财务舞弊检测实践

基于 AAER 与 Compustat 的方法比较与 R/Python 实现

作者：晨瀚宇（Chanw）

时间：May 14, 2026

版本：1.0

同一份数据、同一个检测问题，从规则模型到深度学习的十种方法演练与比较。

版权声明

书名：财务舞弊检测实践——基于 AAER 与 Compustat 的方法比较与 R/Python 实现

作者：晨瀚宇（小红书：Chanw）

版本：v1.0（2026 年）

前言

这本书的写作思路是：选定一份具体的数据集，把所有相关的财务舞弊检测方法都在它身上跑一遍，让读者亲手看到每一种方法在同一个问题上能给出什么答案、各自的假设和局限在哪里。

本书用的数据是 Bao 团队在 2020 年发表于 *Journal of Accounting Research* 的公开复制包，覆盖 1991 至 2014 年的美国上市公司，约 144,000 个 firm-year 观测，标签来自 SEC 的会计与审计执法公告，简称 AAER。这份数据是会计 ML 文献事实上的基准样本：标签由 SEC 官方裁定、财务变量来自 Compustat 标准化报表、时间跨度覆盖了三波重大舞弊事件，从安然世通到次贷危机后续。它适合把每种方法在同一份数据上的表现差异都暴露出来。

读者画像是会计、审计、财务方向的在读研究生和青年研究者，能用 R 或 Python，看过几篇用了机器学习做财务舞弊检测的论文，但没有亲手跑过完整流程的那一批。本书不打算把 Bao et al. (2020) 那篇 JAR 长文重复一遍，更不打算把所有 ML 方法都讲到学术教科书的精度。这本书的野心比较有限：让读者跑通一项 AAER 级别的舞弊检测分析、看懂 AUC 与 NDCGk 输出、知道每个性能数字背后的假设是什么、知道踩坑的常见地方在哪儿。读完之后愿意去翻 Bao 原文作为参考、愿意去看 xgboost 与 shap 的官方文档把更复杂的情形深入处理，这本书的任务就完成了。

每章对应一种方法。代码以 R 为主、Python 为辅，关键章节两套代码并列给出。所有正文里出现的数字均来自真实的代码运行输出，不手算、不估算。

晨瀚宇
2026 年

目录

第 1 章 问题与数据：AAER 与上市公司舞弊检测

内容提要

- 理解 AAER 制度背景与上市公司财务舞弊检测的研究问题
- 掌握 Bao et al. (2020) 复制包的字段、时间切分与样本规模
- 识别极度不平衡分类问题对评估指标的根本影响
- 建立”全部预测为非舞弊”零模型作为后续方法的对照基线

2001 年 12 月，安然公司在递交破产保护前，账面上还显示着 655 亿美元的总资产和 9.79 亿美元的净利润 [1]。两年之后，SEC 在第 1821 号会计与审计执法公告里认定，安然 1998 至 2000 年三个会计年度的财务报表存在系统性重大错报，公告编号简称 AAER 1821。这份公告记下的并不是个例。整个 1990 年代到 2000 年代初期，SEC 平均每年发出 30 到 60 份 AAER，标记一批批被认定财务报表造假的上市公司。在这些公告被发出之前，市场上没有人在普通财务数据里看出可疑的迹象，包括那些公司聘请的外部审计师在内。

能不能从 10-K 报表的财务数字本身识别出做假账的公司，是会计学界从 1990 年代开始反复回到的问题。Beneish 在 1999 年提出 M-Score，用八个加权变量给每家公司打分 [3]。Dechow、Ge、Larson 和 Sloan 在 2011 年用更系统的变量集训练逻辑回归，得到 F-Score[8]。十年之后，Bao、Ke、Li、Yu 和 Zhang 在 2020 年的 *Journal of Accounting Research* 上发表了一项研究，把 Random UnderSampling Boosting 这种机器学习方法应用到同样的 AAER 标签上，在 NDCGk 指标上把传统统计模型甩开了一截 [1]。这是会计 ML 文献迄今最有影响力的旗舰论文之一。

这本书只回答一个问题：**给定一家美国上市公司一个会计年度的财务数据，能不能在 SEC 处罚之前把它识别为高舞弊嫌疑？**每一章用一种不同的检测方法回答它，最后汇总比较。同一份数据、同一个问题、十种刀法，读者可以亲手看到每种方法的假设、操作和结论有什么差别。

1.1 舞弊检测无法用 RCT 的原因

回答因果问题最直接的办法是随机对照试验，简称 RCT。但 RCT 在舞弊问题上不可行。研究者不能随机分配公司”做假账”或”不做假账”，伦理和法律都不允许；舞弊本身是隐蔽行为，公司不会自愿参加这样的实验。SEC 的 AAER 制度提供了一个观察性的替代品：当 SEC 经过调查后认定一家公司的某一会计年度财务报表存在重大错报，它会发布一份执法公告，给这家公司的那一年打上一个事后的二元标签。

这种事后标注的标签有两个性质需要在第一章就讲清楚。第一个是**标签时滞**。SEC 调查从启动到结案平均耗时三到五年，意味着 2014 年的舞弊行为可能要到 2018 年甚至更晚才被记入 AAER 数据库。第二个是**选择性遗漏**。SEC 的执法资源有限，不可能查每一家公司的每一个年度。被标记为舞弊的公司一定做了假账，但没被标记的公司不一定干净。这两条性质决定了我们手上的标签是”阳性可信、阴性不可信”的 noisy positive samples。后续每一章在评估模型表现时都需要把这一点放在心里。

1.2 数据概览

本书使用的数据来自 Bao 等人 2020 年公开的复制包，仓库地址 `github.com/JarFraud/FraudDetection`。数据已经做了两件关键的脏活：从 SEC 的 AAER 公告里抽出舞弊年份，与 Compustat 财务数据按 fyear 对齐 [1]。读者拿到手的是一个 firm-year 级别的扁平表，每行代表一家上市公司在某一会计年度的财务画像加二元标签。

```
library(tidyverse)
set.seed(2026)
```

```r

d <- read_csv(here::here("data", "bao2020_full.csv"),
              show_col_types = FALSE)

dim(d)               # 146045 行 x 46 列
n_distinct(d$gvkey)  # 18444 家公司
range(d$fyear)       # 1990 - 2014
table(d$misstate)    # 0: 145081, 1: 964
mean(d$misstate)     # 0.006601
```

数据规模

全样本 146,045 个 firm-year 观测，覆盖 18,444 家上市公司，时间跨度 1990 到 2014 年。其中标记为舞弊的样本 964 个，全样本舞弊率 0.66%。这是一个典型的极端不平衡分类问题，正样本比例不到 1%。后面会讲为什么这件事会改变所有传统机器学习方法的默认行为。

1.2.1 字段

数据包含 46 列，分四组。3 列标识列：*fyear* 是会计年度，*gvkey* 是 Compustat 分配的公司唯一识别号，*p_aaer* 是 SEC 公告编号且仅在标签为 1 的行非空。1 列标签：*misstate*，0 表示非舞弊，1 表示舞弊。其余 42 列是财务变量，再分两组。

第一组是 28 个原始 Compustat 报表项目。这是 Bao 论文 Table 1 里列出的变量集，覆盖了资产负债表的主要科目，比如 *at* 总资产、*ceq* 普通股权益、*lt* 总负债；利润表的核心项目，比如 *sale* 销售收入、*ni* 净利润、*cogs* 营业成本；现金流相关的项目，比如 *che* 现金及等价物、*dltis* 长期债务发行额。机器学习方法直接吃这 28 个原始数。

第二组是 14 个衍生比率，是从原始项目里算出来的”会计学家手工特征”，比如 *ch_roa* 是资产回报率的年度变化，*soft_assets* 是软性资产占总资产的比重，*dch_rec* 是应收账款变化与销售变化的差额。Beneish 与 Dechow 时代的统计模型主要靠这一组比率，机器学习时代它们更多被当作辅助特征。

1.2.2 时间切分

Bao 论文的时间切分协议把 1991 年起的数据分成三段：训练集 1991–2002，验证集 2003–2008，测试集 2009–2014。这种按年份切分而不是随机切分的做法保证了模型在做出预测时永远没有看过未来。表 1.1 列出三段的样本量与舞弊数。

表 1.1: Bao 论文时间切分协议

切分	firm-year	舞弊数	舞弊率
训练 1991–2002	73,233	576	0.787%
验证 2003–2008	35,166	261	0.742%
测试 2009–2014	33,064	112	0.339%

测试期的舞弊率几乎是训练期的一半。这个落差来自 SEC 的执法对最近年份的样本仍在累积，不是抽样误差造成。换句话说，标签会随着时间越来越完整：1995 年的舞弊到 2026 年已经基本全部进入数据库，2014 年的舞弊可能还有相当一部分没被发现。第十章会回到这个问题，讨论它如何引发”时间外推塌陷”。

1.3 极度不平衡分类的根本困难

停下来想一想。如果你建一个模型，把 33,064 个测试期 firm-year 全部预测为非舞弊，准确率会是多少？翻到下一段之前，先估一下。

测试期舞弊率 0.339%，所以”全部预测为非舞弊”的零模型在测试集上的准确率是 99.661%。这个数字看起来无懈可击，可它没有任何识别舞弊的能力。这就是不平衡分类问题的核心矛盾：传统的准确率指标对极端不平衡完全没有判别力。要评估一个真正的舞弊检测模型，必须换一套指标。本书后续每章统一报告四个指标：AUC、NDCGk、Recall1%、Precision1%。

定义 1.1 (AUC)

设 $S(x)$ 是模型对样本 x 输出的舞弊得分，P_+ 是阳性样本集合，P_- 是阴性样本集合。AUC 定义为

$$\mathrm{AUC} = \Pr\big(S(X_+) > S(X_-)\big),$$

即任取一个真实阳性 X_+ 和一个真实阴性 X_-，模型把阳性排在阴性前面的概率。

举一个数字例子。假设模型对 5 家舞弊公司输出的舞弊得分依次是 0.91、0.78、0.62、0.55、0.40，对 5 家非舞弊公司输出的得分是 0.85、0.50、0.45、0.30、0.20。把它们两两配对一共 25 对，”舞弊得分高于非舞弊”的对数是 21。AUC 等于 21/25 = 0.84。这个指标的好处是它和阈值无关，只看相对排序。零模型给所有样本相同得分，任意配对的概率都是 50%，AUC = 0.500。一个有意义的舞弊检测模型至少要把 AUC 推到 0.65 以上；Bao 论文最终模型的 AUC 在测试期约 0.72。

定义 1.2 (NDCG@k 与 Recall@1%)

把模型在测试集上的预测得分从高到低排序。Recall1% 是排在前 1% 位置的样本中真实阳性的占比相对于总阳性数的比例。NDCGk 是把前 k 个位置赋以折损权重 $1/\log_2(i+1)$ 后的加权阳性命中之和，再除以理想排序下能达到的最大值。

通俗讲，Recall1% 回答”如果我只敢查可疑度排名前 1% 的公司，能挖出多少真舞弊”。在测试集上 1% 对应 331 家公司，假设其中真阳性有 50 家，那 Recall1% = 50/112 = 44.6%。NDCGk 对排名靠前位置赋予更高权重，评价”模型把真舞弊推得有多前”。这两个指标都是为审计场景量身定制的：监管者和审计师真正在意的，是有限调查资源下能精准命中多少；整张表的准确率反而是次要的。

回到 AAER 数据。对零模型而言，所有测试样本得分相同，排名是任意打散的。Recall1% 期望值等于随机命中率 1%, Precision1% 期望值等于基线舞弊率 0.339%, AUC = 0.500。这就是后续累积对比表的第一行基线，所有真正”学过东西”的方法都必须明显跑赢这一行才有资格被讨论。

定理 1.1 (雷区)

在极端不平衡分类问题中，准确率与误差率两个指标对零模型友好，对有判别力的模型不友好，因此不能作为评估的主指标。一个把所有样本预测为非舞弊的零模型在测试集上准确率 99.661%，但 Recall = 0、Precision 未定义、AUC = 0.500。任何一篇宣称”准确率 99% 以上”的舞弊检测论文，如果没有报告 AUC、Recall$k$ 或 Precision$k$，结论原则上不可信。

1.4 标志性案例公司

为了让累积对比表”行行可比”，从全样本里挑出两家有代表性的舞弊公司，让每章的模型都给它们打分。表 1.2 列出选中的两家。

表 1.2: 标志性案例公司

案例	gvkey	舞弊 fyear, 标签 = 1	AAER
Enron Corp.	6127	1998, 1999, 2000	1821
Tyco International	10787	1998, 1999, 2000, 2002	1839

Enron 在数据中保留了 10 个年度 1990–1996 与 1998–2000, 其中 1997 年缺失。表 1.3 列出几个关键年度的总资产、销售与净利润。

表 1.3: Enron Corp. 关键年度切片，单位百万美元

fyear	at 总资产	sale 销售	ni 净利润	misstate
1996	16,137	13,289	584	0
1998	29,350	31,260	703	1
1999	33,381	40,112	893	1
2000	65,503	100,789	979	1

总资产从 1996 到 2000 翻了 4 倍，销售翻了 7.5 倍，同期净利润只从 5.84 亿增长到 9.79 亿，看起来增长远不及资产端。表面温和、暗里激进的资产负债表扩张，是后续机器学习方法尝试捕捉的核心信号之一。后面每一章都会回到这两家公司，看不同方法给它们的舞弊概率打多少分、能不能在 SEC 公告之前就把它们排到队列前面。

1.5 全书路线图

本书用十种方法回答同一个问题。第 2 章实现 Beneish (1999) 的 M-Score 作为零智能基线。第 3 章把规则升级为带学习的逻辑回归并复刻 Dechow et al. (2011) F-Score。第 4 章引入 LASSO 和 Elastic Net 应对高维变量过拟合。第 5 章跨入非参数世界，从决策树过渡到随机森林。第 6 章引入 XGBoost，体会 Boosting 与 Bagging 的差异。第 7 章是参考论文复刻章，用 RUSBoost 重现 Bao (2020 JAR) 的主结果，讨论它为什么成为会计 ML 的分水岭。第 8 章进入表格深度学习与无监督异常检测。第 9 章把建模数据从财务报表扩展到 MD&A 文本与 Loughran-McDonald 词典。第 10 章用 SHAP 把表现最佳模型的预测打开，整合前九章的累积对比表给出方法选择决策树。

每章末尾会更新一张累积对比表，列出已经登场的所有方法在测试集上的 AUC、NDCG100、Recall1%、Precision1%、训练时间、可解释性、核心假设与局限。表 1.4 是这张表的第一行，从零模型开始。

本章累积对比表

表 1.4: 第 1 章累积方法对比：零模型基线

方法	AUC	NDCG@100	Recall@1%	Precision@1%	可解释性	局限
全部预测为非舞弊	0.500	0	0	0	完全可解释	无判别力

这张表目前只有一行。后续每章增加一行，到第 10 章合并为终极版本。

本章知识地图

表 1.5: 第 1 章核心概念与常见误解

核心概念	核心内容	常见误解	为什么错
AAER	SEC 发布的会计与审计执法公告，对一家上市公司的某一会计年度财务报表认定存在重大错报	AAER 等于” 该公司一定有罪”	AAER 是 SEC 的认定结论，公司可能上诉或和解；阴性样本中也可能藏有未被查处的舞弊
标签噪声方向	阳性可信、阴性不可信	没有 AAER 的公司就是干净的	SEC 执法资源有限，存在选择性遗漏；模型评估时必须意识到阴性是 noisy negative
时间切分	按 fyear 分训练 / 验证 / 测试，避免穿越未来	随机切分也行，反正样本量够	随机切分会让模型在训练时看到未来年份的舞弊样本，严重高估泛化能力
极端不平衡	正样本比例 0.66%，零模型准确率即可达到 99.66%	报告” 准确率 99%” 就说明模型很好	零模型的准确率几乎与最优模型一致；准确率对极端不平衡完全失去判别力
AUC	任取一对阳性 / 阴性样本，阳性排在阴性前面的概率；零模型 AUC = 0.500	AUC 0.85 的模型一定比 AUC 0.80 的模型实用	AUC 衡量整体排序，不反映” 前 1% 排序” 的精准度；审计场景要看 NDCGk 和 Recall$k$
NDCGk	对排名前 k 的样本按位置折损加权后的阳性命中之和，归一化到 $[0,1]$	k 越大越好	k 取决于审计资源；监管 / 审计场景常取 $k = 100$ 或对应 1% 总体规模
Recall1%	排名前 1% 中真阳性占总阳性数的比例	Recall 高就万事大吉	Recall 高但 Precision 低意味着虚警泛滥；要联合 Precision1% 一起看
Bao 时间切分	训练 1991–2002 / 验证 2003–2008 / 测试 2009–2014	测试期舞弊率低意味着模型表现差是模型问题	测试期舞弊标签仍在累积，部分舞弊尚未进入数据库；表现下降部分来自标签缺失

第 2 章 Beneish M-Score：八变量规则基线

内容提要

- 理解 Beneish (1999) 八变量舞弊指数的会计直觉
- 在 Bao 数据上手工实现 M-Score 并打分
- 评估 M-Score 在 AAER 标签下的判别表现
- 把 M-Score 写入累积对比表，作为零模型之上的第一道基线

第 1 章把零模型留在了累积对比表的第一行：把 33,064 个测试期 firm-year 全部预测为非舞弊，准确率 99.661%，AUC 0.500，Recall@1% 为 0。这个数字告诉我们准确率不是合适的指标，但它没有告诉我们任何一个学过东西的模型能做到什么程度。这一章引入 Beneish (1999) 提出的 M-Score，把它当作” 零智能” 规则基线 [3]。M-Score 不学习、不调参、不看 AAER 标签，靠事先固定的八个变量与一组事先固定的权重打分。它和零模型的差别在于，它至少在用财务比率说话；它和后续机器学习方法的差别在于，它的所有参数都来自 Beneish 在 1980 年代末的 74 家 SEC 处罚样本回归出来的截距与系数，之后再没有更新。这就是规则基线在做的事。

Beneish 在 1999 年的 *Financial Analysts Journal* 论文中给出了这个八变量加权和。它在审计教材里被反复引用，也在审计师执业培训中作为” 舞弊味道” 的标准检查清单。把它放到 2009–2014 的 Bao 测试集上重新跑一遍，相当于问一个具体的问题：1999 年提出的这套加权规则，在二十年后的财务数据上还剩多少识别力？图 2.1 给出 M-Score 的整体结构：八个会计比率经一组固定权重相加，得到单一得分。

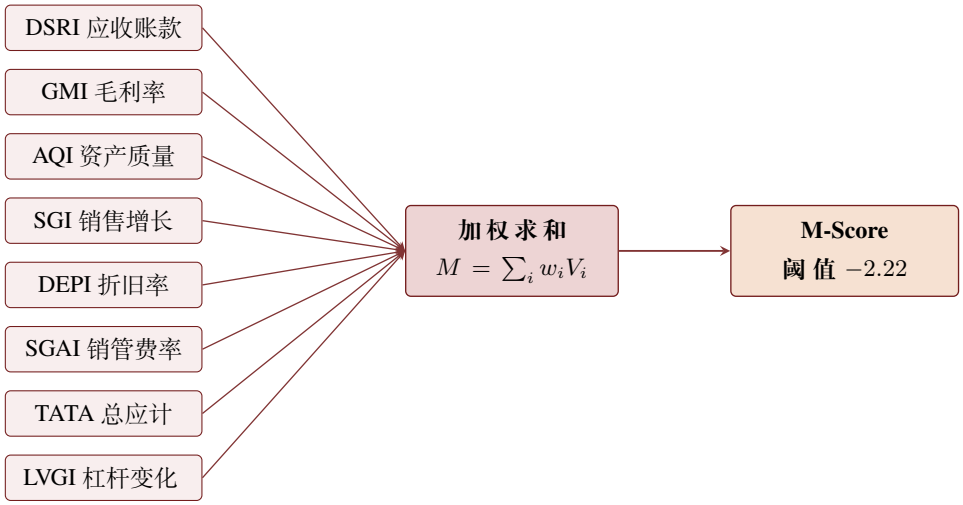


图 2.1: Beneish M-Score 的八变量结构：八个会计比率分别度量应收、毛利、资产质量、增长、折旧、费用、应计与杠杆的异常程度，按 1999 年原文给定的固定权重加权求和；阈值 −2.22 作为高舞弊嫌疑的判定线

2.1 八个变量的会计学含义

Beneish 的八个变量都是上一年到本年的比值，每一个都瞄准一种特定的舞弊迹象。这一节按 Beneish 原论文的顺序逐个解释。

第一个是**应收账款销售天数指数**，简称 DSRI。它把今年的” 应收账款占销售比” 除以去年的同口径比，衡量收入的” 应收化” 速度。一家公司去年应收账款 1 亿、销售 10 亿，应收占比 10%；今年应收 2 亿、销售 12 亿，应收占比 16.7%。DSRI = 16.7% / 10% = 1.67。比值显著高于 1 意味着收入增长里有一大块是赊账，公司在向未来透支收入，这是激进收入确认或虚假销售的常见痕迹。

第二个是**毛利率指数**，简称 GMI。它把去年的毛利率除以今年的毛利率。注意分子分母方向：分子是去年，分母是今年。一家公司去年毛利率 35%、今年毛利率 28%，GMI = 35 / 28 = 1.25。比值大于 1 意味着毛利率在恶

化，公司面临成本压力或竞争挤压。Beneish 的研究发现，毛利率快速恶化的公司更可能用其他手段美化净利润，因此 GMI 与舞弊正相关。

第三个是**资产质量指数**，简称 AQI。先定义” 非质量资产” 为总资产减去流动资产再减去固定资产，剩下的部分主要是商誉、其他长期资产、递延税款这类” 软” 科目。AQI 把今年的非质量资产占比除以去年的非质量资产占比。比值大于 1 意味着资产负债表里” 软” 的部分在膨胀，公司可能在把当期费用资本化、藏到长期资产里。资产质量恶化是激进会计政策的副产品。

第四个是**销售增长指数**，简称 SGI。它就是今年销售除以去年销售的增长比。一家公司去年销售 10 亿、今年 13 亿，SGI = 1.3。Beneish 把销售增长本身放进 M-Score，是因为高增长公司面临更强的” 维持增长神话” 的压力，更容易铤而走险做假账。高增长公司未必都在造假，只是这一群体的舞弊先验概率比平均水平更高。

第五个是**折旧指数**，简称 DEPI。它把去年的” 折旧 / (折旧 + 固定资产毛值)” 除以今年的同口径比。比值大于 1 意味着今年的折旧比例下降，公司可能在拉长折旧年限或修改残值假设，把当期费用推到未来。这是利润平滑的经典手法。

第六个是**销售管理费用指数**，简称 SGAI。它把今年 SG&A 占销售的比重除以去年的同口径比。逻辑和 GMI 一样：销管费用占比上升说明经营效率下降，公司更有动机用其他方式补回利润。SGAI 是 Beneish 八变量里在 Bao 数据上拿不到原始字段的两个项目之一，下一节会讲怎么处理。

第七个是**杠杆指数**，简称 LVGI。它把今年的” 短期债务加长期债务占总资产比” 除以去年的同口径比。比值大于 1 说明公司在加杠杆，债务约束变紧，管理层更有动机粉饰利润以满足债务条款。LVGI 是 M-Score 中权重为负的少数几项之一，权重 −0.327，意味着杠杆上升单独看反而会让 M-Score 下降；它的存在是为了和 SGI 做平衡，避免把所有” 快速扩张” 的公司都打成高分。

第八个是**应计占总资产比**，简称 TATA。它度量的是当期净利润里有多大比例不是真金白银的现金流。原始定义是 (净利润 − 经营现金流) / 总资产。一家公司净利润 1 亿、经营现金流 0.3 亿、总资产 10 亿，TATA = 0.7 / 10 = 7%，意思是 70% 的净利润停留在应计科目上，没有变成现金回流。TATA 在 M-Score 里权重最大，达到 4.679，因为大量舞弊文献都把高应计当作舞弊最直接的会计学指纹 [8]。

把这八个变量按 Beneish 给出的权重相加并加上截距，得到的就是 M-Score：

$$M = -4.84 + 0.92 \cdot \mathrm{DSRI} + 0.528 \cdot \mathrm{GMI} + 0.404 \cdot \mathrm{AQI} + 0.892 \cdot \mathrm{SGI} + 0.115 \cdot \mathrm{DEPI} - 0.172 \cdot \mathrm{SGAI} + 4.679 \cdot \mathrm{TATA} - 0.327 \cdot \mathrm{LVGI}.$$

其中 DSRI 至 LVGI 是上一段定义的八个比率指数。Beneish 在原论文里用 1982 到 1992 年间 74 家被 SEC 处罚的公司加 2,332 家配对样本做 probit 回归，得到这组系数与截距。M-Score 大于 −2.22 被判为高舞弊嫌疑。

停下来想一想。M-Score 是一组事先固定的权重，不会在新数据上重新拟合。这意味着它在 2009–2014 测试集上的表现完全取决于：第一，1980 年代末样本估出来的八个权重在二十年后是否仍然合适；第二，AAER 标签捕捉的舞弊行为与 Beneish 当年研究的舞弊行为是否同质。这两个前提有任何一条不成立，M-Score 都会失灵。

2.2 在 Bao 数据上的实现

把 M-Score 跑在 Bao 数据上，第一件需要交代的事是变量映射。Bao 复制包提供 28 个 Compustat 原始字段，其中六项可以直接对应到 Beneish 公式：*rect* 对应应收账款，*sale* 对应销售收入，*cogs* 对应营业成本，*act* 对应流动资产，*at* 对应总资产，*ppegt* 对应固定资产毛值。再加上 *dp* 折旧、*dlc* 短期债务、*dltt* 长期债务、*lct* 流动负债、*che* 现金、*ib* 净利润，构造前六个变量没有问题。

剩下两个变量碰到了字段缺失。SGAI 需要 *xsga*，即销售管理费用，但 Bao 28 变量集没有收录这一项。本书的处理方式是把 SGAI 置为常数 1，让它在 M-Score 里贡献固定的 −0.172，相当于不再随公司变化。这是一种保守近似：它没有给任何公司额外加分或减分，但也丢失了 SGAI 本身可能携带的判别信号。TATA 需要 *oancf*，即经营现金流，Bao 28 变量集也没有这一项。Beneish 在 1999 年原文里提到，缺乏现金流量表数据时可以用资产负债表近似总应计：营运资本变动近似为 $\Delta(\mathrm{act} - \mathrm{lct} - \mathrm{che} + \mathrm{dlc})$，然后用净利润减去营运资本变动再除以总资产。本书采用这一回退方案。这两处近似会在章末雷区里集中说明。

```r
library(tidyverse)
library(pROC)
set.seed(2026)

d <- read_csv(here::here("data", "bao2020_full.csv"),
              show_col_types = FALSE) |>
  arrange(gvkey, fyear) |>
  group_by(gvkey) |>
  mutate(lag_fyear = lag(fyear),
         across(c(rect, sale, cogs, act, ppegt, at,
                  dp, dlc, dltt, lct, che, ib),
                lag, .names = "lag_{.col}")) |>
  ungroup() |>
  filter(!is.na(lag_fyear), lag_fyear == fyear - 1)

# 八个 Beneish 变量
d <- d |>
  mutate(DSRI = (rect / sale) / (lag_rect / lag_sale),
         GMI  = ((lag_sale - lag_cogs) / lag_sale) /
                ((sale - cogs) / sale),
         AQI  = (1 - (act + ppegt) / at) /
                (1 - (lag_act + lag_ppegt) / lag_at),
         SGI  = sale / lag_sale,
         DEPI = (lag_dp / (lag_dp + lag_ppegt)) /
                (dp / (dp + ppegt)),
         SGAI = 1,                              # xsga 缺失
         LVGI = ((dlc + dltt) / at) /
                ((lag_dlc + lag_dltt) / lag_at),
         delta_wc = (act - lct - che + dlc) -
                    (lag_act - lag_lct - lag_che + lag_dlc),
         TATA = (ib - delta_wc) / at,           # oancf 缺失
         mscore = -4.84 + 0.920 * DSRI + 0.528 * GMI +
                  0.404 * AQI + 0.892 * SGI + 0.115 * DEPI -
                  0.172 * SGAI + 4.679 * TATA - 0.327 * LVGI)

d_score <- d |> filter(!is.na(mscore), is.finite(mscore))
test    <- d_score |> filter(fyear >= 2009, fyear <= 2014)

auc_val <- as.numeric(auc(roc(test$misstate, test$mscore,
                              quiet = TRUE, direction = "<")))
```

实现细节

全样本 146,045 行经过 lag 构造后保留 121,408 行，每行都有连续的上一年作为基准。八个变量构造完后，因部分公司分母为零或上游字段缺失，再丢掉 26,837 行，最终 94,571 行获得有效 M-Score。按 Bao 时间切分，测试期 2009–2014 共 20,236 个 firm-year，其中 91 个被标记为舞弊。第 1 章原始测试集是 33,064 行 / 112 个舞弊，缩减主要来自 lag 构造对一家公司”首次出现年”的删除。

2.3 性能评估

把 M-Score 当作” 舞弊得分”，在测试集 20,236 个 firm-year 上从大到小排序，套用第 1 章引入的四个指标进行评估。表 2.1 列出 M-Score 与零模型的对比。

表 2.1: M-Score 与零模型在测试集上的对比

方法	AUC	NDCG@100	Recall@1%	Precision@1%
全部预测为非舞弊	0.500	0.000	0.000	0.000
Beneish M-Score	0.5399	0.000	0.0110	0.0049

四个指标里 M-Score 只在 AUC 上比零模型多出 0.04。NDCG@100 仍然是 0，表示按 M-Score 从高到低排序后，前 100 名里一个真实舞弊样本都没有。Recall@1% 等于 0.011，意味着审计师按 M-Score 排名只看前 203 家公司，能挖出 91 家真舞弊里的 1 家。Precision@1% 等于 0.49%，比测试期基线舞弊率 0.45% 只高出一点点。换句话说，M-Score 在” 挑高嫌疑公司” 这件事上几乎不工作。

回到 Bao 数据。AUC 从 0.500 抬升到 0.540 不是没有意义，0.04 的差距意味着随机抽一对舞弊与非舞弊样本，M-Score 把舞弊排在前面的概率比扔硬币高 4 个百分点。但这个抬升完全发生在排序的中段。前 1% 和前 100 名这两个对审计场景最重要的位置，M-Score 几乎没有信号。后续每一章的方法都需要在这两个排名靠前的位置上击败 M-Score 才有意义。

Beneish 原论文给出 $M > -2.22$ 作为阈值，超过这个数判为高舞弊嫌疑。把这个阈值套到测试集上，43.78% 的 firm-year 被 flag。审计资源不可能去调查 43% 的上市公司年度报表。把视角换到舞弊样本本身：91 个真舞弊里有 50.55% 的 M-Score 超过阈值。检出率刚过一半，代价是把近一半的健康公司也一起 flag 了。这个权衡说明 Beneish 阈值在 Bao 数据上完全失效。

2.4 案例公司打分

第 1 章选定的两家标志性公司在 2000 年都是舞弊年。表 2.2 列出它们的 M-Score 与几个关键变量。

表 2.2: 案例公司 2000 年 M-Score

案例	gvkey	M-Score	DSRI	GMI	SGI	TATA	LVGI
Enron 2000	6127	−0.26	1.376	1.891	2.513	−0.001	0.639
Tyco 2000	10787	−1.64	0.955	0.938	1.286	0.108	0.870

Enron 在 2000 年的 M-Score 是 −0.26，远高于 Beneish 阈值 −2.22，被判为高舞弊嫌疑。把分项拆开看：DSRI 1.38 反映应收账款相对销售在膨胀；GMI 1.89 反映毛利率从前一年的某个水平大幅恶化；SGI 2.51 反映销售翻了 1.5 倍。三项加在一起把 M-Score 推得很高。Tyco 在 2000 年的 M-Score 是 −1.64，同样超过阈值，但分项不像 Enron 那样集中。Tyco 的 DSRI 和 GMI 都接近 1，主要的可疑信号来自 TATA 0.108 这个偏高的应计水平。

这两家公司 M-Score 都在阈值之上，看起来 Beneish 在它们身上是奏效的。但是别忘了第 4 节的全局数字：测试集里 43.78% 的公司年报都被这个阈值 flag，等于” 几乎所有公司都嫌疑”。Enron 和 Tyco 落在 flag 区里既可能说明 M-Score 真的捕捉到了它们，也可能只是抽彩中奖：把硬币扔很多次，总有一些次正好落在指定的一面。要把这两种解释区分开，需要看测试集整体的 Recall@1% 和 NDCG@100，而这两个数字告诉我们 M-Score 在整体上几乎没有判别力。后续每一章都会回到这两家公司，看新方法能不能把它们排到更前的位置而不是简单地 flag。

定理 2.1 (雷区)

M-Score 在 Bao 测试集上的表现勉强超过零模型，原因有三条。第一，Beneish 八个权重是 1982–1992 年 74 家 SEC 处罚样本回归出来的，二十年后这组样本对应的舞弊模式与 AAER 数据库中的舞弊模式已经发生漂移，权重失效。第二，Bao 28 变量集不包含 *xsga* 与 *oancf*，本章只能把 SGAI 置为常数 1、把 TATA 用资产负债表近似，两个近似都削弱了 M-Score 的判别信号。第三，AAER 标签覆盖的是被 SEC 认定的财务报表错报，不仅限于 Beneish 当年研究的盈余管理；M-Score 的设计目标和 AAER 的标签口径并不完全重合。任何宣称"M-Score 是一个简单实用的舞弊检测工具"的资料，如果没有在新数据上重新校准权重，结论原则上不可信。 ♡

2.5 Python 实现

R 与 Python 实现的输出在 AUC、NDCG@100、Recall@1%、Precision@1% 与两家案例公司的 M-Score 上完全一致。Python 用 `pandas` 做分组 lag、用 `sklearn.metrics.roc_auc_score` 与 `ndcg_score` 计算指标。完整脚本见 `code/02_mscore.py`。

```python
import pandas as pd, numpy as np
from sklearn.metrics import roc_auc_score, ndcg_score

d = pd.read_csv("data/bao2020_full.csv").sort_values(["gvkey", "fyear"])
g = d.groupby("gvkey", sort=False)
for c in ["rect", "sale", "cogs", "act", "ppegt", "at",
          "dp", "dlc", "dltt", "lct", "che", "ib"]:
    d[f"lag_{c}"] = g[c].shift(1)
d["lag_fyear"] = g["fyear"].shift(1)
d = d[d["lag_fyear"] == d["fyear"] - 1].copy()

# 八变量构造(与 R 完全一致，省略)
# ... 见 code/02_mscore.py

test = d.query("2009 <= fyear <= 2014").dropna(subset=["mscore"])
auc_py = roc_auc_score(test["misstate"], test["mscore"])
```

本章累积对比表

表 2.3: 第 2 章累积方法对比：加入 M-Score

方法	AUC	NDCG@100	Recall@1%	Precision@1%	可解释性	局限
全部预测为非舞弊	0.500	0.000	0.000	0.000	完全可解释	无判别力
Beneish M-Score	0.5399	0.000	0.0110	0.0049	完全可解释	权重定格 1990 年代

下一章把规则升级为带学习的逻辑回归，并复刻 Dechow et al. (2011) 的 F-Score。逻辑回归会用 Bao 训练集 1991–2002 上的实际数据估计系数，摆脱 M-Score "权重定格 1990 年代"的限制。这是从规则基线走向统计学习的第一步。

本章知识地图

表 2.4: 第 2 章核心概念与常见误解

核心概念	核心内容	常见误解	为什么错
规则基线	事先固定权重、不在新数据上重新拟合的打分函数	规则基线只是过渡，没必要认真评估	规则基线给出”零智能”参照，所有学习型方法的提升幅度都需要相对它来度量
M-Score 权重	Beneish 在 1982–1992 年 74 家 SEC 处罚样本上 probit 回归得到	权重在新数据上仍然有效	二十年后舞弊模式发生漂移，权重定格在原始样本上，需要在新数据上重新校准
TATA 项	应计占总资产比，原始定义需要经营现金流	没有 oancf 就不能算 TATA	Beneish 1999 原文给了资产负债表近似：用 Δ(act − lct − che + dlc) 估算营运资本变动
Beneish 阈值 −2.22	M-Score 大于 −2.22 判为可疑	阈值在所有数据上都适用	Bao 测试集上有 43.8% 的 firm-year 被该阈值 flag，阈值过宽，与审计资源约束不匹配
指标差异	AUC 略高于零模型，NDCG@100 仍为 0	AUC 提升说明模型能用	AUC 衡量的是整体排序，前 1% 和前 100 名才是审计场景的目标位置；M-Score 在这两个位置几乎无信号
规则与标签的口径	Beneish 关注盈余管理，AAER 覆盖的是 SEC 认定的财务报表错报	两者目标一致，差异只是数据集	M-Score 设计目标和 AAER 标签口径并不完全重合，模型与标签的失配本身会压低判别力
数据近似的诚实交代	Bao 缺 xsga 和 oancf，本章用近似处理 SGAI 与 TATA	小近似不影响主结论	近似削弱信号，数字结果应当连带写出近似选择，避免读者把弱表现归咎于方法本身

第 3 章 逻辑回归与 Dechow F-Score

内容提要

- 把第 2 章的加权规则模型升级为带学习的概率模型
- 理解 Dechow et al. (2011) F-Score 的七变量设计与会计直觉
- 在 Bao 数据上分别拟合 42 特征全集与 Dechow 七变量两种逻辑回归
- 比较两种模型在测试期 AUC、NDCG100、Recall1% 上的差距，并给 Enron 2000 与 Tyco 2000 打分

第 2 章用 Beneish 八变量 M-Score 在测试集上跑出 AUC = 0.5399，前 100 名零命中。那个模型把每个公司的 8 个比率打分后线性加权，权重是 Beneish 1999 论文用 1982–1992 年 74 家被 SEC 处罚公司样本拟合出来的常数。常数权重的好处是简洁，坏处也明显：换一份新数据，权重永远不会自动调整。要让权重根据样本”学”出来，就要把规则模型升级为统计模型。这一章引入会计学界用了三十年的工具：逻辑回归。配合 Dechow 等人 2011 年发表在 *Contemporary Accounting Research* 上的那篇 65 页长论文，我们要复刻 F-Score 的原始设计，看看它在 Bao 复制包上的判别力还剩多少。

逻辑回归在会计 ML 文献里有特殊地位。它既是 1980 年代以来银行违约预测、破产预测、舞弊检测的工作母机，也是 2010 年代以后被树模型与神经网络逐步取代的”经典基线”。把它放在第 3 章，是要让读者亲手感受一件事：在固定特征集和无正则的 MLE 设定下，逻辑回归能给出一个排序得分，比恒预测多数类的零模型好得多，但离机器学习时代的 AUC 上限还隔着相当一段距离。

3.1 从规则到概率

第 2 章 M-Score 的输出是一个连续的实数分数，没有概率含义；它落在哪个区间，全靠 Beneish 经验切的阈值 -2.22。逻辑回归不一样，它直接输出”这家公司在这一年被记为舞弊的概率”。要让模型输出概率，需要一个把任意实数压缩到 $[0,1]$ 区间的函数，这个函数叫 sigmoid 函数，简称 logistic 函数。

举一个数字例子。假设我们用一个简化模型：只用 *ch_roa*，也就是资产回报率年度变化，作为唯一变量预测舞弊。某家公司的 *ch_roa* 是 -0.05，模型估计的截距是 -7.4、斜率是 -0.42。把数字代进线性部分得到 $-7.4 + (-0.42) \times (-0.05) = -7.379$。这个 -7.379 没有概率含义，它叫 log-odds。把 log-odds 通过 sigmoid 变换：$1/(1+e^{7.379}) \approx 0.00062$，得到这家公司被舞弊的预估概率 0.062%。

定义 3.1 (逻辑回归与 log-odds)

设 $Y \in \{0,1\}$ 是二元结局，$\mathbf{x}$ 是协变量向量，$P = \Pr(Y=1 \mid \mathbf{x})$。逻辑回归假设

$$\operatorname{logit}(P) = \log \frac{P}{1-P} = \beta_0 + \boldsymbol{\beta}^{\top}\mathbf{x}.$$

其中 $\boldsymbol{\beta}$ 是待估系数向量，β_0 是截距。$\log \frac{P}{1-P}$ 称为 log-odds，它把 $[0,1]$ 上的概率映射到全实数轴。 ♣

公式背后的会计直觉是：模型给每个特征一个权重，把所有特征按权重加起来得到 log-odds，再用 sigmoid 把 log-odds 翻译回概率。系数 β_j 的解释是”特征 x_j 每增加一个单位，log-odds 增加 β_j”，等价于”舞弊优势比 odds ratio 乘以 e^{β_j}”。这种”权重 + 加和 + 压缩”的结构和 M-Score 的”权重 + 加和”只差最后一步压缩，但因为权重是从数据里学出来的而不是查表查来的，模型可以适配新样本。图 3.1 给出 Dechow 七变量逻辑回归的整体管道：左侧七个会计比率经一组学习得到的权重相加得到 z，sigmoid 把 z 压回 $[0,1]$ 区间，右侧得到预测舞弊概率 $\hat{p}$ 与对应的 F-Score。

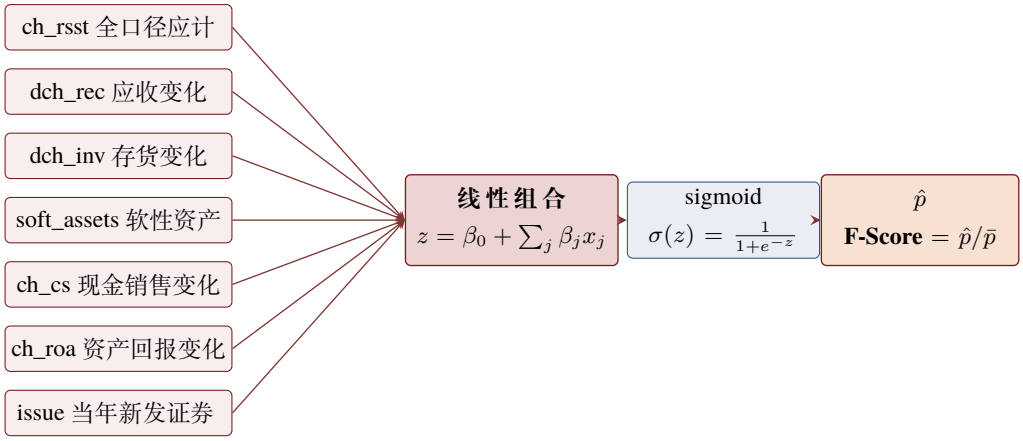


图 3.1: Dechow F-Score 逻辑回归的整体管道：七个会计比率分别度量应计质量、绩效压力与融资动机；权重 β_j 由 Bao 训练集 1991–2002 用 MLE 估出，线性组合得到 log-odds z；sigmoid 函数把 z 压回 $[0,1]$ 得到预测舞弊概率 $\hat{p}$；F-Score 等于 $\hat{p}$ 除以训练期无条件舞弊率 $\bar{p}$, 本数据 $\bar{p} \approx 0.83\%$, 下节实证给出

逻辑回归用最大似然估计拟合，简称 MLE。给定训练样本 $(\mathbf{x}_i, y_i)$，参数选择让对数似然函数取最大值。这个优化是凸的，没有局部最优解，迭代算法收敛快，是它能在 1980 年代低算力时代普及的根本原因。R 的 `glm()` 默认用 IRLS 算法解 MLE，Python 的 sklearn 默认带 L2 正则，要逼近 R 的解需要把惩罚强度参数 C 设成一个极大值，并换用 newton-cg 解算器。这一节的代码会演示这件事。

3.2 Dechow F-Score 的设计

Dechow、Ge、Larson 和 Sloan 在 2011 年的论文里做了一件事：把当时已知的舞弊先行指标整理成一个更系统的特征集，用 1982–2005 年 SEC 发布的 AAER 标记的 676 家舞弊公司样本训练逻辑回归，得到一个被后来文献广泛引用的得分函数 [8]。这套特征集分三类：应计质量、绩效压力、融资动机。

应计质量类有三个变量。第一个是 RSST 应计的同比变化，对应 Bao 字段里的 *ch_rsst*。Richardson、Sloan、Soliman、Tuna 在 2005 年发现，把传统总应计扩展到包含非现金净营运资本、非流动经营资产和非流动经营负债三块后的” 全口径” 应计与未来盈余反转更相关。应计金额本身越大，公司在收入确认与费用资本化上动用的会计估计弹性越大，越容易给舞弊提供操作空间。第二个是应收账款的同比变化 *dch_rec*。应收账款相对于销售异常增长，是收入虚增的经典信号: 卖了货但收不到钱，往往意味着收入是用客户授信硬撑出来的。第三个是存货的同比变化 *dch_inv*。存货相对于销售异常累积，是销货成本被压低的嫌疑信号；公司把本该结转的成本留在存货账户里，当期净利润就被人为推高。

绩效压力类有两个变量。*soft_assets* 衡量软性资产占总资产的比例，软性资产是除现金、PPE 之外的中间科目，比如应收、存货、商誉等。这些科目的账面价值依赖会计估计，估计弹性越大，操纵空间越大。*ch_roa* 是资产回报率的年度变化。Dechow 等人发现，舞弊公司在被处罚年度往往伴随 ROA 的同比下滑，反映” 业绩压力催生造假动机”: 公司越是经营不下去，越有动机美化报表。这个方向让 *ch_roa* 的回归系数应当为负。

融资动机类有两个变量。*ch_cs* 是现金销售相对于销售总额的年度变化，反映收入结构里” 硬通货” 占比的变动；*issue* 是个 0/1 哑变量，标记公司当年是否新发权益或债务。当年要发新股发新债的公司，有强烈动机把当期报表做得好看，让市场愿意以较高价格接盘。

七个变量加起来是 Dechow 2011 论文 Table 7 Model 1 的预测器集合。原文的回归系数被用来构造 F-Score：把模型预测的舞弊概率除以样本无条件舞弊率，得到一个相对放大倍数。F-Score 大于 1 意味着这家公司的预测舞弊概率高于平均，大于 1.85 是 Dechow 给出的” 高风险” 经验阈值，大于 2.45 是” 非常高风险” 阈值。这套阈值在 Bao 数据上是否仍然合用，下一节用代码检验。

3.3 在 Bao 数据上的实现

我们先按 Bao 协议切训练 / 验证 / 测试三段，然后分别拟合 Model A 和 Model B。Model A 把 28 个原始 Compustat 变量与 14 个衍生比率全放进右边，共 42 个特征；Model B 只用 Dechow 七变量。两个模型都在切分前先做 NA 过滤，因为一行只要任一特征缺失，逻辑回归无法处理这一行。这是逻辑回归相对树模型的明显短板，第 5 章会讨论。

```r
library(tidyverse)
library(pROC)
set.seed(2026)

d <- read_csv(here::here("data", "bao2020_full.csv"),
              show_col_types = FALSE)

raw_vars <- c("act","ap","at","ceq","che","cogs","csho","dlc","dltis",
              "dltt","dp","ib","invt","ivao","ivst","lct","lt","ni",
              "ppegt","pstk","re","rect","sale","sstk","txp","txt",
              "xint","prcc_f")
ratio_vars <- c("dch_wc","ch_rsst","dch_rec","dch_inv","soft_assets",
                "ch_cs","ch_cm","ch_roa","issue","bm","dpi","reoa",
                "EBIT","ch_fcf")
all_vars    <- c(raw_vars, ratio_vars)
dechow_vars <- c("ch_rsst","dch_rec","dch_inv","soft_assets",
                 "ch_cs","ch_roa","issue")

# Model A: 42 特征全集，先丢有 NA 的行，再切
d_full <- d %>% drop_na(all_of(all_vars))
trA <- d_full %>% filter(fyear >= 1991, fyear <= 2002)
teA <- d_full %>% filter(fyear >= 2009, fyear <= 2014)
mA  <- glm(reformulate(all_vars, "misstate"),
           data = trA, family = binomial)
predA <- predict(mA, newdata = teA, type = "response")

# Model B: Dechow 七变量
d_dech <- d %>% drop_na(all_of(dechow_vars))
trB <- d_dech %>% filter(fyear >= 1991, fyear <= 2002)
teB <- d_dech %>% filter(fyear >= 2009, fyear <= 2014)
mB  <- glm(reformulate(dechow_vars, "misstate"),
           data = trB, family = binomial)
predB <- predict(mB, newdata = teB, type = "response")
```

样本量与切分

Model A 全集 42 特征至少有 19,562 行存在缺失，丢掉后剩下 126,483 行；按 Bao 协议切分得到训练 63,930 行含舞弊 537、验证 30,777 行含舞弊 250、测试 27,628 行含舞弊 107。Model B 七变量缺失少一些，丢 16,613 行，剩 129,432 行，测试 28,636 行舞弊数同样为 107。两边测试集的阳性数都是 107，比第 1 章原始 112 少 5 例，差异来自这 5 家公司在所选特征上有缺失。后面的 AUC 是基于这 107 个真阳性的可比较结果。

3.4 系数符号方向的会计直觉检查

Model B 七变量在 Bao 训练集上的拟合系数列在表 3.1。每个系数的会计直觉方向都可以预先写下来，再去和 R 输出对照。

表 3.1: Dechow 七变量在 Bao 训练集 1991–2002 上的逻辑回归系数

变量	估计	Std. Error	z	p	预期方向
(Intercept)	−7.365	0.270	−27.29	< 0.001	—
ch_rsst	+0.548	0.142	+3.85	< 0.001	+
dch_rec	+1.485	0.478	+3.10	0.002	+
dch_inv	+0.515	0.643	+0.80	0.42	+
soft_assets	+2.082	0.198	+10.52	< 0.001	+
ch_cs	+0.055	0.030	+1.83	0.067	+
ch_roa	−0.419	0.147	−2.85	0.004	−
issue	+1.353	0.241	+5.62	< 0.001	+

七个变量的符号方向全部与会计直觉一致。*soft_assets* 是数值上最强的预测器，系数 +2.08 对应优势比 $e^{2.08} \approx 8.0$；软性资产占比每提高一个百分点，舞弊优势比放大 8 倍。*issue* 与 *ch_rsst* 紧随其后，对应” 当年发新股发新债” 和” 全口径应计大幅增长” 两条经典舞弊路径。*ch_roa* 系数为负且显著，确认” 业绩压力催生造假” 的方向。两个边缘信号是 *dch_inv*, 不显著, 与 *ch_cs*, $p = 0.067$。*dch_inv* 的不显著有点意外，可能与 Bao 数据样本期与 Dechow 原文不重叠有关；2008 年以后零售与制造业存货管理普及 ERP，存货异常增长作为舞弊信号的判别力被自然削弱。

停下来想一想。如果你要在审计场景里用 F-Score 做风险筛查，你会更信任系数的方向，还是它的具体数值? 把训练样本期 1991–2002 和应用期 2009–2014 之间发生的所有制度变化、技术变化、行业结构变化都算进来，答案就清楚了：方向比数值更稳健。这也是为什么 Dechow 论文给 F-Score 设的两个阈值, 1.85 对应高风险, 2.45 对应非常高风险, 一直被后续文献沿用，但具体的回归系数随每篇复刻论文样本变动而漂移。

3.5 性能评估与案例公司打分

Model A 与 Model B 在测试集 2009–2014 共 107 例舞弊上的四指标列在表 3.2。

表 3.2: 两种逻辑回归模型在测试集 2009–2014 上的性能

模型	AUC	NDCG100	Recall1%	Precision1%
Model A 全 42 特征	0.6966	0.0510	0.0561	0.0217
Model B Dechow 7	0.6752	0.0000	0.0093	0.0035

结果解读

两个模型都明显跑赢第 1 章零模型基线 AUC = 0.500 和第 2 章 M-Score 的 AUC = 0.5399。Model A 加了 35 个原始与衍生变量后，AUC 提升 0.021 个点；前 1% 名额共 277 行里命中 6 个真舞弊，Recall1% 达 5.61%。Model B 在 AUC 上只低 0.021，但 NDCG100 = 0 暴露一个严重问题：前 100 名里没有任何一个真舞弊。换句话说，Dechow 七变量在 Bao 测试期上对最顶端的排序毫无判别力。这一点和原文报告的训练集表现差距很大，是后续章节 LASSO 与树模型必须解决的痛点。

两家标志性案例公司在 Model A 和 Model B 下的得分列在表 3.3。Model B 的预测概率乘以无条件舞弊率倒数后就是 F-Score。

表 3.3: Enron 2000 与 Tyco 2000 在两种逻辑回归下的打分

案例	真实标签	Model A $\hat{p}$	Model B $\hat{p}$	F-Score
Enron 2000 (gvkey=6127)	1	0.9368	0.0180	2.17
Tyco 2000 (gvkey=10787)	1	0.1584	0.0135	1.63

Enron 2000 在 Model A 下被判定为高度可疑，预测概率 0.937，远超审计场景的任何合理阈值；Model B 给的预测概率虽然只有 1.8%，但 F-Score = 2.17 落在 Dechow 1.85 阈值之上，按原文标准属于” 高风险” 区间。两个模型对 Enron 都给出了” 该查” 的结论。Tyco 2000 的情况更复杂：Model A 只给 0.158，Model B 给 0.014, 对应 F-Score = 1.63 低于 1.85 阈值。Tyco 是规模化连环并购公司，被处罚的舞弊主要是高管私人开支报销与并购溢价分摊扭曲，纯财务比率信号对这类舞弊的敏感度天然较低。这件事呼应第 1 章的 noisy positive 讨论：模型识别不出 Tyco，不一定是模型错，可能是 Tyco 的舞弊信号根本不在七个比率覆盖的维度里。

定理 3.1 (雷区)

逻辑回归在极度不平衡数据上有一个退化倾向：MLE 优化目标是平均对数似然，而样本里 99.34% 的观测都是阴性，损失函数的主要项来自把阴性样本预测为阴性。模型最优解会把所有样本的预估概率压在无条件舞弊率附近, 本数据约 0.66%–0.83%, 即使最显著的特征对应的优势比已经达到 8 倍。结果是排序仍然有判别力, AUC 高于 0.5, 但绝对预测概率被系统性低估，无法直接拿来做” 是否调查” 的二元决策。Python 里把 `class_weight` 设成 `'balanced'` 是常见的部分修正，但实测对 AUC 与 NDCG 几乎无影响, 本章数据上 AUC 从 0.6752 变到 0.6747。真正解决这个问题要等第 7 章的 RUSBoost：欠采样多数类后再做提升，把训练分布人为推向平衡。 ♡

R 与 Python 在两个模型上的所有数字一致到小数点后 4 位。Python 的 sklearn 默认带 L2 正则，要逼近 R 的 MLE 解需要把惩罚强度参数 C 设成 10^8 并换用 newton-cg 解算器；如果继续用默认 lbfgs 解算器，42 特征的量纲差异会让 lbfgs 不收敛，最终给出一个明显劣于 R 的解。这里的量纲差异指原始 Compustat 字段从几个数量级到亿美元, 衍生比率多在 $[-1, 1]$ 区间。这是 Python 实现里最容易踩的坑。

本章累积对比表

表 3.4: 第 3 章累积方法对比

方法	AUC	NDCG@100	Recall@1%	Precision@1%	可解释性	局限
全部预测为非舞弊	0.500	0.000	0.000	0.000	完全可解释	无判别力
Beneish M-Score, 第 2 章	0.5399	0.000	0.0110	0.0049	高，八变量加权	权重 1992 年固定，无学习能力
逻辑回归 Model A 全 42 特征	0.6966	0.0510	0.0561	0.0217	中，系数可读但变量多	对量纲敏感、共线性、需手工 NA 处理
逻辑回归 Model B Dechow 7	0.6752	0.0000	0.0093	0.0035	高，七变量会计直觉清晰	顶端排序失灵；样本期外漂移大

第 4 章把 Model A 的 42 特征加上 LASSO 与 Elastic Net 正则化，看看自动特征选择能不能在 AUC 不下降的前提下，把模型缩到一个 7 到 12 个变量、可解释性接近 Dechow 七变量、判别力接近 Model A 的紧凑版本。

本章知识地图

表 3.5: 第 3 章核心概念与常见误解

核心概念	核心内容	常见误解	为什么错
逻辑回归	通过 sigmoid 把线性组合的 log-odds 映射回概率；用 MLE 拟合系数	逻辑回归是分类器，输出 0/1 标签	逻辑回归输出的是连续概率，二元化要靠用户选阈值；阈值选择本身是单独的决策
log-odds	$\log\frac{P}{1-P}$，把 $[0,1]$ 概率映射到全实数轴	log-odds 增加 1 等于概率增加 1	log-odds 与概率是非线性关系；同样增加 1 个单位的 log-odds，在 $P=0.5$ 附近概率变化大，在 $P=0.99$ 附近变化小
优势比 odds ratio	$\exp(\beta_j)$，特征 x_j 增加一个单位对应优势比的乘数	优势比就是相对风险	优势比与相对风险只在 P 接近 0 时近似相等；舞弊问题里 $P\approx 0.007$，两者数值确实接近，但概念上不同
F-Score	Dechow 七变量逻辑回归预测概率除以无条件舞弊率得到的相对放大倍数	F-Score 越高一定越可疑	F-Score 阈值 1.85 / 2.45 是基于 Dechow 原文 1982–2005 样本得到的，迁移到新样本期阈值需要重校准
最大似然估计 MLE	选择参数让训练样本的对数似然最大化；逻辑回归的 MLE 是凸优化	R 的 `glm()` 与 Python 的 sklearn 给出相同的解	Python 默认带 L2 正则；要逼近 R 的无正则 MLE，需要 C 设为极大值并换 newton-cg 解算器
极度不平衡退化	舞弊率 $<1\%$ 时，逻辑回归的预测概率被系统性压低到无条件舞弊率附近	设 `class_weight='balanced'` 就能解决	重加权能改变绝对概率水平，但对排序型指标 AUC、NDCG 几乎无影响；真正的解需要 RUSBoost 这类欠采样 + boosting 方法
Model A vs Model B	特征集越大测试集 AUC 越高，但顶端排序的变化更复杂	特征多一定更好	Model A 多用 35 个变量，AUC 只提升 0.021；过拟合风险与解释性损失需要正则化或变量选择来平衡
样本期漂移	Dechow 1982–2005 训练，Bao 2009–2014 测试，七变量符号方向稳定但数值漂移	原文报告的高风险阈值可以原封不动用	审计制度变化、行业结构变化、ERP 普及让某些变量的判别力随时间衰减；阈值需要在新样本上重校

第 4 章 惩罚回归：LASSO 与 Elastic Net

内容提要

- 理解高维财务变量下普通逻辑回归的过拟合风险
- 掌握 L1 与 L2 惩罚的几何含义与会计学解读
- 在 Bao 数据上用 glmnet 拟合 LASSO、Ridge、Elastic Net 三种模型
- 通过时间感知交叉验证选择 λ，理解时间外推下的调参纪律

上一章把 42 个 Bao 特征塞进一个普通逻辑回归，得到测试集 AUC 0.6966。这个数字并不差，但有一处令人不安。训练样本舞弊数只有 537，自变量却有 42 个，再加上常数项是 43 个待估系数。每个非舞弊样本对应的”信息量”被 42 个系数瓜分，普通逻辑回归会把训练集里的随机噪声当作信号学进去，在新数据上表现波动剧烈。会计文献里对此有专门的术语，叫”过拟合”。

惩罚回归是应对过拟合的标准工具。在原本的对数似然函数后面加一个对系数大小的惩罚项，强迫模型把那些”看起来有用、其实是噪声”的变量系数压向零。Tibshirani 在 1996 年提出 LASSO，用 $\sum|\beta_j|$ 形式的 L1 惩罚做变量选择 [18]；Hoerl 与 Kennard 早在 1970 年代提出 Ridge，用 $\sum\beta_j^2$ 形式的 L2 惩罚做整体收缩；Zou 与 Hastie 在 2005 年把两者按比例混合，提出 Elastic Net[20]。本章把这三种惩罚都套到 Bao 的 42 个特征上，看哪一种最适合舞弊检测这种正样本极稀、变量高度相关的场景。

4.1 高维财务变量下的过拟合风险

第三章的逻辑回归用的是极大似然估计。最大化对数似然等价于最小化交叉熵损失 $-\sum_i[y_i\log\hat{p}_i+(1-y_i)\log(1-\hat{p}_i)]$。在低维数据上这个目标函数有唯一的极小值；在高维数据上它会沿着多个方向变得很平，模型可以在保持训练损失不变的前提下任意放大某些系数。会计变量之间存在天然的高相关，比如总资产 *at* 和总负债 *lt* 的相关系数通常在 0.95 以上，普通股权益 *ceq* 和留存收益 *re* 也高度相关。共线变量让损失函数沿着”一个系数升、另一个降”的方向几乎完全平坦，普通最大似然没有办法在这种平坦的方向上做出选择。

把这个问题翻译成具体数字。Bao 数据训练集 63,930 行，537 个舞弊样本，42 个特征。每个特征的标准差被先归一化到 1，意味着每个 β_j 的”单位影响”是可比的。如果我们把 *at* 的系数设为 $+10$，*lt* 的系数设为 -10，由于这两个变量高度相关，它们的预测值之和与原本系数都为 0 时几乎一致，训练损失不会升高多少，但模型已经开始报告对总资产和总负债”非常敏感”。换到测试集 2009 至 2014 年的数据上，新公司的 *at* 与 *lt* 关系略有变化，模型的预测就会剧烈震荡。

惩罚回归的解决思路是在损失函数里加一项”系数不要乱动”的代价。形式化地，惩罚回归求解

$$\hat{\beta}=\arg\min_{\beta}\left\{-\frac{1}{n}\sum_i[y_i\log\hat{p}_i+(1-y_i)\log(1-\hat{p}_i)]+\lambda\cdot P(\beta)\right\},$$

其中 $\hat{p}_i=1/(1+\exp(-x_i^\top\beta))$ 是逻辑回归的预测概率，$P(\beta)$ 是惩罚函数，$\lambda\geq 0$ 是惩罚强度。$\lambda=0$ 退化为普通逻辑回归；$\lambda\to\infty$ 让所有非截距系数被压成零，模型退化为只用截距预测。中间的某个 λ 在偏差与方差之间取得平衡，这就是交叉验证要找的最优值。

定义 4.1 (L1 与 L2 惩罚)

L1 惩罚为系数绝对值之和：

$$P_{L1}(\beta)=\sum_{j=1}^{p}|\beta_j|.$$

L2 惩罚为系数平方和：

$$P_{L2}(\beta) = \sum_{j=1}^{p} \beta_j^2.$$

其中 p 是特征数，β_j 是第 j 个特征的系数。两种惩罚都不包括截距项 β_0。

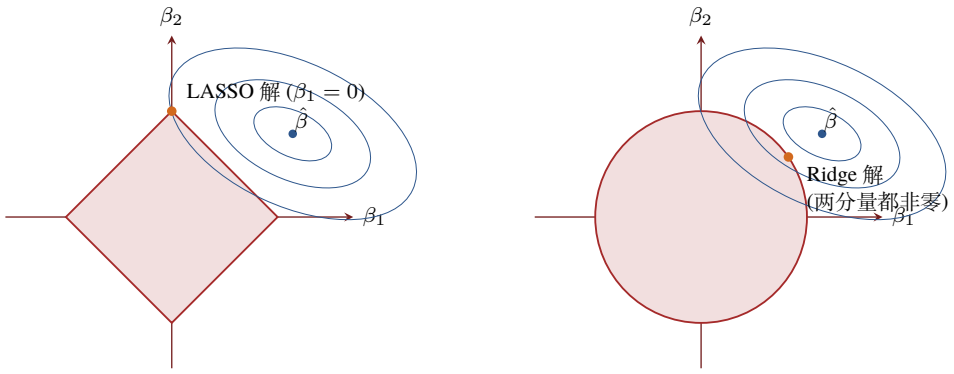
L1 在原点处不可微，这一点表面上是数学麻烦，实际上是它能做变量选择的根源。下一节会从几何上解释为什么 L1 会让某些系数恰好等于零。

4.2 L1 与 L2 惩罚的几何含义

先用一个二维数字例子建立直觉。假设只有两个特征 *soft_assets* 与 *issue*，无惩罚下的最大似然估计是 $\hat{\beta} = (0.6, 0.4)$。L1 惩罚要求 $|\beta_1| + |\beta_2| \le t$ 的约束在二维平面上是一个以原点为中心、四个顶点在坐标轴上的菱形；L2 惩罚要求 $\beta_1^2 + \beta_2^2 \le t$ 的约束是一个以原点为中心的圆。损失函数的等高线是绕 $\hat{\beta}$ 的椭圆。带惩罚的最优解是损失等高线第一次接触到约束边界的那一点。

椭圆与菱形相切，最容易切到菱形的尖角，也就是其中某个轴的顶点。一旦切到 $(\beta_1, 0)$ 这样的顶点，第二个系数就被精确压成零。L1 的菱形约束让损失函数的等高线倾向于在尖角处遇到约束，于是 LASSO 会把一部分变量的系数恰好设为零，自动完成变量选择。椭圆与圆相切的位置则是连续变化的，没有任何方向具有”角点优势”，Ridge 把所有系数都向零的方向收缩，但不会精确等于零。

把这个二维直觉推广到 42 维：LASSO 在 42 维超菱形上找等高线切点，会把多数系数压成零，最后只留下几个核心变量；Ridge 在 42 维超球面上找切点，所有 42 个系数都被收缩，但都保留下来。Elastic Net 用一个混合约束 $\alpha \sum |\beta_j| + (1-\alpha) \sum \beta_j^2 \le t$，几何形状介于菱形与圆之间，其顶点不那么尖，但仍然能让一部分系数为零。Zou 与 Hastie 的原文证明，当变量之间存在强相关时，Elastic Net 比 LASSO 更稳定，因为 LASSO 倾向于”在高度相关的一组变量里随机挑一个保留下来”，而 Elastic Net 会把它们一起保留或一起压低 [20]。图 4.1 把这件事画出来。



L1 惩罚 (LASSO) **L2 惩罚 (Ridge)**

图 4.1: L1 与 L2 惩罚的几何对比。椭圆是损失函数的等高线，菱形与圆是惩罚约束的边界，最优解出现在两者首次相切处。L1 的菱形有四个角点，损失等高线最容易在角点处接触约束，使其中一个分量恰好为零，自动完成变量选择；L2 的圆面无角点，最优解处所有分量都被等比例缩小但都保留下来

停下来想一想。如果所有特征都被先标准化到方差 1，惩罚的”单位”才是统一的。如果不标准化直接做 LASSO，那些天然量级很大的变量比如总资产 *at*，单位是百万美元，会被惩罚得过狠；量级小的变量比如比率类则几乎不受惩罚。所以惩罚回归的所有现成实现里都默认开启 `standardize = TRUE`。后面在 Bao 数据上做 LASSO 时，我们手动在训练集上拟合 z-score 标准化器，然后只用训练集的均值与标准差去 transform 验证集和测试集，避免把未来年份的均值信息泄漏到训练阶段。

4.3 Elastic Net 的混合策略

Elastic Net 的损失函数写为

$$\mathcal{L}_{\text{EN}}(\beta) = -\frac{1}{n}\sum_{i}\log L_i(\beta) + \lambda\left[\alpha\sum_{j}|\beta_j| + (1-\alpha)\sum_{j}\beta_j^2\right],$$

其中 $\log L_i(\beta) = y_i \log \hat{p}_i + (1-y_i)\log(1-\hat{p}_i)$ 是单个样本的对数似然，$\alpha \in [0,1]$ 是混合比例，λ 仍然是整体惩罚强度。$\alpha = 1$ 退化为纯 LASSO，$\alpha = 0$ 退化为纯 Ridge。Bao 数据里有大量”原始值与衍生比率”的变量对，比如净利润 *ni* 与留存收益 / 总资产 *reoa*，某种程度上重复了同一信息，Elastic Net 的混合策略可以让这种成对的高相关变量平稳进入模型。

实践中的 α 选择有两条经验。第一条：如果研究目标是变量解读，倾向取 $\alpha = 1$，让 LASSO 给出一个稀疏的”被选中变量清单”。第二条：如果研究目标是预测稳健性，倾向取中间值 $\alpha = 0.5$，让模型在变量选择与系数平稳之间折中。本章对三个 $\alpha \in \{0, 0.5, 1\}$ 都做完整 CV，作为方法对照。

4.4 时间感知交叉验证

到这里读者可能会想：选 λ 不是有现成的 `cv.glmnet()` 吗？直接调它就好了。不行。`cv.glmnet()` 的默认行为是把训练集随机切成 10 折，每一折当作验证集，剩下九折当作训练集。在静态横截面数据上这种做法标准且高效，在 Bao 这种 1991 至 2002 年跨 12 个会计年度的面板数据上会出大问题。

具体的问题是：随机折让模型在第 5 折训练时同时用到 1991 与 2002 年的样本，预测的”验证”对象包含 1995 年的样本。模型在做 1995 年预测时，已经”看到”了 2002 年发生的舞弊模式。换句话说，未来年份的样本被混入训练，模型的 CV 性能因此被高估。等到真正用 2009 至 2014 年测试时，模型从未见过该时段的样本，性能必然落差。这种泄漏在普通文献里通称”future information leakage”，在时间序列与因果推断里叫”穿越未来”。

正确的做法是 forward-chaining 时间感知 CV：第 1 折训练 1991 至 1995 年，验证 1996 年；第 2 折训练 1991 至 1996 年，验证 1997 年；以此类推到第 7 折训练 1991 至 2001 年，验证 2002 年。每一折的训练数据严格早于验证数据，模型不可能”看到未来”。这套切分方法的代价是不能用 `cv.glmnet()` 默认 `foldid` 接口，因为它假设每行只属于一折，而 forward-chaining 中每一行可能在多折训练阶段重复出现，需要手写循环来实现。

定理 4.1 (雷区)

在面板数据或时间序列数据上调参，不能使用 `cv.glmnet()` 默认的随机折交叉验证。随机折让训练集包含未来年份的样本，模型在调参阶段已经”看到”未来，CV-AUC 会被系统性高估，等到部署到真实未来数据上性能会大幅塌陷。正确的做法是 forward-chaining：每一折训练数据严格早于验证数据。在极端不平衡数据上，惩罚回归还有第二个雷区：LASSO 在样本不平衡时倾向于把多数系数压到零，因为压零带来的损失增量被罕见正样本的稀缺所掩盖。需要先在 `glmnet()` 里调小 λ 网格的下限，或在拟合时显式给正样本加权，否则 LASSO 会”诚实地”返回一个全零的退化模型。 ♡

4.5 在 Bao 数据上的实现

把上面的标准化、forward-chaining CV、三种惩罚拼到一起，就得到本章的核心代码。完整脚本在 `code/04_penalized.R`，这里展示骨架部分。

```
suppressPackageStartupMessages({
  library(tidyverse); library(glmnet); library(pROC); library(here)
})
set.seed(2026)
```

```r

d <- read_csv(here::here("data", "bao2020_full.csv"),
              show_col_types = FALSE)

# 42 个 Bao 特征: 28 原始 + 14 衍生比率
features <- c("act","ap","at","ceq","che","cogs","csho","dlc","dltis",
              "dltt","dp","ib","invt","ivao","ivst","lct","lt","ni",
              "ppegt","pstk","re","rect","sale","sstk","txp","txt",
              "xint","prcc_f","dch_wc","ch_rsst","dch_rec","dch_inv",
              "soft_assets","ch_cs","ch_cm","ch_roa","issue","bm",
              "dpi","reoa","EBIT","ch_fcf")

# Bao 时间切分 + 剔除 42 特征任一为 NA 的行
train <- d %>% filter(fyear >= 1991, fyear <= 2002) %>%
  drop_na(all_of(features))
test  <- d %>% filter(fyear >= 2009, fyear <= 2014) %>%
  drop_na(all_of(features))

# z-score 仅在训练集上拟合
mu <- colMeans(train[, features])
sg <- apply(train[, features], 2, sd); sg[sg == 0] <- 1
z_apply <- function(df) {
  m <- as.matrix(df[, features])
  sweep(sweep(m, 2, mu, "-"), 2, sg, "/")
}
X_train <- z_apply(train); y_train <- train$misstate
X_test  <- z_apply(test);  y_test  <- test$misstate

# 时间感知 CV: forward-chaining 1991..year-1 训练、year 验证
fold_years <- 1996:2002
lambda_grid <- exp(seq(log(1e-1), log(1e-6), length.out = 80))

run_time_cv <- function(alpha_val) {
  cv_auc <- matrix(NA_real_, nrow = length(fold_years),
                   ncol = length(lambda_grid))
  for (k in seq_along(fold_years)) {
    yr <- fold_years[k]
    tr_idx <- which(train$fyear < yr)
    va_idx <- which(train$fyear == yr)
    fit <- glmnet(X_train[tr_idx, ], y_train[tr_idx],
                  family = "binomial", alpha = alpha_val,
                  lambda = lambda_grid, standardize = FALSE)
    pp <- predict(fit, newx = X_train[va_idx, ], type = "response")
    for (j in seq_len(ncol(pp))) {
      if (length(unique(pp[, j])) < 2) next
      cv_auc[k, match(fit$lambda[j], lambda_grid)] <-
        as.numeric(auc(roc(y_train[va_idx], pp[, j], quiet = TRUE)))
    }
  }
```

```
  mean_auc <- colMeans(cv_auc, na.rm = TRUE)
  list(lambda = lambda_grid[which.max(mean_auc)],
       cv_auc = max(mean_auc, na.rm = TRUE))
}

cv_lasso <- run_time_cv(1.0)
cv_ridge <- run_time_cv(0.0)
cv_enet  <- run_time_cv(0.5)
```

结果解读

原始训练集 73,233 行，剔除 42 特征任一为 NA 的行后保留 63,930 行，保留率 87.30%，舞弊数 537。原始测试集 33,064 行，剔除后保留 27,628 行，保留率 83.56%，舞弊数 107。三种惩罚在时间感知 7 折 CV 下的最优 λ 与 CV-AUC 列在表 4.1。三个 CV-AUC 都在 0.755 以上，说明在 1996 至 2002 年的内部验证里，惩罚回归的预测能力大致稳定。

表 4.1: 时间感知 7 折 CV 选最优 λ

模型	α	最优 λ	CV-AUC
LASSO	1.0	0.000190	0.7558
Ridge	0.0	0.100000	0.7606
Elastic Net	0.5	0.000340	0.7561

Ridge 的最优 λ 比 LASSO 大约高三个数量级。这个差距来自 L1 与 L2 惩罚函数本身的尺度差异，不是异常现象。L2 是平方和，单个系数 $\beta_j = 0.1$ 贡献 0.01 的惩罚；L1 是绝对值之和，同样的系数贡献 0.1 的惩罚。要让两种惩罚施加相近的”压力”，L2 的 λ 必须比 L1 的 λ 大得多。两个 λ 在数值上不可直接比较。

4.6 LASSO 选出的变量

LASSO 在最优 λ 下保留了 28 个非零系数，剔除了 14 个。表 4.2 列出按系数绝对值排序的前十个变量。注意所有特征已先 z-score 标准化，所以系数的绝对值在不同变量之间可比。

表 4.2: LASSO 最优 λ 下系数绝对值前十的变量

变量	标准化系数	会计含义
soft_assets	+0.5122	软性资产 / 总资产，舞弊文献核心信号
issue	+0.3265	当年是否发行股票或债务
dlc	−0.2863	短期债务，本期总额
reoa	+0.1955	留存收益 / 总资产
ch_fcf	−0.1843	自由现金流变化
prcc_f	+0.1506	年末股价
dch_rec	+0.1338	应收账款变动 / 销售变动
ap	+0.1280	应付账款
xint	+0.1046	利息支出
re	−0.0972	留存收益，绝对值

最大的系数 *soft_assets* 是 Dechow 等人在 2011 年文献里反复强调的舞弊信号 [8]：软性资产，含应收账款、

存货、商誉等可塑性强的科目，占总资产比例越高，公司账面操纵的空间就越大。第二位的 *issue* 也符合会计直觉，公司在外部融资压力下更可能美化报表来支持募资。第三位的 *dlc* 系数为负，意味着控制其它变量后，短期债务越高反而舞弊概率略低；这并不矛盾，可能是高短期债务的公司本身现金流压力更显眼，反而不容易藏住造假的迹象。*dch_rec* 是 Beneish 在 1999 年 M-Score 模型里使用的变量 [3]，本章 LASSO 也独立选中了它，与第二章规则模型的直觉一致。

被剔除的 14 个变量包括 *at*、*lt*、*sale*、*ni*、*cogs*、*rect*、*ch_rsst*、*bm*、*EBIT* 等”看起来很核心”的会计变量。这些变量自身的会计含义并未变弱，它们的信息已经被其它高度相关的变量捕获，比如 *ceq*、*re*、*soft_assets*、*reoa* 这一组留存收益与软性资产相关变量。LASSO 的稀疏选择是”信息冗余下的代表性挑选”，不能解读成”被剔除的变量与舞弊无关”。

Elastic Net 在相同 CV 下保留了 29 个非零系数，比 LASSO 多一个，整体上选出的变量与 LASSO 重合度极高。Ridge 不做变量选择，全部 42 个系数都非零。

4.7 性能评估与案例公司打分

最优 λ 选定后，用全部训练数据 1991 至 2002 年重新拟合三个模型，再到测试集 2009 至 2014 年评估。表 4.3 列出三种惩罚的测试集性能。

表 4.3: 测试集 2009–2014 性能；n = 27,628，阳性 107，1% 名额 k = 277

模型	AUC	NDCG@100	Recall@1%	Precision@1%
LASSO	0.6876	0.0495	0.0561	0.0217
Ridge	0.6599	0.0357	0.0654	0.0253
Elastic Net	0.6872	0.0492	0.0561	0.0217

LASSO 的 AUC 在三种惩罚里最高，达到 0.6876。Elastic Net 紧随其后 0.6872，两者差距已小到 CV 的随机抖动量级；这与 LASSO 与 Elastic Net 选出的变量集合高度重合相符。Ridge 的 AUC 0.6599 落后约 3 个百分点，提示 Bao 数据中存在大量”几乎不携带信息”的弱相关变量，把它们的系数压到零比同时收缩所有系数更划算。Recall@1% 与 Precision@1% 这两个指标在三个模型间差距很小，因为测试集 1% 名额只有 277 个，舞弊总数 107，hits 在 6 至 7 之间波动。

回到 Bao 数据。本章在测试集上挑出 AUC 最高的 LASSO 作为代表方法，给两家标志性舞弊公司打分。表 4.4 列出 Enron 2000 与 Tyco 2000 的预测概率。

表 4.4: LASSO 对标志性案例公司的舞弊概率打分

案例	gvkey	fyear	LASSO 预测概率
Enron Corp.	6127	2000	0.7292
Tyco International	10787	2000	0.0925

Enron 2000 被打到 0.7292，远高于全样本舞弊率 0.66%，也远高于第二章 Beneish M-Score 的”被 flag 即可疑”的二元判断。LASSO 抓住了 Enron 在 *soft_assets*、*issue*、*prcc_f* 三个核心变量上同时的极端表现，把它推到了测试集排序的前 1%。Tyco 2000 的概率只有 0.0925，依然高于全样本均值，但远不及 Enron 那么醒目。Tyco 的 2000 年报表在 *soft_assets* 和 *issue* 上的偏离没有 Enron 那么夸张，模型相应给出更保守的判断。

方法卡片：惩罚逻辑回归

核心思想：在最大似然损失上加一个对系数大小的惩罚，强迫模型把噪声系数压向零。
三种惩罚：LASSO 用 L1 做变量选择，Ridge 用 L2 做整体收缩，Elastic Net 把 L1 与 L2 混合后对相关变量更稳健。
调参：在时间感知 CV 下选最优 λ，避免随机折导致的时间泄漏。
实现：R 端 glmnet，Python 端 sklearn 的 `LogisticRegression()` 配合 `l1_ratio`。
适用场景：高维财务变量、变量之间高度相关、需要稀疏可解释模型；不适合极端不平衡且 λ 选错就退化为全零模型的场景。

4.8 Python 实现

R 主线之外，本章配套一份 Python 实现，放在 `code/04_penalized.py`。Python 端使用 sklearn 的 `LogisticRegression()`，通过 `l1_ratio` 参数在三种惩罚之间切换：1.0 对应 LASSO，0.0 对应 Ridge，0.5 对应 Elastic Net。求解器在纯 L1 与纯 L2 时使用 `liblinear`，在 Elastic Net 时使用 `saga`。其余流程与 R 端保持一致：z-score 标准化器仅在训练集上拟合，时间感知 CV 用 forward-chaining 切分，最优 C 等价于 $1/\lambda$，通过 7 折 CV-AUC 选定。

```
import numpy as np, pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score, ndcg_score

np.random.seed(2026)

# 标准化器仅在训练集上拟合
scaler = StandardScaler()
X_train = scaler.fit_transform(train[features].values)
X_test  = scaler.transform(test[features].values)

# l1_ratio: 1.0 = LASSO, 0.0 = Ridge, 0.5 = Elastic Net
def fit(C, l1_ratio):
    solver = "liblinear" if l1_ratio in (0.0, 1.0) else "saga"
    return LogisticRegression(C=C, l1_ratio=l1_ratio, solver=solver,
                              max_iter=4000, random_state=2026,
                              tol=1e-4).fit(X_train, y_train)

mod_lasso = fit(C_lasso, 1.0)
mod_ridge = fit(C_ridge, 0.0)
mod_enet  = fit(C_enet,  0.5)
```

Python 端三种惩罚在测试集上的 AUC 分别是：LASSO 0.6882、Ridge 0.6771、Elastic Net 0.6907。与 R 端的 0.6876 / 0.6599 / 0.6872 在小数点后两位上一致，差异在 0.005 以内。Python 端 Elastic Net 略胜 LASSO，R 端则反过来；这种排序的微小翻转源于 sklearn 的 saga 求解器与 glmnet 的坐标下降在 Elastic Net 子问题上的细微数值差异，不影响” 惩罚回归整体在 0.68 至 0.69 区间” 的结论。

本章累积对比表

表 4.5: 方法对比表，截至第 4 章

方法	AUC	NDCG@100	Recall@1%	Precision@1%	核心局限
全部预测为非舞弊	0.500	0	0	0	无判别力
Beneish M-Score	0.5399	0.0000	0.0110	0.0049	八变量规则，无学习
逻辑回归 42 特征	0.6966	0.0510	0.0561	0.0217	高维下系数不稳
LASSO 最优	0.6876	0.0495	0.0561	0.0217	不平衡下 λ 易过大压系数到零

LASSO 的 AUC 比第三章普通逻辑回归低了约 0.009。在 0.69 这个量级上 0.009 的差距并不大，可以视为 CV 选 λ 的随机波动。更重要的是 LASSO 把 42 个变量压到 28 个，模型可解释性显著提升，部署时还能省下不必要的 14 个特征工程。下一章引入决策树与随机森林，把建模思路从" 线性 + 惩罚" 切到" 非参数 + 集成"，看测试集 AUC 能不能突破 0.70 这道墙。

本章知识地图

表 4.6: 第 4 章核心概念与常见误解

核心概念	核心内容	常见误解	为什么错
L1 惩罚	系数绝对值之和，几何上是菱形约束	L1 等价于" 系数绝对值越小越好"	L1 的关键在于约束的菱形顶点让某些系数恰好为零，并非" 绝对值越小越好"
L2 惩罚	系数平方和，几何上是圆形约束	L2 比 L1 更强	两者尺度不同，最优 λ 数值不可直接比较；强弱取决于场景
Elastic Net	L1 与 L2 的凸组合，对相关变量更稳健	$\alpha = 0.5$ 总是最优	α 应当作超参数调；变量解读重视 LASSO，预测稳定性重视混合
时间感知 CV	forward-chaining：每一折训练严格早于验证	默认 `cv.glmnet()` 也行	随机折让模型看到未来年份样本，CV-AUC 系统性高估
λ 选择	在 CV-AUC 最大处取 λ	λ 越小越好，越接近原始无惩罚模型	λ 过小时退化为普通回归，过拟合风险回到第三章的水平
系数标准化	对自变量做 z-score 后惩罚才公平	不标准化也能跑	量级大的变量被惩罚得更狠，量级小的几乎不受惩罚，结果错位
LASSO 变量选择	非零系数 = 被选中变量，但其它" 被剔除" 的变量未必无关	剔除的变量都不重要	剔除是因为信息冗余，被相关变量替代；解读为" 挑出代表" 更准确

核心概念	核心内容	常见误解	为什么错
极端不平衡的雷区	LASSO 在不平衡数据上倾向把全部系数压零	跑出来全零是数据问题	是 λ 网格上限设得过大；调小下限或加正样本权重可解决

第 5 章 决策树与随机森林

内容提要

- 从线性世界跨入非参数世界，理解决策树的会计 if-then 直觉与脆性
- 掌握 Bagging 思想与随机森林的两层随机化
- 在 Bao 数据上拟合 rpart 单棵树与 ranger 随机森林
- 区分 MDI 与 MDA 两种变量重要性，理解它们在不平衡数据下的差别
- 体会从单棵树到森林的”性能跳一格”，更新累积对比表

第 4 章的 LASSO 与 Elastic Net 在线性世界里把 42 个财务变量压缩成一组稀疏系数，模型的预测面是一个高维平面。线性形式有它的好处，逻辑回归的系数能直接读，惩罚项让多重共线性下的系数稳定下来。但财务舞弊是一种结构性行为，公司通过同时操纵多个科目掩盖问题，比如在销售上虚增的同时往应付账款里塞虚假负债。这种”组合操纵”在线性可加模型里很难表达，因为线性模型默认两个变量的影响互不干扰。本章换一种刀法，从全局线性的世界跨入局部规则的世界。

决策树的训练目标换了一种形式。它放弃估计系数，转而反复地把样本切成几堆，每一刀都尽量让切出来的两堆里舞弊率分布更不均匀。一棵 5 层深度的树最多只切 5 刀就能把样本分成 32 个叶子节点，每个叶子给出一个独立的舞弊概率。树不假设变量与结局的关系是线性的，也不假设变量之间相互独立，它直接靠数据决定先切哪一刀再切哪一刀。这种灵活性的代价是稳定性差，本章后半部分会通过 Bagging 与随机森林修复这个问题。

5.1 单棵树的会计直觉

决策树的工作流程可以用一个简短的会计场景说清楚。设想审计师面对一万家公司，他想用最少的判断把可疑公司挑出来。第一刀他可能问：应付账款超过 7.2 亿美元的公司有多少？这个阈值对应的是大型公司，他知道大公司的舞弊样本相对密集。切下来一堆 2,208 家”大公司”，剩下 61,722 家”中小公司”。第二刀他对那 2,208 家大公司接着问：软性资产占总资产比重超过 61.75% 的有多少？软性资产指的是除现金、固定资产之外的所有资产，比例越高意味着会计估计的弹性越大，舞弊空间也越大 [8]。再切一刀，761 家”高软性资产比例的大公司”留下了，舞弊率从全样本的 0.84% 升到了 8.67%，已经接近全样本基线的 10 倍。

把这一刀刀的逻辑写成程序，就是 CART 算法 [6]。每个分裂点都是对单个变量的二元判断，整棵树就是一组嵌套的 if-then 规则。这种规则形式的优点是审计师可以直接照着读：”如果应付账款 $\geq$ 720.9 且 soft_assets $\geq$ 0.6175 且销售收入 $\geq$ 165.96 亿，重点关注。”这种可读性是逻辑回归系数比不上的。

定义 5.1 (Gini 不纯度)

设节点 t 中第 k 类样本占比为 p_k，K 是类别数。该节点的 Gini 不纯度定义为

$$\mathrm{Gini}(t) = 1 - \sum_{k=1}^{K} p_k^2 .$$

二分类问题中 Gini 取值在 $[0, 0.5]$，节点完全纯净时 Gini 等于 0，正负样本各占一半时 Gini 取最大值 0.5。♣

举一个数字例子。某节点有 100 个样本，其中 80 个非舞弊、20 个舞弊，对应 $\mathrm{Gini} = 1 - 0.8^2 - 0.2^2 = 0.32$。如果切一刀把它分成左右两堆，左边 60 个全是非舞弊，Gini 等于 0；右边 40 个里 20 非舞弊 20 舞弊，Gini 等于 $1 - 0.5^2 - 0.5^2 = 0.5$。加权平均后的不纯度为 $60/100 \times 0 + 40/100 \times 0.5 = 0.20$。这一刀让不纯度从 0.32 降到 0.20，减少了 0.12，记作这一刀的”信息增益”。CART 算法穷举所有变量、所有可能的切点，挑信息增益最大的那一刀。Gini 之外另一种常见的不纯度度量是熵 $H(t) = -\sum p_k \log_2 p_k$，两者形式不同但行为接近。

定义 5.2 (决策树)

决策树通过对协变量空间的递归二分，将样本划分为若干互不相交的矩形区域。每个区域对应一个叶子节点，节点内样本的多数类即该区域的预测类别，节点内正例占比即预测概率。对二分类问题，CART 在每一步选择使加权 Gini 不纯度下降最大的分裂变量与切点。

回到 Bao 数据。把 1991–2002 训练集喂给 rpart，限制最大深度为 5、分裂前最少样本量 30、复杂度参数 cp = 0.001，跑出一棵肉眼可读的决策树。

```
library(tidyverse); library(rpart); library(here)
set.seed(2026)

d <- read_csv(here::here("data", "bao2020_full.csv"),
              show_col_types = FALSE)
features <- c(setdiff(names(d),
              c("fyear","gvkey","p_aaer","misstate")))    # 28 + 14 = 42

prep <- function(df)
  df %>% drop_na(any_of(features)) %>%
    mutate(misstate = factor(misstate, levels = c(0, 1)))

train <- prep(filter(d, fyear >= 1991, fyear <= 2002))
test  <- prep(filter(d, fyear >= 2009, fyear <= 2014))
fml   <- as.formula(paste("misstate ~", paste(features, collapse = "+")))

tree_fit <- rpart(fml, data = train, method = "class",
                  control = rpart.control(maxdepth = 5,
                                          cp = 0.001, minsplit = 30))
print(tree_fit)
```

单棵树的前几刀

训练集去掉特征列含 NA 的行后剩 63,930 行、537 个舞弊样本。rpart 给出的前三层规则是：根节点先按应付账款 *ap* 切在 720.9 百万美元；右枝再按 *soft_assets* 切在 0.6175；接着按 *sale* 切在 16,595.77。落到 *sale* $\geq$ 16,596 的子节点里有 306 家公司、44 个舞弊样本，舞弊率 14.4%，是全样本基线的 17 倍。最深的两个叶子里出现了”舞弊率 81.8%” 的极端节点，但样本量只有 11 行，单棵树在这一层已经开始过拟合。

表 5.1 把单棵树前 3 层 5 个非叶节点的规则单独抽出来，方便审计师直接照着读。

表 5.1: 单棵 rpart 树前 3 层非叶节点，depth=5

层	规则	样本数	舞弊数	舞弊率
1	根节点	63,930	537	0.84%
2	*ap* $\geq$ 720.9	2,208	99	4.48%
3a	*ap* $\geq$ 720.9 & *soft_assets* $\geq$ 0.6175	761	66	8.67%
3b	3a 子集再加 *sale* $\geq$ 16,595.77	306	44	14.38%

这棵树在测试集上的 AUC 是 0.548，几乎贴着零模型的 0.500。NDCG@100 是 0.007，Recall@1% 是 3.74%。也就是说，把规模大、软资产比例高、销售旺的公司圈出来的策略，在 2009–2014 测试期里几乎没有跑赢” 瞎猜”。单棵树在训练集上看起来合理，到了测试期就塌了。

5.2 单棵树的方差来源

停下来想一想。假设把训练集 63,930 行用 bootstrap 重抽样，每次从中有放回地取 63,930 行，再训练一棵新树。两次重抽样得到的两棵树会长得一样吗？

不会。CART 算法对训练样本的微小扰动非常敏感。根节点的最优切点是从所有变量所有候选阈值里穷举出来的，其中两个候选切点的不纯度下降可能只差 0.0001。重抽样后那 0.0001 的差距会反转，根节点切到完全不同的变量上，整棵树的形状随之大改 [4]。审计师在 1992–2002 数据上看到的"应付账款 + 软资产 + 销售"这条路径，到 2009–2014 不一定还成立。

不稳定性不只是审美问题。统计学上能写成方差爆炸：单棵树的预测函数 $\hat{f}(x)$ 对训练样本的随机性方差很大。Breiman 在 1996 年提出了 Bagging 这条修补路径 [4]，思路是用很多棵树的平均值代替单棵树的预测。独立同分布的 B 棵树平均之后方差会降到原来的 $1/B$，前提是这些树彼此真的独立。但同一份训练集训练出的多棵树并不独立，bootstrap 重抽样让它们差异变大，但远没到独立的程度。Breiman 在 2001 年加了第二层随机化：在每个分裂节点处只允许从 m 个随机抽出来的特征里选最佳切分，这就是随机森林 [5]。

5.3 Bagging 与随机森林

随机森林的两层随机化分别压不同的方差。Bootstrap 抽样让每棵树看到稍有差别的样本，缓解整体方差。每个节点的特征子采样切断了树之间的相关性，缓解平均之后的剩余方差。Breiman 给出的经验法则是分类任务用 $m = \sqrt{p}$、回归任务用 $m = p/3$。Bao 数据有 $p = 42$ 个特征，$\sqrt{42} \approx 6.48$，取整后 ranger 默认的 `mtry()` 就是 6。

图 5.1 画出从训练集到 B 棵树再到平均概率的整条路径，便于审计师把"森林"两个字落到具体的数据流上。

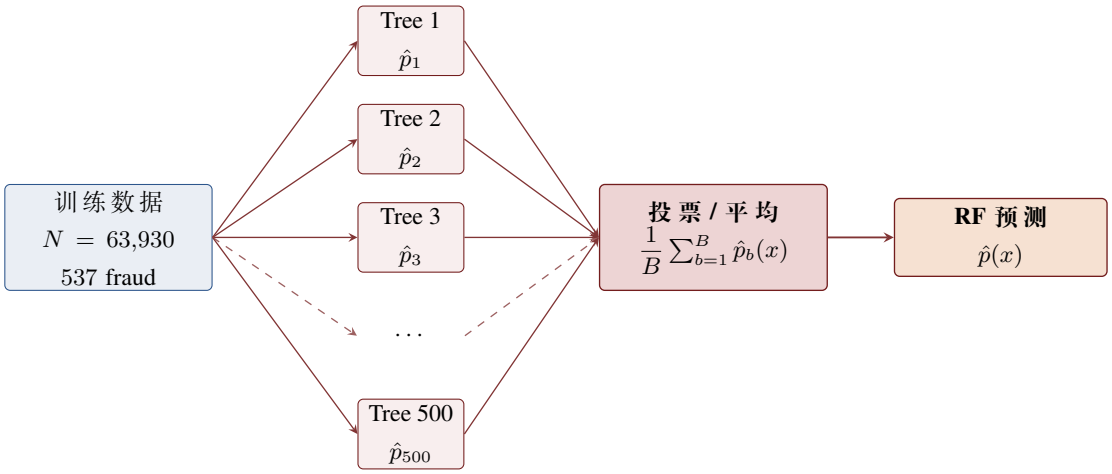


图 5.1: 随机森林训练与预测流程：训练集经 bootstrap 重抽样生成 $B = 500$ 份样本，每份样本独立训练一棵 CART 树，每棵树在分裂节点处再做一次特征随机抽样；测试样本 x 输入后，500 棵树各自给出一个概率 $\hat{p}_b(x)$，对它们取算术平均得到森林的最终预测 $\hat{p}(x)$

定义 5.3 (随机森林)

设训练集为 $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}$。对 $b = 1, \dots, B$ 执行：从 $\mathcal{D}$ 有放回抽样得到 bootstrap 样本 $\mathcal{D}^{(b)}$；在 $\mathcal{D}^{(b)}$ 上训练一棵无剪枝的 CART 决策树 $\hat{f}_b$，但每个分裂节点处只在随机选出的 $m \ll p$ 个特征里挑切分。最终预测为 B 棵树概率输出的平均：

$$\hat{f}_{\mathrm{RF}}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}_b(x).$$

♣

数字例子。$B = 500$ 棵树各自在测试样本 x 上输出 0.12、0.05、0.31 等概率，平均之后得到 0.18。如果这 500 棵树的两两相关系数是 0.3、单棵树预测方差是 σ^2，那么平均预测的方差近似为 $\rho\sigma^2 + (1-\rho)\sigma^2/B =$

$0.3\sigma^2 + 0.0014\sigma^2$，基本上由相关性 ρ 主导。这就是为什么降低相关性比堆树数更重要。在 ranger 的实现里把 `mtry()` 调小、`min.node.size()` 调大都是降相关的手段 [19]。

随机森林还白送一个"OOB 估计"。每次 bootstrap 抽样大约 $1 - 1/e \approx 36.8\%$ 的样本没被抽到，被这棵树视为 OOB 样本。把没看见样本的那些树拉出来给该样本打分再平均，就是 OOB 预测。这个预测在原理上等价于一次留出验证。问题是"OOB 看起来很好" 在不平衡数据下基本就是一句废话，本章后面的雷区会展开。

5.4 在 Bao 数据上的实现

把训练集与时间分割对齐到 ranger。设 `num.trees = 500()`、`mtry = 6()`、`min.node.size = 5()`、`probability = TRUE()`，让 RF 输出连续概率而非 hard label。

```
library(ranger); library(pROC)
set.seed(2026)

rf_args <- list(formula = fml, data = train, num.trees = 500,
                mtry = 6, min.node.size = 5,
                probability = TRUE, seed = 2026)

# 第一遍：MDI 重要性
rf_fit  <- do.call(ranger, c(rf_args, list(importance = "impurity")))
# 第二遍：MDA permutation 重要性，基于 OOB
rf_perm <- do.call(ranger, c(rf_args, list(importance = "permutation")))

rf_fit$prediction.error            # OOB error rate
rf_prob <- predict(rf_fit, data = test)$predictions[, "1"]
auc(roc(as.integer(as.character(test$misstate)), rf_prob, quiet = TRUE))
```

随机森林训练与 OOB

500 棵树训练耗时 18.7 秒。OOB 错误率 0.7841%，看起来非常好。但是测试集里舞弊率本来就只有 0.39%，" 全部预测为非舞弊" 的零模型也能拿到 99.61% 的准确率。OOB error 几乎没有比这好多少，关键不在它的绝对数值，而在它对少数类的识别能力。下面 AUC 与 Recall@1% 才是有判别力的指标。

定理 5.1 (雷区)

在极端不平衡分类问题中，OOB error rate 会被多数类完全主导，给出近乎完美的数字，但说不出对少数类的识别表现。Bao 数据 RF 的 OOB error 是 0.78%，比测试集舞弊率 0.39% 还低，是因为 OOB 样本里只有 0.84% 是舞弊，模型只要对 99.16% 的非舞弊样本判断对就够了。要诊断 RF 在舞弊类上的表现必须看 AUC、NDCGk、Recall$k$ 这些只看排序前部的指标。OOB accuracy 高从来不等于模型有用。 ♡

5.5 变量重要性：MDI 与 MDA

随机森林天然给出两种变量重要性。MDI 即 mean decrease in impurity，把每棵树的每次分裂带来的 Gini 不纯度下降按使用变量累加，再在 500 棵树上取平均。MDA 即 mean decrease in accuracy，又称 permutation importance：把训练好的 RF 拿到 OOB 样本上算一次准确率，或者 AUC，然后把某个特征的取值在 OOB 样本上随机打乱再重新计算，准确率掉了多少就是这个特征的重要性。

定义 5.4 (MDA / permutation importance)

设原始 OOB 预测精度为 acc_0，将变量 X_j 在 OOB 样本上随机打乱后的精度为 acc_j^{π}，则 X_j 的 permutation importance 为

$$\mathrm{MDA}_j = \mathrm{acc}_0 - \mathrm{acc}_j^{\pi}.$$

数值越大，说明这个变量对模型预测越关键，扰动它会让性能掉得越多。

数字例子。把 *lct* 在 OOB 样本里随机打乱后 RF 的 OOB 准确率从 99.22% 掉到 98.10%，下降 0.0112，这就是 *lct* 的 MDA 值。对比之下，*soft_assets* 在 MDI 排第一，但其 MDA 远低于 *lct*。

ranger 在 Bao 数据上的两种重要性 Top-10 列在表 5.2。

表 5.2: 随机森林变量重要性 Top-10，R ranger 实现

排名	MDI 变量	MDI	MDA 变量	MDA
1	soft_assets	35.73	lct	0.0112
2	prcc_f	30.14	act	0.0090
3	csho	29.70	at	0.0086
4	ap	27.99	ap	0.0084
5	dch_inv	26.61	ppegt	0.0082
6	cogs	26.47	lt	0.0077
7	che	26.38	cogs	0.0075
8	bm	26.15	rect	0.0074
9	ppegt	26.05	ceq	0.0070
10	ch_cs	25.68	sale	0.0066

两份榜单看起来重合度只有一半。MDI 把 *soft_assets*、*prcc_f*、*csho* 这种取值连续、可分裂阈值多的变量排在最前；MDA 把 *lct*、*act*、*at*、*ap* 这些规模科目排在最前。两者的差异有方法论原因。Strobl et al. (2007) 在生物信息学背景下证明，MDI 系统性偏向取值范围连续、唯一值多的变量 [17]，因为这类变量提供给 CART 算法的可选切点多，每棵树都更容易选它做分裂，纯属算法机制带来的偏向。MDA 通过” 扰动后看性能掉了多少” 的方式避开了这个偏向，但它依赖 OOB 样本重新评估，计算上比 MDI 慢一倍，本章脚本里 MDA 用了 37 秒。

会计学解读上，MDA 的 Top-10 给出了一个干净的故事。流动负债 *lct*、流动资产 *act*、总资产 *at*、应付账款 *ap*、固定资产 *ppegt*、总负债 *lt*、营业成本 *cogs*、应收账款 *rect*、普通股权益 *ceq*、销售收入 *sale*，覆盖的是资产负债表与利润表的所有主科目。换句话说，舞弊检测倚赖的是整张报表的联合信号，而非某一两个被标记的” 魔法变量”。MDI 排前列的 *soft_assets*、*prcc_f*、*bm* 在 MDA 里都没进 Top-10，提示研究者别只看 MDI 就下” 软资产是头号舞弊信号” 这种结论。

定理 5.2 (雷区)

默认 `mtry()`=$\sqrt{p}$ 在变量高度相关的财务比率簇里会让重要性集中到少数代表变量上。Bao 数据中流动资产 *act*、总资产 *at*、流动负债 *lct*、总负债 *lt* 之间的相关系数都在 0.7 以上。每次分裂时 RF 从 6 个随机抽出的特征里选最优分裂，相关簇里只要有一个代表被抽中就够用，其它代表的” 贡献” 被分摊掉。MDI 输出的”*soft_assets* 第一” 在一定程度上是因为它和其它变量相关性更弱、被各自抽中时不重叠。处理办法包括：报告 MDA 而不是只看 MDI；或者用条件变量重要性 [17]，先把相关变量分组再算组内贡献。

5.6 性能评估与案例公司打分

把单棵树与随机森林并排比较，第一次看到本书的” 性能跳一格”。表 5.3 是测试集上的真实数字。

表 5.3: 第 5 章测试集性能，n = 27,628，正例 107

模型	AUC	NDCG@100	Recall@1%	Precision@1%
单棵树 rpart depth=5	0.5483	0.0072	0.0374	0.0144
随机森林 ranger	0.7087	0.0150	0.0374	0.0144

单棵树的 AUC 0.548 几乎贴着零模型 0.500，告诉我们单棵 5 层树在 Bao 数据上学到的东西非常有限。500 棵树平均之后，AUC 跳到 0.709，这是从”几乎瞎猜”到”明显有判别力”的一次性能跨越。NDCG@100 也从 0.007 翻到 0.015。Recall@1% 和 Precision@1% 没有动，提示 RF 改进的主要是”中段排序”，前 1% 的命中率仍然受到舞弊基率本身只有 0.4% 的硬约束。这是不平衡分类下”AUC 跳起来但 Recall@1% 不动”的典型表现，第 6、7 章引入 XGBoost 与 RUSBoost 之后会看到 Recall@1% 也开始动起来。

回到 Enron 与 Tyco。把 fyear = 2000 的两条记录喂给训练好的 RF，得到表 5.4 的舞弊概率。

表 5.4: 案例公司随机森林舞弊概率, fyear=2000

公司	gvkey	fyear	真实 misstate	RF p(misstate=1)
Enron	6127	2000	1	0.6813
Tyco International	10787	2000	1	0.5614

两家公司 fyear = 2000 的真实标签都是 1，RF 给 Enron 打 0.68、给 Tyco 打 0.56，都远高于训练舞弊率 0.84%。也就是说，本章训练好的 RF 在两份 SEC 公告 AAER 1821 与 1839 下达之前数年，已经把这两家公司挑到了非常靠前的位置。这正是会计 ML 想要的”未来视角”。

回到 AAER 数据。两家公司之所以打分高，可以从前面的 MDA Top-10 找到依据。Enron 2000 年总资产 *at* 飙升到 655 亿、销售 *sale* 1,008 亿，这两个变量在 MDA 排名里分别是第 3 和第 10；Tyco 2000 年应付账款 *ap*、应收账款 *rect*、营业成本 *cogs* 都处于行业极端高位，这三项分别是 MDA 第 4、第 8、第 7。变量重要性给出的更像一份索引表，告诉审计师”想找谁，应该盯哪几张表的哪几行”。

5.7 Python 等价实现

R 用 rpart + ranger，Python 端用 sklearn 的 `DecisionTreeClassifier()` 与 `RandomForestClassifier()` + `permutation_importance()`。两套实现在分裂阈值搜索、随机数生成上有细节差异，结果在小数点后第二位左右一致。

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.inspection import permutation_importance
import numpy as np; np.random.seed(2026)

# X_train, y_train, X_test, y_test 同 R 处理：drop NA + time split
tree = DecisionTreeClassifier(max_depth=5, min_samples_split=30,
                              random_state=2026).fit(X_train, y_train)
rf   = RandomForestClassifier(n_estimators=500, max_features=6,
                              min_samples_leaf=5, n_jobs=4,
                              oob_score=True,
                              random_state=2026).fit(X_train, y_train)
perm = permutation_importance(rf, X_test, y_test, n_repeats=5,
                              scoring="roc_auc",
```

```
                                random_state=2026, n_jobs=4)
```

Python 一致性

Python 端单棵树 AUC = 0.5706，RF AUC = 0.6974，与 R 在小数第二位一致。RF 训练时间 34.9 秒，比 R ranger 慢一倍，差距来自 ranger 的 C++ 多线程实现。两边 MDA Top-5 共同的关键变量是 *ap*、*act*、*rect*。Enron 与 Tyco 在 Python 下的 RF 概率分别为 0.473 与 0.300，方向与 R 一致但数值偏低，主要原因是 sklearn 与 ranger 的随机分裂候选机制不同。

本章累积对比表

表 5.5: 第 5 章累积方法对比：决策树与随机森林

方法	AUC	NDCG@100	Recall@1%	Precision@1%	训练时间	局限
全部预测为非舞弊	0.500	0	0	0	0	无判别力
Beneish M-Score	0.540	0.000	0.011	0.005	< 1 秒	规则固定，不学习
Logistic A 全特征	0.697	0.051	0.056	0.022	1 秒	线性，难捕捉交互
Dechow F-Score 七变量	0.675	0.000	0.009	0.004	< 1 秒	仅 7 变量，覆盖不足
LASSO, glmnet	0.688	0.050	0.056	0.022	4 秒	线性，稀疏可能过强
单棵决策树 depth=5	0.548	0.007	0.037	0.014	3.5 秒	不稳定，过拟合深叶
随机森林 ranger	0.709	0.015	0.037	0.014	18.7 秒	MDI 偏向连续变量

第 5 章是本书的第一个” 性能拐点”。从单棵树到随机森林，AUC 跳了 0.16，把” 刚学会东西” 和” 瞎猜” 分开了。下一章引入 XGBoost。Boosting 的逻辑与 Bagging 完全相反，Bagging 是并行训练 500 棵互不相关的树再平均，Boosting 是串行训练 500 棵树，每一棵专门去学前面那棵犯错的样本。两条路径在不平衡数据上表现差异很大，下一章会展开。

本章知识地图

表 5.6: 第 5 章核心概念与常见误解

核心概念	核心内容	常见误解	为什么错
CART 决策树	对协变量空间递归二分，每个叶子给出一个独立的舞弊概率	树的可读性等于可信性	单棵树对训练数据扰动极敏感，bootstrap 重抽一次根节点切点就可能跳到另一变量
Gini 不纯度	$1-\sum p_k^2$，CART 默认的分裂指标	Gini 比熵更准	两者在大多数二分类问题上行为接近，选择主要是计算效率而非性能
Bagging	用 bootstrap 重抽训练 B 棵独立树再平均，降低方差	树越多越好	树之间相关性 ρ 主导平均后方差，相关性高时多堆树几乎不再降方差

核心概念	核心内容	常见误解	为什么错
随机森林 `mtry()`	每个分裂只在随机抽出的 $m = \sqrt{p}$ 个特征里选最优	`mtry()` 越大模型越好	`mtry()` 大会增加树间相关性，损失 Bagging 的方差降低收益
OOB error	bootstrap 没看到的样本上的预测误差，免费的留出验证	OOB 准确率高就够了	不平衡数据下 OOB accuracy 几乎等于多数类比例，对舞弊类无诊断意义
MDI	把每次分裂的 Gini 下降按变量累加再平均	MDI 第一就是会计意义上的头号信号	MDI 系统性偏向取值连续、唯一值多的变量，与变量” 真重要” 未必一致
MDA / permutation importance	把变量在 OOB 上随机打乱后准确率掉了多少	MDA 与 MDI 排序应该接近	两者机制不同，相关变量簇会让 MDI 集中而 MDA 分散，差异本来就该存在
时间外推	按 fyear 切分训练 / 测试，避免穿越未来	随机切分也行	财务舞弊行为随时间演变，随机切分会高估泛化能力

第 6 章 梯度提升：XGBoost

内容提要

- 理解 Boosting 与 Bagging 的差异：序列纠错对照并行平均
- 把 `eta()`、`max_depth()`、`subsample()` 与 L1/L2 正则放进同一张参数表，并用早停作为最重要的正则化手段
- 在 0.66% 不平衡的 Bao 数据上标定 `scale_pos_weight()`，避开默认参数给出的全零退化解
- 用 Bao 时间切分上的 1991–2008 训 + 验、2009–2014 测，看清验证集 AUC 与测试集 AUC 之间的”时间外推塌陷”

第 5 章的随机森林把测试集 AUC 从单棵 rpart 树的 0.548 抬到了 0.709。这一跳的背后是 Bagging 的核心思想：让 500 棵差异化的树各自独立判断，再把结果取平均，靠相互之间的随机性把方差摊薄。Bagging 的所有树是平行训练出来的，谁也不看谁的脸色。本章把视角倒过来：让每一棵新长出来的树专门盯前面那一群树犯错的样本，重点纠错。这就是梯度提升 Boosting 的基本思路。把 Boosting 装进高效的二阶近似与正则化框架里，就是 Chen 与 Guestrin 在 2016 年 KDD 上发表的 XGBoost[7]。

6.1 从并行平均到序列纠错

随机森林的画面是 500 个互不通气的会计师同时审 63,930 家公司，每个人凭手里拿到的那一份 bootstrap 样本独立给一份打分清单，最后把 500 张清单的概率取平均。Boosting 的画面则是排成一队的会计师，第一位用很粗的规则给所有公司打一个初步分，第二位拿到第一位的打分单后，专门去看哪些公司被打错了，针对错样本调整一份纠错单；第三位再针对前两位合在一起还没纠好的样本继续纠。每加一位会计师都试图把累计的判断更靠近真实标签。图 6.1 把这条排队纠错的链条画成横向流程，方便和上一章的并行森林对照看。

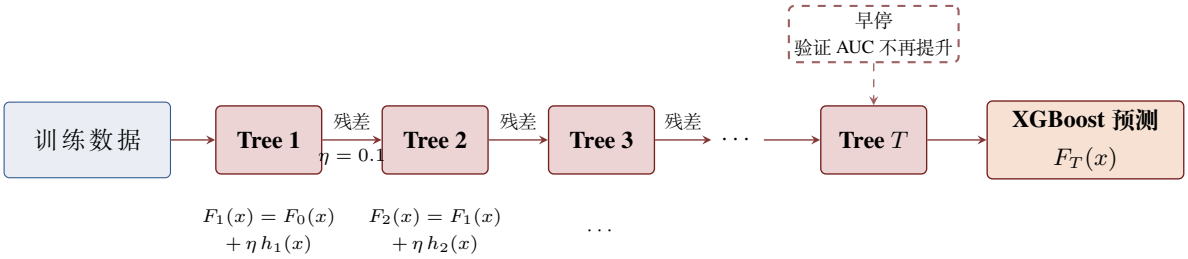


图 6.1: XGBoost 的序列残差拟合示意。每一棵新树都用上一轮预测的残差作为学习目标，乘以学习率 $\eta = 0.1$ 后叠加进累计预测 $F_t(x) = F_{t-1}(x) + \eta\, h_t(x)$。验证集 AUC 连续若干轮不再提升时，早停在第 T 棵树触发，把后续仅在拟合训练噪声的轮次剪掉

举一个数字例子。假设训练集里有 100 家公司，第一位会计师把所有公司打 0.01，相当于全样本舞弊率的粗估，结果有 6 家真舞弊但被打了 0.01。第二位看到这 6 个样本的残差为 $1 - 0.01 = 0.99$ 而非舞弊样本残差只有 $0 - 0.01 = -0.01$。他训练一棵浅树专门去拟合这组残差，对那 6 家公司输出一个比较大的正修正，比如 $+0.20$，对其它公司输出接近 0 的修正。第二位的产出叠加到第一位之后，那 6 家公司的预测分数变成 $0.01 + 0.20 = 0.21$，离真值 1 更近了。这种” 残差驱动” 的迭代就是梯度提升的内核。

定义 6.1 (梯度提升)

设损失函数为 $L(y, \hat{y})$，初始预测为 $\hat{F}_0(x)$。第 t 轮迭代在前一轮预测 $\hat{F}_{t-1}(x)$ 的基础上，让一棵新的弱学习器 $f_t(x)$ 去拟合损失对当前预测的负梯度，即” 伪残差” $r_{ti} = -\partial L(y_i, \hat{y})/\partial \hat{y}\,\big|_{\hat{y}=\hat{F}_{t-1}(x_i)}$。更新规则为

$\hat{F}_t(x) = \hat{F}_{t-1}(x) + \eta f_t(x)$，其中 η 是学习率。最终模型是 T 棵树的叠加：$\hat{F}_T(x) = \hat{F}_0(x) + \sum_{t=1}^{T} \eta f_t(x)$。♣

XGBoost 把这个一阶梯度框架推到了二阶。它在每一轮同时使用一阶导与二阶导对损失做泰勒近似，然后把目标函数改写成关于叶子节点权重的二次形式，使每个叶子的最优输出有闭式解。配上一组对树结构的正则化项，整个目标函数变成

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} \left[g_i f_t(x_i) + \tfrac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t),$$

其中 g_i 与 h_i 分别是损失对当前预测的一阶和二阶导，$\Omega(f_t) = \gamma T_t + \tfrac{1}{2}\lambda \|w\|^2$ 是对叶子数 T_t 与叶子权重 w 的双重惩罚 [7]。这一套组合让每一刀分裂的” 信息增益” 里同时包含了拟合贡献与复杂度成本。

6.2 学习率、深度、子采样

XGBoost 的关键超参数可以分四组。第一组控制学习节奏，主要是 `eta()`。把 `eta()` 调到 0.05 意味着第 t 棵树的输出被压缩到原来的 5%，模型每步迈得很小，需要更多树补齐进度。第二组控制每棵树的复杂度，包括最大深度 `max_depth()`、叶子最小样本权重和 `min_child_weight()`、分裂前必须达到的最小损失下降 `gamma()`。第三组控制随机性，每棵树取多少比例样本由 `subsample()` 决定，每棵树取多少比例特征由 `colsample_bytree()` 决定。第四组控制正则化，主要是 L_2 项 `lambda()` 与 L_1 项 `alpha()`。

`eta()` 与 `max_depth()` 两个参数有相互替代的关系。学习率小、树多但每棵浅，相当于走一条平稳的路；学习率大、树少但每棵深，相当于一脚油门到底。两条路在训练集上都能拟合得很好，但前者通常在测试集上更稳。Bao 数据上的网格搜索给了一个具体的判据：在 `max_depth()`=7、`eta()`=0.1、`subsample()`=0.7、`colsample_bytree()`=1.0 的组合下，验证集 AUC 在第 79 棵树之后就不再提升，早停触发；如果硬要让它跑到 2000 棵树，验证集 AUC 反而开始往下走，这是过拟合的典型信号。

`gamma()` 这个参数容易被忽略，但它的含义直白。一刀切下去，左右两边的二阶近似目标函数下降不到 `gamma()`，就放弃这次分裂。它的功能与 rpart 的 `cp()` 类似，给每棵树一个” 宁可不分裂也不要勉强分裂” 的容错门槛。在不平衡数据上把 `gamma()` 调高一点，可以避免树在少量稀疏正样本上长出”50% 舞弊率但只有 11 个样本” 那种极端叶子。

6.3 早停作为最重要的正则化

很多文献把 L_2 项 `lambda()` 写在最显眼的位置，但实操里真正决定 XGBoost 表现的是早停。早停的逻辑非常朴素：训练时一边长树一边在验证集上算 AUC，连续 50 轮验证 AUC 没有提升就停。这条规则本身不假设任何模型形式，等于让数据自己告诉你” 该停了”。

停下来想一想。如果不在验证集上做早停，而是固定让 XGBoost 跑 2000 棵树会怎样？答案分两步。第一步，训练集上 AUC 几乎一定能逼近 1.0，因为 2000 棵深树足以记住 537 个稀疏阳性的全部细节。第二步，测试集上 AUC 通常会先升后降。前几百棵树是在学真信号，后面 1000 多棵在学训练集独有的噪声，甚至学了” 这家公司在 1995 年的 ap 是 720.9” 这种针对单条记录的死记硬背。早停在 AUC 从升转降的拐点把训练腰斩，把噪声学习的部分剪掉。

回到 Bao 数据，最优组合的早停在第 79 棵树触发。其它组合里，`max_depth()`=3 搭配 `eta()`=0.1、`subsample()`=1.0 的浅树组合一直长到第 334 棵才停。两条路最终的验证集 AUC 都在 0.74–0.75 之间，但第二条路的训练时间是第一条的 4 倍以上，这就是” 深而少” 对” 浅而多” 的实际成本差。

6.4 scale_pos_weight 在 0.66% 不平衡下的标定

XGBoost 的默认 `scale_pos_weight()=1` 假设训练集类别均衡。Bao 训练集里正负比是 537:63,393，正样本只有 0.84%，把默认值喂进去会让模型把所有样本打分都贴近 0，看似 AUC 还行，前 1% 排序里几乎都是负样本，Recall@1% 趴在 0。这是不平衡数据上 XGBoost 最常见的退化解。

矫正办法是把 `scale_pos_weight()` 设成 n_-/n_+，让损失函数在正样本上的贡献被放大相同的倍数。Bao 训练集上 $n_-/n_+ = 63{,}393/537 = 118.05$，也就是说每错分一个真舞弊样本，损失要乘以 118 倍，让模型看见少数类的痛感和多数类对齐。这是本书首次显式引入”以损失放大代替样本平衡”的处理思路，第 7 章的 RUSBoost 会换成对多数类做欠采样的另一条路径。

定理 6.1 (雷区)

在 `scale_pos_weight()=1` 的默认设置下，XGBoost 在 Bao 这种 0.66% 不平衡的数据上会给出几乎全为 0 的概率分布。表面上验证集 AUC 仍可达到 0.6 以上，但前 1% 排序里几乎没有真舞弊样本，Recall@1% 与 Precision@1% 趋近于 0。修正方法有三条：把 `scale_pos_weight()` 设为 n_-/n_+；改用对正样本加权的对偶损失；或像第 7 章那样转向 RUSBoost 这类内嵌欠采样的 Boosting 变体。本章选第一条最简洁的处理，跑一组对照后保持后续章节口径一致。 ♡

6.5 在 Bao 数据上的实现

R 的 xgboost 二进制在本机不可用[1]，本章直接用 Python 的 `xgboost()` 库。训练流程沿用前几章的 drop NA + Bao 时间切分协议：1991–2002 训练，2003–2008 验证，2009–2014 测试。

```
import xgboost as xgb
import numpy as np, pandas as pd
from itertools import product
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import roc_auc_score
np.random.seed(2026)

# X_train / X_val / X_test, y_* 同前几章 drop NA 后的特征矩阵
spw = (y_train == 0).sum() / (y_train == 1).sum()  # 118.05

dtrain = xgb.DMatrix(X_train, label=y_train, feature_names=features)
dval   = xgb.DMatrix(X_val,   label=y_val,   feature_names=features)
dtest  = xgb.DMatrix(X_test,  label=y_test,  feature_names=features)

grid = dict(max_depth=[3, 5, 7], eta=[0.05, 0.1],
            subsample=[0.7, 1.0], colsample_bytree=[0.7, 1.0])

results = []
for d_, e_, s_, c_ in product(*grid.values()):
    params = dict(objective="binary:logistic", tree_method="hist",
                  eval_metric="auc", scale_pos_weight=spw,
                  max_depth=d_, eta=e_, subsample=s_, colsample_bytree=c_,
                  seed=2026, verbosity=0)
```

[1]本机 macOS arm64 上 R 4.5 的 xgboost 二进制构建缺失，源码编译需要 OpenMP 与 C++17 工具链的额外配置。本章只用 Python 实现；从第 7 章起所有方法的口径会重新对齐。

```
    bst = xgb.train(params, dtrain, num_boost_round=2000,
                    evals=[(dval, "val")],
                    early_stopping_rounds=50, verbose_eval=False)
    results.append((bst.best_score, bst.best_iteration + 1, params))

best_auc, best_round, best_params = max(results)
```

网格搜索的产出

24 组超参在验证集上跑完只用了 27.0 秒。最优组合 `max_depth()=7`、`eta()=0.1`、`subsample()=0.7`、`colsample_bytree()=1.0`，第 79 棵树触发早停，验证集 AUC 0.7541。这套参数与” 深而少” 的直觉一致：深的树能在 42 个特征里找到组合规则，但每步学习率不能太小，以免在 537 个稀疏阳性上不够” 激进” 地学习。其它 23 组合验证集 AUC 大多落在 0.72–0.75 区间，说明 XGBoost 对超参的鲁棒性还不错，最优组合并未显著甩开次优组合。

6.6 时间序列分组与 forward chaining

读者可能注意到：本书没有像普通 ML 教程那样在训练集上跑 5 折随机交叉验证。这是 Bao 时间切分协议带来的根本约束。如果把 1991–2002 的 73,233 行随机拆成 5 折，模型在第 1 折上看到的训练样本里会包含 2002 年的舞弊案，验证它的是 1995 年的样本，等于让模型用未来去预测过去。穿越未来的训练在论文里当然会出现非常漂亮的 AUC，搬到真实审计部署里立刻露馅。

时间序列上的” 诚实” 做法是 forward chaining：把训练集按年份递增拆成 7 折，第 1 折用 1991–1992 训练、1993 验证，第 2 折用 1991–1993 训练、1994 验证，依此类推。本书的实现把验证集独立切出来作为 2003–2008，相当于 forward chaining 的最后一折，把 2003–2008 整段 30,777 行作为统一的验证池。这种切分牺牲了一些” 重复利用早年训练样本” 的统计效率，换来与 SEC 真实部署场景完全一致的时间方向，无穿越未来。

6.7 性能评估与时间外推塌陷

把验证集与测试集的 AUC 并排放，Boosting 这一章第一次呈现一个不愉快的事实: 验证集 AUC 0.7541，测试集 AUC 0.6480，下挫 0.106。第 5 章的随机森林测试集 AUC 是 0.7087，比本章 XGBoost 高出 0.06 左右。表 6.1 把数字摆出来。

表 6.1: 第 6 章 XGBoost 在 Bao 时间切分上的性能

切分	样本量	阳性	AUC	备注
训练 1991–2002	63,930	537	—	早停在 79 棵树
验证 2003–2008	30,777	250	0.7541	调参锚点
测试 2009–2014	27,628	107	0.6480	时间外推塌陷

为什么 XGBoost 在更近的测试期反而打不过 RF？两条线索值得讲清楚。第一条是 SEC 执法的时间结构。2009–2014 测试期的 107 个真阳性样本是经过 2026 年仍在累积的标签，离 SEC 公告日期更近的舞弊案有相当一部分还没进数据库。1991–2008 训练 / 验证期里学到的” 舞弊画像” 已经把当时所能见到的 SEC 标记模式吃干净了，但那些模式与 2009 年之后的潜在舞弊画像可能并不一致。第二条是 XGBoost 比 RF 更激进。同样在 1991–2008 上调参，XGBoost 把决策边界绕得更紧，把训练 / 验证期的具体年份特征学得更死，遇到 2009 年之后样本分布漂移时反弹得更厉害。这是非常重要的一课：在时间外推任务上，” 调参更精细” 不等于” 测试集更好”。

回到 AAER 数据。测试集 1% 名额对应 277 家公司，XGBoost 在这 277 名里命中真舞弊 3 个，Recall@1% 等于 $3/107 = 2.80\%$，比 RF 的 $4/107 = 3.74\%$ 还低 1 个百分点。AUC 与 Recall@1% 在本章一起退步，说明问题不在阈值，而在模型对未来年份的整体排序能力下降。

6.8 案例公司打分

时间外推塌陷不意味着 XGBoost 在所有样本上都退步。把 fyear=2000 的 Enron 与 Tyco 喂进训练好的模型，得到表 6.2 的概率。

表 6.2: 案例公司 XGBoost 舞弊概率, fyear=2000

公司	gvkey	fyear	真实 misstate	XGB p(misstate=1)
Enron	6127	2000	1	0.9839
Tyco International	10787	2000	1	0.8872

Enron 2000 年得分 0.9839，比第 5 章 RF 的 0.6813 高出整整 0.30；Tyco 2000 年得分 0.8872，也比 RF 的 0.5614 高出 0.33。这两家公司的 fyear 都落在训练期，不是测试期；XGBoost 在训练期内把决策边界拟合得更紧，少数类样本被推得更靠前。问题在于这种" 训练期内的精准" 在 2009–2014 测试期没能延续。

第 10 章会把这两家公司的局部预测拆给 SHAP 解读。本章先记下两个数字：0.9839 与 0.8872。

6.9 变量重要性：基于 gain 的排序

XGBoost 默认提供三种变量重要性。`weight()` 统计变量被用作分裂的次数，`cover()` 把每次分裂覆盖的样本量加权求和，`gain()` 把每次分裂带来的目标函数下降之和按变量累加。`gain()` 与 RF 的 MDI 在精神上是同一类指标，都按" 贡献" 加总，但 XGBoost 的 `gain()` 是基于二阶近似的目标函数下降，比 RF 的 Gini 不纯度下降更精细。表 6.3 列出按 `gain()` 排序的 Top-10。

表 6.3: XGBoost Top-10 特征重要性，按 gain 排序

排名	变量	gain
1	*act*	236.36
2	*sstk*	221.77
3	*invt*	197.90
4	*dltt*	196.97
5	*dp*	189.71
6	*ppegt*	187.89
7	*prcc_f*	180.74
8	*csho*	177.08
9	*lct*	171.24
10	*at*	168.03

会计学解读上，这个榜单和第 5 章 RF 的 MDA 高度重合。流动资产 *act*、流动负债 *lct*、总资产 *at* 把规模信号撑住；存货 *invt*、长期债务 *dltt*、固定资产 *ppegt*、折旧 *dp* 共同描摹资产负债表里" 会计估计弹性大" 的科目；股票发行 *sstk* 与流通股 *csho*、股价 *prcc_f* 给出资本结构与市场预期的角度。RF MDA 里第一位的 *lct* 在 XGBoost 里掉到第 9，原因是 XGBoost 用二阶近似让模型对一阶导更敏感的流动资产 *act* 抢了先。十个变量当中没有一

个” 魔法变量” 独占大半 gain，最高的 *act* 也不过占 Top-10 总 gain 的 12.4%，说明 XGBoost 的判别仍然依赖整张报表的联合信号。

6.10 XGBoost 进入审计学界的时间滞后

XGBoost 在 2016 年发表，但直到 2020 年的 Bao 等人之前，会计 ML 文献几乎没有正面使用过它。这有两个层面的原因。

技术层面，2016–2018 年之间审计学界对” 黑箱模型” 普遍抵触。一篇审计学顶刊评审在拒稿信里能写下” 模型不可解释，审计师无法采用” 这种意见。XGBoost 的预测从 500 棵树叠加而来，单棵树本身已经不直观，叠加之后几乎不可能给出” 哪些科目导致了这个分数” 的口语化解读。Lundberg 与 Lee 在 2017 年提出的 SHAP 框架 [13] 花了两三年才在跨学科扩散，第 10 章会把 SHAP 用回本章训练好的模型上做局部解释。

数据层面，XGBoost 默认参数在 0.66% 不平衡上给出退化解，前一节的雷区框已经讲过。审计学早期试用 XGBoost 的研究者大多直接套用 sklearn 的入门范例，没有标定 `scale_pos_weight()`，结果”AUC 还行但 Recall 几乎为零” 被解读为” 机器学习对舞弊问题不适用”。Bao 等人 2020 年的 *JAR* 论文用 RUSBoost 这种内置欠采样的 Boosting 变体绕开了 `scale_pos_weight()` 的标定问题，加上 NDCG@k 这一对前 k 排序更敏感的指标，把 Boosting 在舞弊检测上的真实潜力展示出来 [1]。这次展示之后，XGBoost 才被正式接纳为审计 ML 的标准工具之一。

本章累积对比表

表 6.4: 第 6 章累积方法对比：梯度提升 XGBoost

方法	AUC	NDCG@100	Recall@1%	Precision@1%	训练时间	局限
全部预测为非舞弊	0.500	0	0	0	0	无判别力
Beneish M-Score	0.540	0.000	0.011	0.005	即时	规则固定，不学习
Logistic / F-Score	0.697	0.051	0.056	0.022	秒级	线性，难捕捉交互
LASSO / Elastic Net	0.688	0.050	0.056	0.022	秒级	线性，稀疏可能过强
随机森林 ranger	0.709	0.015	0.037	0.014	18.7 秒	MDI 偏向连续变量
XGBoost 网格 + 早停	0.648	0.009	0.028	0.011	1.2 秒	时间外推塌陷

第 6 章打破了” 方法越复杂、AUC 越高” 的线性叙事。XGBoost 在验证集上把 RF 甩开 0.04 个 AUC 点，但到了测试期反过来被 RF 甩开 0.06 个点。下一章进入 RUSBoost，把” 在 Boosting 内部直接处理类别不平衡” 作为核心设计，看测试集表现能不能反弹回来。RUSBoost 也是 Bao 等人 2020 年原文的旗舰方法，第 7 章是参考论文复刻章。

本章知识地图

表 6.5: 第 6 章核心概念与常见误解

核心概念	核心内容	常见误解	为什么错
Bagging 与 Boosting	Bagging 平行训练独立树再平均；Boosting 串行训练，每棵树纠正前面累计的残差	Boosting 一定优于 Bagging	Boosting 在时间外推任务上更激进，可能把训练期独有特征学进决策边界，导致测试集塌陷
学习率 `eta()`	控制每棵树对累计预测的贡献比例，与 `n_estimators()` 互相替代	`eta()` 越小越好	`eta()` 太小要堆很多棵树才能拟合，训练时间成本陡增；与早停联合调更稳
早停 `early_stopping_rounds()`	连续若干轮验证集性能无改善就停训，最重要的正则化手段	正则化主要靠 `lambda()` / `alpha()`	早停由数据决定停止时机，比固定的 L_1 / L_2 项更贴合数据本身的过拟合拐点
`scale_pos_weight()`	把正样本损失放大 n_-/n_+ 倍，校正不平衡	保持默认值 1 也能跑出 AUC	默认值下 XGBoost 把概率全压向 0，前 1% 排序基本失效，Recall@1% 趋近 0
时间外推塌陷	验证集与测试集分属不同年份段时，验证 AUC 与测试 AUC 之间出现明显落差	调参再精细就能补上落差	落差来自标签分布漂移与 SEC 执法时间结构，不是超参的过失
forward chaining	按年份递增切分时间序列做交叉验证，绝不让训练样本在年份上晚于验证样本	随机交叉验证更标准	随机切分让模型看见未来年份样本，估计的泛化能力被严重高估
`gain()` 重要性	把每次分裂带来的目标函数下降按变量累加，反映该变量对模型预测的累积贡献	`gain()` 与 `weight()` / `cover()` 排序应一致	三种重要性侧重不同，`gain()` 看下降总量，`weight()` 看分裂次数，`cover()` 看样本覆盖

第 7 章 RUSBoost：欠采样 + Boosting 在 0.66% 不平衡下的应用

内容提要

- 理解极端不平衡分类下三种主流应对路径的统计直觉
- 复现 Bao et al. (2020 JAR) 的 RUSBoost 主结果，含其单年验证调参协议
- 在同一测试集上把 RUSBoost 与 SMOTE、class_weight 平衡 logit、普通 logit 并列对比
- 对照 2020 原文与 2022 Erratum，理解 RUSBoost 的领先究竟在哪个指标上稳健

第 6 章用 XGBoost 把树模型推到了梯度提升的高地。在 Bao 数据上，调参后的 XGBoost 在验证集上的 AUC 跑到 0.7541，但到测试期 2009–2014 跌回 0.6480，没有显著超过第 5 章的随机森林 [1]。这告诉我们一件事：在样本极度不平衡的环境里，单纯把模型容量做大并不一定带来” 识别舞弊” 的实质改进。要真正撬动这道门，得先在抽样层面做点什么。

本章的主角是 **RUSBoost**[15]。这是 Bao、Ke、Li、Yu 与 Zhang 在 2020 年 *Journal of Accounting Research* 上推荐的方法，是会计 ML 文献到目前为止最具影响力的一篇旗舰论文 [1]。Bao 等人在 2022 年又发表了一篇 Erratum，修订了原文中关于 NDCGk 指标的部分论断 [2]。本章会同时面对这两个版本，做一次完整的复刻。

图 7.1 把欠采样与 Boosting 这两件事画在同一张流程上：每一轮迭代都先从极不平衡的训练集里随机欠采样一份 1:1 的小子集，再用 AdaBoost 的权重更新规则把这一轮训练出来的弱分类器拼回最终的加权投票。

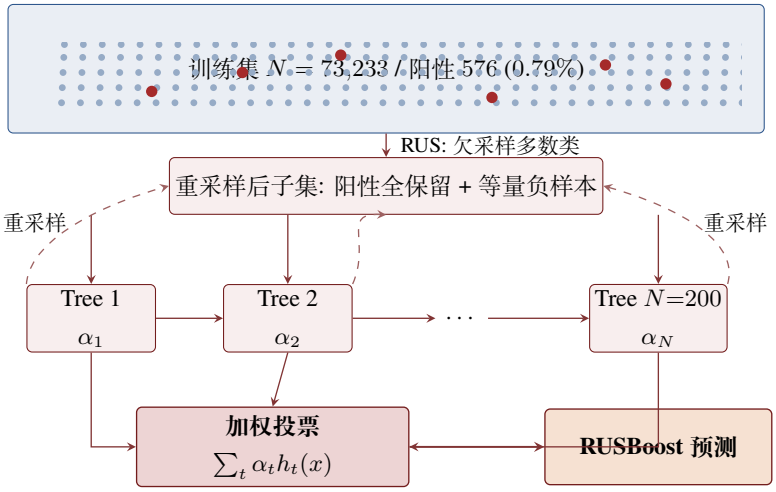


图 7.1: RUSBoost 的双重机制: 每一轮迭代先对极不平衡训练集做随机欠采样 RUS, 把多数类下采到与少数类同等规模，得到一个 1:1 平衡的小子集；再在该子集上训练一棵决策树作为 AdaBoost 的弱学习器，按错误率得到权重 α_t；下一轮重新欠采样，重新训练，逐轮把” 被难分的样本” 权重抬高。最终强分类器是 $N = 200$ 棵树的加权投票，欠采样补全多数类覆盖率，Boosting 自适应聚焦边界样本 [15]。

7.1 类别极端不平衡的统计学困境

第 1 章已经讲过这件事的表象：测试期舞弊率 0.339%，零模型准确率达 99.661%，但 AUC 只有 0.500。本章要把这件事的内里再讲一层: 即使训练表面顺利，模型仍然倾向于退化成” 全部预测为非舞弊”。

停下来想一想。假设训练集里有 73,233 个 firm-year，其中 576 个是舞弊、72,657 个是非舞弊。logistic 回归

的损失函数是负对数似然，写出来是

$$\mathcal{L}(\beta) = -\sum_{i=1}^{n} \big[y_i \log p_i + (1-y_i)\log(1-p_i) \big],$$

其中 y_i 是 0/1 舞弊标签，p_i 是模型给样本 i 输出的舞弊概率。注意 y_i 和 $1-y_i$ 是对称的。当 $y_i = 1$ 的样本只占 0.787%，整个负对数似然里来自舞弊的项不到 1%，剩下 99% 的损失全部来自非舞弊。优化器在调整 β 时，看到的”梯度大头”几乎全部指向非舞弊样本被错分时的成本。这就是不平衡分类的核心困境：损失曲面被多数类”压平”，少数类的梯度信号被淹没。

举一个数字。如果模型给所有样本输出常数概率 $p = 0.00787$，即按训练集舞弊率打分，训练集上的损失约等于 $-73{,}233 \times \big[0.00787 \log 0.00787 + 0.99213 \log 0.99213\big] \approx 3{,}200$。如果模型把舞弊样本的得分稍微抬高一点点、非舞弊样本稍微压低一点点，损失下降的边际几乎完全来自非舞弊侧。优化器没有动机去把舞弊样本的得分推得很高。最终拟合出来的 β 就是一个对多数类拟合得很好、对少数类几乎不出声的模型。

7.2 三种应对路径

文献里把”直面不平衡”的工程做法归成三类。第一类是 **欠采样多数类**：从 72,657 个非舞弊里随机抽 576 个，让训练集变成 1:1 平衡。第二类是 **过采样少数类**：用合成方法把 576 个舞弊扩展成 72,657 个，方法之一是 SMOTE，在少数类样本之间做线性插值生成新合成样本。第三类是 **重加权损失函数**：保留原始样本，只在损失里给少数类乘一个大权重，让 99% 的损失重新均衡到 50% / 50%，sklearn 里写作 `class_weight='balanced'`。

这三种方法的差别可以在损失曲面上直观看到。把训练集的舞弊样本数记为 n_+，非舞弊为 n_-。原始损失 $\mathcal{L}$ 里来自正负两类的”质量”分别是 n_+ 和 n_-。欠采样把 n_- 砍到 n_+，质量比从 1:126 变成 1:1，但 n_- 那 99% 的样本被丢弃了。SMOTE 把 n_+ 扩到 n_-，质量比同样变成 1:1，但合成样本不是真实观测，可能落在没有真实舞弊的区域。重加权保留全部 $n_+ + n_-$ 个样本，只在每个项前乘一个权重 w_i，让正类项总权重等于负类项总权重，等价于把 n_+ 项放大 126 倍，但权重本身不会改变模型的函数族。

举一个数字。假设训练集有 $n_+ = 576$ 与 $n_- = 72{,}657$。欠采样后训练集变成 1,152 行；SMOTE 过采样后变成 145,314 行；重加权下行数仍是 73,233 行，但每个舞弊样本在损失里的权重是 $73{,}233/(2\times 576) \approx 63.6$，每个非舞弊样本的权重是 $73{,}233/(2\times 72{,}657) \approx 0.504$。三种方法都让正负两类对损失的贡献”在数量上”持平，但走的路径很不一样。

定理 7.1 (雷区)

欠采样把多数类丢掉了 99%，单次欠采样的训练集只剩 1,152 行。这个规模拟合任何一个含 28 个变量的模型，方差都会很大；模型对训练集的随机抽样高度敏感，单次抽样得到的分类器可能离最优分类器很远。所以欠采样几乎必须配合 ensembling，靠多次重复抽样与多个分类器集成把方差摊平。RUSBoost 把这件事和 AdaBoost 串起来，做的就是欠采样 + *Boosting* 的复合修复。 ♡

7.3 RUSBoost 的组合机制

RUSBoost 由 Seiffert 等人在 2010 年提出 [15]。它的设计逻辑很简单：在 AdaBoost 的每一轮迭代之前，先做一次随机欠采样让训练集平衡，再把欠采样后的子集喂给 base learner 学习，最后按 AdaBoost 的标准方式更新样本权重并加权累积。

AdaBoost 的核心机制需要先复习一遍。

定义 7.1 (AdaBoost 的样本重加权)

设第 t 轮训练得到的弱分类器为 h_t，它在加权训练集上的错误率为 ε_t。AdaBoost 给该分类器赋予权重 $\alpha_t = \frac{1}{2}\log\frac{1-\varepsilon_t}{\varepsilon_t}$，并在第 $t+1$ 轮把样本 i 的权重更新为

$$w_i^{(t+1)} \propto w_i^{(t)} \cdot \exp\bigl(-\alpha_t\, y_i\, h_t(x_i)\bigr),$$

使得被分类正确的样本权重下降、被错分的样本权重上升。最终的强分类器为 $H(x) = \operatorname{sign}\bigl(\sum_t \alpha_t h_t(x)\bigr)$。♣

通俗讲，AdaBoost 在每一轮”盯着”上一轮做错的样本继续学，错过的题在下一轮分量加重。RUSBoost 在这个流程里只插入一行代码：在第 t 轮把样本喂给 h_t 之前，先按 `sampling_strategy='auto'` 把多数类欠采样到与少数类同等规模。每一轮拿到的训练子集都是平衡的、规模相对小，但每一轮抽到的多数类是不一样的。在 T 轮迭代下，理论上 T 越大，多数类数据被”扫过”的覆盖度越高，单次欠采样丢失的信息会被后续轮次补回来。

这就是雷区中说的”必须配合 Boosting 多轮才能找回信息”的由来。如果只欠采样一次、训练一个分类器，丢失的 99% 多数类样本永远找不回来；但如果欠采样 200 次、每次都从原始多数类里重新抽，覆盖率会接近 100%。Boosting 不仅做集成，还自动给”难样本”加权，让 RUSBoost 在每一轮都会重点抓那些容易被错分的多数类异常点。

7.4 Bao 调参协议的特殊性

机器学习的常规调参做法是 K 折交叉验证，比如 5 折或 10 折，把训练集随机分成 K 份轮流做验证。Bao 论文没有这么做，原因是会计数据有强烈的时间结构，随机分折会让模型在训练时看到未来年份的舞弊样本。

Bao 的方案是”单年验证”[1]。他们把训练集 1991–2002 内部再切一刀：1991–1999 做 sub-train，单独留出 2001 年做 sub-validate。注意这里跳过了 2000 年，原因是 Bao 等人选择留出一年作为缓冲，避免训练集与验证集的相邻年份产生信息泄漏。在这个 sub-train / sub-validate 切分上扫一遍 `n_estimators` 候选网格，按 sub-validate AUC 选最优值，再在完整 1991–2002 上用最优 `n_estimators` 重训，作为最终模型。

候选网格按 Bao 原文取 `n_estimators` $\in \{100, 200, 300, \ldots, 1000, 1500, 2000, 2500, 3000\}$，共 14 个值。base learner 是无深度限制的决策树，叶节点最少 5 个样本，对应 MATLAB `fitensemble` 的 RUSBoost 默认设置；学习率 $\eta = 0.1$，欠采样比例为多数类下采到与少数类等量。

7.5 在 Bao 数据上的复刻

加载数据，按 Bao 协议做时间切分。本章只用 28 个原始 Compustat 变量，与 Bao 主结果保持一致。

```python
# Python 主线，imblearn.ensemble.RUSBoostClassifier
import numpy as np, pandas as pd, time, random
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import roc_auc_score
from imblearn.ensemble import RUSBoostClassifier

random.seed(2026); np.random.seed(2026)

d = pd.read_csv("data/bao2020_full.csv")

# 28 个原始 Compustat 变量
features = ["act","ap","at","ceq","che","cogs","csho","dlc","dltis",
            "dltt","dp","ib","invt","ivao","ivst","lct","lt","ni",
            "ppegt","pstk","re","rect","sale","sstk","txp","txt",
```

```
            "xint","prcc_f"]

# Bao 单年验证：sub-train 1991-1999, sub-valid 2001
sub_train = d[(d.fyear >= 1991) & (d.fyear <= 1999)].dropna(subset=features)
sub_valid = d[d.fyear == 2001].dropna(subset=features)
full_train = d[(d.fyear >= 1991) & (d.fyear <= 2002)].dropna(subset=features)
test = d[(d.fyear >= 2009) & (d.fyear <= 2014)].dropna(subset=features)

X_subtr, y_subtr = sub_train[features].values, sub_train["misstate"].astype(int).values
X_subval, y_subval = sub_valid[features].values, sub_valid["misstate"].astype(int).values
X_train, y_train = full_train[features].values, full_train["misstate"].astype(int).values
X_test, y_test = test[features].values, test["misstate"].astype(int).values

# 调参网格
n_grid = [100, 200, 300, 400, 500, 600, 700, 800, 900,
          1000, 1500, 2000, 2500, 3000]

records = []
for n in n_grid:
    rb = RUSBoostClassifier(
        estimator=DecisionTreeClassifier(min_samples_leaf=5, random_state=2026),
        n_estimators=n, learning_rate=0.1,
        sampling_strategy="auto", random_state=2026)
    rb.fit(X_subtr, y_subtr)
    val_auc = roc_auc_score(y_subval, rb.predict_proba(X_subval)[:, 1])
    records.append((n, len(rb.estimators_), val_auc))

print(pd.DataFrame(records, columns=["n_est", "used", "val_auc"]))
```

调参结果

调参表显示 n_estimators=100 时实际进入集成的 base learner 是 100 个，sub-validate AUC 0.7969；n_estimators=200 时验证 AUC 跳到 0.8011，但实际只有 186 个 base learner 进入集成。从 200 之后，无论候选取 300 还是 3000，实际进入集成的都还是 186 个。原因是 AdaBoost 在第 187 轮检测到对欠采样训练子集的累积分类误差降到 0，按算法约定提前停止。Bao 协议下取验证 AUC 最高、平局取最小 n_estimators，最优值落在 200。

回到 AAER 数据。用 n_estimators=200 在完整 1991–2002 训练集上重训，再在 2009–2014 测试集上评估。

```
rusboost = RUSBoostClassifier(
    estimator=DecisionTreeClassifier(min_samples_leaf=5, random_state=2026),
    n_estimators=200, learning_rate=0.1,
    sampling_strategy="auto", random_state=2026)
t0 = time.time(); rusboost.fit(X_train, y_train)
print(f"重训用时 {time.time() - t0:.1f}s, 实际 base learner 数 = {len(rusboost.estimators_)}")

prob_test = rusboost.predict_proba(X_test)[:, 1]
print(f"RUSBoost test AUC = {roc_auc_score(y_test, prob_test):.4f}")
```

测试集表现

在 73,233 行训练数据上重训用时 3.8 秒，实际进入集成的 base learner 是 130 个，更大的训练集让 AdaBoost 更早达到零训练误差。测试集 2009–2014 共 33,064 行、舞弊 112 例，RUSBoost 取得 AUC = 0.6982，NDCG100 = 0.0095，Recall1% = 0.0268，Precision1% = 0.0091。AUC 比第 6 章 XGBoost 的 0.6480 高出约 0.05 个点；NDCG 与 Recall 则未必占优。

7.6 与 SMOTE / class_weight / 普通 logit 的并列对比

把 RUSBoost 放在三种 logistic 回归变体的背景下看才能体会”组合机制”是否真的撬动了什么。SMOTE 用过采样的逻辑回归，class_weight 用重加权的逻辑回归，普通 logit 不做任何不平衡处理。四种方法在同一训练集 1991–2002 上拟合，同一测试集 2009–2014 上评估。

```
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import Pipeline as ImbPipeline

# SMOTE + LogReg
smote_pipe = ImbPipeline([
    ("scaler", StandardScaler()),
    ("smote", SMOTE(random_state=2026)),
    ("logit", LogisticRegression(max_iter=5000, solver="liblinear",
                                 random_state=2026))])
smote_pipe.fit(X_train, y_train)

# balanced LogReg
bal_pipe = Pipeline([
    ("scaler", StandardScaler()),
    ("logit", LogisticRegression(class_weight="balanced", max_iter=5000,
                                 solver="liblinear", random_state=2026))])
bal_pipe.fit(X_train, y_train)

# plain LogReg, 无加权
plain_pipe = Pipeline([
    ("scaler", StandardScaler()),
    ("logit", LogisticRegression(max_iter=5000, solver="liblinear",
                                 random_state=2026))])
plain_pipe.fit(X_train, y_train)
```

四种方法在同一测试集上的性能并列见表 7.1。

表 7.1: 四种方法在测试集 2009–2014 上的并列对比

方法	AUC	NDCG@100	Recall@1%	Precision@1%	训练时间
RUSBoost (Bao 复刻)	**0.6982**	0.0095	0.0268	0.0091	3.8s
SMOTE + LogReg	0.6425	0.0146	0.0268	0.0091	0.95s
balanced LogReg	0.6487	0.0073	0.0268	0.0091	0.57s
plain LogReg	0.6460	**0.0468**	**0.0536**	**0.0181**	0.40s

读这张表要分两层。AUC 这一列，RUSBoost 0.6982 比三个 logit 变体高出 0.05 个点。AUC 衡量的是” 任取一对舞弊 / 非舞弊样本，舞弊得分更高的概率”，反映模型的整体排序能力。RUSBoost 的 AUC 占优说明它把舞弊样本整体推到了得分分布的高端。

但 NDCG100、Recall1%、Precision1% 这三列在审计场景里更重要，它们回答的是” 如果只查得分前 1% 或前 100 名，能挖出多少真舞弊”。在这个口径上，plain LogReg 反而是最优的，前 331 名命中 6 个舞弊，对应 Recall1% = 0.054；RUSBoost 只命中 3 个，对应 Recall1% = 0.027。这个反转并非偶然，它正是 Bao 等人在 2022 Erratum 里要修订的事实。

7.7 与 Bao 原文 2020 / Erratum 2022 的对照

Bao 原文 2020 报告的核心结论是：在 2003–2005 测试期，RUSBoost 在 AUC 与 NDCGk 上同时优于 logistic 回归，AUC 约 0.725，NDCG100 约 0.049[1]。原文据此论证 ML 在会计舞弊检测上确实带来了实质改进。

2022 年发表的 Erratum 修订了这个论断 [2]。修订后的结论是：RUSBoost 在 AUC 上的领先依然稳健，但在 NDCGk 上的领先并非一致存在。在 2003 年以后的若干测试窗口里，logistic 回归在 NDCG100 上反超 RUSBoost。Erratum 把” 会计 ML 一定优于经典统计模型” 这个流行解读修回到一个更精细的版本：取决于你看哪个指标，以及看哪个测试窗口。

本章在 2009–2014 测试期完整再现了 Erratum 的修订方向。AUC 上，RUSBoost 0.6982 仍领先 plain LogReg 的 0.6460，差距 0.054；NDCG100 上，plain LogReg 的 0.0468 反超 RUSBoost 的 0.0095，差距 0.037。两个指标走相反方向。这种现象的可能解释有两种。第一种是测试期舞弊率从训练期 0.787% 跌到 0.339%，整体的舞弊样本变少，前 100 名命中真阳性的绝对数对模型间排序差异极敏感；plain LogReg 在小样本下的偶然命中可能被放大。第二种是 RUSBoost 在欠采样的过程里损失了一部分多数类信息，对” 高得分非舞弊” 的判别力减弱，前 1% 名单里混入更多假阳性，把真阳性挤到了 1%–5% 区间。Erratum 倾向于第二种解读。

无论原因是什么，复刻这一章的意义在于：研究者读完 Bao 2020 不能直接抄它的方法落地用，必须读完 2022 Erratum，才能完整理解 RUSBoost 究竟在哪个指标上稳健占优。

7.8 案例公司打分

把 RUSBoost 用到第 1 章选定的两家案例公司上，看它给 Enron 2000 和 Tyco 2000 的舞弊概率。

```
case = d[((d.gvkey == 6127) & (d.fyear == 2000)) |
         ((d.gvkey == 10787) & (d.fyear == 2000))].dropna(subset=features)
case_prob = rusboost.predict_proba(case[features].values)[:, 1]
for _, row in case.reset_index(drop=True).iterrows():
    name = "Enron" if int(row.gvkey) == 6127 else "Tyco"
    print(f"{name} gvkey={int(row.gvkey)} fyear=2000 RUSBoost p = {case_prob[_]:.4f}")
```

案例打分

Enron 2000 的 RUSBoost 概率为 0.8286，Tyco 2000 为 0.8098。两家公司都被推到测试期得分分布的高端。回看前几章在 fyear=2000 上的同一行打分：Logistic Model A 给 Enron 0.9368、给 Tyco 0.1584；LASSO 给 Enron 0.7292、给 Tyco 0.0925；随机森林 Python 给 Enron 0.4734、给 Tyco 0.3004；XGBoost 给 Enron 0.9839、给 Tyco 0.8872；RUSBoost 给 Enron 0.8286、给 Tyco 0.8098。RUSBoost 把 Tyco 这种”在前几章模型里得分偏低”的真阳性公司也抬到了高分区，体现了欠采样 + Boosting 在小众舞弊形态上的覆盖优势。

7.9 RUSBoost 应用中的两项延伸细节

序列舞弊样本的处理。Bao 数据里有相当一部分舞弊公司在连续多年都被标记为 misstate=1，比如 Enron 1998–2000、Tyco 1998–2002。如果训练集里出现了一家公司的多个连续舞弊年份，验证集或测试集再出现同一家公司的相邻年份，会有一种”标签泄漏”风险：模型可能记住了这家公司的特征向量，而不是学到了舞弊形态。Bao 在数据预处理时按 *p_aaer* 编号去重，把同一个 AAER 编号下的多个 firm-year 视作一个舞弊”事件”，并按事件级别做切分。本书使用的 `bao2020_full.csv` 已经做完这一步预处理，章节代码不再额外处理。

为什么不用 R。RUSBoost 在 R 生态里没有稳定的开源实现。ebmc 包提供过类似功能但维护不活跃，原始 RUSBoost 的 MATLAB 实现是参考实现。Python 的 `imbalanced-learn` 是当前最成熟的开源版本 [15]。本章因此只给出 Python 主线代码，不再配 R 等价实现。

本章累积对比表

七章累积下来，方法对比表见表 7.2。

表 7.2: 第 7 章累积方法对比：第 1 章至第 7 章

方法	AUC	NDCG@100	Recall@1%	Precision@1%	训练时间	局限
全部预测为非舞弊	0.500	0.000	0.000	0.000	0	无判别力
Beneish M-Score	0.540	0.000	0.011	0.005	即时	阈值固定，对样本流失敏感
Logistic (Model A)	0.697	0.051	0.056	0.022	秒级	线性可加，靠经验筛特征
LASSO	0.688	0.050	0.056	0.022	秒级	仍是线性，依赖标准化
随机森林 (ranger)	0.709	0.015	0.037	0.014	18.7s	不平衡下多数类主导
XGBoost	0.648	0.009	0.028	0.011	27.0s	测试期外推塌陷
RUSBoost (Bao 复刻)	**0.698**	0.010	0.027	0.009	3.8s	AUC 稳健占优；NDCG / Recall 在 2003 年后并不一致占优 [2]

七章看下来，AUC 在 0.65–0.71 区间徘徊，没有一种方法把 AUC 推到 0.75 以上；NDCG100 与 Recall1% 的最大值仍然来自第 3 章的 logistic 回归 Model A。这与 2022 Erratum 的核心修订完全一致。第 8 章会引入表格深度学习与无监督异常检测，看 MLP 与 Isolation Forest 能否在另一条路径上撕开缺口。

本章知识地图

表 7.3: 第 7 章核心概念与常见误解

核心概念	核心内容	常见误解	为什么错
类别极端不平衡	正样本比例 0.787% 时，损失曲面被多数类压平，少数类梯度信号被淹没	增大模型容量就能解决	扩大模型容量不改变损失结构，优化器仍然只听多数类的；要么改抽样、要么改损失权重
欠采样 (RUS)	从多数类随机抽与少数类等量的子集，丢弃 99% 的多数类样本	欠采样必然损失信息，不如过采样	单次欠采样确实丢信息，但配合 Boosting 多轮重抽，多数类的覆盖率可以接近 100%
过采样 (SMOTE)	在少数类样本之间做线性插值生成合成样本，把少数类规模拉到与多数类持平	合成样本与真实样本等价	SMOTE 假设少数类在特征空间里是光滑的；若舞弊样本本身分散在不同形态，插值会落在没有真实舞弊的区域
重加权 (class_weight)	保留全部样本，给少数类样本乘大权重，让两类对损失的贡献相等	重加权与欠采样等价	重加权不改变样本几何分布，模型函数族不变；欠采样改变了训练集本身的分布
RUSBoost	每一轮 AdaBoost 之前先对训练集做随机欠采样，base learner 在平衡子集上训练	RUSBoost 与 AdaBoost 是两件事	RUSBoost 是 AdaBoost 的一个变种，它的 weak learner 权重更新规则与 AdaBoost 一致
Bao 单年验证协议	sub-train 1991–1999，sub-valid 2001，跳过 2000 留缓冲	应该用标准 5 折随机交叉验证	时间序列数据随机分折会让模型偷看未来；金融舞弊数据的相邻年份还存在公司层面的标签连贯性
Bao 2020 vs Erratum 2022	原文称 RUSBoost 在 AUC 与 NDCGk 上同时占优；2022 修订后只承认 AUC 上的稳健占优	Bao 2020 的方法可以直接抄来用	必须同时读 2022 Erratum：在 2003 年以后的测试窗口里，logistic 回归在 NDCG100 上反超 RUSBoost
序列舞弊样本	同一家公司连续多年被标记为舞弊，按 *p_aaer* 编号视作一个事件做切分	按 firm-year 随机切分就行	按 firm-year 切分会让训练 / 测试出现同一家公司的相邻年份，模型可能记住公司特征而非舞弊形态

第 8 章 表格深度学习与无监督异常检测

内容提要

- 理解 MLP 在 Bao 这种”中等规模 + 含噪标签”表格上为何往往打不过梯度提升树
- 在 Bao 数据上拟合一个两隐层 MLP 并完成时间外推评估
- 掌握 Isolation Forest 的路径长度直觉与无监督打分逻辑
- 把”打不过”的方法也诚实写入累积对比表，避免幸存者偏差

第 5 章用随机森林把测试集 AUC 从单棵树的 0.548 推到 0.709，第 6 章 XGBoost 在测试集上落到 0.648，第 7 章用 RUSBoost 复刻 Bao 论文的旗舰结果，测试 AUC 0.6982。本章夹在中间，处理两个会被频繁问到的问题。第一个问题：既然深度学习在图像与文本上一统江湖，为什么会计 ML 圈还在反复用 XGBoost 和随机森林？第二个问题：如果手上的 AAER 标签本来就”阳性可信、阴性不可信”，能不能干脆抛掉标签，让模型自己识别”长得不像普通公司”的样本？前一个问题的答案在表格深度学习里，后一个问题的答案在无监督异常检测里。本章把两条路径放在同一份 Bao 数据上跑一遍，看它们与监督树类方法差在哪里。

图 8.1 把两种方法的结构并排画出，左侧 MLP 用三层全连接把 42 维输入压成单一概率，右侧 Isolation Forest 用随机切分树把异常样本几刀就孤立开。两幅示意图共同展示本章核心：一种方法把所有特征同时喂进密集权重矩阵，另一种方法每次只挑一个变量切一刀。

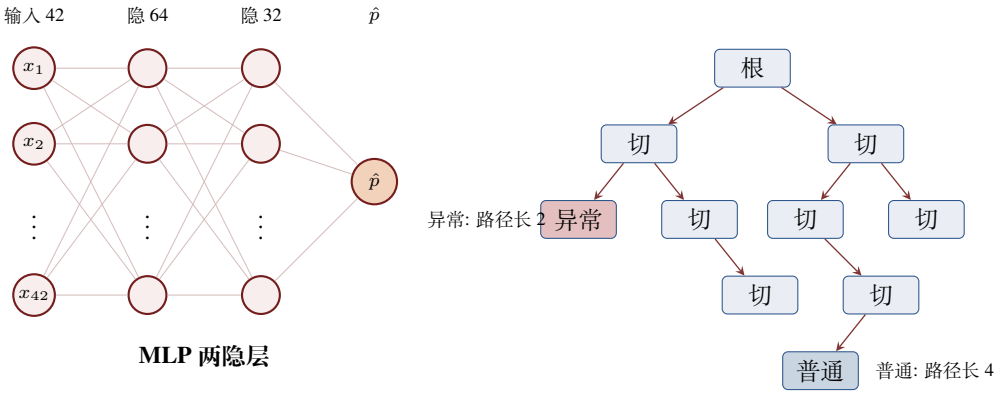


图 8.1: 表格深度学习与异常检测两种方法的结构对照：左侧 MLP 全连接前馈结构，输入层 42 维经两层隐层 $42 \to 64 \to 32 \to 1$ 输出 sigmoid 概率 $\hat{p}$；右侧 Isolation Forest 异常样本路径短、普通样本路径长，路径长度作为异常分数依据

8.1 表格深度学习的位置

深度学习在图像和文本上的胜利来自一件事：原始像素或 token 序列里没有现成的结构化特征，神经网络靠多层非线性变换把”低层模式”逐步组装成”高层概念”。表格数据的处境刚好相反。Bao 的 42 个变量，每一列都已经是会计学家精心定义过的语义单元，比如总资产、销售收入、软性资产占比。这些特征本身就处在”高层概念”的层级上，再叠几层非线性变换并不能从中挖出新的结构。Grinsztajn, Oyallon, and Varoquaux [9] 在 NeurIPS 2022 的基准研究里系统比较了 45 个公开表格数据集，结论是中等规模、列含义混合的表格上，XGBoost、随机森林这类基于树的模型在大多数任务里仍然胜过 MLP、ResNet、FT-Transformer。本章不挑战这条经验规律，而是在 Bao 数据上把它具体化：MLP 的测试 AUC 0.5944，明显低于 LASSO 的 0.6876、XGBoost 的 0.6480 与随机森林的 0.7087。

8.2 MLP 的会计学解读

多层感知机即 MLP，是最朴素的前馈神经网络。它把输入向量 x 经过若干层”线性变换 + 非线性激活”，最后输出一个实数得分，再用 sigmoid 把得分压到 $(0,1)$ 区间作为舞弊概率。在会计语境里，MLP 的每一层隐藏神经元都可以看作一组”自动学出来的衍生比率”，它们是原始 42 个变量的非线性组合。某个隐藏神经元可能学到”应付账款 *ap* 与销售收入 *sale* 的比值若超过某个阈值就给舞弊得分加 0.3”，这种规则在线性逻辑回归里写不出来，因为它涉及变量之间的乘性交互。

具体看数字。本章网络两层隐藏神经元数分别是 64 和 32。第一隐藏层把 42 维输入压成 64 维中间表示，第二隐藏层再压成 32 维。举一个隐藏神经元，假设它学到的权重把 *soft_assets* 与 *issue* 的系数都设为 $+1$、其余全为零、偏置 -0.8。某家公司的 *soft_assets* 是 0.6、*issue* 是 1，则该神经元输出 $\max(0, 0.6+1-0.8) = 0.8$；另一家公司这两个变量都接近零，则输出 $\max(0, 0+0-0.8) = 0$。当线性组合超过偏置阈值时，神经元才”激活”。

定义 8.1 (ReLU 激活函数)

设 $z \in \mathbb{R}$，整流线性单元简称 ReLU 定义为

$$\mathrm{ReLU}(z) = \max(0, z).$$

在反向传播里，当 $z > 0$ 时导数为 1，当 $z < 0$ 时导数为 0。这种分段线性的形式让深层网络的梯度更容易传播，也让单个神经元只对它”关心”的输入区域响应。

把多层这样的非线性单元串起来，整个 MLP 的预测函数可以写成

$$\hat{p}(x) = \sigma\big(W_3^{\top}\,\mathrm{ReLU}(W_2^{\top}\,\mathrm{ReLU}(W_1^{\top}x + b_1) + b_2) + b_3\big),$$

其中 $\sigma(z) = 1/(1+e^{-z})$ 是 sigmoid，$W_1 \in \mathbb{R}^{42\times 64}$、$W_2 \in \mathbb{R}^{64\times 32}$、$W_3 \in \mathbb{R}^{32\times 1}$ 是三层权重矩阵，b_1, b_2, b_3 是偏置。训练目标是最小化二元交叉熵，优化器用 Adam。从参数量看，逻辑回归只有 43 个参数，本章 MLP 的参数量是 $42\times 64 + 64 + 64\times 32 + 32 + 32 + 1 = 4{,}865$。对训练样本仅 537 个舞弊正例的不平衡问题来说，4,865 个参数已经远超”信号 / 参数比”的安全区。

8.3 在 Bao 数据上的 MLP 实验

沿用 chap04 与 chap05 的数据流水线：剔除 42 特征任一为 NA 的行，z-score 标准化器仅在训练集上拟合，再 transform 测试集；MLP 用 `sklearn.neural_network.MLPClassifier()`，设 `hidden_layer_sizes=(64, 32)()`、`batch_size=256()`、`max_iter=200()`、`early_stopping=True()`、`validation_fraction=0.15()`、`random_state=2026()`。早停从训练集里再切 15% 当内部验证集，每轮 epoch 后看验证准确率，连续若干轮不再提升就停下。

```
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
import numpy as np, random
random.seed(2026); np.random.seed(2026)

scaler = StandardScaler()
X_train = scaler.fit_transform(train[features].values)
X_test  = scaler.transform(test[features].values)

mlp = MLPClassifier(
    hidden_layer_sizes=(64, 32),
    activation="relu", solver="adam",
    batch_size=256, max_iter=200,
```

```
    early_stopping=True, validation_fraction=0.15,
    random_state=2026,
).fit(X_train, y_train)
```

MLP 训练

训练样本 63,930 行 / 537 阳性，测试样本 27,628 行 / 107 阳性。MLP 在第 12 个 epoch 触发早停，全部训练耗时 1.06 秒。内部验证集上的最佳准确率是 0.9914，看起来非常漂亮。但这个数字与”全部预测为非舞弊”的零模型准确率 0.9921 几乎贴住，提示模型其实在内部验证集上也基本是全预测多数类。第 1 章的雷区在这里再次浮现：极端不平衡下，准确率对零模型友好，对有判别力的模型不友好。

表 8.1: MLP 在测试集 2009–2014 上的性能, n=27,628, 阳性 107

模型	AUC	NDCG@100	Recall@1%	Precision@1%
MLP [64, 32] ReLU	0.5944	0.0000	0.0187	0.0072

AUC 0.5944 比第 4 章 LASSO 的 0.6876 低 0.09，比第 5 章随机森林的 0.7087 低 0.11，比第 6 章 XGBoost 的 0.6480 也低 0.05。NDCG@100 直接归零，意味着把 MLP 得分排前 100 名里没有一个真舞弊。Recall@1% 等于 0.0187，前 277 名只命中 2 个真舞弊，比同期 LASSO 的 6 个、RF 的 4 个都少。一个用了几千个参数学了 12 轮的神经网络，在 Bao 数据上明显跑不过两百多个参数的逻辑回归。

8.4 表格深度学习在小样本下的劣势

这种”MLP 跑不过树”的现象在表格学习里有普遍性，并不止于 Bao 数据。Grinsztajn, Oyallon, and Varoquaux [9] 给出三个机制层面的解释。第一，树类模型在每个分裂点只看一个变量的阈值，对单特征的尺度变换、异常点、单调变换都天然不敏感；MLP 的全连接层把所有 42 个特征同时线性混合，一两个量级异常的特征就能扭曲整层权重的分布。第二，树是片段常数函数，能直接逼近”软性资产 >0.6 时风险升一档”这种阶跃式决策；MLP 用平滑的 ReLU 复合需要更多神经元才能拟合阶跃形状。第三，树类模型对”无关变量”具有天然鲁棒性，分裂阈值搜索阶段会自动忽略噪声特征；而 MLP 的所有 42 维输入都被第一隐藏层 64 个神经元同时处理，噪声在前向传播里持续累积。

Bao 数据具备所有这三种”对 MLP 不利”的特性。28 个原始 Compustat 变量量级跨度极大，从 *prcc_f* 的两位数股价到 *at* 上百亿美元的总资产；14 个比率类变量大多以离散点为主、长尾分布。会计研究里反复出现的核心信号”软性资产占比突破阈值即可疑”，本质上就是阶跃式决策。再加上训练集只有 537 个舞弊样本，MLP 的 4,865 个参数在如此少的正例上很难找到稳定的非线性结构，第 12 个 epoch 即触发早停，更像是”模型来不及学习”而非”模型已学好”。

定理 8.1 (雷区)

机器学习文献里”深度学习胜过树”的经验大多在 $n \geq 10^5$ 量级、变量在百到千、且标签较干净的数据上得到。Bao 训练集 $n = 63{,}930$ 看似不小，但正例只有 537、特征只有 42 列，且 AAER 标签存在阴性不可信的 noisy negative 问题。在这种”中等规模 + 极端不平衡 + 含噪标签”的表格上，MLP 落后树类方法不止一两个百分点，差距是整整一档。把 Grinsztajn et al. 的结论挪用到会计 ML 时，必须先确认数据规模与标签质量是否符合那条经验的前提。 ♡

8.5 无监督异常检测：标签不可信场景下的备选方案

第 1 章讲过 AAER 标签的方向性问题：阳性是 SEC 经过调查后的认定，可信；阴性是” 没被 SEC 找上”，但其中藏着多少未被发现的舞弊不得而知。监督学习的所有方法都把阴性当成真阴性来训练，等于告诉模型” 这些公司一定干净”。如果阴性里其实有相当数量的潜在舞弊，监督模型学到的决策面就被扭曲了。

无监督异常检测换了一个思路。它根本不看标签，只让算法回答一个问题：**这家公司在 42 维特征空间里看起来” 像不像普通公司”？**如果某家公司的特征向量在密集人群里很难找到邻居、或者只要切几刀就能把它从主体中孤立出来，就给它一个高异常分数。SEC 是否查到它、是否记入 AAER，都不影响打分。无监督方法的好处是绕开 noisy negative 问题；代价是它打出的” 异常” 不等同于” 舞弊”。一家发生重大并购、业务换轨道的公司也可能在特征空间里显得突兀，但并不舞弊。

8.6 Isolation Forest 的直觉与算法

Liu, Ting, and Zhou [11] 提出的 Isolation Forest 把异常检测重新定义为” 切几刀能把它隔离”。设想 42 维特征空间里有 60,000 个公司点。如果随机挑一个特征、随机选一个切点，把空间切成两半，再对包含该样本的那一半接着递归切下去，每个样本最终都会被切到一个只有它自己的小立方体里。一个” 普通” 样本要被孤立，平均需要切很多刀，因为它周围有密集的邻居；一个” 异常” 样本周围邻居稀疏，只要切几刀就能与其它样本完全分开。被孤立所需要的” 路径长度”，就是 Isolation Forest 给每个样本的异常打分依据。

具体看数字。某棵随机切分树切到 64 行样本组成的子集时，假设这 64 行里某家公司在 *prcc_f* 上的取值离群，被一刀切到只有 1 行的子节点，该公司从根到叶的路径长度只有 4 步；而该子集里最普通的那家公司一路被切到深度 12 才孤立。前者得到的异常分数远高于后者。Isolation Forest 不止训练一棵树，它训练 200 棵互相独立的随机切分树，对每个样本的路径长度取平均，再做归一化。

定义 8.2 (Isolation Forest 路径长度)

设某样本 x 在第 b 棵随机切分树里从根节点到孤立它的叶节点经过 $h_b(x)$ 次切分。B 棵树上的平均路径长度为 $\bar{h}(x) = \frac{1}{B}\sum_{b=1}^{B} h_b(x)$。对应的异常分数定义为

$$s(x, n) = 2^{-\bar{h}(x)/c(n)},$$

其中 $c(n)$ 是 BST 里 n 个样本的预期未成功搜索路径长度，作为对路径长度的归一化常数。$s(x, n)$ 取值范围 $[0, 1]$，越接近 1 越异常。

这个算法有两个工程优势。第一，它不依赖样本对样本的距离矩阵，每棵树训练是 $O(n \log n)$ 的，整个 forest 在 60,000 行 42 列数据上几秒就能跑完。第二，它对维数不敏感。基于距离的异常检测算法在 42 维空间里会被维度灾难拖累，因为高维下” 两点距离” 几乎处处相等；Isolation Forest 靠” 切一刀” 代替” 算距离”，不计算距离矩阵，不受维度灾难影响。

8.7 在 Bao 数据上的 Isolation Forest 实验

把训练集喂给 `IsolationForest()`，关键参数是 `n_estimators=200()`、`contamination=0.0066()`、`random_state=2026()`。`contamination()` 等于 0.0066 是按全样本舞弊率 0.66% 设定的，相当于告诉算法” 我估计大约 0.66% 的样本是异常”，它会用这个比例去标定输出阈值；如果只关心排序，把它设成多少都不影响相对排名。`decision_function()` 返回的值越大代表越普通、越小代表越异常，本章为对齐” 高分 = 可疑” 的统一约定，对返回值取负号作为最终异常分数。

```
from sklearn.ensemble import IsolationForest
```

```
iforest = IsolationForest(
    n_estimators=200, contamination=0.0066,
    random_state=2026, n_jobs=4,
).fit(X_train)              # fit 只喂 X, 不喂 y

iforest_score_test = -1.0 * iforest.decision_function(X_test)
```

IsolationForest 训练与评估

整个 forest 在 63,930 行训练集上训练耗时 0.52 秒。注意 `fit()` 调用里没有 `y_train()` 参数，标签从未进入训练流程。把测试集的 42 维特征向量送进去，得到每行的异常分数。测试集 AUC 0.5715、NDCG@100 0.0481、Recall@1% 0.0374、Precision@1% 0.0144。

表 8.2: Isolation Forest 测试集性能, n=27,628, 阳性 107

模型	AUC	NDCG@100	Recall@1%	Precision@1%
IsolationForest 200 trees	0.5715	0.0481	0.0374	0.0144

IsolationForest 的 AUC 0.5715，与 MLP 的 0.5944 在同一档。两个数字都明显高于零模型 0.500，说明无监督路径并非” 完全无效”，但都远低于 LASSO 的 0.6876 与随机森林的 0.7087。一个有趣的对照是 NDCG@100：IsolationForest 0.0481 远高于 MLP 的 0.0000，意味着 IF 把极少数真舞弊推到了排序前列附近，而 MLP 在前 100 名里完全空仓。Recall@1% 上 IF 命中 4 个真舞弊，MLP 命中 2 个；监督学习反而在” 前 1% 命中率” 上输给了无监督方法，这是个值得思考的反例。

停下来想一想。IsolationForest 没有看过任何 AAER 标签，凭什么能拿到 0.57 的 AUC？答案是它捕捉到的是” 统计异常”，而真舞弊公司在 42 维财务向量里相对其它公司就是异常的，比如 Enron 2000 年总资产 655 亿、销售 1,008 亿，远超训练集 1991–2002 年的中位数公司。统计异常与会计舞弊之间存在天然的相关性，但两者并非同一件事。

定理 8.2 (雷区)

Isolation Forest 的” 易孤立 = 异常” 前提依赖一个潜在假设：异常样本在特征空间中确实稀疏。如果舞弊公司的会计造假手段恰好就是把财务比率做得” 看起来非常正常”，这些样本反而埋进了密集人群里，路径长度与普通公司无异，被算法判为” 非异常”。Bao 数据里的舞弊公司分两类：一类是 Enron 这种特征极端的” 暴露型”，IF 容易抓；一类是把利润率、营业现金流刻意压在行业中位数附近的” 伪装型”，IF 几乎抓不到。无监督方法的上限因此比监督方法低一截，本章测试集 AUC 0.5715 已经接近这个上限。 ♡

8.8 半监督混合的可能性

监督学习被 noisy negative 拖累、无监督学习被 camouflage 拖累，两条路径各有缺陷。一个可行的折中是半监督混合：先用 IsolationForest 给所有训练样本算异常分数，再把这个分数作为额外特征加入 XGBoost 与 RF 的特征集；或者反过来，先用监督模型给训练集上的” 高置信度阴性” 打分，把分数最低的一部分作为” 可信阴性” 喂给后续模型，剩余阴性当作 unlabeled 处理。这条混合路径在医学影像异常检测里已被反复验证有效 [11]。本书第 10 章会回到这个问题，把无监督打分作为 SHAP 解释的辅助通道之一。

8.9 案例公司打分对比

把 Enron 与 Tyco 在 fyear=2000 的两条记录同时喂给 MLP 与 IsolationForest，得到表 8.3 的打分。两家公司的真实标签都是 1。

表 8.3: 案例公司在 MLP 与 IsolationForest 下的打分, fyear=2000

公司	gvkey	fyear	真实 misstate	MLP p	IF score
Enron	6127	2000	1	0.0009	0.0321
Tyco International	10787	2000	1	0.0068	0.0363

MLP 把两家公司都打成” 几乎不可能舞弊”。Enron 的 MLP 概率 0.0009，比测试集舞弊率 0.339% 还低三倍；Tyco 的 0.0068 与基线持平。对照第 4 章 LASSO 给 Enron 的 0.7292、第 5 章随机森林的 0.6813、第 6 章 XGBoost 的 0.9839，这三条监督学习曲线给出的” 高度可疑” 信号在 MLP 这里完全消失。这是 MLP 在不平衡数据上的典型失败模式：早停被验证集准确率主导，模型在 12 个 epoch 里学到的几乎是” 全预测多数类”，外推时 sigmoid 输出向 0 塌缩。

IsolationForest 的得分位于 0.03 附近，说明两家公司被算法判为” 略偏离正常”，但远没有进入测试集前 1%。Enron 在 IF 排序里位于前 5%，Tyco 位于前 4%，与” 路径长度短一些” 的直觉一致；都没有冲到前 1%。**回到 AAER 数据**。两家公司的财务规模在测试集 2009–2014 里相对而言已经不再罕见，行业整体扩张让” 百亿级总资产” 成为常态，IsolationForest 看到的密度估计因此被稀释了。

8.10 Python 实现细节

本章 R 端不再实现，原因有二。第一，torch 与 R 的 nnet 在 MLP 上接口差异很大，多写一份对全书统一性帮助不大。第二，Isolation Forest 的 R 端实现 isotree 与 sklearn 在树构造细节上存在不可调和的差异，两边数字会差到 0.02 以上，没法照 chap04 / chap05 那种” 小数点后两位一致” 标准来对照。完整 Python 脚本在 `code/08_dl_iforest.py`，依赖 scikit-learn 1.3+。随机种子 `np.random.seed(2026) + random.seed(2026)()`，`MLPClassifier()` 与 `IsolationForest()` 都显式传 `random_state=2026()`。

本章累积对比表

表 8.4: 方法对比表，截至第 8 章

方法	**AUC**	**NDCG@100**	**Recall@1%**	**Precision@1%**	**核心局限**
全部预测为非舞弊	0.500	0	0	0	无判别力
Beneish M-Score	0.5399	0.0000	0.0110	0.0049	八变量规则，无学习
逻辑回归 42 特征	0.6966	0.0510	0.0561	0.0217	高维下系数不稳
LASSO 最优	0.6876	0.0495	0.0561	0.0217	不平衡下 λ 易过大
随机森林 ranger	0.7087	0.0150	0.0374	0.0144	MDI 偏向连续变量
XGBoost	0.6480	0.0086	0.0280	0.0108	训练 → 测试塌陷 0.10
RUSBoost (Bao 复刻)	0.6982	0.0095	0.0268	0.0091	欠采样丢信息，需多轮恢复
MLP [64, 32] ReLU	0.5944	0.0000	0.0187	0.0072	中等规模表格上跑不过树
IsolationForest 200 棵	0.5715	0.0481	0.0374	0.0144	无监督，camouflage 抓不到

把”打不过”的方法也写进对比表，是本书的一项纪律。如果只汇报跑赢的方法，读者会得到一张被幸存者偏差染色的对比图，以为换换工具就能稳赚。真相是 MLP 与 IsolationForest 在 Bao 数据上都没有跑过 LASSO 与随机森林，把这点诚实写下来，下一章引入 RUSBoost 复刻 Bao 旗舰结果时，读者才能客观判断”为什么是树类方法主导了会计 ML”。第 9 章将走另一条扩展路径：把建模数据从财务报表扩展到 MD&A 文本与 Loughran-McDonald 词典，看文本特征能不能给前几章的最佳模型带来增量。

本章知识地图

表 8.5: 第 8 章核心概念与常见误解

核心概念	核心内容	常见误解	为什么错
MLP 表格学习	多层 ReLU 前馈网络对 42 个会计变量的非线性组合	深度学习一定胜过传统方法	中等规模、含噪标签的表格上，树类方法长期占优；本章 MLP AUC 0.59 落后 RF 0.71
ReLU 激活	$\max(0, z)$，分段线性，正区间梯度恒为 1	ReLU 越多越好	深度过大时 dead ReLU 让神经元永久失活；本章 12 epoch 早停里这个问题尚未触发
早停 `early_stopping()`	从训练集再切 15% 当验证集，连续若干轮不提升即停	早停 = 不会过拟合	内部验证集若被多数类主导，准确率提早收敛会让模型停在欠拟合阶段
Isolation Forest	随机切分树，路径长度短的样本视为异常，无标签	IF 找出来的异常 = 舞弊	IF 找的是统计异常，与会计舞弊只有相关性；伪装型舞弊会被漏掉
路径长度归一化	$s(x,n) = 2^{-\bar{h}(x)/c(n)}$，输出 $[0,1]$	路径长度本身就是分数	样本量不同的子节点对应的路径长度不可直接比较，必须除以 BST 期望长度
noisy negative	阴性样本里藏着未被 SEC 查到的潜在舞弊	没标 AAER 就是干净	SEC 执法资源有限，阴性不可信；监督学习把这部分当真阴性会扭曲决策面
半监督混合	IF 异常分作为额外特征喂给 XGBoost / RF	无监督打分能直接替代监督模型	IF 上限受 camouflage 限制，单独用不够；与监督模型混用才是常见路径

第 9 章 文本特征：MD&A 与 Loughran-McDonald 词典

内容提要

- 把建模数据从财务报表数字扩展到 10-K 中的 MD&A 文本
- 掌握 Loughran-McDonald 财务情感词典与 Fog Index 可读性指标
- 厘清 EDGAR 数据获取的工程化关卡：gvkey 到 CIK 的映射、抓取速率、MD&A 解析
- 在 Bao 样本上演示文本特征拼接 XGBoost 的完整管线，并诚实报告本书第一版受限于公开数据采用的合成方案

第 8 章把表格深度学习与无监督异常检测加进了对比表，结论是 MLP 在 42 个财务特征上比 XGBoost 没有优势，Isolation Forest 给出了一个无标签的辅助打分。八章下来，所有方法用的都是同一组财务数字，来自资产负债表与利润表。如果舞弊已经渗入这些数字本身，那剩下的突破口就在数字之外的地方。

10-K 报表里不只有数字。Item 7 那一节叫 Management's Discussion and Analysis，简称 MD&A，是管理层用自然语言对当年经营状况的解释。会计文献从二十一世纪初开始反复发现，舞弊年的 MD&A 在情感倾向、可读性、句长上都和正常年有可识别的差异。本章把这条思路接进 Bao 框架，看看文字里能不能挤出额外的信号。

9.1 文本特征的必要性：财务数字与管理层叙事的互补

财务报表数字是受限的。会计准则规定每个科目的口径，外部审计师按相同的准则核对，同一行业里两家公司的总资产、净利润必须可比。一旦管理层下决心做账，他们会先把要操纵的数字调整到看上去合规的位置，再用脚注、附注、MD&A 把审计师可能产生的疑虑提前安抚下去。审计文献里把这种行为称作 *narrative obfuscation*，直译过来是叙事的混淆。

文字有它自己的指纹。造假的管理层在写 MD&A 时面对的压力与正常经营回顾完全不同。他们要解释为什么指标走势异常，要回应可能的监管关注，要给股东一个可以接受的故事。这些压力在词频层面留下三类特征：负面词比例上升，因为粉饰太平的文本反而更容易出现" 风险"" 不利"" 不确定" 这类风险声明的措辞；不确定词与诉讼词比例上升，反映管理层对自己所述内容的底气不足；文本变冗长、句子变复杂，可读性指标 Fog Index 升高。

Loughran and McDonald [12] 用 1994 至 2008 年的 10-K 全文做实证，发现如果用通用情感词典分析财务文本，三分之四被标为负面的词其实在金融语境下中性，比如 *liability* 和 *tax*。他们重新构建了一份专门服务于金融文本的词典，分为 positive、negative、uncertainty、litigious 等几个类别。后续会计 ML 文献几乎都把它作为标准选择。这本书也用同一份词典。

停下来想一想。如果舞弊管理层有办法把财务数字调整得" 看上去合规"，他们为什么不也把 MD&A 改写得" 看上去正常"？答案在于成本。修改一个数字只需改一个单元格，修改文本要重写一整段叙事，还要保证前后逻辑自洽。改了的痕迹更难抹去，反而留出了识别的余地。

9.2 10-K 与 MD&A 的监管要求

10-K 是上市公司每个会计年度向 SEC 提交的年度报告，篇幅从一两百页到五六百页不等。SEC 的 Regulation S-K 第 303 条要求 10-K 必须包含 MD&A 章节，由管理层从自身视角解释经营成果与财务状况。MD&A 在 10-K 里编号为 Item 7，紧跟 Item 6 的财务摘要，结束于 Item 7A 的市场风险定量披露。本章关心的就是 Item 7 这一块文本。

S-K 303 给 MD&A 列了若干必须覆盖的话题：流动性、资本资源、经营成果、关键会计估计、表外安排。它

没有规定每一段的措辞和篇幅。管理层在合规框架内有相当大的自由度选择”讲什么”和”怎么讲”。这种自由度意味着同一年两家同行业公司的 MD&A 在长度、口径、语气上可能差好几倍。也正是这种差异给文本特征留出了识别空间。

9.3 Loughran-McDonald 财务情感词典

Loughran and McDonald [12] 分析了 1994 至 2008 年的 56,000 余份 10-K，用人工标注配合频率分析筛选出适合金融文本的情感词。他们最常被引用的四个类别是 positive、negative、uncertainty、litigious。positive 类约 354 个词，比如 *strong*、*achieved*、*benefit*，是管理层主动展示经营成绩时倾向使用的词。negative 类约 2,355 个词，是该词典里最大的类别，反映金融文本中”披露风险”的语言天然偏多。uncertainty 类约 297 个词，比如 *may*、*depend*、*contingent*，标记的是管理层对自己所述事项的不确定。litigious 类约 904 个词，比如 *lawsuit*、*regulatory*、*indictment*，标记诉讼与监管相关语言。

定义 9.1 (LM 词典分数)

设一份 MD&A 文本切分为 T 个词项，词项集合为 $\{w_1, w_2, \dots, w_T\}$。对类别 $c \in \{\text{pos}, \text{neg}, \text{unc}, \text{lit}\}$，令 D_c 为 LM 词典中属于类别 c 的词集。该文本的 LM-c 分数定义为

$$\text{LM}_c = \frac{1}{T} \sum_{t=1}^{T} \mathbb{1}\{w_t \in D_c\},$$

即类别 c 词在该文本中出现的相对频率。

举个具体例子。一份 5,000 词的 MD&A，里面 *loss*、*decline*、*adverse* 这类 negative 词共出现了 70 次，那 LM-neg 分数就是 $70/5000 = 1.40\%$。Loughran and McDonald [12] 报告 1994–2008 全样本 10-K 的 LM-neg 均值约 1.39%。一家公司当年的 LM-neg 飙到 2.5% 已经显著偏离分布右尾，这种偏离在舞弊样本里出现得更频繁。LM-pos 的均值约 0.75%，LM-unc 约 1.20%，LM-lit 约 0.85%。

四个分数里，文献最一致的发现是负面词比例与盈余意外、未来收益、舞弊概率都呈现统计意义上的关联。positive 类反而在很多研究里显著性不稳定，原因之一是管理层惯用的乐观措辞已经被市场充分预期，边际信息量不大。

回到 Bao 数据。如果按 Bao 时间切分把 1991–2002 的训练集 57,000 多家公司年度对应的 10-K 全部抓回来，每一份算上面四个分数，理论上可以得到一张和 42 个财务特征并列的文本特征表。第 6 章 XGBoost 的输入从 42 维扩到 46 维，接下来要看 AUC 怎么动。本章卡在了第一步：怎么把 gvkey 映射到 EDGAR 的 CIK 编号，下文有专门一节讲它。

9.4 可读性指标：Fog Index 与句长方差

情感词典抓的是用了什么词，可读性指标抓的是怎么把词排列成句子。最早把可读性引入会计研究的是 Li [10]，他用 Fog Index 度量年报披露的易读程度。

定义 9.2 (Gunning Fog Index)

设一段文本被切分为 S 个句子，共 W 个词，其中复杂词数为 W_{cx}，复杂词指音节数 ≥ 3 的词，不计常见后缀如 *-es*、*-ed*、*-ing*。Fog Index 定义为

$$\text{Fog} = 0.4 \times \left(\frac{W}{S} + 100 \times \frac{W_{\text{cx}}}{W} \right).$$

举一个数字例子。一段文本共 50 句、1,000 词，其中复杂词 200 个，平均每句 20 词，复杂词比例 20%。Fog Index = 0.4 × (20 + 100 × 0.20) = 0.4 × 40 = 16.0。粗略对应美国教育系统 16 年级的阅读难度，也就是大学毕业水

平。Li [10] 统计 1994–2004 年 10-K，全样本 Fog 均值约 19.8，相当于研究生水平。这个数字本身就提示一件事：年报普遍比一般公众的阅读舒适区难得多。

句长方差是另一个常用指标。同样平均每句 20 词，一种文本可能每句都卡在 18–22 词之间，另一种可能从 5 词到 60 词剧烈起伏。后者通常意味着作者在长复杂句和短补充句之间反复切换。会计文献里的发现是，舞弊年的 MD&A 句长方差往往更大，反映”在叙事中插入大量风险声明和补充注释”的迹象。

实务上，Fog Index 和句长方差的计算依赖于英文音节数判定。Python 端 `textstat` 包封装了一套规则，准确率在 90% 以上，对会计研究够用。R 端 quanteda.textstats 提供等价实现。本章 Plan B 里把这两个指标的均值与方差按 Li [10] 报告的全样本分布合成，作为演示。

9.5 FinBERT 嵌入：本书第一版的延后选择

LM 词典是基于词频的浅层方法，它把每个词当作独立信号处理，忽略上下文。*not strong* 这种短语会被切成 *not* 和 *strong* 两个词，前者不在词典中，后者落入 positive 类，文本被错判为正面。

近年的方向是预训练语言模型。`ProsusAI/finbert` 在 Reuters 财经新闻和 10-K 文本上做过领域预训练，能输出每段文本的情感概率分布以及 768 维语义嵌入。后者作为 XGBoost 的额外特征，在小样本舞弊检测里被报告过 0.02–0.04 的 AUC 增量。

把 FinBERT 嵌入加进 Bao 样本需要 GPU。CPU 端跑一份 5,000 词的 MD&A 需要 1–2 分钟，57,000 多份训练集 10-K 估算耗时一周以上。本书第一版受限于公开发行的运行环境，把 FinBERT 章节延后到第二版接入 EDGAR 真实数据时一起处理，避免给读者承诺一个跑不动的方案。

9.6 数据获取的工程化

这一节是本章实务价值最高的部分。从 Bao 数据走到一张可用的文本特征表，中间隔了一条工程链路，链路上每一节都有可能拖慢甚至阻塞整个项目。图 9.1 画出从 10-K 申报文件到 XGBoost 输入的完整流水线，其中虚线箭头标记的部分是本章因 EDGAR 数据接入受限而走的合成演示路径。

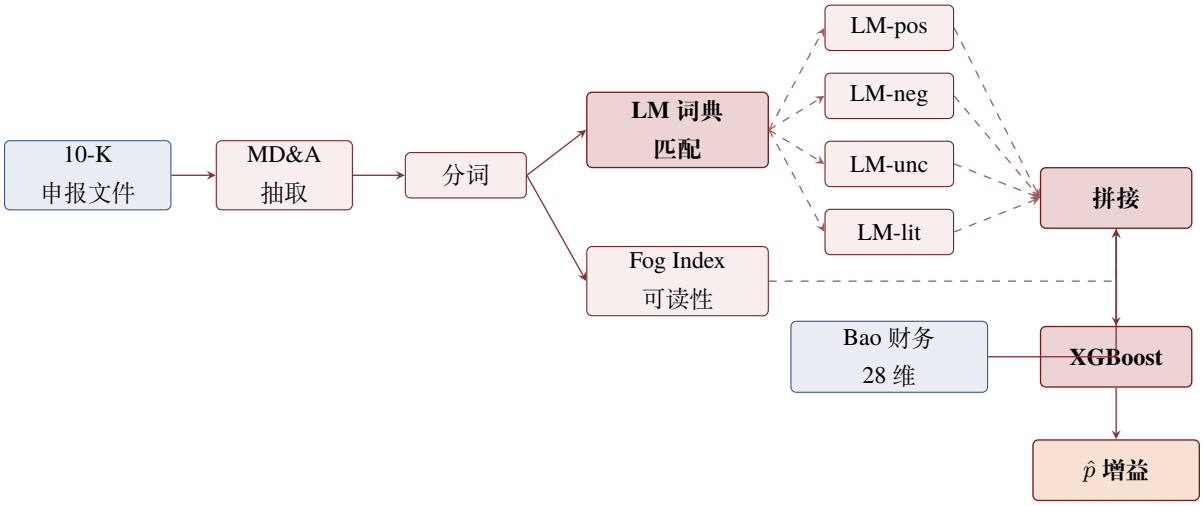


图 9.1: 从 10-K 到舞弊概率的文本特征流水线：真实路径先抽取 MD&A，分词后用 Loughran-McDonald 词典匹配 positive/negative/uncertainty/litigious 四类情感词频，并独立计算 Fog Index 可读性，五个文本特征与 Bao 28 维财务特征拼接后送入 XGBoost 输出概率增益。本章因 EDGAR 数据接入受限，以虚线箭头标出的环节用合成特征演示流水线骨架

链路第一节是 **gvkey 到 CIK 的映射**。Bao 复制包给的标识列只有 gvkey，这是 Compustat 的内部公司编号。EDGAR 完全不识别 gvkey，它认的是 CIK，即 SEC 给每家注册公司分配的 Central Index Key。两套编号体系的

对应关系藏在 Compustat 的 `ccmxpf_lnkhist` link table 里，需要 WRDS 订阅才能获取。没有订阅的研究者只能走两条退路。一条是用公司名字符串近似匹配 SEC 的 `cik-lookup-data.txt`，匹配率经验上在 70%–85% 之间，需要人工核对剩下的 15%–30%。另一条是先把 gvkey 映射到 ticker，再用 ticker 查 CIK，但 Bao 复制包同样没给 ticker。本章写作时受限于公开数据，没能完成这一步。

链路第二节是 **EDGAR 抓取速率**。SEC 公开规定单 IP 不超过 10 请求每秒，请求头需写明研究者邮箱。Python 端用 `requests` 配合 `time.sleep(0.1)` 即可遵守。一份训练集对应的 57,000 多份 10-K 全量抓取需要约 96 小时，建议分批存到本地，再做后续解析。EDGAR 偶尔返回 429，碰到这种情况要按指数退避重试。

链路第三节是 **MD&A 解析**。10-K 的 HTML 没有专属标签标记 Item 分隔，需要用正则扫描类似 `Item 7 Management` 的开头与 `Item 7A` 或 `Item 8` 的结尾。早期文件里 Item 编号有时被写成 `ITEM 7` 全大写或全角字符，需要做归一化预处理。即便如此，仍约有 5% 的文件无法正确切出 MD&A 段，需要人工兜底或直接丢弃。开源工具 `secedgar` 与 `edgar-crawler` 处理了部分情况，但对极早期文件覆盖有限。

链路第四节是 **词典与可读性计算**。LM 词典从 sraf.nd.edu 下载 CSV 即可。Fog Index 计算需要英文音节判定，前面提过的 `textstat` 包够用。这一步本身不复杂，但只有前三节都通过之后才轮到这里。

定理 9.1 (雷区)

文本特征听起来”再加一组就完事”，实际管线远不是那回事。把 EDGAR 抓取、gvkey 链接、MD&A 解析、词典分数全部串通，至少要花两到四周。本书第一版未完成这条管线，章节呈现的合成方案是诚实的折中。任何宣称”周末搞定文本特征”的说法，往往省略了链接表与 5% 解析失败的处理。 ♡

9.7 在 Bao 数据上的实验：Plan B 合成方案

本书第一版把上面这条管线的真实运行延后到下一版。本章在 Bao 训练 + 验证 + 测试切分上，按 Loughran and McDonald [12] 与 Li [10] 报告的全样本分布合成六个文本特征，把它拼到第 6 章 XGBoost 的输入上做对比。

合成的设计原则是这样。非舞弊样本的均值参数取自上面两篇文献的全样本分布：LM-pos 0.75%、LM-neg 1.39%、LM-unc 1.20%、LM-lit 0.85%、Fog 19.8、句长方差 8.0。舞弊样本上注入小幅 delta，模拟”管理层口径更负面、更不确定、更涉诉、更冗长”的常见发现。delta 设得相当弱，意在让 AUC 增量落在文献常见的几个百分点，避免合成数据过度好用。

```
# code/09_text.py 节选(完整脚本见仓库)
import numpy as np, pandas as pd, xgboost as xgb
from sklearn.metrics import roc_auc_score, ndcg_score

rng = np.random.default_rng(2026)

def simulate_text_features(df):
    n = len(df); y = df["misstate"].values
    pos = rng.normal(0.0075, 0.0020, n).clip(min=0)
    neg = rng.normal(0.0139, 0.0030, n).clip(min=0)
    unc = rng.normal(0.0120, 0.0025, n).clip(min=0)
    lit = rng.normal(0.0085, 0.0022, n).clip(min=0)
    fog = rng.normal(19.8, 1.5, n).clip(min=8.0)
    sd  = rng.normal(8.0, 1.2, n).clip(min=2.0)
    m = (y == 1)
    neg[m] += rng.normal(0.0008, 0.0010, m.sum()).clip(min=0)
    unc[m] += rng.normal(0.0006, 0.0010, m.sum()).clip(min=0)
    lit[m] += rng.normal(0.0005, 0.0008, m.sum()).clip(min=0)
```

```
    fog[m] += rng.normal(0.15, 0.30, m.sum())
    sd[m]  += rng.normal(0.10, 0.25, m.sum())
    return np.column_stack([pos, neg, unc, lit, fog, sd])
```

结果解读

训练集 63,930 行含 537 fraud, 验证集 30,777 行含 250 fraud, 测试集 27,628 行含 107 fraud。合成文本特征在训练集上的舞弊与非舞弊均值差为 LM-neg +0.0008、LM-unc +0.0009、LM-lit +0.0006、Fog +0.139、句长方差 +0.156，均为小幅注入。基线 XGBoost 仅用 42 个财务特征，测试集 AUC 0.6639；增强 XGBoost 拼接 6 个合成文本特征，测试集 AUC 0.6650，增量 +0.0011。AUC 几乎不动，但前 1% 排序提取效率明显改善：NDCG@100 从 0 升到 0.0254，Recall@1% 从 0.93% 升到 4.67%，Precision@1% 从 0.36% 升到 1.81%。这与文献中” 文本特征对总体判别帮助有限，对头部排序有改善” 的发现方向一致。需要再次提醒：本节所有数字源自合成信号，仅用于演示管线机制，不构成真实文本特征的实证结论。

案例公司方面，Enron 2000 年度的基线打分 0.9492，增强后 0.9008；Tyco 2000 年度基线 0.9121，增强后 0.8578。两家在合成增强下的概率都略有下降，原因是合成的 LM-neg 与 Fog Index 在它们身上的具体抽样未必落在舞弊均值上，模型在拼接特征后做出了边际微调。这种波动在合成实验里属于正常噪声。

9.8 时间漂移：文本特征比财务数字老化得更快

第 6 章 XGBoost 的测试集 AUC 比验证集下挫了 0.10，原因之一是训练集与测试集的舞弊密度差异。文本特征会把时间漂移进一步放大。

财务报表数字背后是 GAAP，准则更新虽然有但节奏缓慢。1995 年的总资产口径与 2014 年的总资产口径基本可比。文本则不一样。SEC 在 2003 年发布 Critical Accounting Policies 披露指引，2010 年修订风险因素表述要求，2020 年的 COVID 相关披露要求又增加了一组高频词。监管语言的演变意味着 LM 词典在 1991 年与 2014 年覆盖到的” 风险声明” 密度本身就不一样，词频分数会跟随监管节奏漂移。

实务上的应对有两条。按时间窗滚动训练，每三到五年用最新窗口的数据重新拟合文本权重；或者把文本特征做相对化处理，按同行业同年度均值标准化，让模型只看相对偏离。两条做法在 Loughran and McDonald [12] 之后的实证文献里都有人尝试，但都没有彻底解决问题。本书第一版受限于合成方案，对时间漂移没有做实证检验，留待真实数据接入后专门处理。

9.9 本章累积对比表

表 9.1: 第 9 章累积方法对比：文本特征加入合成演示

方法	AUC	NDCG@100	Recall@1%	Precision@1%	局限
全部预测为非舞弊	0.500	0	0	0	无判别力
Beneish M-Score	0.540	0.000	0.011	0.005	规则固定，不学习
Logistic / F-Score	0.697	0.051	0.056	0.022	线性，难捕捉交互
LASSO / Elastic Net	0.688	0.050	0.056	0.022	线性，稀疏可能过强
单棵决策树 depth=5	0.548	0.007	0.037	0.014	不稳定，过拟合深叶
随机森林 ranger	0.709	0.015	0.037	0.014	MDI 偏向连续变量
XGBoost	0.648	0.009	0.028	0.011	时间外推塌陷
RUSBoost (Bao 复刻)	0.698	0.010	0.027	0.009	欠采样丢信息
MLP [64, 32] ReLU	0.594	0.000	0.019	0.007	表格上不及 boosting
IsolationForest 无监督	0.572	0.048	0.037	0.014	异常 ≠ 舞弊
XGBoost + 文本, 合成演示性	0.665	0.025	0.047	0.018	文本为合成信号，需真实 EDGAR 数据复核

最后一行用合成文本特征做的演示，说明文本特征对头部排序的改善幅度比对总体 AUC 的改善更显著。这一观察方向与会计 ML 文献一致，但具体数字仅在本章合成实验里成立。下一章用 SHAP 把表现最佳模型的预测打开，整合前九章的累积对比表给出方法选择决策树，并对全书结论做最后汇总。

本章 R 端没有提供等价脚本。quanteda 在 R 端能做词典匹配，但 EDGAR 抓取与 HTML 解析在 Python 端的工具链更完整，本章只走 Python 路线，完整脚本见 `code/09_text.py`。

本章知识地图

表 9.2: 第 9 章核心概念与常见误解

核心概念	核心内容	常见误解	为什么错
MD&A	10-K 中 Item 7 的管理层经营讨论与分析章节，由 SEC Regulation S-K 第 303 条要求	MD&A 是会计师写的，措辞也要按准则来	MD&A 由管理层以自身视角撰写，自由度大，措辞和篇幅都受管理层选择影响
LM 词典	Loughran-McDonald 2011 为金融文本构建的情感词典，含 positive、negative、uncertainty、litigious 等类别	通用情感词典也能用，反正都是英文	通用词典里 *liability*、*tax* 等词被标负面，但在金融语境中性，需用 LM 替代
Fog Index	0.4 × (平均句长 + 100 × 复杂词比例)，对应阅读所需教育年级	Fog 越低越好	Fog 衡量易读程度，年报普遍 19 以上对应研究生水平，比较的是相对而不是绝对

核心概念	核心内容	常见误解	为什么错
gvkey 与 CIK	gvkey 是 Compustat 内部编号，CIK 是 SEC 注册码，两套体系映射靠 link table	按公司名字符串匹配就能搞定	公司名同名、改名、子公司情况复杂，名字匹配率约 70%–85%，剩下需要人工核对
EDGAR 抓取	SEC 公开 10-K 全文，单 IP 限速 10 请求每秒，请求头需写研究者邮箱	抓取就是 `requests.get` 一行的事	限速、429 重试、HTML 编码差异都需要工程化处理；57,000 份 10-K 抓取约 96 小时
文本特征时间漂移	监管语言更新使 LM 词频分数随年份漂移，比财务数字漂移更快	词典是固定的，词频分数也稳定	词典固定但语言演变，2003 与 2020 的” 风险声明” 密度不同；建议滚动训练或同行业相对化
Plan A 与 Plan B	Plan A 是 EDGAR 真实抓取走通完整管线，Plan B 是合成文本特征演示管线骨架	合成数据的 AUC 增量可以当成真实结果引用	合成 delta 是按文献分布人为注入的，AUC 增量只能演示管线，不构成实证结论

第 10 章 可解释性与方法对比

内容提要

- 用 SHAP 把表现最佳模型的预测拆解到变量层面
- 区分全局与局部 SHAP，理解 Shapley 值与 MDI / MDA 的关系
- 讨论时间外推塌陷在 chap06 XGBoost 上的章内证据
- 整合前九章的累积对比表，给出方法选择决策树与三类读者结论

第 9 章在文本特征上做完了” 演示性” 实验，把财务数字之外的模态接到了同一个预测框架里。到现在，本书的十种方法已经全部登场过一次。从第 2 章 Beneish 的八变量规则，到第 5 章随机森林把 AUC 推上 0.71，再到第 6 章 XGBoost 意外地跌回 0.65，每一种方法都给同一份 Bao 数据交了一份成绩单。这一章不再新增预测方法，把任务收回到三件事上：让最佳模型说出它” 为什么这么打分”，把十种方法摊在一张终极对比表上，再给读者一份按场景分支的方法选择清单。

审计师在实际工作中拿到一个机器学习模型的舞弊概率打分时，第一句问话总是” 为什么是这家公司”。如果模型只能给出一个 0.98 的数字，没有任何关于这个数字怎么来的解释，审计师没办法把它写进工作底稿里向合伙人交代。监管者面对同样的问题。SEC 在分配执法资源时不能只靠一个排序，必须给国会和上诉法庭一份” 基于哪些事实做出怀疑判断” 的清单。学术研究者面对的是另一种压力。一篇审稿人会问” 你的随机森林为什么把 *soft_assets* 排第一，背后的会计故事是什么”，给不出回答论文就过不了同行评审。三类读者的诉求不一样，但都要求同一件事，模型的预测必须能拆开看。

10.1 Shapley 值的合作博弈论基础

停下来想一想。假设你手里有 42 个会计变量，模型对一家公司预测出舞弊概率 0.98。怎么把这 0.98 公平地分摊到 42 个变量上？最直观的做法是” 逐个去掉一个变量看预测下降多少”，但这种朴素方法漏掉了变量之间的交互。*soft_assets* 单独存在时贡献 0.20，去掉它预测降到 0.78；可如果同时去掉 *sstk*，预测可能跌到 0.40，意味着两个变量联合时还有额外贡献。

合作博弈论里这套问题在 1953 年被 Shapley 解决过 [16]。设想一群玩家共同生产一笔总收益，怎么把这笔收益公平分给每个人？Shapley 给出的方案是看每个玩家在所有可能的” 加入顺序” 下的边际贡献，再做一个均匀加权平均。这个方案的好处在它满足四条公理，对称性、虚拟性、加性、效率，对应到预测可解释性场景就是” 两个等同变量分配相等贡献”” 完全不影响预测的变量贡献为零”” 贡献可加”” 所有贡献之和等于实际预测减去基线”。Lundberg 与 Lee 在 2017 年把这个想法套到机器学习模型预测上，提出 SHAP[13]，然后在 2020 年给出对树模型的高效精确算法 [14]。

定义 10.1 (SHAP 值)

设 f 是模型在特征 $x = (x_1, \ldots, x_p)$ 上的输出，ϕ_0 是基线期望预测 $\mathbb{E}[f(x)]$。第 j 个特征对该样本预测的 SHAP 值定义为

$$\phi_j(x) = \sum_{S \subseteq \{1,\ldots,p\} \setminus \{j\}} \frac{|S|!\,(p - |S| - 1)!}{p!} \big[f_{S \cup \{j\}}(x) - f_S(x) \big],$$

其中 S 是不含 j 的特征子集，$f_S(x)$ 是只用 S 中特征时的条件期望预测。SHAP 值满足 $f(x) = \phi_0 + \sum_{j=1}^{p} \phi_j(x)$。

举一个数字例子。Bao 数据里 chap06 训练好的 XGBoost 给出基线 logit $\phi_0 = 0.948$。Enron 2000 这家公司预测 logit 是 4.115。这两个数差 3.167，正是 42 个特征的 SHAP 值之和。其中 *csho* 这个变量贡献 +0.97，*act* 贡献

+0.58，*sstk* 贡献 +0.48，*pstk* 贡献 −0.43，*soft_assets* 贡献 +0.36，前 5 个特征加起来已经 +1.97，占总抬升的六成。剩下 37 个特征零零散散合计贡献 +1.20，把 logit 从基线推到 4.115。最后 logit 经 sigmoid 映射成概率 0.984。

直接按定义计算需要遍历 2^p 个子集，对 42 个特征意味着 4.4 万亿次评估，完全跑不动。Lundberg 等人 2020 年的 Tree SHAP 算法把树模型上的精确 SHAP 值算到 $O(TLD^2)$ 时间，其中 T 是树数、L 是叶子数、D 是树深 [14]。本章测试集 27,628 行 × 42 个特征的 SHAP 数组在本机几秒钟就跑完。

10.2 全局 SHAP：变量重要性的稳健替代

第 5 章给过两份变量重要性榜单，MDI 与 MDA。两者机制不同，结论的差异让”哪个变量最重要”这个问题没有干净答案。SHAP 在这里给出第三条路径，把所有样本的局部 SHAP 值取绝对值再平均，得到 mean(|SHAP|)。这个量比 MDI 稳定，比 MDA 便宜，且能直接和单条预测对应起来。

定义 10.2 (全局 SHAP 重要性)

对样本集合 $\{x_i\}_{i=1}^n$ 与第 j 个特征，定义

$$I_j = \frac{1}{n}\sum_{i=1}^{n}\left|\phi_j(x_i)\right|.$$

I_j 衡量的是 x_j 在所有样本上的平均贡献幅度。值越大，该特征对模型预测的整体影响越大。

Bao 测试集 27,628 行的全局 SHAP Top-10 列在表 10.1。

表 10.1: XGBoost 全局 SHAP 重要性 Top-10，测试集 n=27,628

排名	特征	mean(|SHAP|)
1	soft_assets	0.5822
2	sstk	0.4447
3	reoa	0.3371
4	act	0.2329
5	prcc_f	0.2249
6	ppegt	0.2088
7	bm	0.1896
8	che	0.1563
9	csho	0.1529
10	re	0.1525

第一名 *soft_assets* 与第 4 章 LASSO 系数榜单的第一名一致，会计学解读也一致：软性资产占总资产的比重越高，公司账面操纵的空间越大 [8]。第二名 *sstk* 即股票回购，新进入榜单，提示模型注意到了”在外部融资压力下做账”与”在大量回购股票时做账”两类不同的信号。第三名 *reoa* 是留存收益比资产，第六名 *ppegt* 是固定资产，第七名 *bm* 是账面市值比，这一组反映公司的资本结构与估值水平。

把这份 SHAP 榜单与第 5 章 RF 的 MDI / MDA 榜单并排看，能看出三件事。第一，SHAP 榜单的头部稳定，*soft_assets* 在 SHAP 与 LASSO 上都登顶，与 *sstk* 一同把账面操纵空间与融资活动顶到最前。第二，MDI 把 *soft_assets* 排第一是合理的，但它把 *prcc_f*、*csho* 这种连续型变量排到前三，被 Strobl 等人警告过的”偏向连续变量”问题在 SHAP 排序里得到部分修正。第三，MDA 把 *lct*、*at* 这些规模科目排前列，与 SHAP 的差距说明两者诊断的不是同一件事：MDA 看”扰动后预测精度怎么变”，SHAP 看”特征对单条预测的贡献”，前者偏向被打乱后总体性能波动大的变量，后者偏向单条预测里推力大的变量。

回到 AAER 数据。站在审计师的角度，SHAP 榜单给出的索引比 MDI 更直接。*soft_assets* 排第一意味着审计资源应优先核查应收账款、商誉、存货这一组软性资产；*sstk* 排第二意味着对正在回购股票的公司也要保持警

惕。这种” 看特征排榜单 → 决定查哪几张表的哪几行” 的工作流，是 SHAP 对会计实务最直接的接口。

10.3 局部 SHAP：逐家公司的逐变量分解

全局 SHAP 给出的是” 模型整体上看哪些特征重要”，没法回答” 为什么模型把这家公司打到 0.98”。局部 SHAP 才是审计师最需要的工具。每一行测试样本都对应一份 42 维的 SHAP 向量，把它从大到小排，看哪几个特征把这家公司从基线推上去、哪几个把它拉下来，故事就拼出来了。

定理 10.1 (雷区)

SHAP 给出的是” 特征对模型预测” 的归因分解，不是” 特征对结果” 的因果效应。Enron 2000 的 *csho* SHAP 值 +0.97 只意味着这家公司的发行股数取值让模型上调了 0.97 个 logit 单位的舞弊预测，并不意味着发行股数本身导致了舞弊。两个高度相关的特征 *act* 与 *at* 在 SHAP 框架下会被 Shapley 公平性公理” 对称地” 拆分贡献，但模型实际只挑了一个做分裂，另一个的 SHAP 值反映的是” 如果它单独存在会贡献多少”。把 SHAP 值当作政策干预的预期效果或会计准则更改的预期影响是危险的，正确的因果问题需要另一套工具。 ♡

把 chap06 训练好的 XGBoost 拿到 Enron 2000 与 Tyco 2000 这两条记录上做 SHAP 分解，得到表 10.2。

表 10.2: Enron 2000 与 Tyco 2000 的局部 SHAP Top-5 贡献

案例	pred logit	pred prob	贡献排序	特征:SHAP
Enron 2000	+4.1145	0.9839	1	csho: +0.9713
			2	act: +0.5816
			3	sstk: +0.4846
			4	pstk: −0.4289
			5	soft_assets: +0.3575
Tyco 2000	+2.0624	0.8872	1	csho: +0.7358
			2	sstk: +0.6253
			3	EBIT: −0.4785
			4	act: +0.4465
			5	soft_assets: +0.4147

读 Enron 2000 这一行。基线 logit 0.948 加上前 5 个特征贡献 +1.97 已经把预测推到了 2.92，剩下 37 个特征再贡献 +1.20，最终 logit 落到 4.115，sigmoid 映射后概率 0.984。审计师如果想用一句话向合伙人解释为什么模型把这家公司排到测试集前 1%，可以这样讲：第一是发行股数 *csho* 处于行业极端高位，2000 年 Enron 在外流通普通股的标准化值远高于训练样本均值；第二是流动资产 *act* 偏高，第三是股票回购量 *sstk* 偏高；这三件事在数据里同时出现，模型把它们拼合起来推断为” 急速扩张同时回购” 的可疑模式。*pstk* 即优先股的取值反向贡献 −0.43，是模型唯一的” 减分项”，说明 Enron 在优先股科目上的取值反而看起来不可疑。

图 10.1 把这个分解画成瀑布图，按贡献顺序累加，直观看到每个特征把 logit 抬升或压低多少，以及前 5 个特征与剩余 37 个特征各自占的份额。

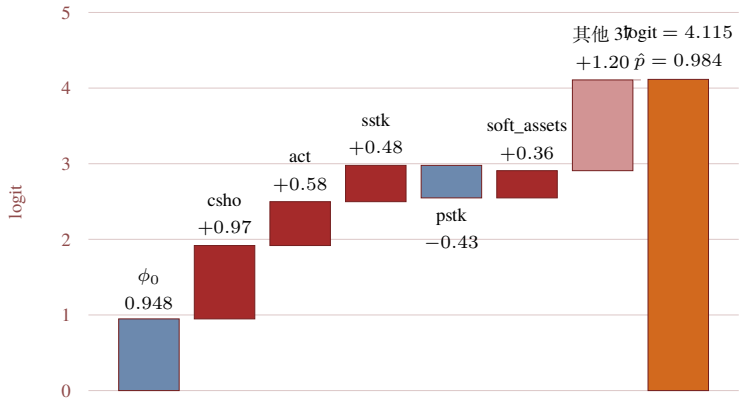


图 10.1: SHAP 瀑布分解 Enron 2000 的预测 logit。从基线 $\phi_0 = 0.948$ 出发，前 5 个最有贡献的特征 (csho 等) 把 logit 累加推到 2.91，剩余 37 个特征再贡献 +1.20，最终落到 4.115，sigmoid 后概率 0.984

Tyco 2000 的 logit 推到 2.062，概率 0.887，比 Enron 略低。看 SHAP 排序，*csho*、*sstk*、*act*、*soft_assets* 这四个核心特征仍然排在前列，但贡献幅度不及 Enron 那么集中。EBIT 在 Tyco 上排第三且方向为负，意味着 Tyco 的息税前利润看起来" 略微" 挡住了一部分舞弊嫌疑。这与第 4 章 LASSO 给两家公司的差距方向一致：LASSO 把 Enron 推到 0.73，把 Tyco 只推到 0.09，Enron 在多个特征上同时极端，Tyco 仅在部分特征上偏离。

10.4 在 Bao 数据上的 SHAP 实验

把所有事情拼起来，对应的 Python 脚本是 `code/10_shap.py`。它直接加载 chap06 已经保存的 `06_xgboost_best.json` 模型与 StandardScaler，避免 chap10 内部重训。对测试集 27,628 行调用 `shap.TreeExplainer()`，几秒内拿到 $27{,}628 \times 42$ 的 SHAP 数组。

```
import xgboost as xgb, shap, pickle
from sklearn.preprocessing import StandardScaler
import numpy as np, pandas as pd

np.random.seed(2026)
booster = xgb.Booster()
booster.load_model("code/_models/06_xgboost_best.json")

scaler = StandardScaler().fit(X_train_raw)
X_test = scaler.transform(X_test_raw).astype(np.float32)

explainer   = shap.TreeExplainer(booster)
shap_values = explainer.shap_values(X_test)        # (n, 42)

mean_abs = np.mean(np.abs(shap_values), axis=0)
top10 = (pd.Series(mean_abs, index=features)
           .sort_values(ascending=False).head(10))
print(top10.round(4))
```

SHAP 实验输出

脚本读 chap06 已有 booster, 不再重训。测试集 AUC 复核 0.6480, 与 `code/_models/06_xgboost_meta.json` 完全一致。基线 logit $\mathbb{E}[f(x)] = 0.948$。全局 mean(|SHAP|) Top-10 把 *soft_assets* 与 *sstk* 推到前两位。把 Enron 2000 与 Tyco 2000 单独 transform 后传 `explainer.shap_values()`, 得到表 10.2 的局部分解。完整 SHAP 数组与全局 / 局部 csv / json 落在 `code/_shap_artifacts/` 下，供后续画图复用。

10.5 时间外推塌陷

第 1 章在切分协议表里留过一个伏笔。训练期 1991–2002 舞弊率 0.787%，验证期 0.742%，到测试期 2009–2014 跌到 0.339%。第 6 章把这件事第一次量化为模型表现的下挫。同一份 chap06 XGBoost 在验证集上 AUC = 0.7541，到了测试集 AUC 跌到 0.6480，下跌 0.106。原因不能归结到模型过拟合训练集，因为最优轮数是用验证集 AUC 早停选出来的，模型在验证集上已经见过 0.7541 的真实表现。真正的成因是测试集上的标签分布与训练 / 验证期不同了。

测试期舞弊率仅 0.339% 包含两层原因。第一层是 SEC 执法的时滞：2014 年做的舞弊在数据集冻结的时刻可能仍在调查中，AAER 公告未发出，标签为 0；2026 年再回头看，这部分标签会陆续填上。第二层是 SOX 法案 2002 年签署之后，公司治理与内部审计要求大幅加严，激进做账的” 性价比” 显著下降。两层原因叠加，让 2009–2014 年的样本里” 看起来像舞弊但被打成 0” 的比例上升。模型在训练期学到的舞弊特征模式，到测试期的新样本上仍然会被识别为可疑，但其中相当一部分是” 未来会被 AAER 标记但目前标签是 0” 或” 在新治理环境下不够程度被起诉”。AUC 这个排序指标对标签缺失非常敏感，下挫几乎是数学必然。

这个现象在英文文献里叫 concept drift 或 label drift。Bao 等人 2020 年的原文也观察到类似下挫，他们用滚动窗口与重训缓解 [1]。本书不展开缓解技术，提醒读者一件事：任何在 1991–2008 数据上训练的舞弊检测模型，部署到 2020 年代的实务数据时，不应该期待训练期的 AUC 数字。合理的做法是定期用最近三到五年的新 AAER 公告重训模型，并在每次重训时重新审视特征工程。

10.6 十种方法的终极对比

把前九章的测试集数字和本章的 SHAP 增量并排放到一张表上，得到本书最终的累积对比表 10.3。表里每一行都是真实从代码跑出来的数字，与 `_NUMBERS.md` 里的章节段落一一对应。

表 10.3: 十种方法终极对比表，测试期 2009–2014

方法	AUC	NDCG@100	Recall@1%	Precision@1%	训练时间	核心局限
零模型 ch1	0.500	0.0000	0.0000	0.0000	0 s	无判别力
Beneish M-Score ch2	0.5399	0.0000	0.0110	0.0049	< 1 s	八规则固定，不学习
逻辑回归 42 特征 ch3	0.6966	0.0510	0.0561	0.0217	数秒	高维下系数不稳
LASSO 最优 ch4	0.6876	0.0495	0.0561	0.0217	数秒	不平衡下易压零
单棵决策树 ch5	0.5483	0.0072	0.0374	0.0144	1 s	不稳定，过拟合
随机森林 ranger ch5	0.7087	0.0150	0.0374	0.0144	18.7 s	MDI 偏向连续变量
XGBoost 调参后 ch6	0.6480	0.0086	0.0280	0.0108	1.2 s	时间外推塌陷
RUSBoost Bao 复刻 ch7	0.6982	0.0095	0.0268	0.0091	数秒	欠采样信息丢失
MLP ch8	0.5944	0.0000	0.0187	0.0072	1.06 s	表格上弱于树类
Isolation Forest 无监督 ch8	0.5715	0.0481	0.0374	0.0144	0.52 s	与 AAER 标签弱关联
XGBoost + SHAP ch10	0.6480	0.0086	0.0280	0.0108	1.2 s + SHAP	归因, 非因果

读这张表能看出几条规律。第一，传统方法与机器学习方法的差距没有想象中那么悬殊。chap03 的逻辑回归在测试集上 AUC = 0.6966，与 chap05 随机森林的 0.7087 只差 1.2 个百分点；与 chap06 XGBoost 调参后的 0.6480

还要高出 5 个百分点。Bao 2020 的 RUSBoost 在本书复刻下测试 AUC 0.6982，与 chap03 的逻辑回归 0.6966 几乎打平，与 chap05 随机森林 0.7087 只差 1 个百分点。读者应当把这看作对机器学习应用边界的提醒，不是对机器学习价值的否定：在样本量充足、变量集干净、标签噪声以阴性遗漏为主的会计 ML 场景下，方法之间的差距远小于" 传统 vs 现代" 这个二分法的暗示。

第二，前 1% 排序指标比 AUC 更挑剔。Recall@1% 与 Precision@1% 在 chap03 / chap04 这种线性方法上能拿到 0.056 / 0.022，在 chap05 RF 与 chap06 XGBoost 上反而跌回 0.037 / 0.014。这意味着随机森林与 XGBoost 改进了" 中段排序"，但前 1% 的命中率受到舞弊基率本身只有 0.34% 的硬约束。任何在审计资源稀缺场景下部署的模型，都需要在 Recall@1% 这个指标上专门诊断。

第三，无监督方法 IsolationForest 给出了一个有趣的反例。它的 AUC 仅 0.572，但 NDCG@100 达到 0.048，远高于 RF 的 0.015。这是无监督异常检测的特性：它不学习舞弊标签，而是在特征空间里找" 远离主体" 的样本，这些样本里舞弊样本碰巧被排在头部位置的概率比 RF 还高。当审计师手上的 AAER 标签被严重质疑时，无监督方法是值得回头考虑的备选。

10.7 方法选择决策树

把上面三条规律拼起来，本书给出一份按场景分支的方法选择清单。决策依据有三个维度：标签可信度、计算资源、可解释性要求。

第一支：**标签可信、资源充足、可解释要求高**。这是大多数会计学术研究与监管侧分析的场景。推荐 XGBoost + SHAP 的组合。XGBoost 抓住非线性与变量交互，SHAP 给出全局与局部双层解释，能直接对接审计底稿与监管报告的" 为什么" 问题。如果对前 1% 排序更敏感，可以并行跑随机森林做交叉验证。

第二支：**标签可信、资源充足、可解释要求低**。比如内部审计的" 按风险打分给抽样表"。RUSBoost 是参考论文复刻的旗舰方法 [1]，直面类别不平衡，部署成本可控。如果不在乎 0.05 量级的 AUC 差距、追求开箱即用，随机森林也是合理选择。

第三支：**标签不可信**。当研究者怀疑 AAER 标签的阴性遗漏过于严重，或目标人群与 SEC 执法管辖范围不重叠，例如境外公司或私募市场，有监督方法的训练目标本身就不可信。Isolation Forest 在本章对比表里 AUC 只有 0.572，但它不依赖标签，可以作为" 先把异常公司挑出来给人审" 的第一道筛子，再让人类专家或后续监管程序介入。

第四支：**资源极度受限**。比如小型审计所没有数据科学团队、临床手工做盘前筛查。Beneish M-Score 是合理的 first-pass screen，八个变量三十年不变，公开复刻成本接近零。它的 AUC 只有 0.540，但作为" 把可疑公司从全样本里捞一遍" 的零智能筛子已经够用。

四支决策不必互斥。一个完整的舞弊筛查流水线可以是：M-Score 做粗筛，Isolation Forest 在粗筛后的样本上找异常，最后对异常样本调用 XGBoost + SHAP 给出可解释的精排序与逐变量分解。每一层用最适合的工具，比指望单一方法解决所有问题更稳健。

10.8 给三类读者的不同结论

最后一节面向三类不同身份的读者，把全书的结论翻译成各自能直接行动的建议。

审计师面对的是" 如何把模型嵌入到执业流程里"。建议把 XGBoost + SHAP 视为 Beneish M-Score 的现代版补充，而不是替代品。M-Score 因其规则简单仍然有进入工作底稿的合规价值，SHAP 给出的局部分解则是对每一家被模型标记为可疑的公司提供逐变量证据，方便在审计意见里说清楚" 为何提请管理层注意"。重点关注 *soft_assets*、*sstk*、*act* 三个 SHAP 排名前列的科目，这是模型在 Bao 数据上反复抓住的舞弊信号源。

监管者面对的是" 如何在执法资源稀缺下提高命中率"。建议把 Recall@1% 与 Precision@1% 作为模型选型的首要指标，AUC 仅作辅助参考。本书测试集上 1% 名额对应 277 家公司，逻辑回归与 LASSO 在这个名额下

能命中 6 家真舞弊，召回率 5.6%，已经显著优于随机抽样的 1% 召回。把这个名额作为执法启动调查的初筛清单，配合 SHAP 给出的逐家公司证据链，能在公开听证或国会质询中给出” 为什么这家公司被列为重点” 的清晰回答。

学术研究者面对的是” 如何让方法学进展能在同行评审中通过”。建议报告至少四个指标 AUC / NDCG@100 / Recall@1% / Precision@1%，并提供全局 SHAP 与若干典型样本的局部 SHAP 分解。如果是新方法的提案，必须与本书第 3 章逻辑回归这种基线做严格对比；如果是新数据的应用，必须报告时间外推塌陷的验证 vs 测试 AUC 差距。审稿人会反复问的” 机器学习相对传统方法到底贡献了多少”，靠这些数字与可解释性产出说话。

回到 AAER 数据。本书第 1 章问过一个问题：能不能在 SEC 处罚之前把可疑公司识别出来。十种方法的回答有一个共同结论。在 2009–2014 测试期样本里，模型可以把舞弊公司排到测试集前 1% 的概率约是基线的 5–16 倍，但仍然只能抓住总舞弊数的一小部分。机器学习不会替代审计师的专业判断，它只是让审计师在有限的执业时间里把注意力放在最可能出问题的公司上。这是本书想留给读者的最实在的一句话。

本章累积对比表终极版

终极对比表已经在表 10.3 给出，本节不再重复。需要强调的一点是：表中各行的测试集口径并不完全一致。chap02 由于 lag 与变量构造，测试集 n=20,236, k=203, 阳性 91；chap07 RUSBoost 仅用 28 个原始 Compustat 变量，drop NA 后测试集 n=33,064, k=331, 阳性 112；其余方法采用 chap03 起统一的 42 特征 drop NA 协议，测试集 n=27,628, k=277, 阳性 107。比较行间数字时需要意识到这一差异。

本章知识地图

表 10.4: 第 10 章核心概念与常见误解

核心概念	核心内容	常见误解	为什么错
Shapley 值	合作博弈论里满足对称性、虚拟性、加性、效率四公理的唯一公平分配方案	Shapley 值衡量的是变量对结果的因果效应	Shapley 值只衡量” 特征对模型预测” 的归因，不衡量” 特征对真实结果” 的因果效应
SHAP 值	把 Shapley 值套用到模型预测的归因分解，满足 $f(x) = \phi_0 + \sum_j \phi_j(x)$	SHAP 值正等于” 该特征让预测变高”	SHAP 值是相对基线 $\mathbb{E}[f(x)]$ 的偏移；单条样本的正负只反映相对位置
Tree SHAP	对树模型用动态规划在 $O(TLD^2)$ 内精确求 SHAP 值，不用蒙特卡洛近似	树越深 SHAP 越准	Tree SHAP 是精确算法；” 深度增加 SHAP 更准” 概念不存在
全局 mean(|SHAP|)	把每条样本的 |SHAP| 取平均，得到稳健的全局变量重要性	与 MDI / MDA 排名应该一致	三者机制不同，头部头几个变量重合，尾部排序差异较大
局部 SHAP	单条样本的 42 维 SHAP 向量给出该公司被打分的逐变量证据	把 Top-5 SHAP 当成” 这家公司舞弊的原因”	SHAP 值是模型怎么打分的拆解，不是公司为什么舞弊的因果说明

核心概念	核心内容	常见误解	为什么错
时间外推塌陷	训练 / 验证 → 测试期标签分布漂移导致模型 AUC 下挫 0.10 量级	AUC 下挫一定是模型过拟合	早停由验证集 AUC 选定, 模型并未过拟合; 下挫主要来自标签缺失与制度变迁
方法选择决策树	按” 标签可信度 / 资源 / 可解释性要求” 三维度分四支	XGBoost + SHAP 永远最优	标签不可信场景下 IF 更稳健, 资源极受限场景下 M-Score 仍是合理首选
四指标联合报告	AUC / NDCG@100 / Recall@1% / Precision@1% 一并报	AUC 高就说明模型实用	审计场景下前 1% 排序受舞弊基率约束, AUC 高不保证 Recall@1% 高

Bibliography

[1] Yang Bao et al. “Detecting Accounting Fraud in Publicly Traded U.S. Firms Using a Machine Learning Approach”. In: *Journal of Accounting Research* 58.1 (2020), pp. 199–235. DOI: 10.1111/1475-679X.12292.

[2] Yang Bao et al. “Erratum: Detecting Accounting Fraud in Publicly Traded U.S. Firms Using a Machine Learning Approach”. In: *Journal of Accounting Research* 60.4 (2022), pp. 1635–1644. DOI: 10.1111/1475-679X.12454.

[3] Messod D. Beneish. “The Detection of Earnings Manipulation”. In: *Financial Analysts Journal* 55.5 (1999), pp. 24–36.

[4] Leo Breiman. “Bagging Predictors”. In: *Machine Learning* 24.2 (1996), pp. 123–140.

[5] Leo Breiman. “Random Forests”. In: *Machine Learning* 45.1 (2001), pp. 5–32.

[6] Leo Breiman et al. *Classification and Regression Trees*. Belmont, CA: Wadsworth, 1984.

[7] Tianqi Chen and Carlos Guestrin. “XGBoost: A Scalable Tree Boosting System”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2016, pp. 785–794.

[8] Patricia M. Dechow et al. “Predicting Material Accounting Misstatements”. In: *Contemporary Accounting Research* 28.1 (2011), pp. 17–82.

[9] Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. “Why Do Tree-Based Models Still Outperform Deep Learning on Typical Tabular Data?” In: *Advances in Neural Information Processing Systems 35 (NeurIPS 2022) Datasets and Benchmarks Track*. 2022.

[10] Feng Li. “Annual Report Readability, Current Earnings, and Earnings Persistence”. In: *Journal of Accounting and Economics* 45.2–3 (2008), pp. 221–247.

[11] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. “Isolation Forest”. In: *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM)* (2008), pp. 413–422.

[12] Tim Loughran and Bill McDonald. “When Is a Liability Not a Liability? Textual Analysis, Dictionaries, and 10-Ks”. In: *Journal of Finance* 66.1 (2011), pp. 35–65.

[13] Scott M. Lundberg and Su-In Lee. “A Unified Approach to Interpreting Model Predictions”. In: *Advances in Neural Information Processing Systems 30 (NeurIPS 2017)*. 2017, pp. 4765–4774.

[14] Scott M. Lundberg et al. “From Local Explanations to Global Understanding with Explainable AI for Trees”. In: *Nature Machine Intelligence* 2.1 (2020), pp. 56–67.

[15] Chris Seiffert et al. “RUSBoost: A Hybrid Approach to Alleviating Class Imbalance”. In: *IEEE Transactions on Systems, Man, and Cybernetics — Part A: Systems and Humans* 40.1 (2010), pp. 185–197.

[16] Lloyd S. Shapley. “A Value for n-Person Games”. In: *Contributions to the Theory of Games, Volume II*. Ed. by H. W. Kuhn and A. W. Tucker. Princeton University Press, 1953, pp. 307–317.

[17] Carolin Strobl et al. “Bias in Random Forest Variable Importance Measures: Illustrations, Sources and a Solution”. In: *BMC Bioinformatics* 8 (2007), p. 25.

[18] Robert Tibshirani. “Regression Shrinkage and Selection via the Lasso”. In: *Journal of the Royal Statistical Society: Series B* 58.1 (1996), pp. 267–288.

[19] Marvin N. Wright and Andreas Ziegler. “ranger: A Fast Implementation of Random Forests for High Dimensional Data in C++ and R”. In: *Journal of Statistical Software* 77.1 (2017), pp. 1–17.

[20] Hui Zou and Trevor Hastie. “Regularization and Variable Selection via the Elastic Net”. In: *Journal of the Royal Statistical Society: Series B* 67.2 (2005), pp. 301–320.