2022 年第七届“数维杯”大学生

数学建模挑战赛论文

题　　目 基于 XGBoost 和增量学习的银行倒闭风险模型研究

摘 要

国际银行由于其倒闭概率较高，波动幅度较广，因而如何进行银行效率评价与倒闭成因分析十分重要。本题从波兰 2017 年至 2021 年的现存或倒闭银行的 64 项指标数据入手，完成相应分析任务。

对于任务 1，我们首先进行了指标的选取，并评价了银行效率的同时确定其倒闭临界线。首先通过**独立样本 *t* 检验**得出了有显著差异的投入产出项，然后使用**熵权法**和**TOPSIS 方法**量化了其权重，并据此给出银行效率的综合评价及倒闭分界线。

对于任务 2，我们需要从所给的 64 项指标中提取出最重要的五项。因而在倒闭与否的分类问题中建立了基于 **XGBoost 的特征工程模型**，对节点权重与特征进行排序，得到最重要的五项数据分别为 X34，X56，X26，X27 和 X5。归一化后权重分别为 0.287、0.220、0.178、0.175、0.138。

对于任务 3，进一步在分类任务的基础上，对比多种不同的机器学习模型求解从而分析出倒闭风险。最终发现 **XGBoost 分类器模型**表现最优，准确率和 AUC 均高达 0.99。

对于任务 4，则需要进行二分类的小样本推断。考虑到数据体量的失衡，使用**基于增量学习的小样本推断模型**，将选择的 40 个样本推广到 2021 年的全样本并进行了测试，准确率和 F1 分数均达到 0.91，具有一定准确性和可扩展性。

对于任务 5，构建了基于 **KM 算法**的最大权匹配模型，通过余弦相似度与最大权匹配推断同一实体。在此基础上，利用**线性回归**来预测倒闭趋势，筛选出 2146 家具有倒闭趋势的银行。

模型综合运用了多种方法，表现良好，对银行倒闭的预测问题起到了有效预测作用。

关键词：银行效率与倒闭；TOPSIS；XGBoost；增量学习；KM 算法

目 录

一、问题重述

1.1 问题背景

银行业在在金融领域乃至整个经济领域一直处于核心的地位。国际银行的倒闭有可能导致银行系统危机的出现。为此，加强对银行的监管，特别地妥善处理面临倒闭的银行，及早采取措施，发出预警，甚至提早关闭面临倒闭的商业银行，对保障金融秩序，维护经济稳定十分重要。对银行效率进行测评,并分析影响其倒闭的因素，既可为银行调整资源配置和经营策略提供数据支持，也便于相关监管机构对银行实行有针对性的监管措施，这对于完善银行的组织模式，减少其倒闭概率具有十分重要的意义，因而受到了国内外学者的广泛关注。基于此，我们需要对目前的银行效率评价与倒闭成因问题进行相关的分析和预测。

1.2 问题提出

本题给出了 2017~2021 年的银行信息，要求综合考虑其 64 个相关财务指标的因素，对银行的效率进行评价，分析其倒闭的原因和趋势并进行量化评估，同时根据数据预测出银行的倒闭风险。

因此，本题要求建立数学模型，解决以下问题：

（1）考虑适合的投入产出数据，评价对应的银行效率，并提供分界线。

（2）分析银行倒闭的原因，并对各项指标数据进行权重分析并排序。

（3）基于倒闭原因的分析结果，预测银行倒闭风险。

（4）在 2021 年的数据中分别选取 20 家最有代表性的现存银行和倒闭银行，并针对其进行风险预测。

（5）判断数据来源并据此预测该银行的倒闭趋势。

二、问题分析

2.1 任务 1 的分析

我们采用假设检验的方式，用独立样本 t 检验来对各指标进行比对，从而得出在是否倒闭的银行间有显著差异的投入产出的数值项，即二者的关键差别指标所在，并由临界值得出分界线标准。为了界定这些指标的权重，可对这些指标与理想化目标的接近程度进行排序，因而可以使用基于熵权法和 TOPSIS 的效率评价模型来比较数据项权重，从而综合得出银行倒闭效率的阈值，对银行是否倒闭做出界定。

2.2 任务 2 的分析

题中所给数据属于时序数据，具有数据量大、指标多、非线性等特点，因此本文选

用对多变量处理能力强、运算速度快的 XGBoost 模型。运用 Boosting 的思想，在此基础上通过叶子权重，则可以得出相关样本的预测值并进行排序，从而在 64 项指标中得出对银行倒闭的原因最具有显著性影响的因素，得到最为重要的 5 项数据。

2.3 任务 3 的分析

在任务 1 和任务 2 基础上，继续采用 XGBoost 模型。基于混淆矩阵，计算 F1 值，从而综合了精准率和召回率两个指标，从而综合构建模型来比对银行倒闭结果。为了避免 F1 分数和召回率过低的问题（这是由于该样本中倒闭的银行为少部分），拟采用重复采样的预处理，并采用网格搜索法调整参数，以得到更精确的倒闭风险预测模型。

2.4 任务 4 的分析

由于我们需要基于选取的 40 个小样本的银行数据对其它大样本的银行倒闭风险进行预测，即极小批量样本上学习的模型迁移到大批量样本，因而传统的机器学习方法由于学习的样本数过少而不再适用。基于该问题的特殊性，我们拟采用小样本推断的方法，采用增量式学习的策略解决。从而分批次进行学习和推广，从而解决对大样本的预测问题。

2.5 任务 5 的分析

我们需要基于 2017-2021 五年不同的数据识别匿名数据下的同一实体，因而需要通过相似度来进行识别，即进行二者的最优匹配，故而我们选取 KM 算法。在此基础上，采用线性回归的方法来对银行倒闭情况进行识别。

三、模型假设

1. 假设题目所给的数据真实可靠。
2. 假设无突发事件或意外事件发生。
3. 不考虑商业环境变化的影响。
4. 不考虑国家政策等方面的影响。

四、定义与符号说明

符号定义	符号说明
Z	归一化的决策矩阵
z_{ij}	决策矩阵中第 i 个决策项第 j 个指标
w_j	熵权法求解的第 j 个指标的权重系数

Z^+, Z^-	正理想解和负理想解
D^+, D^-	决策项到正理想解和负理想解的距离
p_{ij}	对于指标 j，第 i 类所占比例
e_j	指标 j 的熵值
n	指标 j 有多少类
$p(x)$	x 的概率
$H(X)$	X 的信息熵
$H(X\|Y)$	按照 Y 划分的条件信息熵
$ID(X,Y)$	信息增益
$L(\Phi)$	XGBoost 的损失函数
$f_t(x)$	学习的基学习器
$\Omega(f_t(x))$	正则化项，与基学习器的深度与权值范数有关
λ	常数项
w	XGBoost 中的各项权值系数
accuracy	准确率
F1-score	F1 分数
precision	查准率
recall	查全率
TP	预测为正实际也为正
FP	预测为负实际为正
TN	预测为正实际为负
FN	预测为负实际也为负
Gini	基尼指数

五、模型的建立与求解

数据预处理：我们将五年期数据拼接为一份样本整体，注意到数据中有些样本是存在缺失的，我们利用均值填充法对数据进行了填充，排除异常点后绘制了图 5-1 所示的热力图分析指标间相关性。

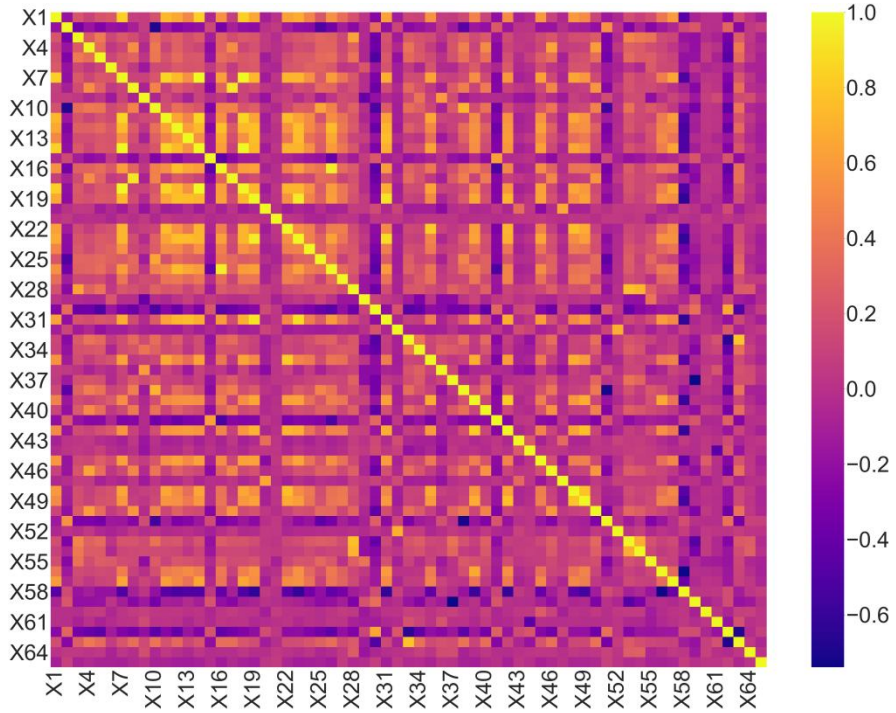

图 5-1　指标相关性

可以看到整体样本之间实际上有不少指标的关联性是比较强的，这说明在后续任务中大概率会存在特征冗余的现象，需要进行降维与分析。

由于样本总体中未倒闭银行和倒闭银行比例达到将近 20:1，这一严重的样本失衡将对后续任务产生重要影响。为了解决样本失衡问题，我们对原始数据进行了一次过采样处理，将倒闭银行的数据重复采样 20 次放入构造新数据集，使正负样本数量达到平衡。

5.1 任务 1 的模型建立与求解

5.1.1 基于熵权法和 TOPSIS 的效率评价模型的建立

评价类问题常用的方法有层次分析法（AHP）、模糊综合判别法、灰色关联分析（GRA）、主成分分析（PCA）和优劣解距离法（TOPSIS）等[1]，但这些方法侧重点各有不同，前两者属于主观赋权评价法，在该问题中由于缺乏运输订购类问题中相关变量的权重分配研究，进行应用时权重会带有一定主观性和随机性，不利于量化。灰色关联分析更多用于分析序列之间的相关性而非重要性，主成分分析更多则用于降维减少变

量维度，并且要求构建的主成分有充分的可解释性，在这一问题中不能很好地达到我们的目的，而 TOPSIS 方法是多目标决策分析中一种常用的有效方法，是一种趋近于理想解的排序法。它根据有限个评价对象与理想化目标的接近程度进行排序，在现有的对象中进行相对优劣的评价[2]，故我们选用 TOPSIS 方法进行求解。

TOPSIS 评价法是有限方案多目标决策分析中常用的一种科学方法，其基本思想为，对原始决策方案进行归一化，然后找出最优方案和最劣方案，对每一个决策计算其到最优方案和最劣方案的欧几里得距离，然后再计算相似度。若方案与最优方案相似度越高则越优先。基本流程如下

Step 1：根据归一化得到的决策矩阵（这里我们选取 min-max 归一化）和权重向量构造规范化权重矩阵 Z：

$$Z = [w_j z_{ij}] \tag{1}$$

Step 2：确定正理想解Z^+和负理想解Z^-,其中J^+, J^-分别表示效益型指标和成本型指标：

$$\begin{aligned} Z^+ &= \left\{ \max_{j \in J^+} z_{ij}, \min_{j \in J^-} z_{ij} \right\} \\ Z^- &= \left\{ \max_{j \in J^+} z_{ij}, \min_{j \in J^-} z_{ij} \right\} \end{aligned} \tag{2}$$

Step 3：计算各评价对象 i（$i = 1, 2, 3, \ldots, 402$）到正理想解和负理想解的欧几里得距离D_i：

$$\begin{aligned} D_i^+ &= \sqrt{\sum_{j=1}^{n} [z_{ij} - Z_j^+]^2} \\ D_i^- &= \sqrt{\sum_{j=1}^{n} [z_{ij} - Z_j^-]^2} \end{aligned} \tag{3}$$

Step 4：计算各评价对象的相似度W_i

$$W_i = \frac{D_i^-}{D_i^+ + D_i^-} \tag{4}$$

可以看到，相似度是与负理想解与两理想解距离之和的比值，若占比越大则说明离负理想解越远，越优先选择。

Step 5：根据W_i大小排序可得到结果。

权重向量的构建是 TOPSIS 应用的核心，需要尽可能削弱其主观性。这里我们使用

熵权法构建权重向量。熵权法基于信息论，基于信息论的熵值法是根据各指标所含信息有序程度的差异性来确定指标权重的客观赋权方法，仅依赖于数据本身的离散程度[5]。熵用于度量不确定性，指标的离散程度越大，说明不确定性越大，则最终熵值越大，该指标值提供的信息量越多，则权重也相应越大。

TOPSIS 的思想流程大致如图 5-2 所示：

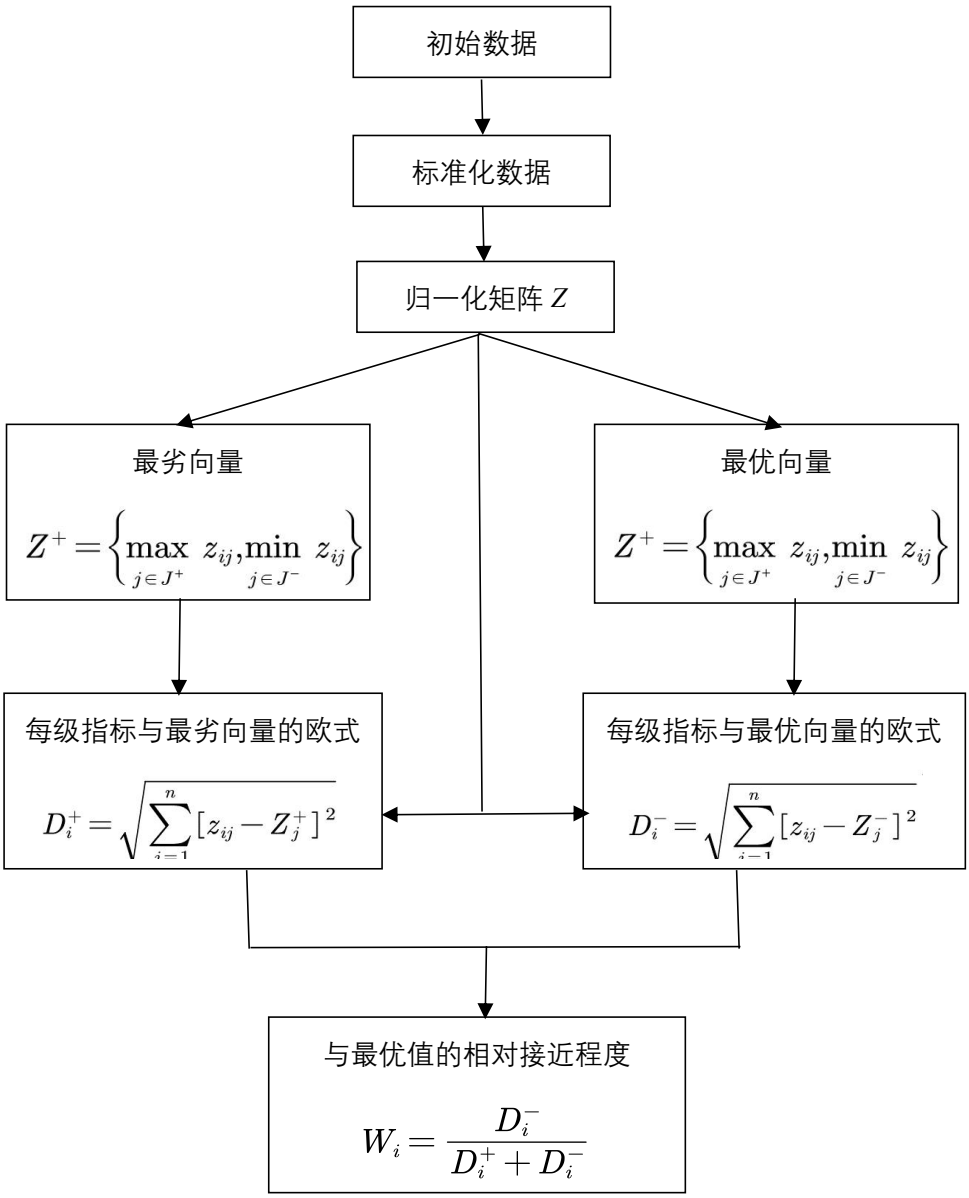

图 5-2　TOPSIS 流程示意图

根据信息论中对熵的定义[3]，熵值 e 的计算如下所示

$$e_j = -\frac{\sum_{i=1}^{n} p_{ij} \ln p_{ij}}{\ln n} \tag{5}$$

式子(4)中代表对于某一个属性 j，第 i 类占样本的比例。n 为属性 j 的取值数量。所以权重系数 w 定义为：

$$w_j = \frac{1 - e_j}{\sum_{i=1}^{m} (1 - e_i)} \tag{6}$$

这样，我们构建了 TOPSIS 综合评价与熵权分析法的综合模型。

5.1.2 基于熵权法和 TOPSIS 的效率评价模型的求解

我们经过对 64 项指标作独立样本 *t* 检验，分析结果发现 X11，X18，X22，X24，X29，X35，X48，X55 八项指标存在显著性差异，具有统计学意义，是投入产出的核心，可作为投入产出指标。将预处理数据带入上述模型，通过熵权法确定 8 项指标权重分别为 [0.161623795, 0.106874094, 0.544739146, -0.109166049, 0.000409266543, 0.213142174, 0.0673550358, 0.0150225386]。

表 5-1　指标的 *t* 检验结果

独立样本检验						
		莱文方差等同性检验		平均值等同性 *t* 检验		
		F	显著性	*t*	自由度	Sig.（双尾）
X11	假定等方差	8.649	.003	3.336	43403	.001
	不假定等方差			1.769	2139.764	.077
X18	假定等方差	10.073	.002	3.344	43403	.001
	不假定等方差			1.900	2148.896	.058
X22	假定等方差	7.327	.007	3.134	43403	.002
	不假定等方差			1.740	2145.748	.082
X24	假定等方差	3.597	.058	3.191	43403	.001
	不假定等方差			2.379	2201.430	.017
X29	假定等方差	6.165	.013	11.623	43403	.000
	不假定等方差			11.277	2292.301	.000
X35	假定等方差	9.814	.002	3.750	43403	.000

	不假定等方差			1.935	2136.471	.053
X48	假定等方差	13.076	.000	3.310	43403	.001
	不假定等方差			1.450	2120.304	.147
X55	假定等方差	26.382	.000	4.605	43403	.000
	不假定等方差			10.066	3598.672	.000

我们将部分样本（2021 年前 10 条样本）的结果展示如表 5-2 所示：

表 5-2　TOPSIS 解的综合得分

样本 id	正理想解	负理想解	综合得分	排序
1	0.756024158	0.519409366	0.407241425	3814
2	0.756058801	0.519616033	0.407326396	1550
3	0.756026881	0.519250202	0.407166575	5011
4	0.756086827	0.51902369	0.407042121	5541
5	0.75610117	0.518850082	0.4069568	5688
6	0.755955012	0.519355221	0.407238339	3907
7	0.756045724	0.519389484	0.407225299	4173
8	0.756089923	0.51898411	0.407022727	5582
9	0.756082911	0.519146316	0.40710039	5396
10	0.756020318	0.519502449	0.407285909	2491

从表 5-2 可以看出，经过对倒闭样本和非倒闭样本的对比分析，我们解得最终效率阈值为 0.40724。

5.1.3 结果

因此，经过独立样本 *t* 检验筛选出 X11，X18，X22，X24，X29，X35，X48，X55 这八个对应的投入产出数据项，由此各银行的效率展开对应评价，得到综合得分。经过对倒闭样本和非倒闭样本的对比分析，我们得到银行倒闭效率的分界线为 0.40724，即当综合得分低于该值时，银行很有可能面临倒闭。

5.2 任务 2 的模型建立与求解

对于是否倒闭这一状态，可以将其视作一个二分类问题。对于分类问题的特征筛选，可以采用机器学习的策略进行自动化特征工程。

5.2.1 基于 XGBoost 的特征工程模型的建立

特征工程分为数据预处理、特征抽取、特征构造和特征选择等几个步骤[4]，使用

机器学习算法自动进行特征筛选的一类重要模型是树模型，这里选用 XGBoost 算法进行测试。XGBoost 算法是一类基于树结构的算法，而树结构为每个特征分配重要性分数的核心在于信息增益的计算。

由 Chen 等人提出的 XGBoost 是对梯度提升树框架(Gradient Boost Decision Tree Framework)的一种具体实现，以 CART 决策树作为基学习器，既可以用于分类问题也可以用于回归问题[5]。XGBoost 的基本形式为：

$$\begin{cases} L(\phi)=\sum_i l(\hat{y}_i,y_i)+\sum_k \Omega(f_k) \\ \Omega(f)=\gamma T+\frac{1}{2}\lambda\,||\omega||^2 \end{cases} \tag{7}$$

在迭代过程中的损失函数表达式如下：

$$\begin{aligned} L^{(t)} &= \sum_{i=1}^{n} l\left(y_i,\hat{y}_i^{(t-1)}+f_t(x_i)\right)+\Omega(f_t) \\ &\simeq \sum_{i=1}^{n}\left[l\left(y_i,\hat{y}^{(t-1)}\right)+g_i f_t(x_i)+\frac{1}{2}h_i f_t^2(x_i)\right]+\Omega(f_t) \end{aligned} \tag{8}$$

不同于普通的 GBDT，XGBoost 在 L 后加入了与决策树深度 T 和分类器分数 w 有关的正则化项，使得精度本就出众的模型更具有鲁棒性。此外从某种意义上讲，它本身还具有降维的效果，并且对类别失衡的数据效果较好。

XGBoost 由于是基于 Boosting 原理训练基学习器，每一步的学习器是在上一步学习器的基础上改进得到。其迭代训练算法如下所示：

Algorithm 1: Exact Greedy Algorithm for Split Finding

Input: I, instance set of current node
Input: d, feature dimension
$gain \leftarrow 0$
$G \leftarrow \sum_{i\in I} g_i,\ H \leftarrow \sum_{i\in I} h_i$
for $k = 1$ ***to*** m **do**
  $G_L \leftarrow 0,\ H_L \leftarrow 0$
  for j *in sorted(*I, *by* $\mathbf{x}_{jk}$*)* **do**
    $G_L \leftarrow G_L + g_j,\ H_L \leftarrow H_L + h_j$
    $G_R \leftarrow G - G_L,\ H_R \leftarrow H - H_L$
    $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
  end
end
Output: Split with max score

图 5-3　XGBoost 算法训练过程

XGBoost 使用了和 CART 回归树一样的想法，利用贪婪算法，将所有特征升序排序，然后遍历所有特征的所有特征划分点，不同的是使用的目标函数不一样。具体做法就是求分裂后的目标函数值比分裂前的目标函数的增益。而 XGBoost 虽然采用了 Boosting 系列[6]方法，但同样可以在特征级上做到并行训练，对不同特征的基尼指数进行并行化计算。

信息增益是指在按照某一条件划分以后的信息熵相较划分前信息熵的增量，能够衡量划分的效果，此外，信息增益率和基尼指数也可以衡量划分能力。

$$IG(X,Y)=H(Y)-H(Y|X)=\sum_{x,y}p(x)\cdot p(y|x)\cdot \log p(y|x)-\sum_{y}p(y)\cdot \log p(y) \quad (9)$$

使用信息熵我们就能够计算出，根据哪一特征划分信息增益最大，然后按照信息增益从大到小排序即可得到最重要的特征。

5.2.2 基于 XGBoost 的特征工程模型的求解

我们通过构建 XGBoost 分类器模型，将预处理后的所有数据输入 XGBoost 模型进行训练，经过 10 折交叉验证后得到性能较为鲁棒的 XGBost 模型。通过将基学习器的特征节点权重与特征进行排序，获得分数最高的 10 个特征如图 5-4 所示：

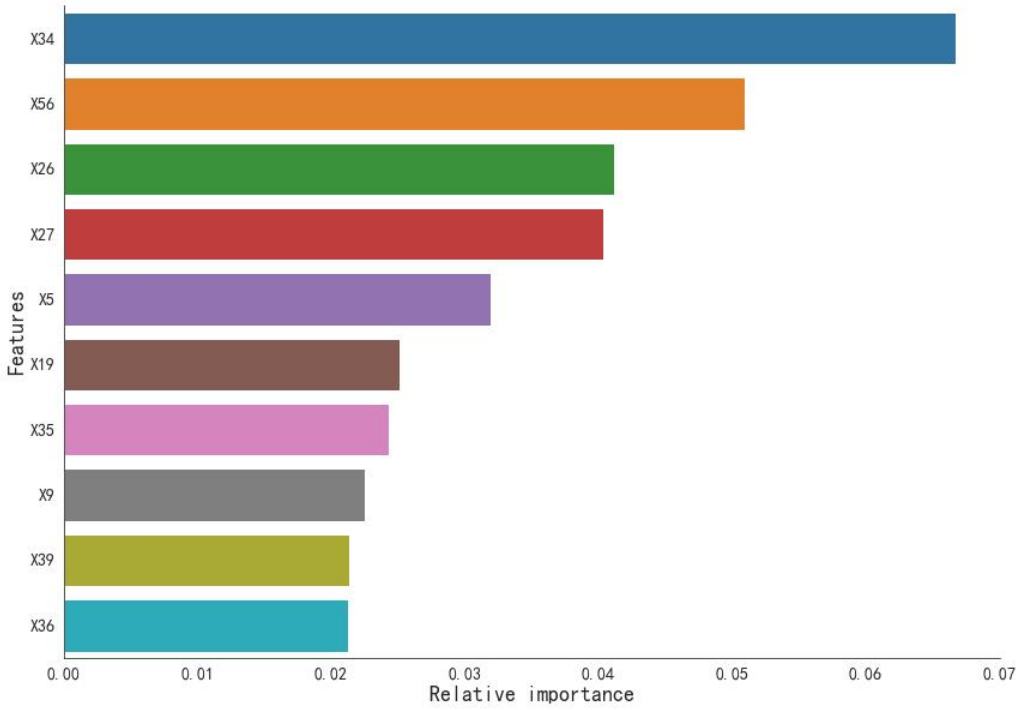

图 5-4 XGBoost 筛选排名前十的特征

从图中可以很清晰看到排名前五的特征分别为 X34,X56，X26，X27 和 X5。通过将其特征分数进行归一化我们为其分配的权重分别为 0.287、0.220、0.178、0.175、0.138。

5.2.3 结果

对于问题二中我们筛选的五项指标，我们认为，这些指标的结果是具备一定经济学意义的，具备一定可解释性。五项指标的具体含义与解释见表 5-3：

表 5-3 五项指标的含义与权重

指标	含义	权重
X34	营业费用/负债总额	0.287
X56	(销售收入-销售产品成本)/销售收入	0.220
X26	(净利润+折旧)/负债总额	0.178
X27	经营活动利润/财务费用	0.175
X5	[(现金+短期证券+应收款-短期负债)/(营业费用-折旧)]*365	0.138

银行效率同样是考察产出与投入之间的关系，反映的是对所有投入产出项目进行综合评价的结果，是银行综合竞争实力的一种体现。Berger 和 Mester（1997）将银行效率分为成本效率、标准利润效率和替代利润效率。有研究发现，利润方面的效率比成本方面的效率更为重要，原因是利润不但考虑了生产过程中的投入，而且还考虑了产出，而成本方面的效率只考虑了生产过程中的投入，没有考虑产出，从而对银行利润方面效率的测度显得更为重要[7]。我们可以看出，所选择的指标大多形容银行的获利能力、管理控制、协调能力和运作效率，从银行的一般性和获利性两个角度来选取相应指标，反映了银行重要的效率特征。

5.3 任务 3 的模型建立与求解

5.3.1 基于 XGBoost 的分类模型的建立

根据任务 1 的结论和任务 2 中筛选的特征，我们需要构建一个倒闭风险的预测问题。而倒闭与不倒闭我们将其抽象为一个二分类问题，并将分类为倒闭的概率视为倒闭风险。这一问题我们将对比多种不同的机器学习模型求解。

决策树是一类基于树结构的监督学习算法，可以进行回归也可以用来分类。在决策树的算法中引入了信息论的方法，用熵来衡量非叶节点的信息量的大小，决策树中的非叶节点表示属性，叶子节点表示样本实例所属类别[8]。通过输入一组带有类别标记的数据，输出一棵二叉或多叉的树，优点是能够直观展示其结构。

随机森林是 Breiman[9]提出的基于树的集成学习算法，根据特征数对每个样本选取分裂指标进而构建单棵子树去完成分类或回归任务。随机森林通过集成多性能较弱的多个基学习器来构建一个强学习器，各个基学习器之间相互互补，降低了方差以及过拟合的风险，从而提高模型的性能。由于随机森林是基于 Bagging 策略[10]的的算法，利用 Bootstrap 采样方法从原始样本中抽取多个样本，对每个样本进行决策树建模，然后综合多棵决策树的结果，通过投票得出最终预测结果，具有很高的预测准确率,对异常值和噪声具有很好的容忍度,其基学习器之间并无直接关联，于是可以很方便地进行并行化计算，且不容易出现过拟合[11]。

由于随机森林基于 Bagging 策略，其基学习器只会抽样式从服从同一分布的随机向量中对数据进行采样并学习基学习器，最终应用投票法将多个基学习器的结果投票进行输出。其学习算法如下：

Step 1：采用 Bootstrap 策略从原始数据集中采样生成 *N* 个子数据集，并在每个子数据集上训练 CART 决策树[12]。

Step 2：随机森林由 i 棵分类树构成，每棵分类树的子节点在进行分裂时随机选择分裂指标数，根据衡量指标大小选择最优分割指标进行划分。

Step 3：重复第二步，使每棵子树都构建完成。

Step 4：将生成子树进行投票获得最终学习器：

$$P_{rf}(X)=arg\max_{Y}\sum_{i=1}^{n}I(w(X,\theta_i)=Y)NY_c \tag{10}$$

由于我们在问题二中已经介绍过 XGBoost 模型的结构，这里不再赘述。

5.3.2 基于 XGBoost 的分类模型的求解

对于分类问题，预测值和真实值之间可以构建混淆矩阵。而基于混淆矩阵可以得到以下一些指标：

准确率即为预测值与真实值相同的样本在样本整体中占比：

$$accuracy=\frac{TP+FN}{TP+TN+FP+FN} \tag{11}$$

在此基础上我们提出了查准率和查全率的概念。查准率为分类正确的正样本数（TP）占预测为正样本的总样本数（$TP+FP$）的比例，其定义为：

$$Precision=\frac{TP}{TP+FP} \tag{12}$$

而查全率为分类正确的正样本数（TP）占实际的正样本总数（$TP+FN$）的比例，

其定义为：

$$\mathrm{Re}call = \frac{TP}{TP + FN} \tag{13}$$

F1 分数即为查准率和查全率的调和平均：

$$F_1 = \frac{2}{\frac{1}{Precision} + \frac{1}{\mathrm{Re}call}} \tag{14}$$

理想时，查准率和查全率二者都越高越好。然而，二者实际是一对矛盾的度量。一个原因是样本存在误差，尤其是临界边缘的数据由于存在误差，导致互相渗透；另一个原因是模型拟合能力有限，非线性模型以超平面划分样本，若扩大召回，可能导致混入更多错误样本，若提升精度，必然召回会下降。例如，若某模型很贪婪，想要预测尽可能多的样本，那么犯错的概率就会随样本数增加而提升；相反，若某模型很保守，会尽量在“更有把握”时才把样本预测为正样本，但会因过于保守而漏掉很多“没有把握”的正样本，导致 Recall 值降低。通常，只有某些简单任务才会二者都高。因此，在不同场合需根据实际需求判断哪个指标更紧要。

AUC 值以 TPR 和 FPR 为横纵坐标绘制曲线：

$$\begin{cases} TPR = \dfrac{TP}{TP + FN} \\ FPR = \dfrac{FP}{FP + TN} \end{cases} \tag{15}$$

曲线的面积即为 AUC 值。

我们构建不同的机器学习模型，通过筛选出的五项指标将预处理样本输入模型进行对比以后，利用 AUC 曲线进行评估得到如图 5-5 所示的 AUC 曲线。

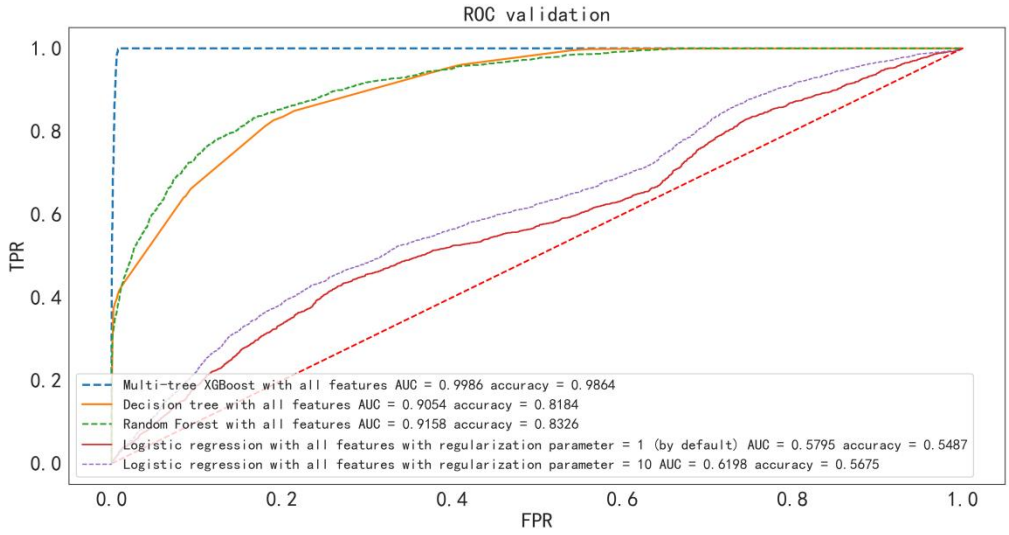

图 5-5　多种模型对比的 ROC 曲线

从图 5-5 中我们可以看到 XGBoost 模型的 AUC 值和准确率最好，分别为 0.9986 和 98.64%，其次分别为决策树、随机森林、逻辑回归（C=1），逻辑回归（C=10）最差，为 0.6198 和 0.5675。

对于这一数据而言，由于本身数据中倒闭数据与非倒闭数据存在样本失衡，所以单纯用 AUC 或准确率描述都并不是最合理的。出于这一考虑，我们也对比了三者的 F1 分数，并将指标对比结果展示在如表 5-4 所示的表格中：

表 5-4　不同算法的指标对比

算法	Accuracy	AUC	F1-score
XGBoost	0.9864	0.9986	0.9185
决策树	0.8184	0.9054	0.8120
随机森林	0.8326	0.9158	0.8245
逻辑回归 (C=1)	0.5487	0.5795	0.5472
逻辑回归 (C=10)	0.5675	0.6198	0.5472

可以看到不同模型 F1 值的大小分布与其 AUC 和准确率的分布有相似性，XGBoost 最好，为 0.9185，随机森林和决策树次之，逻辑回归（C=1）和逻辑回归（C=10）最差，为 0.5472。故而综合来看，XGBoost 依然为最优的模型选择。

5.3.3 结果

对于任务 3，我们通过对比几种不同的分类算法，认为 XGBoost 模型在这一问题上取得了较好的效果。我们认为基于 XGBoost 的分类器模型相比于传统集成学习模型表现更加准确，同时也更加鲁棒。通过约束基学习器的结构和增大基学习器的集成数量可以在保证一定准确性的同时降低过拟合，提高模型的鲁棒性。

我们也将模型的基学习器结构可视化为如图 5-6 的树形：

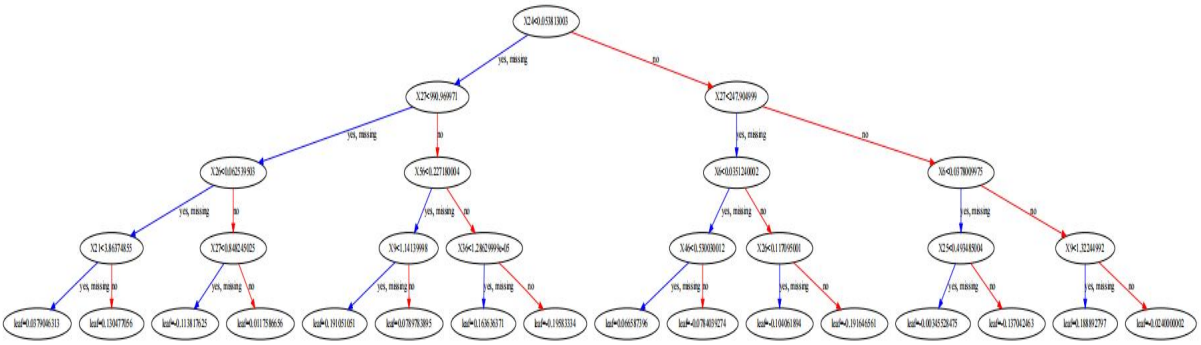

图 5-6　基学习器的结构

从图 5-6 中可以看出对于这一基学习器而言，最核心的特征是 X24，其次 X27、X26 在树中也多次作为节点出现。可以看到基学习器即使是一个比较浅层的模型结构，选取仅仅少量的几个变量同样能够得到相当优秀的表现。而对于 X24 为何会成为这一基学习器的根节点，因为 X24 是毛利三年总资产，资产多少是银行营业的根本，若资产过少则倒闭风险会理所应当地升高，符合我们的认知。

通过构建 XGBoost 分类器模型，我们能够给出样本被分类为倒闭的概率。将此概率作为倒闭风险，经上述方法使用 AUC、F1 分数和准确率多重衡量以后取得了相比其他机器学习模型都要优秀的效果，AUC 最终达到了 0.9986。

5.4 任务 4 的模型建立与求解

5.4.1 基于增量学习的小样本推断模型的建立

根据任务 3 中构建的分类模型我们可以获得每个样本的倒闭概率。我们认为，倒闭概率最高的前 20 位和最低的前 20 位样本是倒闭群体和正常群体中最具代表性的样本，可以以它们作为训练集。

非倒闭样本中最具代表性的 20 个样本编号分别为：698，800，4653，1416，2963，467，1861，345，3486，5397，332，774，5126，207，4285，4231，497，618，525，2507。

倒闭样本中最具代表性的前 20 个样本编号分别为：5870，5789，5621，5793，5765，

5877，5857，5805，5710，5715，5604，5627，5638，5695，5821，5603，5896，5906，5897，5909。

但我们认为，在抽样得到的小批量样本上学习到的模型在总样本上直接进行测试的结果并不鲁棒也并不准确。这是因为小样本的统计特性是对总体的有偏分布，并不能较为精准地推断样本总体分布。将在极小批量样本上学习的模型迁移到大批量样本上的问题被称作小样本推断问题，而对此我们选择采用增量式学习的策略解决。

增量学习是由 Kuncheva[13]于 2001 年提出的概念，其基本思想在于：每当新增数据时，并不需要重建所有的知识库，而是在原有的知识库的基础上，仅对新增数据所引起的变化进行更新。在这一问题中，我们在抽取 40 个样本以后剩余 5890 个样本，我们将其分为 100 批分批学习。

我们对增量学习的小样本推断建模过程如下：

Step 1：初始化：基于抽取的 40 个样本生成初始 XGBoost 模型。

Step 2：分批迭代：对于每一批模型，基于上一步数据和新的数据生成新的 XGBoost 模型，并更新数据集。

Step 3：最终集成：直到数据集遍历完毕，将每一步学习的 XGBoost 模型进行集成作为最终输出。

5.4.2 基于增量学习的小样本推断模型的求解

我们将数据分为 100 批，在每一批数据上进行一次增量学习，实际上是在一边测试的过程中一边利用测试样本扩增训练集对模型进行修正。绘制了模型的对数损失和准确率变化曲线如下：

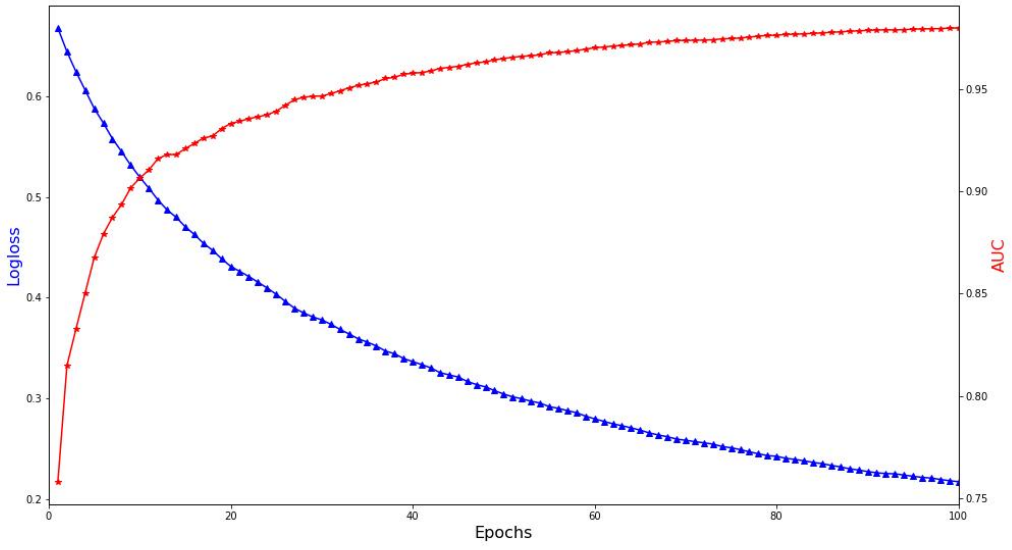

图 5-7　增量学习的基学习器准确率变化和对数损失变化

从图 5-7 中我们可以看到，随着迭代次数的增加，增量学习的基学习器准确率逐渐上升，损失率逐渐下降，并最后趋于稳定。

我们也将最终增量学习的 XGBoost 模型迁移到整体数据集上进行测试，并将指标对比结果展示在如表 5-5 所示的表格中：

表 5-5　增量学习的 XGBoost 测试效果

类别	precision	recall	F1-score	support
未倒闭	0.94	0.87	0.90	5500
倒闭	0.88	0.95	0.91	410
accuracy			0.91	5910
Macro-avg	0.91	0.81	0.91	5910
Micro-avg	0.91	0.91	0.91	5910

从表 5-5 可以看到，模型整体表现仍然较为优秀，准确率和 F1 分数都达到了 0.91。但不同之处在于，相比于任务 2 中构建的特征工程模型和任务 3 中构建的分类器其准确率相比较低，并且整体集成以后的整体表现是不如最后几轮基学习器单体效果的。

5.4.3 结果

对于小样本上的小样本推断任务，我们通过构建增量学习的 XGBoost 模型，较为成功地对 2021 年的全样本进行了测试。

首先，对于整体学习器与基学习器的效果差异，我们认为，尽管增量学习在最后的几轮基学习器都已经达到了 0.98 的 AUC 值，但由于数据在增量学习的过程中逐渐形成有偏分布，单一基学习器的效果并不够鲁棒。而将迭代过程中所有的学习器进行集成能够对模型及时进行修正，通过牺牲一定准确率的方式保证了模型的鲁棒性和可扩展性，结果更具备说服力。

其次，对于增量学习和大规模学习的效果差异，我们认为，小样本上学习的模型本身会与学习样本的采样有关，基于一个小的有偏分布上学习的模型效率必定是只能不断逼近大样本学习。而我们在增量学习的过程中迭代到最后，基学习器已经与大样本学习效果并没有太大区别，只是我们为了保证模型可迁移于是牺牲了一定准确率，实际上产生的准确度差异仍然是在可接受范围内的。

故我们初步认为这一模型取得成功。根据选择的 40 个小样本学习整个数据集的分类能够取得一个较好的效果，F1 分数达到 0.91。

5.5 任务 5 的模型建立与求解

5.5.1 基于 KM 算法的最大权匹配模型的建立

任务 5 需要我们基于 2017~2021 五年不同的数据识别匿名数据下的同一实体。不失一般性，我们先考虑相邻两年数据的同一实体识别。

衡量两个样本是否为同一样本我们考虑用相似度表示，衡量相似度的方法有很多，有基于欧几里得相似度、基于余弦相似度、基于 Jaccard 相似度等[14]，这里我们选择使用基于余弦相似度的算法。余弦相似度定义形如：

$$\cos sim = \frac{x \cdot y}{|x| \cdot |y|} \tag{16}$$

我们基于相邻两年的样本可以算出前一年样本和后一年样本之间的余弦相似度矩阵，规划目标则转化为一个带权二部图的最大权匹配问题，或者视作一个指派问题：

$$\begin{aligned} &\max\ f = D^T W \\ &s.t. \begin{cases} \sum_i D_i = 1 \\ \sum_j D_j \leqslant 1 \\ D_{ij} = 0, 1 \end{cases} \end{aligned} \tag{17}$$

KM 算法是通过给每个顶点一个标号（叫做顶标）来把求最大权匹配的问题转化为求完备匹配的问题的。设顶点X_i的顶标为 A[i]，顶点 Yj 的顶标为 B[j]，顶点 Xi 与 Yj 之间的边权为 w[i, j]。在算法执行过程中的任一时刻，对于任一条边(i, j)，A[i] + B[j] ≥ w[i, j]始终成立[15]。KM 算法的正确性基于以下定理：

若由二分图中所有满足 A[i] + B[j] = w[i, j]的边(i, j)构成的子图（称做相等子图）有完备匹配，那么这个完备匹配就是二分图的最大权匹配。流程图如图 5-8 所示：

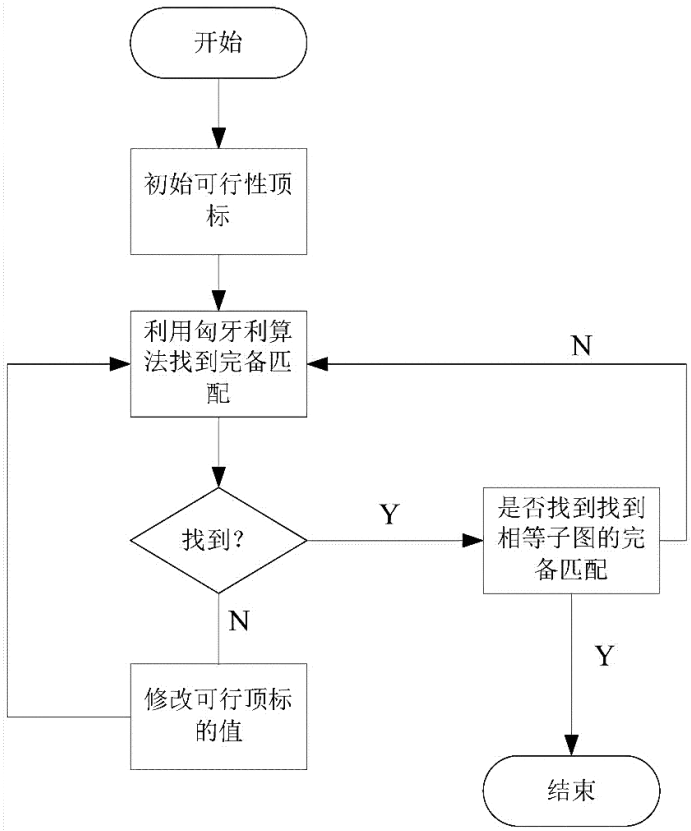

图 5-8　KM 算法的流程

最终，我们将检索到长度为 5 的银行进行筛选，若出现 0~1 交替的序列则我们认为此类样本被匹配错误应该舍弃（因为不可能样本在前一年的分类结果为倒闭但在下一年分类结果为不倒闭）。将样本对应年份的倒闭风险输入以后进行线性回归观察斜率即可。

5.5.2 基于增量学习的小样本推断模型的求解

我们利用余弦相似度求解出 2017~2018、2018~2019、2019~2020、2020~2021 四个年份间样本的余弦相似度矩阵，随后利用 KM 算法得到了 3996 家银行样本，部分样本编号如表 5-6 所示：

表 5-6　KM 算法的部分匹配结果

银行 id	2017-id	2018-id	2019-id	2020-id	2021-id
1	0	1988	5245	826	5672
2	2	1609	6254	866	2816
3	6	221	2247	1311	3452
4	8	5317	6762	5635	2751
5	10	1211	5261	1512	3857

我们对这些样本进行线性回归，共筛选出 2146 家有倒闭趋势的银行。将部分样本的线性拟合结果绘制在图 5-9 中：

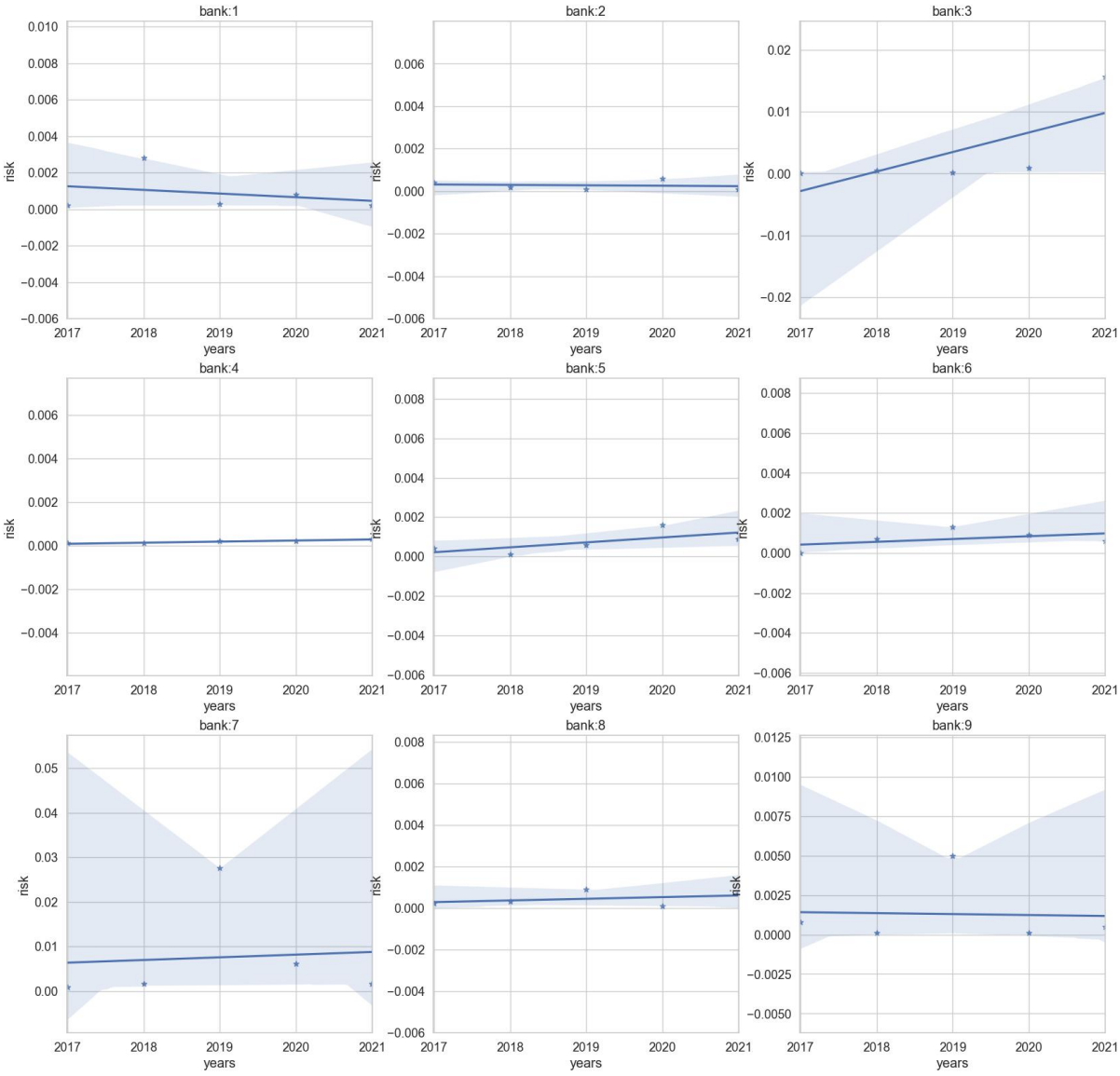

图 5-9　部分样本的线性回归

可以看到，大部分样本线性回归的结果置信区间段较窄，但也有不少样本置信区间段较宽，这说明这一部分样本具有很强的不确定性。而且从随机抽样的样本作图情况来看，也确实是具有倒闭风险的银行在整体的占比更高。不同银行的风险概率随着时间有着不同的变化趋势。斜率大于零说明银行倒闭的风险随时间愈大，越有可能倒闭。尽管在个别年份的银行数据产生了较大程度的偏离，但总体而言保持着一个较为鲁棒的趋势，具有良好的预测效果。

5.5.3 结果

对于不同数据中同一实体的匹配我们将其转化为二部图的最大权匹配问题并利用

KM 算法进行求解，得到完整的匹配结果。而对于匹配后的 3996 家银行数据中，我们通过线性回归的方法筛选出 2146 家具有倒闭趋势的银行。尽管这些银行的预测结果存在一定不确定性，我们根据指标分析仍然认为总体的分析是稳健的。

六、模型的评价及优化

6.1 误差分析

6.1.1 针对于任务 1 的误差分析

任务 1 是一个评价模型，其误差主要集中在基于综合评分指数进行阈值界定的问题上。事实上，以排序后的指数作为划分阈值仍然存在一定的误报率，被区分错误的概率仍然达到 12%。一种更为科学的方法是将 TOPSIS 得分与是否倒闭之间构建逻辑回归模型，通过求方程解的形式确定阈值。

6.1.2 针对于任务 2 的误差分析

任务 2 是一个特征工程问题，由于我们使用了基于 XGBoost 的自动特征工程方法并且训练出的 XGBoost 分类器是较为高性能的（训练准确率高达 100%）并经过交叉验证，我们认为指标筛选的误差非常小。而筛选的 5 项指标具备一定经济学意义的同时在问题三中也发挥了巨大作用，也证明我们的方法误差是非常小的。

6.1.3 针对于任务 3 的误差分析

任务 3 是分类指标问题，分类的 AUC 和准确率上 XGBoost 表现无疑是非常优秀的，已经超过了 0.95。但在 F1 分数的问题上整体表现召回率还是略低。尽管如此，和其他算法的对比也体现了 XGBoost 模型的优势。

6.1.4 针对于任务 4 的误差分析

任务 4 是一个小样本推断问题，这一问题的误差会略大一些。主要误差仍然源于初始数据样本太小，采集样本是有偏的无法反映整体的分布情况。尽管我们在迭代过程中对模型逐步修正和集成已经较为良好地逼近了大样本学习的效果，但与实际的大样本学习还存在一定差距。

6.1.5 针对于任务 5 的误差分析

任务 5 我们通过将同一银行实体的识别问题抽象为二部图的最大权匹配问题，利用 KM 算法进行求解，相比于基于启发式算法的指派问题求解更准确。而主要误差来源存在于线性回归误差，某些点的风险变化本身较小，所以可能会存在被误分为存在倒闭趋势的样本。但整体的平均 R2 分数达到了 0.78，我们认为整体样本都是具有一定线性特征的，使用线性回归的结论仍然具备一定意义。

6.2 模型的优点

经过合理评估分析，我们认为我们的模型具有以下优点：

（1）采用了合适的预处理，如重复采样和均值插补，使结果更合理，避免了由于倒闭银行与未倒闭银行样本数的极大失衡带来的较大误差。

（2）所选取的指标既考虑了其在数据意义上的权重，又具有一定的经济意义，从而使问题描述更加清晰，具有更加明确的应用意义，较好地解决了银行效率及其倒闭原因的分析与预测问题。

（3）所用模型考虑全面，模型简洁高效，便于理解和运用：熵权法和 TOPSIS 的效率评价模型避免了数据方差不齐的问题；采用机器学习的方法，问题解答层层递进，有充分的理论依据；采用基于增量学习的小样本推断模型的方法，实现了小样本数据的合理推广，得到求解结果与实际情况较为接近，结果较为合理。

6.3 模型的缺点

客观而言，我们认为目前的模型仍然存在这样的局限性：

（1）由于初期的样本量过小，尽管使用了小样本推断模型进行增量学习，但是在推广的过程中仍存在着较大的不确定性。

（2）本文的模型中没有考虑不同项指标间的交互作用。

6.4 模型的推广

该模型只基于了五年的样本给出，因而在未来可基于更多的历史数据对原因及变化趋势做出预测。作为国际银行，其是否倒闭不仅与资产本身的收支情况有关，还可能源于国际形势、管理政策等的变化，且存在突发事件因子，因而在未来可以考虑将社会效益因素量化，并将突发事件纳入到模型中，能够很好地识别环境和特殊时间的影响，从而作为银行是否倒闭的重要预测因素。

参考文献

[1] 刘雪梅. 上市公司治理结构与盈余管理关系研究[D].华北电力大学(北京), 2008.

[2] AAAI 2019: National Conference on Artificial Intelligence. 2019.

[3] 谢宏, 程浩忠, 牛东晓. 基于信息熵的粗糙集连续属性离散化算法[J]. 计算机学报, 2005(09): 1570-1574.

[4] Dong G, Liu H. Feature engineering for machine learning and data analytics[M],2018

[5] Chen T , Guestrin C. XGBoost: A Scalable Tree Boosting System[C]// the 22nd ACM SIGKDD International Conference. ACM, 2016.

[6] Friedman J, Hastie T, Tibshirani R. Additive logistic regression: a statistical view of boosting(With discussion and a rejoinder by the authors)[J]. Annals of Statistics, 2000, 28(2): 337-374.

[7] 张冠龄. 中国银行业效率的实证研究及其影响因素的分析[D]. 重庆大学, 2006.

[8] Wu Cong,Li Hongxin, Ren Jiajia. Research on hierarchical clustering method based on partially-ordered Hasse graph[J]. Future Generation Computer Systems,2021(prepublish)

[9] Breiman L. Random forest[J]. Machine Learning, 2001, 45: 5-32.

[10] Quinlan J R. Bagging, Boosting, and C4.5[C]// Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 96, IAAI 96, Portland, Oregon, August 4-8, 1996, Vol. 1. 1996.

[11] 李欣海. 随机森林模型在分类与回归分析中的应用[J]. 应用昆虫学报, 2013, 50(4): 8.

[12] 陈辉林, 夏道勋. 基于 CART 决策树数据挖掘算法的应用研究[J]. 煤炭技术, 2011, 30(10): 3.

[13] R. Polikar, L. Udpa, S. S. Udpa, and V. Honavar, “Learn++: An incremental learning algorithm for supervised neural networks,” IEEE Trans. Syst., Man Cybern. Part C: Appl. Rev., vol. 31, no. 4, pp. 497–508, Nov. 2001.

[14] Lin D. An information-theoretic definition of similarity[J]. Proc. international Conf. on Machine Learning, 1998.

[15] 杨胜超, 张瑞军. 基于二分图最优匹配算法的毕业论文选题系统[J]. 计算机系统应用, 2008(7): 5.

附 录

软件：VSCode+ Python 3.7.6

➢ KM 算法代码

```
import pandas as pd
import numpy as np
from scipy.optimize import linear_sum_assignment
import time
data_2017=pd.read_excel("附件 1.xlsx",sheet_name="2020")
data_2018=pd.read_excel("附件 1.xlsx",sheet_name="2021")
data_2017=data_2017.replace("?",np.nan)
data_2017=data_2017.fillna(data_2017.mean())
data_2018=data_2018.replace("?",np.nan)
data_2018=data_2018.fillna(data_2018.mean())
arr1=np.array(data_2017[['X%d'%i for i in range(1,65)]])
arr2=np.array(data_2018[['X%d'%i for i in range(1,65)]])
def cosine_similarity(x,y):
    num = x.dot(y.T)
    denom = np.linalg.norm(x) * np.linalg.norm(y)
    return num / denom
matsim=cosine_similarity(arr1,arr2)
class KM:
    def __init__(self):
        self.matrix = None
        self.max_weight = 0
        self.row, self.col = 0, 0   # 源数据行列
        self.size = 0    # 方阵大小
        self.lx = None   # 左侧权值
        self.ly = None   # 右侧权值
        self.match = None    # 匹配结果
        self.slack = None    # 边权和顶标最小的差值
        self.visx = None      # 左侧是否加入增广路
        self.visy = None      # 右侧是否加入增广路
    # 调整数据
```

```
        def pad_matrix(self, min):
            if min:
                max = self.matrix.max() + 1
                self.matrix = max-self.matrix

            if self.row > self.col:    # 行大于列，添加列
                self.matrix = np.c_[self.matrix, np.array([[0] * (self.row -
self.col)] * self.row)]
            elif self.col > self.row:   # 列大于行，添加行
                self.matrix = np.r_[self.matrix, np.array([[0] * self.col] *
(self.col - self.row))]
        def reset_slack(self):
            self.slack.fill(self.max_weight + 1)
        def reset_vis(self):
            self.visx.fill(False)
            self.visy.fill(False)
        def find_path(self, x):
            self.visx[x] = True
            for y in range(self.size):
                if self.visy[y]:
                    continue
                tmp_delta = self.lx[x] + self.ly[y] - self.matrix[x][y]
                if tmp_delta == 0:
                    self.visy[y] = True
                    if self.match[y] == -1 or self.find_path(self.match[y]):
                        self.match[y] = x
                        return True
                elif self.slack[y] > tmp_delta:
                    self.slack[y] = tmp_delta
            return False
        def km_cal(self):
            for x in range(self.size):
                self.reset_slack()
                while True:
```

```
                        self.reset_vis()
                        if self.find_path(x):
                            break
                        else:   # update slack
                            delta = self.slack[~self.visy].min()
                            self.lx[self.visx] -= delta
                            self.ly[self.visy] += delta
                            self.slack[~self.visy] -= delta
    def compute(self, datas, min=False):
        """
        :param datas: 权值矩阵
        :param min: 是否取最小组合，默认最大组合
        :return: 输出行对应的结果位置
        """
        self.matrix = np.array(datas) if not isinstance(datas, np.ndarray) else
datas
        self.max_weight = self.matrix.sum()
        self.row, self.col = self.matrix.shape   # 源数据行列
        self.size = max(self.row, self.col)
        self.pad_matrix(min)
        print(self.matrix)
        self.lx = self.matrix.max(1)
        self.ly = np.array([0] * self.size, dtype=int)
        self.match = np.array([-1] * self.size, dtype=int)
        self.slack = np.array([0] * self.size, dtype=int)
        self.visx = np.array([False] * self.size, dtype=bool)
        self.visy = np.array([False] * self.size, dtype=bool)
        self.km_cal()
        match = [i[0] for i in sorted(enumerate(self.match), key=lambda x: x[1])]
        result = []
        for i in range(self.row):
            result.append((i, match[i] if match[i] < self.col else -1))   # 没
有对应的值给-1
        return result
```

```
if __name__ == '__main__':
    a = matsim
    '''
    km = KM()
    min_ = km.compute(a.copy(), True)
    print("最小组合:", min_, a[[i[0] for i in min_], [i[1] for i in min_]])
    max_ = km.compute(a.copy())
    print("最大组合:", max_, a[[i[0] for i in max_], [i[1] for i in max_]])
    '''
    begin=time.time()
    row, col = linear_sum_assignment(a, True)
    print("行坐标:", row, "列坐标:", col, "最大组合:", a[row, col])
    end=time.time()
    print("运行时间：",end-begin)
    pipei={}
    pipei['2020']=row
    pipei['2021']=col
    pipei=pd.DataFrame(pipei)
    pipei.to_csv("匹配-2020-2021.csv")
```

➢ TOPSIS 代码

```
# -*- coding: utf-8 -*-
"""
Created on Sun May   8 04:29:20 2022

@author: Mashituo
"""
import pandas as pd
'''
data_2017=pd.read_excel("附件 1.xlsx",sheet_name="2017")
data_2018=pd.read_excel("附件 1.xlsx",sheet_name="2018")
data_2019=pd.read_excel("附件 1.xlsx",sheet_name="2019")
data_2020=pd.read_excel("附件 1.xlsx",sheet_name="2020")
data_2021=pd.read_excel("附件 1.xlsx",sheet_name="2021")
data=pd.concat([data_2017,data_2018,data_2019,data_2020,data_2021])
```

```
    '''
    data=pd.read_excel("附件 1.xlsx",sheet_name="2021")
    import numpy as np
    data=data.replace("?",np.nan)
    data=data.fillna(data.mean())
    def entropyWeight(data):
          data = np.array(data)
          # 归一化
          P = data / data.sum(axis=0)
          # 计算熵值
          E = np.nansum(-P * np.log(P) / np.log(len(data)), axis=0)
          # 计算权系数
          return (1 - E) / (1 - E).sum()
    def topsis(data, weight=None):
          # 归一化
          data = data / np.sqrt((data ** 2).sum())
          # 最优最劣方案
          Z = pd.DataFrame([data.min(), data.max()], index=['负理想解', '正理想解'])
          # 距离
          weight = entropyWeight(data) if weight is None else np.array(weight)
          Result = data.copy()
          Result['正理想解'] = np.sqrt(((data - Z.loc['正理想解']) ** 2 *
weight).sum(axis=1))
          Result['负理想解'] = np.sqrt(((data - Z.loc['负理想解']) ** 2 *
weight).sum(axis=1))
          # 综合得分指数
          Result['综合得分指数'] = Result['负理想解'] / (Result['负理想解'] + Result['
正理想解'])
          Result['排序'] = Result.rank(ascending=False)['综合得分指数']
          return Result, Z, weight
    #X=data[['X%d'%i for i in range(1,65)]]
    X=data[['X11','X18','X22','X24','X29','X35','X48','X55']]
    Result,Z,weight=topsis(X)
    Result.to_csv("topsis-8.csv")
```