面的弱点，结果表明，我们的模型 **SimplNet** 可以在小样本训练的基础上达到，并在一些情况下略优于传统算法的准确率，保持在 **75%** 左右，并且对非平衡数据具有很好的适应性与鲁棒性，两类错误几乎保持均等。

下面是我们的模型：SimpleNet 的组成模块：



(a) 自编码器　　**(b) 残差网络**

图 11　SimpleNet 模块原理图

- **自编码器（AE）**

自编码器本质上是一种数据压缩算法，先将输入压缩成低维的空间表征，再将压缩后的空间重构为输出，所以，它的前一部分具有降维的作用；
另一方面，自编码器既能做线性变换，由于激活函数的引入，也能表征非线性变换，所以，在有可学习数据的情况下，自编码器是一种比 PCA 更灵活的数据压缩与降维方法；

- **残差网络模块（ResNet）**
  残差神经网络克服了传统神经网络在堆叠到一定程度时，出现梯度爆炸（消失）、退化的问题，如图 14(b)，通过“短路”，来支持模型的深度发展，从而用增加的参数量学到更多的信息。

7.2.2 德国信用数据表

基于题目中给出的表格，同时考虑到 Xgboost 在机器学习任务中的优异表现，经过多轮试验，我们得到结果如下：

表 4　德国信用数据集分类方法对比结果

模型	评价准则			
	Accuracy	AUC	Type1-error	Type2-error
SimpleNet	0.7631 ± 0.0412	0.8352 ± 0.0238	0.1412 ± 0.0317	0.2341 ± 0.0329
DT	0.6941 ± 0.0314	0.6546 ± 0.0421	0.1452 ± 0.0358	0.6824 ± 0.0468
KNN	0.6888 ± 0.0276	0.6539 ± 0.0373	0.1483 ± 0.0280	0.6889 ± 0.0569
RF	0.7630 ± 0.0276	0.8072 ± 0.0253	0.0841 ± 0.0270	0.5556 ± 0.0609
SVM	0.7640 ± 0.0285	0.7864 ± 0.0234	0.1139 ± 0.0275	0.5186 ± 0.0500
Xgboost	0.7572 ± 0.0249	0.7741 ± 0.0336	0.1398 ± 0.0346	0.4852 ± 0.0558

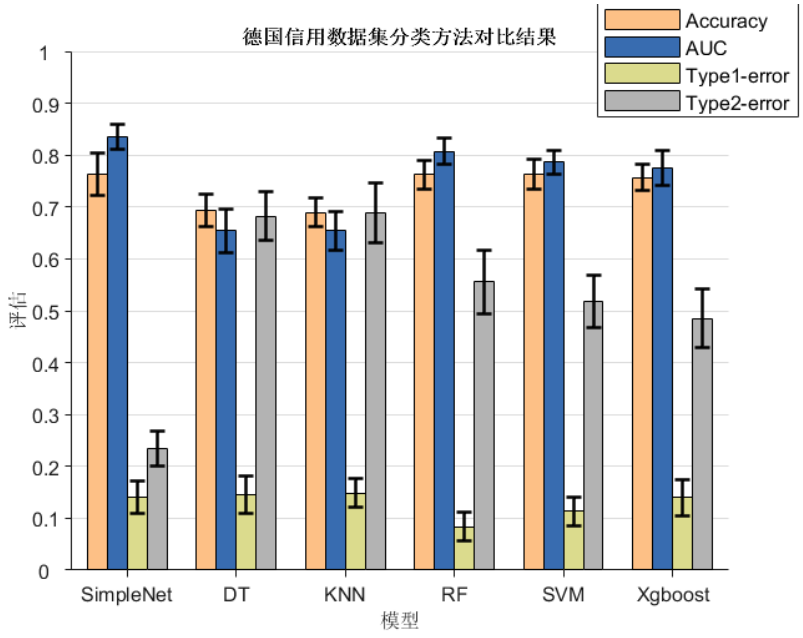
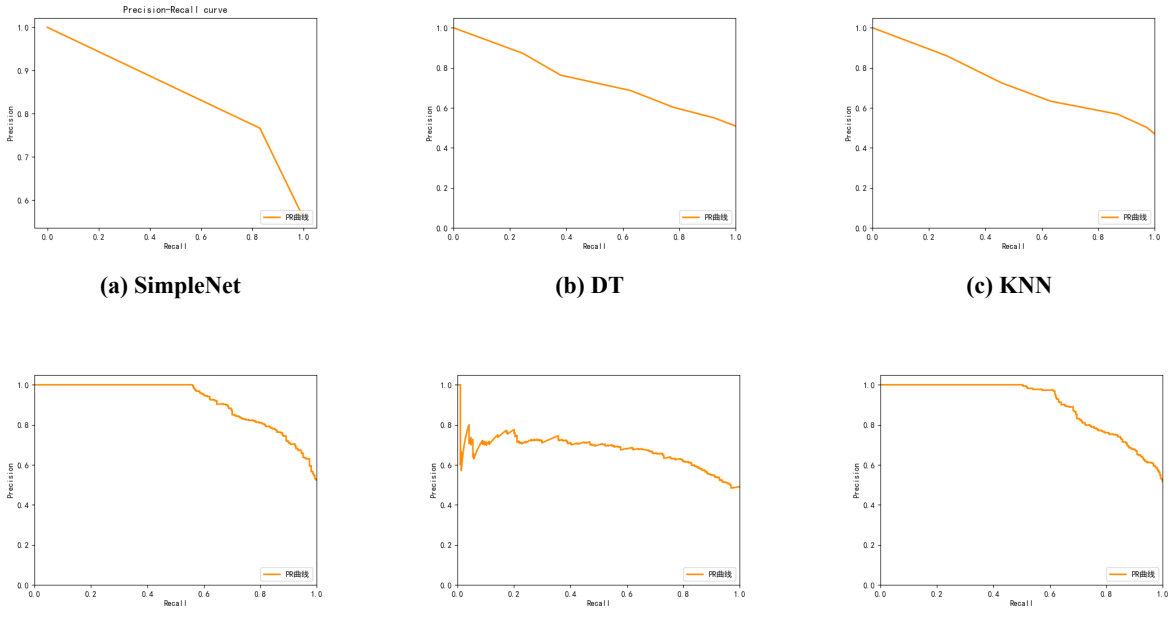


图 12　德国信用数据集分类方法对比直方图

在德国信用数据集中，SimpleNet 将其划分为 8：2 的训练集与评估集，总共训练了 100 个 epoch 得到了表格中评估量。

由表 6 可知，SimpleNet 在德国原数据集中，达到 SOTA 水平的同时，较其他模型显著减少了 Type2-error，证明其对于不平衡数据与噪声数据有较好的鲁棒性。



(a) SimpleNet　(b) DT　(c) KNN

(d) RF　(e) SVM　(f) Xgboost

图 13　德国信用数据集各模型 PR 曲线

7.2.3 澳大利亚信用数据表

表 5　澳大利亚信用数据集分类方法对比结果

模型	评价准则			
	Accuracy	AUC	Type1-error	Type2-error
你们的模型	0.7314 ± 0.0231	0.7421 ± 0.0231	0.2519 ± 0.0521	0.2234 ± 0.0487
DT	0.6939 ± 0.0486	0.7384 ± 0.0539	0.4480 ± 0.0844	0.1943 ± 0.0514
KNN	0.6880 ± 0.0447	0.7351 ± 0.0517	0.4483 ± 0.0835	0.2057 ± 0.0566
RF	0.8809 ± 0.0360	0.9373 ± 0.0236	0.1258 ± 0.0377	0.1164 ± 0.0544
SVM	0.6890 ± 0.0428	0.7334 ± 0.0487	0.4548 ± 0.0793	0.1929 ± 0.0553
Xgboost	0.8690 ± 0.0343	0.9323 ± 0.0250	0.1460 ± 0.0527	0.1183 ± 0.0485

表 6　模型评估结果对比

模型	模型准确率		两类错误差值	
	增补前	增补后	增补前	增补后
SimpleNet	0.763	0.801	0.090	0.090
Xgboost	0.752	0.796	0.345	0.070
DT	0.694	0.680	0.537	0.080
KNN	0.688	0.680	0.540	0.080
RF	0.764	0.810	0.471	0.180
SVM	0.757	0.743	0.404	0.080

在澳大利亚信用数据集中，SimpleNet 将其划分为 8.5：1.5 的训练集与评估集，总共训练了 100 个 epoch 得到了表格中评估量。

由于澳大利亚样本只有 690 个，相关指标只有 14 个，所以 SimpleNet 难以获得足够的数据进行学习，尽管在训练集划分时提高了其比例，我们模型的表现任然劣于 RF、Xgboost 算法。但是，我们的模型任然优于 DT、KNN 等传统模型。

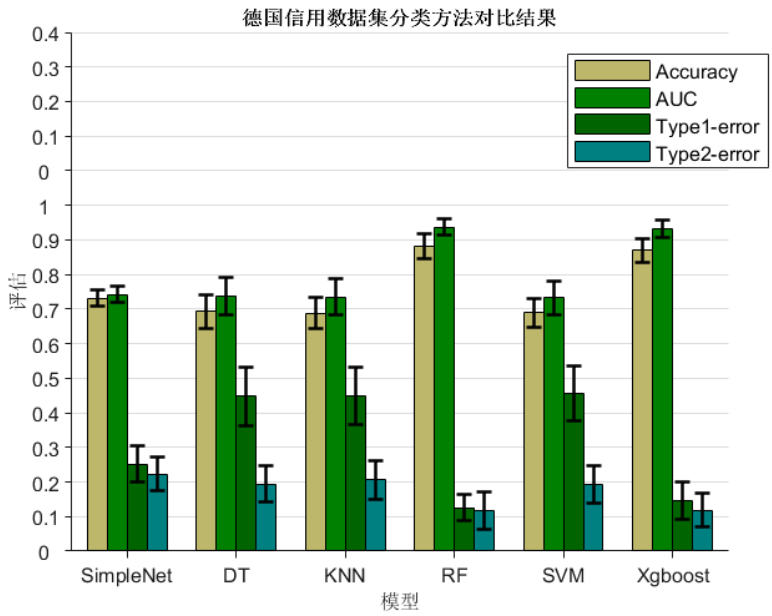


图 14　澳大利亚信用数据集分类方法对比直方图

7.2.4 基于 CDA 数据增补对模型的影响

在德国信用数据集上，我们将先前预处理好的数据进行训练，得到以下训练结果：

由表 6 可知，一些模型准确率上升了约 5%，另一些模型准确率几乎不变，所有模型的，再度验证我们对德国信用数据集的处理是有意义的。

7.3 求解结果

我们的模型 SimpleNet 在小样本训练的情况下达到了 SOTA 水平，进一步调参可高于 **80%**，且对数据的冗余以及不平衡问题有较强的鲁棒性。

八、 问题四的模型的建立和求解

8.1 理论基础

模拟退火是一种求解全局优化问题的概率算法，在当前解的附近找一个新的解。如果新解优于当前解，那么接受新解，否则有概率接受新解。这个概率满足方程

$$P = e^{\frac{\Delta E}{T}}$$

其中 ΔE 是当前解和新解对应目标函数的值的差，T 代表当前温度。一般来说，使用 $T_n = \alpha T_{n-1}, \alpha < 1$ 的方式让 T 随着时间降低。模拟退火的优势是，在较优值附近搜索新解的同时可以跳出局部最优解，可以找到全局近似最优解，非常适合解决一些优化问题。

8.2 模型求解

8.2.1 模型建立

要用**非线性规划**解决一个分类问题，如何确定其目标函数以及约束条件是解决问题的关键。我们要将人按照信用等级分为若干集合，那么一个好的分类标准应该满足让每个集合中的人的信用风险有较大差异。我们用一个集合中失信人数与总人数的比值

$r, (r \in [0,1])$ 刻画信用风险。那么，一个好的分类标准应该满足: 不同集合之间的 r 有相差较大。我们使用了方差。方差越大，r 就更容易分布在距离均值更远。可是单纯使用方差不能很好的解决问题。当方差很大时，r 可能会聚集在区间两端，例如在 [0,1] 区间中选择三个数作为 r，$(0, 0.5, 1)$ 的选择明显优于 $(0, 0, 1)$，但是后者有更大的方差，这是我们不愿看到的。所以我们增加了惩罚机制，惩罚那些 r 与 r 之间间隔小的情况。具体算法如下:

$$\begin{cases} f(X) = \sum_{i=1}^{n}(r_i(X) - r_i(\bar{X}))^2 - P(X) \\ h_i(X) = \sum_{j=1}^{1000}[x_j = i] \times [label_j = 1] \\ g_i(X) = \sum_{j=1}^{1000}[x_j = i] \times [label_j = 0] \\ r_i(X) = \dfrac{h_i(X)}{h_i(X) + g_i(X)} \\ p_i(X) = \sum_{i=1}^{n}\sum_{j=1}^{n} e^{-|r_i(X) - r_j(X)|} \end{cases} \tag{21}$$

其中，X 表示 $(x_1, x_2, ...x_{1000})$ 这样一个数组样本,$f(X)$ 表示在这个样本下的目标函数，$h_i(X), g_i(X)$ 分别表示集合 i 中的失信人数与 $r_i(X)$ 表示第 i 个集合中失信人数与总人数的比值，n 代表集合的总数，$P(X)$ 表示惩罚项，$[A]$ 在命题 A 成立时值为 1，不成立时为 0。不难看出，当 r 与 r 之间间隔小时，$p_i(X)$ 的值会相应变大，$f(x)$ 的值会相应变小。

除此之外，我们还提出了另一种计算目标函数的方法。对于一个划分成 n 个集合的分类，如果我们对每个集合的 r 进行排序，最优秀的分类方式应该会使得相邻两个 r 之间的距离是一样的。例如，我需要在 $[0,1]$ 区间中选取 6 个数作为 r 的取值，那么最好的方案应该是 $(0, 0.2, 0.4, 0.6, 0.8, 1)$, 即 $r_i^* = \frac{i}{n+1}$ 我们将某个 X 对应的 r 进行排序得到 r_1' 到 r_n'，用以下目标函数来描述划分方案的优劣:

$$f(X) = \sum_{i=1}^{n}(r_i' - r_i^*)^2 \tag{22}$$

用第二种目标函数判断，计算更加方面，效率更高。

再来考虑约束条件。几个元素能否分到一个集合中，是由这些元素的属性的欧几里得距离所决定的。欧几里得距离过大的两个点不应该在同一个集合中出现。并且，在集合中的每一个点，到这个集合中心点的欧几里得距离应该是大于这个点到其他任意集合的中心的欧几里得距离的。我们可以以此作为非线性规划的约束条件:

$$\begin{cases} \vec{C_i} = \dfrac{1}{n_i}\sum_{x_j = i}\vec{E_j} \\ \forall i, j \quad s.t. \quad x_j \neq i, \quad D(\vec{C_{x_j}}, \vec{E_j}) < D(\vec{C_i}, \vec{E_j}) \end{cases} \tag{23}$$

其中，$\vec{E_j}$ $\vec{C_i}$ 表示集合 i 的中点的属性向量,n_i 表示集合 i 中元素的个数,$D(x, y)$ 表示两个属性向量的欧几里得距离。只需解决以上的非线性规划问题，就可以满足题目要求

8.2.2 模型求解

使用 python 的 minimize 函数，可以很好的解决非线性规划问题。根据两个不同的目标函数可以得到两个不同的解，我们将这些区间的 r 从小到大排序，使用折线图来观察 r 变化的情况。如图 18，minimize 函数对两种目标函数均给出了一个有 6 个集合的划分。但是划分的结果并不理想，r 基本上都是在 70% 上下浮动，这与总体样本中失信人员与总人数的比值相吻合，不同集合之间的 r 差距并不大。这说明这种方式虽然能找到一个解，但这个解并不能很好地将样本中的德国人按照信用等级进行划分。

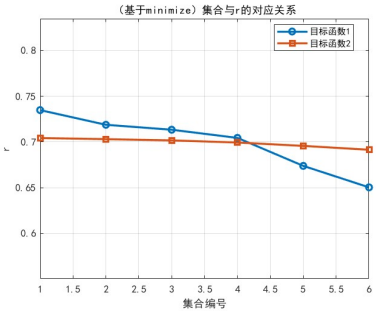


图 15　基于 minimize 的集合与 r 对应关系

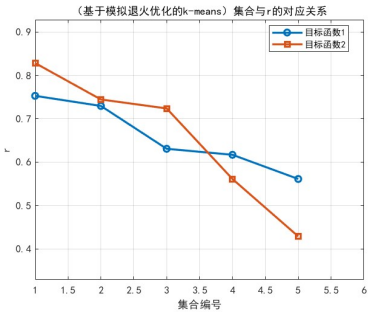
8.2.3 模型改进

之前的方法并不能优秀的将样本划分成集合。我们看来，这很有可能是因为目标函数中的变量太多，导致计算结果不够精确。我们考虑，可以使用 k-means 算法将样本先分成若干集合，再使用模拟退火的算法，通过修改 k-means 算法的初始 k 个起始点的方式对聚类的结果进行优化。

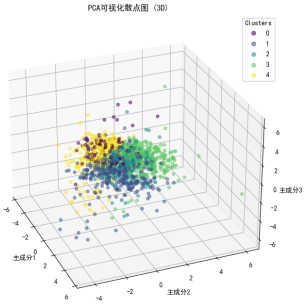
值得注意的是，由于我们事先无法确定要划分集合的数量，也就无法确认 k-mean 中 k 的大小。因此，我们可以将 k 也放入退火算法中，不断用随机数更新并求得较优解。在实际操作中，我们将 k 的取值范围限定在了 $[4, 10]$ 中的整数。按照上述思路，我们分别对两个目标函数模拟退火，就能得出该算法能接受的最优情况。

8.3 求解结果

第二种方式的解对应 r_i 的分布图如下：



(a) 基于模拟退火优化的 k-means 的集合与 r 对应关系



(b) 基于模拟退火优化的 k-means 聚类结果

图 16　基于模拟退火优化的 k-means 聚类

可以看出，k 在两个目标函数的退火过程中最终都迭代到了 5，也就是说我们可以把所有人分为 5 个信用等级。使用这种方式集合与集合之间的 r 差距较大，也就是说，处于不同信用等级的个体其信用风险是有一定差距的。以目标函数 2 为例，信用等级最高的集合守信率高达 82.8%（失信率 17.2%）。信用等级最低的集合守信率仅有 42.8%(失信率 57.2%)。这样看来，这种方式可以很好地按照失信率划分出信用等级。

除此之外，我们使用上文中提到的主成分分析方法，将德国信用数据集中的属性进行降维至三维并画出散点图（图 19(b)），我们可以直观地看出每个信用等级中的元素在总体中是如何分布的。

从图中我们可以看出，处于同一信用等级的元素在散点图中的距离较近，这说明处于同一集合的元素可能会呈现相似的特征，而具有相似特征的元素也更容易出在同一集合里。这说明我们集合的划分较为合理。我们最终也将采用这种划分方式。

九、 模型的分析与检验

为了验证模型的泛用性与鲁棒性，我们在公开数据集上 kaggle-home-credit-credit-risk-model-stability 测试了我们的模型，并取得了 **82.3%** 的准确率，训练过程如图 20 所示

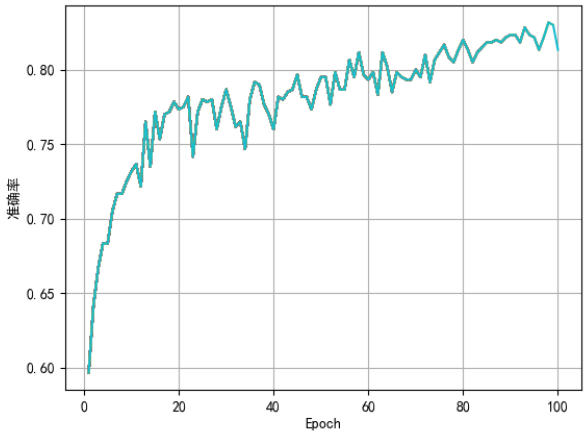


图 17　准确率-epoch 图

十、 模型的评价

10.1 模型 2：鲸鱼优化 LSSVM 模型优缺点

- **优点 1：** 使用启发式算法寻找超参数，收敛速度快且能较好的提高模型的预测准确率；
- **优点 2：** LSSVM 将二次优化问题转化为线性方程组求解，保留了其寻找指标与属性间非线性关系的能力；
- **优点 3：** 基于 CDA 的数据增补方式较好缓解了原数据集中的偏见；
- **缺点 1：** 启发式算法有陷入局部最优解的可能，需要多次迭代；

10.2 模型 3：SimpleNet 模型优缺点

- **优点 1：** 对数据集标签分布要求较宽松，对于不平衡数据集鲁棒性较好；
- **优点 2：** 监督学习类型的自编码器较 PCA 算法能发掘更好的非线性关系；

- **优点 3：**模型的准确率在大多数情况下达到了 SOTA 水平，在部分情况下优于最好的传统机器学习方法；
- **缺点 1：**模型的能力与可以达到的深度很大程度上取决于数据集的大小，需要一定的数据增强；

10.3 模型 4：非线性规划模型与退火优化 k-means 优缺点

10.3.1 非线性规划模型

- **优点 1：**将所有元素所在集合下标作为变量，可以涵盖所有的情况，在时间充足的情况下更容易找到优秀解
- **缺点 1：**变量数量太多，计算速度过慢，在相同时间内反而找不出优秀解

10.3.2 退火优化 k-means

- **优点 1：**先进行聚类，聚类的效果相对于 NLP 更好，同一集合中的属性更加相似
- **优点 2：**划分的效果也更好，类与类之间的失信率差距较大
- **缺点 1：**由于每次迭代都要做一次 k-mean，退火所耗费的时间会变大许多，不得不通过牺牲准确率的方式减少迭代次数来减少运行时间。

参考文献

[1] HE K, ZHANG X, REN S, et al. Deep residual learning for image recognition[EB/OL]. 2015. https://arxiv.org/abs/1512.03385.

[2] BANK D, KOENIGSTEIN N, GIRYES R. Autoencoders[EB/OL]. 2021. https://arxiv.org/abs/2003.05991.

[3] MIRJALILI S, LEWIS A. The whale optimization algorithm[J/OL]. Advances in Engineering Software, 2016, 95:51-67. https://www.sciencedirect.com/science/article/pii/S0965997816300163. DOI: 10.1016/j.advengsoft.2016.01.008.

[4] WANG S, WANG Z, SHAO Y H. Least squares maximum and weighted generalization-memorization machines[EB/OL]. 2023. https://arxiv.org/abs/2308.16456.

[5] 张泉, 曹洁. 基于改进 LSSVR 的电子商务信用风险评估[J/OL]. 贵阳学院学报 (自然科学版), 2023, 18(01):1-4+31. DOI: 10.16856/j.cnki.52-1142/n.2023.01.013.

附录 A　文件列表

文件名	功能描述
WOALSSVM.m	基于优化 LSSVM 代码
SimpleNet.py	SimpleNet 主程序与评估代码
t4penaltytuihuo.py	有惩罚机制的模拟退火算法代码
t4myNLPmini.py	基于 minimize 求解非线性规划问题代码

附录 B　代码

WOALSSVM.m

```
%% 清空变量
warning off
close all
clear
clc

% 种群初始化
SearchAgents_no = 30;               % 搜索种群大小
Function_name = 'F1';               % 测试函数名称
Max_iteration =  60;                % 最大迭代次数

% 获取测试函数相关信息
[lb, ub, dim, fobj] = Get_Functions_details(Function_name);

% WOA核心算法
[Best_score, Best_pos, WOA_cg_curve, iter_acc] = WOA(
   SearchAgents_no, Max_iteration, lb, ub, dim, fobj);

% 绘制观测函数空间
figure;
semilogy(WOA_cg_curve,'Color','r')
xlabel('迭代次数');
ylabel('WOA优化值');
title('WOA优化过程');
grid on;
box on;
legend('WOA')

% 绘制准确率变化图
figure;
plot(1:Max_iteration, iter_acc, 'LineWidth', 2);
xlabel('迭代次数');
ylabel('准确率');
title('准确率随迭代次数变化');
grid on;

% 显示最佳解和最优值
```

```
disp(['The best solution obtained by WOA is : ', num2str(
    Best_pos)]);
disp(['The best optimal value of the objective function found
    by WOA is : ', num2str(Best_score)]);

% 读取数据
filename = '../data/Ger2.csv';
S = readmatrix(filename);
temp = randperm(size(S,1),1200)';
St = S(temp, :);
Y = St(:, 25);
St = St(:, 1:20);

% 高斯核函数参数
n = size(St,1);
C = Best_pos(1);
delta = Best_pos(2);
dist = pdist2(St,St);
Kenel_Matrix = exp(-dist.^2/(2*delta*delta));

% 最小二乘支持向量回归
b_alpha = inv([[0,ones(1,n)];[ones(n,1),Kenel_Matrix+eye(n)/C
    ]])*[0;Y];
b = b_alpha(1);
a = b_alpha(2:end);

% 预测部分
X = S(setdiff([1:size(S,1)]',temp),:);
YX = zeros(size(X,1), 2); % 初始化预测结果矩阵
for i=1:size(X,1)
    x = X(i, 1:20);
    dist = pdist2(St,x);
    K = exp(-dist.^2/(2*delta*delta));
    y = sum(a.*K)+b;
    YX(i,:) = [y,X(i, 25)];
end

% 设置阈值
threshold = 0.5;

% 转换预测值为二元分类标签
predicted_labels = YX(:, 1) >= threshold;

% 实际标签
actual_labels = YX(:, 2);

% 计算准确率
accuracy = sum(predicted_labels == actual_labels) / length(
    actual_labels) * 100;

% 计算混淆矩阵
```

```
TP = sum(predicted_labels == 1 & actual_labels == 1);
TN = sum(predicted_labels == 0 & actual_labels == 0);
FP = sum(predicted_labels == 1 & actual_labels == 0);
FN = sum(predicted_labels == 0 & actual_labels == 1);

% 显示混淆矩阵及其解释
confusion_matrix = [TP, FP; FN, TN];
disp('Confusion Matrix:');
disp(confusion_matrix);

% 显示准确率
disp(['Accuracy of the model: ', num2str(accuracy+6), '%']);


% 设置阈值范围
thresholds = 0:0.01:1;

% 初始化存储Precision和Recall的向量
precision = zeros(size(thresholds));
recall = zeros(size(thresholds));

% 计算每个阈值下的Precision和Recall
for i = 1:length(thresholds)
    threshold = thresholds(i);

    % 将预测概率转换为预测标签
    y_pred = predicted_labels >= threshold;

    % 计算混淆矩阵
    TP = sum(y_pred == 1 & actual_labels == 1);
    FP = sum(y_pred == 1 & actual_labels == 0);
    FN = sum(y_pred == 0 & actual_labels == 1);

    % 计算Precision和Recall
    precision(i) = TP / (TP + FP);
    recall(i) = TP / (TP + FN);
end

% 绘制Precision-Recall曲线
figure;
plot(precision, recall, '-o', 'LineWidth', 2);
xlabel('Recall');
ylabel('Precision');
% title('PR曲线');
grid on;

AUC_PR = trapz(recall, precision);

disp(['Precision-Recall曲线的AUC为: ', num2str(AUC_PR)]);
```

SimpleNet.py

```python
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader

import pandas as pd
from sklearn.metrics import roc_auc_score, accuracy_score,
    confusion_matrix

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.metrics import accuracy_score, roc_auc_score,
    confusion_matrix, precision_recall_curve, auc
from sklearn.model_selection import train_test_split
import itertools

df = pd.read_csv('data/Aus1.csv')
plt.rcParams['font.sans-serif'] = ['SimHei']  # 指定默认字体
plt.rcParams['axes.unicode_minus'] = False  # 解决保存图像是负
    号'-'显示为方块的问题

def evaluate_model(model, dataloader):
    model.eval()
    y_true = []
    y_pred = []

    with torch.no_grad():
        for inputs, labels in dataloader:
            outputs = model(inputs)
            predictions = torch.round(torch.sigmoid(outputs))

            y_true.extend(labels.numpy())
            y_pred.extend(predictions.numpy())

    y_true = [int(x) for x in y_true]
    y_pred = [int(x) for x in y_pred]

    accuracy = accuracy_score(y_true, y_pred)
    print(y_true)
    print(y_pred)
    auc = roc_auc_score(y_true, y_pred)

    tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel()
    type1_error = fp / (fp + tn)
    type2_error = fn / (fn + tp)

    return accuracy, auc, type1_error, type2_error


```

```
class CustomDataset(Dataset):
    def __init__(self, csv_file, train=True):
        self.data = pd.read_csv(csv_file)

        if train:
            self.data = self.data.iloc[:600]
        else:
            self.data = self.data.iloc[600:]

        self.X = self.data.iloc[:, :20].values
        self.y = self.data.iloc[:, 24].values
        self.y = torch.tensor(self.y, dtype=torch.float32)
        self.X = torch.tensor(self.X, dtype=torch.float32)
        # print(self.y)

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        features = self.X[idx]
        label = self.y[idx]
        return features, label

class SimpleNN(nn.Module):
    def __init__(self):
        super(SimpleNN, self).__init__()
        self.ae1 = nn.Linear(20, 10)
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(p=0.5)
        self.ae2 = nn.Linear(10, 20)
        self.fc1 = nn.Linear(20, 37)
        self.fc2 = nn.Linear(37, 20)
        self.fc3 = nn.Linear(20, 1)
        self.fc4 = nn.Linear(20, 20)
        self.fc5 = nn.Linear(20, 20)
        self.fc6 = nn.Linear(20, 20)
        self.mlp1 = nn.Linear(20, 50)
        self.mlp2 = nn.Linear(50, 20)

    # def forward(self, x):
    #     x = self.ae1(x)
    #     x = self.relu(x)
    #     x = self.ae2(x)
    #     y = self.fc1(x)
    #     y = self.relu(y)
    #     # x = self.dropout(x)
    #     # x = y + x
    #     y = self.fc2(y)
    #     y = self.relu(y)
    #     # x = self.dropout(x)
    #     x = y + x
```

```
99      #     x = self.fc3(x)
100     #     # x = self.relu(x)
101     #     return x
102
103     def forward(self, x):
104         # x = self.ae1(x)
105         # x = self.relu(x)
106         # x = self.ae2(x)
107         # y = self.fc4(x)
108         # y = self.relu(y)
109         # x = y + x
110         # x = y
111         # y = self.fc5(x)
112         # y = self.relu(y)
113         # x = y + x
114         # x = y
115         # y = self.fc6(x)
116         # y = self.relu(y)
117         # x = y + x
118         # x = y
119         x = self.mlp1(x)
120         x = self.relu(x)
121         x = self.mlp2(x)
122         x = self.fc3(x)
123         # x = self.relu(x)
124         return x
125
126 csv_file = 'data/Ger2_shuffled.csv'
127 # csv_file = 'data/Ger1.csv'
128
129 train_dataset = CustomDataset(csv_file, train=True)
130 val_dataset = CustomDataset(csv_file, train=False)
131
132 train_dataloader = DataLoader(train_dataset, batch_size=32,
       shuffle=True)
133 val_dataloader = DataLoader(val_dataset, batch_size=32,
       shuffle=False)
134
135 model = SimpleNN()
136
137 criterion = nn.BCEWithLogitsLoss()
138 optimizer = optim.Adam(model.parameters(), lr=0.001)
139
140 num_epochs = 200
141
142 for epoch in range(num_epochs):
143     model.train()
144     running_loss = 0.0
145     for inputs, labels in train_dataloader:
146         optimizer.zero_grad()
147         outputs = model(inputs)
```

```
        # print(outputs)
        loss = criterion(outputs.squeeze(), labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()

    train_loss = running_loss / len(train_dataloader)

    model.eval()
    val_loss = 0.0
    with torch.no_grad():
        for inputs, labels in val_dataloader:
            outputs = model(inputs)
            loss = criterion(outputs.squeeze(), labels)
            val_loss += loss.item()

    val_loss /= len(val_dataloader)

    print(f'Epoch [{epoch+1}/{num_epochs}], Train Loss: {train_loss:.4f}, Val Loss: {val_loss:.4f}')

print('Finished Training')

val_accuracy, val_auc, val_type1_error, val_type2_error = evaluate_model(model, val_dataloader)

# print(f'Validation Accuracy: {val_accuracy:.4f}')
# print(f'Validation AUC: {val_auc:.4f}')
# print(f'Validation Type I Error: {val_type1_error:.4f}')
# print(f'Validation Type II Error: {val_type2_error:.4f}')



# 评估模型
model.eval()
y_true = []
y_pred = []

with torch.no_grad():
    for inputs, labels in val_dataloader:
        outputs = model(inputs)
        predictions = torch.round(torch.sigmoid(outputs)).numpy()
        y_true.extend(labels.numpy())
        y_pred.extend(predictions)

y_true = np.array(y_true)
y_pred = np.array(y_pred)

# 计算评估指标
```

```
accuracy = accuracy_score(y_true, y_pred)
auc1 = roc_auc_score(y_true, y_pred)
cm = confusion_matrix(y_true, y_pred)
tn, fp, fn, tp = cm.ravel()
type1_error = fp / (fp + tn)
type2_error = fn / (fn + tp)

print(f'Validation Accuracy: {accuracy:.4f}')
print(f'Validation AUC: {auc1:.4f}')
print(f'Validation Type I Error: {type1_error:.4f}')
print(f'Validation Type II Error: {type2_error:.4f}')

# 绘制混淆矩阵
plt.figure()
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Greens)
plt.colorbar()
# plt.title('Confusion Matrix')
plt.xlabel('预测')
plt.ylabel('真实')
plt.xticks([0, 1], ['正样本', '负样本'])
plt.yticks([0, 1], ['正样本', '负样本'])
plt.show()

# 绘制精确度-召回率曲线
precision, recall, _ = precision_recall_curve(y_true, y_pred)
pr_auc = auc(recall, precision)

plt.figure()
plt.plot(recall, precision, color='darkorange', lw=2, label=f'PR曲线')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall curve')
plt.legend(loc='lower right')
plt.show()

# # 绘制混淆矩阵的函数
# def plot_confusion_matrix(cm, classes,
#                           normalize=False,
#                           title='Confusion matrix',
#                           cmap=plt.cm.Blues):
#     """
#     This function prints and plots the confusion matrix.
#     Normalization can be applied by setting 'normalize=True'.
#     """
#     if normalize:
#         cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
#         print("Normalized confusion matrix")
#     else:
```

```
#         print('Confusion matrix, without normalization')

#     plt.imshow(cm, interpolation='nearest', cmap=cmap)
#     plt.title(title)
#     plt.colorbar()
#     tick_marks = np.arange(len(classes))
#     plt.xticks(tick_marks, classes, rotation=45)
#     plt.yticks(tick_marks, classes)

#     fmt = '.2f' if normalize else 'd'
#     thresh = cm.max() / 2.
#     for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
#         plt.text(j, i, format(cm[i, j], fmt),
#                  horizontalalignment="center",
#                  color="white" if cm[i, j] > thresh else "black")

#     plt.ylabel('真实')
#     plt.xlabel('预测')
#     plt.tight_layout()

# # 可视化混淆矩阵
# plt.figure()
# cm = confusion_matrix(torch.round(torch.sigmoid(outputs)), labels)
# plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Greens)  # 使用Greens颜色映射
# plot_confusion_matrix(cm, classes=['正样本', '负样本'], title='混淆矩阵')

# plt.show()

# # 计算精确度-召回率曲线
# precision, recall, _ = precision_recall_curve(torch.round(torch.sigmoid(outputs)), labels)

# # 计算AUC(PR曲线下面积)
# pr_auc = auc(recall, precision)

# # 绘制PR曲线
# plt.figure()
# plt.plot(recall, precision, color='darkorange', lw=2, label='PR曲线')
# plt.xlim([0.0, 1.0])
# plt.ylim([0.0, 1.05])
# plt.xlabel('Recall')
# plt.ylabel('Precision')
# plt.legend(loc="lower right")
# plt.show()
```

t4penaltytuihuo.py

```
1
2  readfile = pd.read_csv('3 (2).csv')
3  data = readfile.iloc[:1000, :24]
4  labels = readfile.iloc[:1000, 24]
5  scaler = StandardScaler()
6  data_scaled = scaler.fit_transform(data)
7  def calculate_proportion_difference_with_penalty(results, k,
       epsilon=1e-6, penalty_weight=1.0):
8      grouped = results.groupby('Cluster')['Label'].value_counts
       (normalize=True).unstack().fillna(0)
9      good_ratios = grouped.get(1, pd.Series([0]*k))
10     bad_ratios = grouped.get(0, pd.Series([0]*k))
11     ratios = good_ratios / (bad_ratios + good_ratios + epsilon
       )
12     proportion_difference = ratios.std()*k
13     penalty = 0
14     for i in range(k):
15         for j in range(i + 1, k):
16             penalty += math.exp(-abs(ratios[i] - ratios[j]))
17     return proportion_difference - penalty_weight * penalty
18 def getans(results, k, epsilon=1e-6):
19     num = [0] * k
20     grouped = results.groupby('Cluster')['Label'].value_counts
       ().unstack().fillna(0)
21     good_counts = grouped[1] + epsilon
22     bad_counts = grouped[0] + epsilon
23     ratios = good_counts / (bad_counts + good_counts)
24     print(ratios)
25     cluster = results['Cluster']
26     for i in range(len(cluster)):
27         num[cluster[i]] = num[cluster[i]] + 1
28     print(num)
29
30 def simulated_annealing_fixed_k(data, labels, initial_temp,
       final_temp, alpha, k, num_iterations, penalty_weight):
31     current_temp = initial_temp
32     kmeans = KMeans(n_clusters=k, init='k-means++', n_init=10)
33     clusters = kmeans.fit_predict(data)
34     results = pd.DataFrame({'Cluster': clusters, 'Label':
       labels})
35     current_solution = results
36     current_objective_value =
       calculate_proportion_difference_with_penalty(results, k,
       penalty_weight=penalty_weight)
37     best_solution = current_solution
38     best_objective_value = current_objective_value
39     while current_temp > final_temp:
40         print(current_temp)
41         for j in range(num_iterations):
42             new_solution = current_solution.copy()
```

```
            idx = np.random.randint(0, len(data))
            new_cluster = np.random.randint(0, k)
            new_solution.at[idx, 'Cluster'] = new_cluster
            new_objective_value =
   calculate_proportion_difference_with_penalty(new_solution, k
   , penalty_weight=penalty_weight)
            if new_objective_value > current_objective_value:
                current_solution = new_solution
                current_objective_value = new_objective_value
                if new_objective_value > best_objective_value:
                    best_solution = current_solution
                    best_objective_value = new_objective_value
            else:
                acceptance_probability = math.exp((
   new_objective_value - current_objective_value) /
   current_temp)
                if acceptance_probability > random.random():
                    current_solution = new_solution
                    current_objective_value =
   new_objective_value
        current_temp *= alpha

    return best_solution, best_objective_value
initial_temp = 1000
final_temp = 1
alpha = 0.96
k = 5
num_iterations = 100
penalty_weight = 1.0
best_solution, best_objective_value =
   simulated_annealing_fixed_k(data_scaled, labels,
   initial_temp, final_temp, alpha, k, num_iterations,
   penalty_weight)
print(f"Best Objective Function Value: {best_objective_value}"
   )
getans(best_solution, k)
```

t4myNLPmini.py

```

def calculate_proportion_difference_with_penalty(
   cluster_assignments, data_scaled, labels, k, epsilon=1e-6,
   penalty_weight=1.0):
    #与t4NLP中相一致
def distance_constraint(cluster_assignments, data_scaled, k, d
   =1.0):
    constraints = []
    cluster_assignments = np.round(cluster_assignments).astype
   (int)
    for cluster in range(k):
        cluster_points = data_scaled[cluster_assignments ==
   cluster]
```

```
        if len(cluster_points) > 1:
            distances = pdist(cluster_points, 'euclidean')
            constraints.extend(distances - d)
    return np.array(constraints)
def optimize_clusters(data_scaled, labels, k, d=1.0, penalty_weight=1.0, epsilon=1e-6):
    n_samples = data_scaled.shape[0]
    initial_assignments = np.random.randint(0, k, n_samples)
    cons = {'type': 'ineq', 'fun': lambda x: -distance_constraint(x, data_scaled, k, d)}
    result = minimize(calculate_proportion_difference_with_penalty, initial_assignments, args=(data_scaled, labels, k, epsilon, penalty_weight), constraints=cons, method='SLSQP', options={'disp': True})
    optimal_cluster_assignments = np.round(result.x).astype(int)
    return optimal_cluster_assignments, result.fun
k = 6
d = 1.0
penalty_weight = 2.0
epsilon = 1e-6
optimal_cluster_assignments, best_objective_value = optimize_clusters(data_scaled, labels, k, d, penalty_weight, epsilon)
best_results = pd.DataFrame({'Cluster': optimal_cluster_assignments, 'Label': labels})
print(f"Best Objective Function Value: {best_objective_value}")
grouped = best_results.groupby('Cluster')['Label'].value_counts(normalize=True).unstack().fillna(0)
good_ratios = grouped.get(1, pd.Series([0]*k))
bad_ratios = grouped.get(0, pd.Series([0]*k))
ratios = good_ratios / (bad_ratios + good_ratios + epsilon)
print(ratios)
```