用于财务造假预测的集成模型研究

林昭成 202002020113

摘要—针对上市公司的财务数据造假问题，提出一种基于 GBDT 算法、XGBoost 模型、LightGBM 模型和随机森林算法的二级 Stacking 模型。构建该模型突出了各个算法和模型的优势，并将其紧密结合，构成新的财务造假预测融合模型，用于投资决策和市场监管。将所提出的算法和其他推荐方法进行直接对比，实验结果证明，模型收敛速度较快，且预测模型有较高的精确度和稳定性。基于 Stacking 的集成模型对财务数据有较高的适应度，能够有效解决多特征高维非线性财务造假问题，为财务造假预测建模提供了一种行之有效的方法。

关键字—财务造假；GBDT；随机森林算法；特征选择；Stacking 集成模型

I. 概述

近年来，我国上市公司财务造假事件数见不鲜，如震惊资本市场的雅百特财务造假案、造假数额巨大的康美药业、獐子岛“扇贝出逃”事件等[13]。财务造假事件不仅严重影响了我国资本市场的发展，而且损害了广大投资者的利益[1]。针对上市公司的财务数据造假问题，本文基于树模型的特征选择法，采用 GBDT、LightGBM、XGBoost[16]、基于决策树的 Adaboost 和随机森林[10]的树模型特征选择法来选择特征，统计五种机器学习方法获取的特征[2]。通过选择出现次数大于等于 3 次的特征挑选为主要影响指标，利用 F1-score 和 AUC 指标，并基于 k 折交叉验证和网格搜索对效果较好的基学习器 GBDT 模型、XGBoost 模型和随机森林算法进行超参数调优。在调试好的机器的基础上，建立 Stacking 集成模型，第 1 层基学习器选择 GBDT、XGBoost 和随机森林模型，第 2 层元学习器选择了随机森林模型，从而确定了最优的 Stacking 集成学习分类模型。Stacking 集成模型在测试集上的 F1-score 得分为 0.83，AUC 为 0.92，该模型在各项评估指标上均有提升，且拥有更好的模型泛化能力，能够广泛应用于市场财务造假情况的研究[3]。集成模型的代码可以在 Github 上找到（Lin-Johnson/Paper (github.com)）。

II. 问题背景

2.1 研究背景

随着中国经济的快速发展，不同规模、不同行业的上市公司数量不断增加。截至目前，上市公司数量已超过 4400 家。同时监管机构对上市公司的监督管理日趋严格，在被处罚的违法违规案件中，财务造假案件数量和占比均呈上升趋势。

尽管各交易所和相关监管部门已建立了上市公司内幕交易、市场操纵和信息披露方面的监管体系，但由于信息不对称，隐蔽的上市公司财务数据造假及爆雷很难被预防和预测，对于出现严重财务数据造假、丧失持续经营能力的上市公司，强制退市是唯一的选项。然而上市公司的退市有可能产生多米诺骨牌效应，严重损害投资者利益[4]。屡见不鲜的财务造假提醒相关部门需要对上市公司进行有效的监督和管理，并选择适当的数据特征指标对其财务数据报告进行跟踪、分析和研究，以避免投资暴雷。

2.2 文献综述

洪文洲[5]等选取 2004-2013 年间财务报表造假的 44 家上市企业作为样本，构建了包含 27 个指标的 Logistic 回归模型，研究发现可以帮助识别财务报告造假的 11 个显著性指标；Mukherjee[6]等通过 Logistic 回归模型、朴素贝叶斯算法和随机森林、决策树两类非参数化方法建立财务造假预测模型，发现随机森林与决策树分类器性能最佳；王圣洁[7]以 A 股市场上市公司作为研究样本，结合监管动态更新指标体系选取 XGBoost 算法，并针对识别领域的实际需求优化模型参数，构建性能更优异的识别模型；李爱华[4]等选择决策树和四种基于树的集成学习分类模型，构建基于数据融合的财务造假异常检测研究框架，得出各种数据类型的特征对财务造假异常识别的贡献。

此外，财务造假的预测模型通常需要在已有的样本数据集上进行学习后，对新样本进行预测和分析，因此现有的预测模型多采用有监督的机器学习模型，如 Logistic 回归、决策树分类器、朴素贝叶斯分类器、神经网络等。随着

Z.Lin Studied in Shenzhen University of Science and Technology, China
(e-mail: JohnsonLin0216@gmail.com)

研究的深度和算法的迭代，学者们开始使用识别效果和预测效果更好的集成算法，如随机森林、GBDT 等。而 XGBoost、LightGBM 等较新颖的集成算法也逐渐应用到各个科研分析领域。

2.3 研究设想

现有研究具有以下不足：①模型选择和预测模型的指标选取过度依赖过往文献，没有针对现有的社会环境态势建立新的指标选择模型；②研究选取的数据集时间维度较长，难以针对现今社会情况做出准确预测；③选取的预测指标多为低维线性数据，不能发挥机器学习算法处理高阶数据的优越性，以这些指标构建的指标体系的可解释性不强[7]；④当前研究多使用单一机器学习模型或多种机器学习组合模型，容易放大单一模型造成的预测结果误差，在预测效率上具有一定的缺失。

鉴于此，本文以近年来 19 个行业中 4153 家上市公司的财务数据作为研究样本，根据 363 个财务指标自身的经济学意义和数理统计方法，选择各行业与财务造假相关的数据指标，以完善财务造假的识别体系。在算法使用上，本文选取基于 GBDT 模型、XGBoost 模型和随机森林算法的二级 Stacking 模型，针对该领域的实际需求优化模型参数，试图构建性能更为优异的财务造假预测模型。

III. 方法

3.1 财务造假指标选取

本文采用 GBDT、LightGBM、XGBoost、基于决策树的 Adaboost 和随机森林的树模型特征选择法对财务指标（见 TABLE I，详见附件 1）选择特征，对五种机器学习方法获取的特征进行统计，基于投票机制筛选数据指标。

TABLE I. 财务指标（部分）

指标名称	指标含义及解释	指标名称	指标定义及解释
TICKER_SYMBOL	股票代码	REPORT_TYPE	报告类型
ACT_PUBTIME	实际披露时间	FISCAL_PERIOD	会计区间
CASH_C_EQUIV	货币资金	TRADING_FA	交易性金融资产
SETT_PROV	结算备付金	NOTES_RECEIV	应收票据
LOAN_TO_OTH_BANK_FI	拆出资金	AR	应收账款

上市公司财务数据中，交通运输、仓储和邮政业等 4 个行业无财务造假数据，卫生和社会工作、水利、环境和公共设施管理业等 9 个行业财务造假数据量较少，不足以筛选出和确定这些行业财务数据造假的有效指标，因此这 13 个行业的财务数据指标不做财务造假相关分析。最后选择制造业财务数据作为数据样本范例，建立数据筛选模型和财务造假预测模型。

3.2 数据预处理

根据统计，在 363 个财务指标中，包含 10 个上市公司身份特征指标和 353 个上市公司财务特征指标，制造业财务数据分布情况和缺失程度如 Figure I所示。

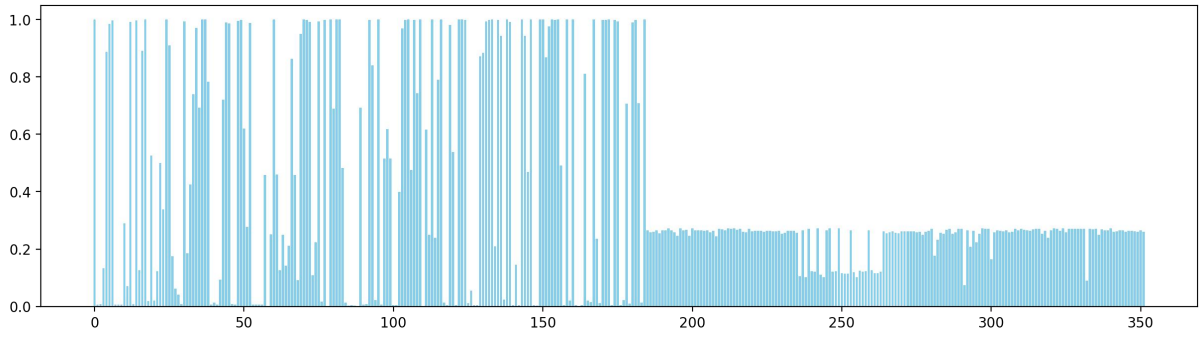

Figure I. 制造业财务数据保有率示意图

首先剔除与财务造假无关，且无实际意义的一些身份特征指标，如 TABLE II所示。

TABLE II. 剔除特征示意表

指标名称	指标含义及解释	指标名称	指标定义及解释
TICKER_SYMBOL	股票代码	REPORT_TYPE	报告类型
ACT_PUBTIME	实际披露时间	FISCAL_PERIOD	会计区间
PUBLISH_DATE	发布时间	MERGED_FLAG	合并标志：1-合并，2-母公司
END_DATE_REP	报告截止日期	ACCOUTING_STANDARDS	会计准则
END_DATE	截止日期	CURRENCY_CD	货币代码

剔除无关数据后，需要对数据集的缺失值进行处理。现行缺失值处理方法大致划分为三类，即缺失值插补，删除缺失记录和保留缺失值为有效值。我们在进行数据处理前需要问自己一个问题，为什么这些数据缺失了，缺失数据的原因具有其自身的含义。

① 保留缺失值为有效值

当你要处理的缺失数据不是 MAR 的分类变量时，最好的方法是保留该缺失值，因为你很有可能在破坏变量。有时候把缺失值当作为分类变量的另一个取值是有意义的。一个很有意思的故事，有一个为一家大型连锁超市研究客户流失的项目，客户流失的最佳预测指标是他在注册会员卡时是否填写了他的电话号码。

② 缺失值插补

缺失值处理最常用的方法是插补缺失值，这个方法非常简单和快速，但同样存在一些弊端。如果我们希望建立一个好的模型去插补缺失值，那无疑是十分浪费时间的，容易将大量的时间花费在模型建立和参数调整上，这不是我们工作的重点。预测同时，缺失值插补在某种意义上人为干扰了数据变量，极大程度增加了解释模型的难度。

③ 删除缺失记录

缺失值处理最常用的方法是插补缺失值，这个方法非常简单和快速，但同样存在一些弊端。如果我们希望建立一个好的模型去插补缺失值，那无疑是十分浪费时间的，容易将大量的时间花费在模型建立和参数调整上，这不是我们工作的重点。预测同时，缺失值插补在某种意义上人为干扰了数据变量，极大程度增加了解释模型的难度。

我们分析该数据集数据缺失的原因，很清楚的知道数据很少是 MAR，更多原因是由于信息无法获取或某些数据的信息获取代价太大。我们认为，缺失值大于 75%的特征指标属于严重缺失，该类指标已经没有应用意义，所以我们对该类特征选择删除。缺失率在 35%到 75%的特征指标也属于较大缺失，因为我们会采用高级的机器学习模型进行财务造假预测，所以填充过多的缺失值容易对模型带来噪音从而影响模型的训练过程，干扰到模型的准确性，因此使用 0 值对缺失值进行填充。缺失率在 35%以下的特征指标选择 KNN 算法进行填补[8]，缺失值处理的数学模型如下所示：

$$\begin{cases} 75\% < \Gamma \le 100\% \text{，舍弃该特征} \\ 35\% < \Gamma \le 75\% \text{，用 0 值填充} \\ 0\% < \Gamma \le 40\% \text{，用 KNN 算法填补} \end{cases} \tag{3.1}$$

3.3 机器学习模型介绍

① K 最近邻分类算法

K-Neares Neighbor（K 最近邻, KNN）分类算法是应用最为广泛的分类算法。这是一种监督学习的分类算法，基本思想为根据距离函数计算待分类样本 x 和每个训练样本之间的距离，选择与待分类样本 x 距离最近的 K 个样本作为 x 的 K 个最近邻，根据这 K 个最近邻的类别从而计算和判断 x 的类别[9]。本文使用 KNNImputer 处理连续变量的缺失值，KNNImputer 通过欧几里德距离矩阵寻找 K 最近邻，帮助估算数据中出现的缺失值。

② 随机森林

随机森林（Random Forest, RF）是一种以决策树为基学习器的 Bagging 算法，传统决策树在选择划分属性时，每次选择一个最优属性，但是随机森林在决策树的训练过程中引入了随机属性选择，对异常值和噪声具有很高的容忍度，且不容易出现过拟合[11]。随机森林的最终分类结果采取投票法，最终分类决策为：

$$H(x) = arg\max_{Y} \sum_{i=1}^{k} I(h_i(x) = Y) \tag{3.2}$$

其中，h_i是单决策树分类模型，Y表示输出变量，$I(\bullet)$为线性函数。

③ 梯度提升决策树

梯度提升决策树（Gradient Boosting Decision Tree, GBDT）是一种典型的 Boosting 集成学习算法，它内部集成多棵 CART 回归树，并把多棵决策树不断累加和优化所得的结果作为最终的预测输出[12]。Boosting 集成算法的核心思想在于通过对前一棵决策树的残差进行学习进而不断降低损失函数，使模型变得越来越精确[12]。使用该模型能有效避免单决策树的过拟合问题。

④ XGBoost

XGBoost（eXtreme Gradient Boosting）是提升树的一种实现，它对损失函数进行了二阶泰勒展开，同时用到了一阶和二阶导数，并引入了适用于树模型的正则项用于控制模型的复杂度[16]。XGBoost 的正则项里包含了树的叶子节点个数、每个叶子节点输出分数的 L_2 平方和[14]。假设第 t 次迭代要训练的树模型是 $f_t(x_i)$，则有：

$$\hat{y}_i^{(t)} = \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i) \qquad (3.3)$$

⑤ LightGBM

轻量梯度提升机（Light Gradient Boosting Machine Method, LightGBM）是由 Ke 等人 2017 年提出的针对 GBDT 的一种改进算法。当特征维度很高或数据量很大时，GBDT 的有效性和可拓展性受限。而 LightGBM 以 GBDT 为核心，通过多方面的优化使得整个算法具有以下优势：

a) 更快的训练效率；
b) 低内存使用；
c) 更高的准确率；
d) 支持并行化学习。

3.4 模型融合

针对上市公司的财务数据造假问题，本文基于树模型的特征选择法，采用 GBDT、LightGBM、XGBoost、基于决策树的 Adaboost 和随机森林的树模型特征选择法来选择特征。在调试好的机器的基础上，建立 Stacking 集成模型，第 1 层基学习器选择 GBDT、XGBoost 和随机森林模型，第 2 层元学习器选择了随机森林模型，如图 2 所示。从而确定了一个用于财务造假预测的 Stacking 集成学习预测模型。

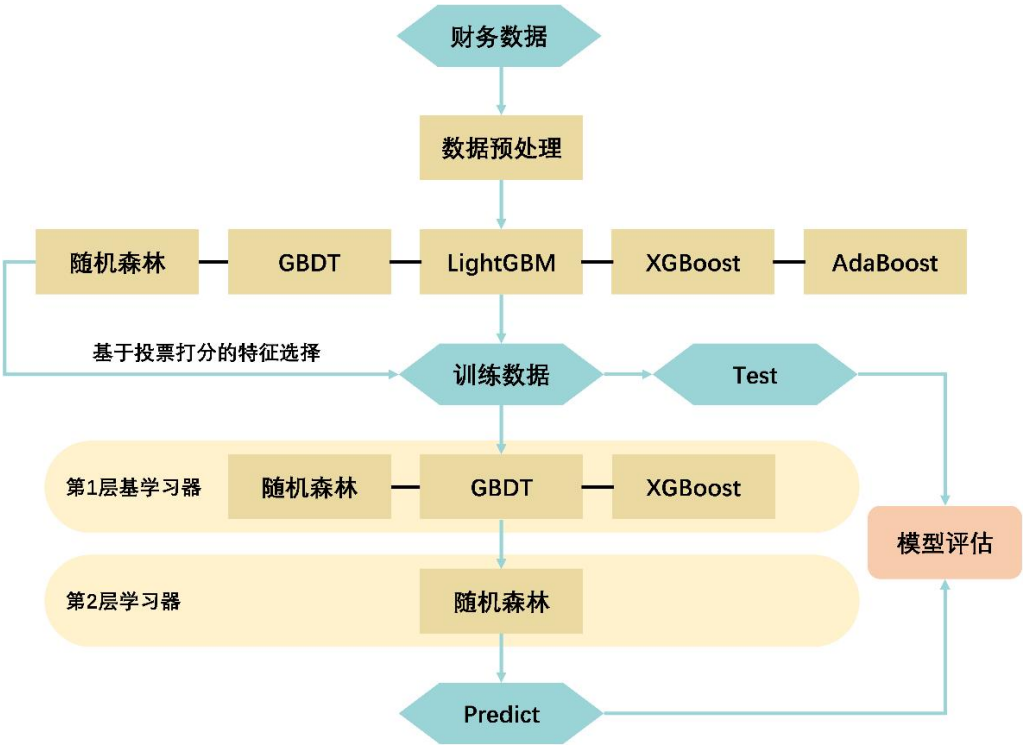

Figure II. 用于财务造假预测的 Stacking 集成学习预测模型

IV. 实验与结果分析

实验选择仿真平台为：Intel 4 核 CPU 1.90GHz，8GB RAM，Windows 10 操作系统，采用 Python 版本为 3.7.0，运行环境为 Jupyter 4.4.0。

4.1 指标选取模型实验结果

本文采用 GBDT、LightGBM、XGBoost、基于决策树的 Adaboost 和随机森林的树模型来选择特征，结果发现这 6 个模型选择的前 30 个指标具有一定的重合度。在此基础上，挑选出 6 个模型共同确定的特征指标如下图所示。

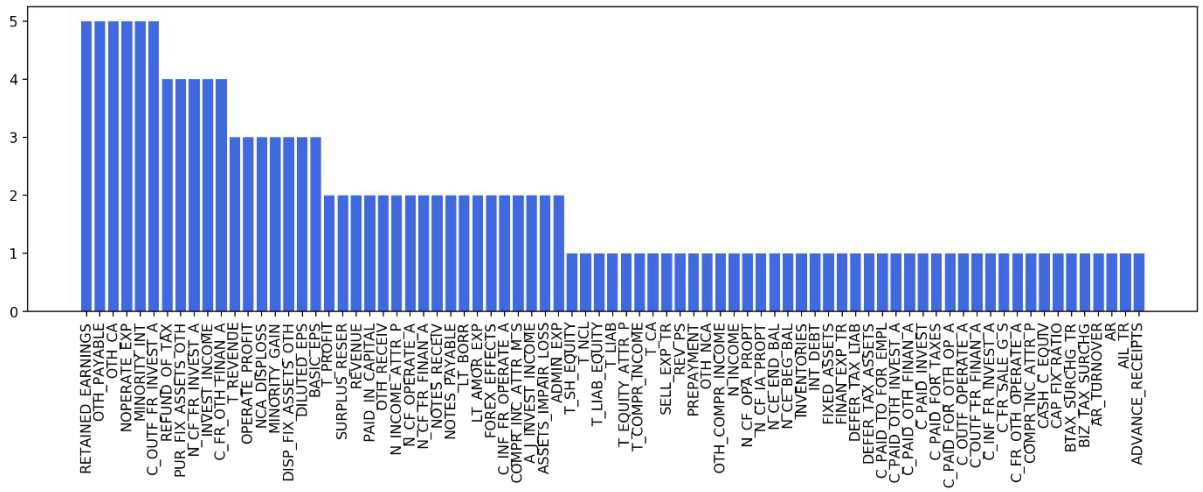

Figure III. 特征指标出现次数

统计所有机器学习算法获取的特征，选择出现次数大于等于 3 次的特征作为最终的指标，共 18 个。

TABLE III. 特征指标选择示意表

指标名称	指标含义及解释	指标名称	指标定义及解释
RETAINED_EARNINGS	未分配利润	INVEST_INCOME	对联营企业和合营企业的投资收益
OTH_PAYABLE	其他应付款	C_FR_OTH_FINAN_A	收到其他与筹资活动有关的现金
OTH_CA	其他流动资产	T_REVENUE	营业总收入
NOPERATE_EXP	营业外支出	OPERATE_PROFIT	营业利润(亏损以“－”号填列)
MINORITY_INT	少数股东权益	NCA_DISPLOSS	非流动资产处置损失
C_OUTF_FR_INVEST_A	投资活动现金流出小计	MINORITY_GAIN	少数股东损益
REFUND_OF_TAX	收到的税费返还	DILUTED_EPS	稀释每股收益
PUR_FIX_ASSETS_OTH	购建固定资产、无形资产和其他长期资产支付的现金	DISP_FIX_ASSETS_OTH	处置固定资产、无形资产和其他长期资产收回的现金净额
N_CF_FR_INVEST_A	投资活动产生的现金流量净额	BASIC_EPS	基本每股收益

4.2 造假预测模型实验结果

根据上文筛选出的数据特征指标，使用 F1-score 和 AUC 作为评价指标，对 Logistic 回归、支持向量机、随机森林、GBDT 和 XGBoost 共 5 种机器学习算法进行训练，结果如下表所示。

TABLE IV. 机器学习模型初步训练结果

机器模型	F1-score	AUC
逻辑回归模型	0.17	0.54
支持向量机模型	0.17	0.54
随机森林模型	0.22	0.99

GBDT 模型	0.35	0.98
XGBoost 模型	0.58	0.99

从上表可以看出，逻辑回归模型和支持向量机模型效果较差，原因是财务数据不具有十分明显的线性关系，这两个模型难以作为最优算法预测，故不做考虑。

同时发现，剩下的 3 个模型的 AUC 都很高，但是 F1-score 的水平不高。这是由于 AUC 希望训练一个尽量不误报的模型，也就是知识外推的时候倾向保守估计，而 F1-score 希望训练一个不放过任何可能的模型，即知识外推的时候倾向激进，这就是这两个指标的核心区别。F1-score 认为召回率和精确率同等重要，在造假预测模型中模型应更注重于对财务造假行为的识别而并非更高的 AUC，本着宁抓错不放过的根本原则，所以优先选择 F1-score 作为评价指标。

根据模型特点分别对 3 个机器学习模型设定其调参范围，调用函数进行网格搜索和交叉验证寻找最优结果参数，各模型调参结果如下表所示。

TABLE V. 随机森林模型调参结果

参数名称	参数含义及解释	调参结果	调参后 F1-score	调参后 AUC
n_estimators	决策树数目	600	0.54	0.99
max_depth	决策树最大深度	5		
min_samples_split	分裂所需最小样本数	2		
min_samples_leaf	叶节点最少承载样本数	1		
max_features	寻找最佳分裂点时考虑的特征数目	‘auto’		
bootstrap	是否考虑放回样本	False		

TABLE VI. GBDT 模型调参结果

参数名称	参数含义及解释	调参结果	调参后 F1-score	调参后 AUC
n_estimators	决策树数目	400	0.67	0.97
max_depth	决策树最大深度	8		
min_samples_split	分裂所需最小样本数	35		
min_samples_leaf	叶节点最少承载样本数	35		
learning_rate	学习率	0.05		
subsample	子采样率	0.5		

TABLE VII. XGBOOST 模型调参结果

参数名称	参数含义及解释	调参结果	调参后 F1-score	调参后 AUC
n_estimators	决策树数目	600	0.68	0.98
max_depth	决策树最大深度	5		
learning_rate	学习率	0.5		
subsample	子采样率	0.2		

4.3 模型融合实验结果

在集成模型中，第 1 层学习器使用随机森林模型、GBDT 模型和 XGBoost 模型。本文采用 3 折交叉验证来划分训练集的数据，从而训练出第 2 层所需数据。第 2 层使用随机森林模型进行训练，并计算集成模型的 F1-score 和 AUC，如 TABLE VIII所示。

TABLE VIII. 集成模型预测结果

第 1 层	第 2 层	F1-score	AUC
随机森林模型 GBDT 模型 XGBoost 模型	随机森林模型	0.83	0.92

发现 Stacking 集成模型具有更高的 F1-score 评分并且保持了良好的 AUC，说明模型预测结果良好。

V. 结论

本文基于上市公司财务造假数据，按照构建的模型提前相关特征指标，并选取适合的预处理方法完成对数据的预处理。针对财务造假规律建立基于 Stacking 算法的财务造假预测模型，实现对可能存在的财务造假行为预测。实验表明，财务造假预测模型在制造业财务造假数据上表现良好，在测试集上的 F1-score 得分为 0.83，AUC 为 0.92，且拥有更高的模型泛化能力，便于扩充为其他行业财务造假行为进行预测。该模型可以及时预测和评估财务数据的主要影响指标和造假情况，为相关部门对财务数据的监督和管理提供服务，防止财务造假事情的产生。

VI. 参考文献

[1] 陈振.上市公司财务造假分析——以浪奇为例[J].老字号品牌营销,2022(17):114-116.

[2] 杨宇超. 基于树集成模型的共享住宿日租价格预测[D].东北财经大学,2022.

[3] 周芃,王勇.基于集成学习的用户信用卡违约预测模型研究[J].井冈山大学学报(自然科学版),2022,43(04):51-56.

[4] 李爱华,王迪文,续维佳,李子沫,姚思涵.基于多数据源融合的创业板上市公司财务造假异常检测[J/OL].数据分析与知识发现:1-18[2022-11-14].

[5] 洪文洲,王旭霞,冯海旗.基于 Logistic 回归模型的上市公司财务报告舞弊识别研究[J].中国管理科学,2014,22(S1):351-356.

[6] Upasana Mukherjee,Vandana Thakkar,Shawni Dutta,Utsab Mukherjee,Samir Kumar Bandyopadhyay. Emerging Approach for Detection of Financial Frauds Using Machine Learning[J]. Asian Journal of Research in Computer Science,2021.

[7] 王圣洁.基于 XGBoost 算法的财务造假识别研究[J].山东理工大学学报(自然科学版),2022,36(06):74-79.

[8] 王婧怡,陈胤佳,袁野,陈辰,王国仁.面向 K-近邻学习模型的高效数据清洗框架[J/OL].计算机科学与探索:1-15[2022-12-04].

[9] 闭小梅,闭瑞华.KNN 算法综述[J].科技创新导报,2009(14):31.

[10] 姚登举,杨静,詹晓娟.基于随机森林的特征选择算法[J].吉林大学学报(工学版),2014,44(01):137-141.

[11] 方匡南,吴见彬,朱建平,谢邦昌.随机森林方法研究综述[J].统计与信息论坛,2011,26(03):32-38.

[12] 刘云菁,伍彬,张敏.上市公司财务舞弊识别模型设计及其应用研究——基于新兴机器学习算法[J].数量经济技术经济研究,2022,39(07):152-175.

[13] 刘菁.科研项目预算绩效管理之思考[J].中国农业会计,2021(06):60-62.

[14] 陈天光,陈典红,吕瑞峰.基于集成学习算法的轴承故障诊断方法研究[J].科技通报,2021,37(04):57-61+93.

[15] 蔡景波,蔡志杰.基于深度学习模型预测财务造假的上市公司[J].数学建模及其应用,2021,10(03):54-59.

[16] 李占山,刘兆赓.基于 XGBoost 的特征选择算法[J].通信学报,2019,40(10):101-108.

VII. 附录

附件 1：上市公司财务数据指标明细

Paper/Data_Mining/Data at main · Lin-Johnson/Paper (github.com)

附件 2：实验代码

Paper/Data_Mining at main · Lin-Johnson/Paper (github.com)

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import xgboost as xgb
import operator
import time
import warnings
warnings.filterwarnings("ignore")

from sklearn.impute import KNNImputer
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier, AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from xgboost.sklearn import XGBClassifier
from sklearn import metrics
from sklearn.metrics import mean_squared_error, accuracy_score, recall_score, precision_score, f1_score, roc_curve
from sklearn import preprocessing

start_time = time.time()
data = pd.read_excel('data/paper_data.xlsx')
print('---------- Reading file use %.2fs ----------' % (time.time()-start_time))

# Delete features with identical values
data.drop(['END_DATE', 'REPORT_TYPE', 'FISCAL_PERIOD',
           'MERGED_FLAG', 'ACCOUTING_STANDARDS', 'CURRENCY_CD'], axis='columns', inplace=True)
# Delete features that have no practical significance
data.drop(['TICKER_SYMBOL', 'ACT_PUBTIME', 'PUBLISH_DATE', 'END_DATE_REP'], axis='columns', inplace=True)
data["FLAG"] = data["FLAG"].astype(int)
data.info()
```

```
# Data visualization
def missing_visualization(data):
    missing_value = []
    retention_value = []
    missing_light = []
    missing_moderate = []
    missing_severe = []
    num_col = data.shape[0]
    for column in data.columns:
        num_null = data[column].isnull().sum(axis=0)
        missing_rate = num_null/num_col
        retention_rate = 1 - missing_rate
        if missing_rate < 0.35 or num_null == 0:
            missing_light.append(column)
        elif missing_rate < 0.75:
            missing_moderate.append(column)
        else:
            missing_severe.append(column)
        missing_value.append(missing_rate)
        retention_value.append(retention_rate)

    missing_value = missing_value[:-1]
    retention_value = retention_value[:-1]
```

```
    index = np.arange(data.shape[1] - 1)
    plt.figure(figsize=(15, 4), dpi=200)
    plt.bar(index, retention_value, color='skyblue')
    # plt.bar(index, missing_value, color='darkred')

    return missing_light, missing_moderate, missing_severe

missing_light, missing_moderate, missing_severe = missing_visualization(data)
```

```
# Optimization of KNN algorithm parameters using random forests
rmse = lambda y, yhat: np.sqrt(mean_squared_error(y, yhat))

def optimize_knn(data, target):
    errors = []
    for k in range(1, 20, 2):
        imputer = KNNImputer(n_neighbors=k)
        imputed = imputer.fit_transform(data)
        df_imputed = pd.DataFrame(imputed, columns=data.columns)

        x = df_imputed.drop(target, axis=1)
        y = df_imputed[target]
        x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

        model = RandomForestClassifier()
        model.fit(x_train, y_train)
        y_pred = model.predict(x_test)
        auc = accuracy_score(y_test, y_pred)
        errors.append({'K': k, 'Auccuary': auc})
        # recall = recall_score(y_test, y_pred)
        # precision = precision_score(y_test, y_pred)
        # errors.append({'K': k, 'Auccuary': auc, 'Recall': recall, 'Percision': precision})

    return errors
```

```
# Preprocess data
start_time = time.time()

# Delete features with a missing rate greater than 0.75
for column in missing_severe:
    if column in data.columns:
        data.drop(column, axis='columns', inplace=True)
print('---------- End of phase I processing ----------')

# The missing value of features with a missing rate greater than 0.35 but less than 0.75 shall be supplemented with 0
for column in missing_moderate:
    if column in data.columns:
        data[column].fillna(0, inplace = True)
print('---------- End of phase  II  processing ----------')

# Use KNN to supplement features with missing rate less than 0.35
# Automatically adjust KNN parameters
# k_errors = optimize_knn(data, 'FLAG')
# k_errors = sorted(k_errors, key=operator.itemgetter('Auccuary'))
# k_best = k_errors[0]['K']
# imputer = KNNImputer(n_neighbors=k_best)
# imputed = imputer.fit_transform(data)
# data = pd.DataFrame(imputed, columns=data.columns)

imputer = KNNImputer(n_neighbors=15)
imputed = imputer.fit_transform(data)
data = pd.DataFrame(imputed, columns=data.columns)
print('---------- End of phase III processing ----------')
```

```
print('---------- Preprocessing data %.2fs ----------' % (time.time()-start_time))

data.info()
```

```
# Feature selection
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
import lightgbm as lgb

start_time = time.time()

x, y = data.iloc[:, :-1].values, data.iloc[:, -1].values
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3, random_state = 42)
feat_labels = data.columns[:-1]
```

```
# Random Forest
rfc_model = RandomForestClassifier(n_estimators=1000, random_state=42, n_jobs=-1)
rfc_model.fit(x, y)

rfc_imp = rfc_model.feature_importances_
indices = np.argsort(rfc_imp)[::-1]
rfc_imp_30 = []
for f in range(x.shape[1]):
    print("%2d) %-*s %f" % (f + 1, 30, feat_labels[indices[f]], rfc_imp[indices[f]]))
    if f < 30:
        rfc_imp_30.append(feat_labels[indices[f]])
```

```
# Gradient Boosting Decision Tree
gbdt_model = GradientBoostingClassifier(n_estimators=30, learning_rate=0.5, max_depth=10, min_samples_leaf=50,
                                        min_samples_split=40, subsample=0.4, random_state=42)
gbdt_model.fit(x, y)

gbdt_imp = gbdt_model.feature_importances_
indices = np.argsort(gbdt_imp)[::-1]
gbdt_imp_30 = []
for f in range(x.shape[1]):
    print("%2d) %-*s %f" % (f + 1, 30, feat_labels[indices[f]], gbdt_imp[indices[f]]))
    if f < 30:
        gbdt_imp_30.append(feat_labels[indices[f]])
```

```
# Light Gradient Boosting Machine
lgb_params = {
    'boosting_type': 'gbdt',
    'objective': 'binary',
    'num_leaves': 30,
    'num_round': 360,
    'max_depth':8,
    'learning_rate': 0.01,
    'feature_fraction': 0.5,
    'bagging_fraction': 0.8,
    'bagging_freq': 12,
    'verbose': -1
}
target = 'FLAG'
lgb_train = lgb.Dataset(x, y)
lgb_model = lgb.train(lgb_params, lgb_train)

lgb_imp = lgb_model.feature_importance()
indices = np.argsort(lgb_imp)[::-1]
```

```
lgb_imp_30 = []
for f in range(x_train.shape[1]):
    print("%2d) %-*s %f" % (f + 1, 30, feat_labels[indices[f]], lgb_imp[indices[f]]))
    if f < 30:
        lgb_imp_30.append(feat_labels[indices[f]])
```

```
# eXtreme Gradient Boosting
xgb_model = XGBClassifier(max_depth=6, learning_rate=0.8, n_estimators=50, subsample=0.8, min_child_weight=6,
random_state=42)
xgb_model.fit(x, y)

xgb_imp = xgb_model.feature_importances_
indices = np.argsort(xgb_imp)[::-1]
xgb_imp_30 = []
for f in range(x.shape[1]):
    print("%2d) %-*s %f" % (f + 1, 30, feat_labels[indices[f]], xgb_imp[indices[f]]))
    if f < 30:
        xgb_imp_30.append(feat_labels[indices[f]])
```

```
# Adaptive Boosting
bdt_model = AdaBoostClassifier(DecisionTreeClassifier(max_depth=2, min_samples_split=20, min_samples_leaf=5),
                               algorithm="SAMME", n_estimators=200, learning_rate=0.8)
bdt_model.fit(x, y)

bdt_imp = bdt_model.feature_importances_
indices = np.argsort(bdt_imp)[::-1]
bdt_imp_30 = []
for f in range(x.shape[1]):
    print("%2d) %-*s %f" % (f + 1, 30, feat_labels[indices[f]], bdt_imp[indices[f]]))
    if f < 30:
        bdt_imp_30.append(feat_labels[indices[f]])
```

```
# Select the top 30 features of each algorithm for voting and scoring
imp_30 = np.array([rfc_imp_30, gbdt_imp_30, lgb_imp_30, xgb_imp_30, bdt_imp_30])
imp_feature, imp_score = np.unique(imp_30, return_counts=True)
Z = zip(imp_score, imp_feature)
Z = sorted(Z, reverse=True)
imp_score, imp_feature = zip(*Z)

plt.figure(figsize=(15, 4), dpi=200)
plt.xticks(rotation=90)
plt.bar(imp_feature, imp_score, color='royalblue')

sel_feature = []
for i in range(0, len(imp_feature)):
    if imp_score[i] >= 3:
        sel_feature.append(imp_feature[i])
print(sel_feature)

sel_feature.append('FLAG')
data = data[sel_feature]

print('Select %d features' % len(sel_feature))
print('---------- Feature selection use %.2fs ----------' % (time.time()-start_time))
```

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier

x, y = data.iloc[:, :-1].values, data.iloc[:, -1].values
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 42)

def grid_display(grid):
    means = grid.cv_results_['mean_test_score']
    params = grid.cv_results_['params']
    for mean, param in zip(means,params):
        print("%f  with:   %r" % (mean,param))
```

```
# Random Forest
param = {'n_estimators':[int(x) for x in np.linspace(start = 200,stop = 800, num = 4)],
         'max_depth':[5, 10, 15],
         'min_samples_split':[1, 2, 5],
         'min_samples_leaf':[1, 2, 5],
         'max_features':['auto', 'sqrt'],
         'bootstrap':[True, False]}
grid = GridSearchCV(RandomForestClassifier(criterion='gini', max_features='auto', bootstrap=False, random_state=42),
                    param_grid=param, scoring='f1', cv=3)
grid.fit(x, y)
rfc_parameters = grid.best_estimator_.get_params()
print('最优分类器:',grid.best_params_,'最优分数:', grid.best_score_)
grid_display(grid)


# Gradient Boosting Decision Tree
param = {'n_estimators':[int(x) for x in np.linspace(start = 200,stop = 800, num = 4)],
         'max_depth':[5, 8, 10],
         'subsample':[0.3, 0.5, 0.7],
         'learning_rate':[0.02, 0.05, 0.1, 0.5],
         'min_samples_split':[30, 35, 40],
         'min_samples_leaf':[30, 35, 40]}
grid = GridSearchCV(GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', max_features=None,
                                               max_leaf_nodes=None, min_impurity_decrease=0.0, random_state=42),
                    param_grid=param, scoring='f1', cv=3)
grid.fit(x, y)
gbdt_parameters = grid.best_estimator_.get_params()
print('最优分类器:',grid.best_params_,'最优分数:', grid.best_score_)
grid_display(grid)

# eXtreme Gradient Boosting
param = {'n_estimators':[int(x) for x in np.linspace(start = 200,stop = 800, num = 4)],
         'max_depth':[3, 5, 8],
         'subsample':[0.1, 0.2, 0.3],
         'learning_rate':[0.5, 1]}
grid = GridSearchCV(XGBClassifier(booster='gbtree', gamma=0, colsample_bytree=0.8, alpha=0, random_state=42),
                    param_grid=param, scoring='f1', cv=3)
grid.fit(x, y)
xgb_parameters = grid.best_estimator_.get_params()
print('最优分类器:',grid.best_params_,'最优分数:', grid.best_score_)
grid_display(grid)
```

```
def get_stacking(clf, x_train, y_train, x_test, n_folds= 5):
    import numpy as np
    from sklearn.model_selection import KFold

    train_num, test_num = x_train.shape[0], x_test.shape[0]
    second_level_train_set = np.zeros((train_num,))
    second_level_test_set = np.zeros((test_num,))
    test_nfolds_sets = np.zeros((test_num, n_folds))
    kf = KFold(n_splits=n_folds)

    for i,(train_index, test_index) in enumerate(kf.split(x_train)):
        x_tra, y_tra = x_train.iloc[train_index,:], pd.DataFrame(y_train).iloc[train_index, :]
        x_tst, y_tst =  x_train.iloc[test_index,:], pd.DataFrame(y_train).iloc[test_index,:]
```

```
        clf.fit(x_tra, y_tra)

        second_level_train_set[test_index] = clf.predict(x_tst)
        test_nfolds_sets[:,i] = clf.predict(x_test)

    second_level_test_set[:] = test_nfolds_sets.mean(axis=1)
    return second_level_train_set, second_level_test_set
```

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from xgboost.sklearn import XGBClassifier

rfc_model = RandomForestClassifier(criterion='gini', max_features='auto', bootstrap=False, n_estimators=600,
                            max_depth=10, min_samples_split=2, min_samples_leaf=1, random_state=42)
gbdt_model = GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', max_features=None, n_estimators=400,
max_depth=8,
                                subsample=0.5, learning_rate=0.05, min_samples_split=35, min_samples_leaf=35,
                                max_leaf_nodes=None, min_impurity_decrease=0.0, random_state=42)
xgb_model = XGBClassifier(booster='gbtree', gamma=0, colsample_bytree=0.8, alpha=0, n_estimators=600, max_depth=3,
                    subsample=0.2, learning_rate=1, random_state=42)
lr_model = LogisticRegression(penalty='l2', dual=False,tol=1e4, fit_intercept=True, intercept_scaling=1,
                                class_weight=None, random_state=42, solver='liblinear', max_iter=100,
                                multi_class='ovr', verbose=0, warm_start=False, n_jobs=1)
x_train = pd.DataFrame(x_train)
y_train = pd.DataFrame(y_train)
x_test = pd.DataFrame(x_test)

train_sets = []
test_sets = []
for clf in [rfc_model, gbdt_model, xgb_model]:
    train_set, test_set = get_stacking(clf, x_train, y_train, x_test)
    train_sets.append(train_set)
    test_sets.append(test_set)

meta_train =  pd.DataFrame(train_sets).T
meta_test =   pd.DataFrame(test_sets).T

# Second
rfc_model.fit(meta_train, y_train)
y_predict = pd.Series(rfc_model.predict(meta_test))

res = pd.DataFrame({'real_label':y_test,
                'predict_label':y_predict.map(lambda x:float(x)).tolist()})
```

```
from sklearn.metrics import accuracy_score , precision_score ,recall_score , f1_score, roc_curve, auc
fpr, tpr, thresholds = metrics.roc_curve(res['real_label'] , res['predict_label'])
print('accuracy_score' , accuracy_score(res['real_label'] , res['predict_label']))
print('precision_score' , precision_score(res['real_label'] , res['predict_label']))
print('recall_score' , recall_score(res['real_label'] , res['predict_label']))
print('f1_score' , f1_score(res['real_label'] , res['predict_label']))
print('AUC', auc(fpr,tpr))
```