软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2024,35(9):4448−4468 [doi: 10.13328/j.cnki.jos.006972]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

规则与概率相结合的不一致数据子集修复方法*

张安珍, 司佳宇, 梁天宇, 朱 睿, 邱 涛

(沈阳航空航天大学 计算机学院, 辽宁 沈阳 110136)

通信作者: 朱睿, E-mail: zhurui@sau.edu.cn

摘 要: 不一致数据子集修复问题是数据清洗领域的重要研究问题, 现有方法大多是基于完整性约束规则的, 采用最小删除元组数量原则进行子集修复. 然而, 这种方法没有考虑删除元组的质量, 导致修复准确性较低. 为此, 提出规则与概率相结合的子集修复方法, 建模不一致元组概率使得正确元组的平均概率大于错误元组的平均概率, 求解删除元组概率和最小的子集修复方案. 此外, 为了减小不一致元组概率计算的时间开销, 提出一种高效的错误检测方法, 减小不一致元组规模. 真实数据和合成数据上的实验结果验证所提方法的准确性优于现有最好方法.

关键词: 不一致数据; 函数依赖; 子集修复; 概率图网络

中图法分类号: TP311

中文引用格式: 张安珍, 司佳宇, 梁天宇, 朱睿, 邱涛. 规则与概率相结合的不一致数据子集修复方法. 软件学报, 2024, 35(9): 4448–4468. http://www.jos.org.cn/1000-9825/6972.htm

英文引用格式: Zhang AZ, Si JY, Liang TY, Zhu R, Qiu T. Subset Repair Method Combining Rules and Probabilities for Inconsistent Data. Ruan Jian Xue Bao/Journal of Software, 2024, 35(9): 4448–4468 (in Chinese). http://www.jos.org.cn/1000-9825/6972.htm

Subset Repair Method Combining Rules and Probabilities for Inconsistent Data

ZHANG An-Zhen, SI Jia-Yu, LIANG Tian-Yu, ZHU Rui, QIU Tao

(School of Computer Science, Shenyang Aerospace University, Shenyang 110136, China)

Abstract: Subset repair for inconsistent data is an important research problem in the field of data cleaning. Most of the existing methods are based on integrity constraint rules and adopt the principle of the minimum number of deleted tuples for subset repair. However, these methods take no account of the quality of deleted tuples, and the repair accuracy is low. Therefore, this study proposes a subset repair method combining rules and probabilities. The probability of inconsistent tuples is modeled so that the average probability of correct tuples is greater than that of wrong tuples, and the optimal subset repair with the smallest sum of the probability of deleted tuples is calculated. In addition, in order to reduce the time overhead of calculating the probability of inconsistent tuples, this study proposes an efficient error detection method to reduce the size of inconsistent tuples. Experimental results on real data and synthetic data verify that the proposed method outperforms the state-of-the-art subset repair method in terms of accuracy.

Key words: inconsistent data; functional dependency; subset repair; probabilistic graph network

数据在集成和处理过程中经常产生各种各样的错误, 这些错误可能导致数据违反给定的完整性约束规则, 造成数据不一致. 不一致数据极大地降低了数据的可用性, 甚至引发错误决策, 造成严重的经济损失. 在数据清洗领域, 一种常见的完整性约束规则是函数依赖 (functional dependency, FD), FD 可以很好地表达属性之间的语义约束关系, 为方便讨论, 本文采用 FD 作为约束规则, 其他规则可以通过一定的扩展适用. FD 的一般形式为 $X \to Y$, 表示 X 中属性可以函数确定 Y 中属性, X 和 Y 为关系 R 上的属性集合, X 称为左部属性, Y 称为右部属性. 若数据中存在两个元组的左部属性值相同, 而右部属性值不同, 则这两个元组违反 FD, 是不一致元组.

* 基金项目: 国家自然科学基金青年基金 (62102271, 62002245); 辽宁省教育厅基础研究项目 (JYT2020027)
收稿时间: 2022-03-27; 修改时间: 2022-06-28, 2022-11-10, 2023-03-04; 采用时间: 2023-05-22; jos 在线出版时间: 2023-09-27
CNKI 网络首发时间: 2023-10-31

例如, 图 1(a) 为部分订单信息表, ZIP、CT、STR 和 AC 分别代表邮政编码、城市名称、街道名称和区域编号. 该表中共有 4 个错误元组, 其中 t_3 和 t_9 的 ZIP 真实值为 2135, 由于发生替换错误 (替换为值域范围内的另一个值) 变成了 5039; t_6 的 CT 真实值为 Brighton, 由于发生拼写错误变为 Brightn (少写了一个字母 o); 类似地, t_{13} 也发生了拼写错误, ZIP 值由 2135 变为 2135x. 假设该表上只有一条 FD: ZIP→CT, 根据 FD 的语义可知, ZIP 值相同的元组其 CT 值也应相同, 因此将表中元组按 ZIP 值划分到图 1(b) 中的不同分组中时, 每个分组中元组的 CT 值应相同. 然而, 在 ZIP=5039 分组中, t_1 的 CT 值为 Corinth, t_3 与 t_9 的 CT 值为 Brighton, 因此 ZIP=5039 分组中的元组不一致, 同理可知, ZIP=2135 分组中的元组不一致, 而 ZIP=2135x 分组中只有一条元组 t_{13}, 因此是一致的.

	ZIP	CT	STR	AC
t_1	5039	Corinth	Pond	340
t_2	2135	Brighton	Abby	857
t_3	5039(2135)	Brighton	Abby	857
t_4	2135	Brighton	Vinal	857
t_5	2135	Brighton	Vinal	857
t_6	2135	Brightn(Brighton)	Abby	857
t_7	2135	Brighton	Vinal	857
t_8	2135	Brighton	Vinal	857
t_9	5039(2135)	Brighton	Vinal	857
t_{10}	2135	Brighton	Vinal	857
t_{11}	2135	Brighton	Vinal	857
t_{12}	2135	Brighton	Vinal	857
t_{13}	2135x(2135)	Brighton	Vinal	857
t_{14}	2135	Brighton	Vinal	857

(a) 订单信息表

ZIP值	元组集合
ZIP=5039	t_1, t_3, t_9
ZIP=2135	t_2, t_4, t_5, t_6, t_7 t_8, t_{10}, t_{11}, t_{12}, t_{14}
ZIP=2135x	t_{13}

(b) 按ZIP分组的结果

保留元组	t_2, t_3, t_4, t_5, t_7, t_8 t_9, t_{10}, t_{11}, t_{12}, t_{13}, t_{14}
删除元组	t_1, t_6

(c) 子集修复方案

图 1 部分订单信息表及基于最小删除元组数量原则的最优子集修复方案

针对数据不一致问题, 一种常见的解决方法是子集修复[1−5], 通过删除一些元组使得保留的元组满足 FD. 理想情况下, 删除的元组都是错误元组, 保留的元组都是正确元组, 此时修复准确性最高. 一般来说, 不一致数据上存在多种子集修复方案, 为了减小对原始数据的改动, 现有方法[4,6]通常采用最小删除代价原则, 将删除代价最小的作为最优方案, 常见的删除代价为删除元组的数量. 例如, ZIP=5039 分组中有两种子集修复方案, 一是删除 t_1 保留 t_3 和 t_9, 二是删除 t_3 和 t_9, 保留 t_1, 根据最小删除元组数量原则, 现有方法将删除 t_1; 同理可知, 对于 ZIP=2135 分组, 现有方法将删除 t_6; 而对于 ZIP=2135x 分组, 由于 t_{13} 是一致的, 因此现有方法将其保留, 最终的修复结果如图 1(c) 所示. 对比图 1(a) 中的真实值可知, 该修复方案删除的 2 个元组中只有 t_6 是错误的, 因此, 删除错误元组的准确率为 1/2, 此外, 数据中还有 3 个错误元组没有被删除 (t_3, t_9, t_{13}), 因此, 删除错误元组的召回率为 1/4.

由此可见, 现有子集修复方法的修复准确性不高, 经分析, 主要原因有两个: (1) 当左部属性发生替换错误时, 一些分组内的元组分布发生了较大改变, 此时错误元组的数量大于正确元组的数量, 若按照最小删除元组数量原则进行子集修复, 将删除数量较少的正确元组而保留数量较多的错误元组. 例如, 图 1(b) 的 ZIP=5039 分组, 当数据中没有错误时, 该分组中只有 t_1, 此时, 正确元组的数量是最多的; 然而, 当左部属性 ZIP 发生替换错误时, t_3 和 t_9 进入该分组, 导致正确元组数量不再是最多的. (2) 当左部属性发生拼写错误时, 会出现一些异常分组, 这些分组中通常只有一个元组, 因此不会违反 FD, 现有方法无法将其删除. 例如, 图 1(b) 中的 ZIP=2135x 分组, 该分组是由于 t_{13} 的左部属性 ZIP 出现拼写错误而生成的.

针对第 1 个问题, 我们有如下观察, 错误元组的数量虽然多, 但是其概率通常较小, 若能合理地建模不一致元组概率, 使得正确元组的平均概率大于错误元组的平均概率, 则可将删除元组的平均概率与数量之积 (即概率之和) 作为删除代价, 求解删除代价最小的子集修复方案, 从而解决错误元组数量较多的问题. 然而, 在子集修复过程中引入概率信息将带来以下挑战: 一方面, 如何设计不一致元组概率模型使得正确元组的平均概率大于错误元组的平均概率是一个巨大挑战. 例如, 假设数据中只有一条 FD: $X \to A$, 对于不一致元组 t, t 出错的属性可能是左部属性 X, 也可能是右部属性 A, 还有可能是 X 和 A 都出错, 令 $Y = U - X - A$, U 为全部属性集合, 则 Y 中属性没有错误. 此时, 一种直观的想法是将 t 的概率建模为给定正确属性 Y 的取值时, 可能出错的 X 属性和 A 属性取当前值

的概率. 值得一提的是, 该模型常被用在数据清洗领域[5,7,8]中. 然而, 我们通过理论分析发现, 若 X 属性与 Y 属性之间存在相关性, 则该模型无法保证错误元组的平均概率小于正确元组的平均概率. 另一方面, 引入概率后子集修复成本将大大提高, 计算全部不一致元组概率的时间开销不可小觑, 如何保证子集修复的效率是另一大挑战. 针对第 2 个问题, 一种直观的想法是通过异常分组检测将异常分组识别出来, 如何正确识别异常分组特征是我们面临的又一挑战.

针对上述挑战, 本文提出规则与概率相结合的子集修复方法 HybridOSR (hybrid optimal subset repair), 兼顾删除元组的质量与数量, 提高子集修复准确性. HybridOSR 主要包括 3 个阶段: 错误检测、概率计算及子集修复. 首先, 错误检测阶段将删除异常分组和大分组 (分组规模大于平均分组规模 2/3) 中的错误元组, 减小不一致元组规模, 具体如下: 对于异常分组, 我们观察发现其通常具备两个特征, 一是分组的规模较小, 二是异常分组的名字与正常分组的名字的编辑距离较小, 因此, 我们可以设置分组规模与编辑距离的阈值将异常分组检测出来. 对于大分组, 由于数据中的错误通常较少 (实际应用中的错误率通常在 5% 左右, 最多不会超过 30%[9,10]), 因此发生错误时, 大分组中的元组分布通常不会受到很大影响, 正确元组数量仍然是最多的, 换言之, 数量较少的即为错误元组. 例如, 图 1(b) 中的平均分组规模为 14/3, 而 ZIP=2135 分组中的元组数量为 10, 大于平均分组规模 2/3, 是一个大分组, 该分组中 CT 值出现频率最多的是 Brighton, 较少的是 Brightn, 我们 Brightn 对应的 t_6 元组删除后该分组中其他元组不再违反 FD, 此时不一致元组只剩 t_1 , t_3 和 t_9 , 不一致元组规模大大减小.

HybridOSR 概率计算阶段首先将不一致元组概率建模为给定正确属性取值时, 可能出错的左部属性取当前值的概率, 然后利用极大似然估计方法估计元组概率, 子集修复阶段将删除元组的平均概率与数量之积 (即概率之和) 作为删除代价, 近似计算删除代价最小的子集修复方案. 例如, 图 1(b) 中的 ZIP=5039 分组中有 2 个错误元组和 1 个正确元组, 根据本文概率模型与计算方法可知, 正确元组概率为 1 , 而错误元组 t_3 和 t_9 . 的概率分别为 0.33 和 0.1 (详细计算过程见第 3.3 节), 删除 t_1 的代价为 1, 删除 t_3 和 t_9 的代价为 0.43, 因此, HybridOSR 将删除代价较小的错误元组 t_3 和 t_9 , 而保留正确元组 t_1 . 本文主要贡献如下.

(1) 提出基于阈值的异常分组检测方法, 有效解决由拼写错误引发的异常分组问题, 提高子集修复召回率.

(2) 提出大分组中错误元组的检测方法, 有效减小不一致元组规模, 并通过理论证明该方法的正确性.

(3) 设计不一致元组概率模型, 并证明该模型可以保证正确元组的平均概率大于错误元组的平均概率.

(4) 设计规则与概率相结合的子集修复代价模型, 兼顾删除元组的质量与数量, 有效解决由替换错误引发的错误元组数量大于正确元组数量问题, 提高子集修复的准确率和召回率.

(5) 在真实数据集和合成数据集上将本文方法与多种方法对比, 实验结果表明本文方法的修复准确性最高.

本文第 1 节介绍子集修复相关工作. 第 2 节介绍预备知识, 并给出规则与概率相结合的最优子集修复问题形式化定义. 第 3 节介绍不一致数据子集修复框架 HybridOSR 的总体流程及每个阶段的具体实现方法. 第 4 节通过实验验证本文提出方法的有效性. 第 5 节总结全文.

1 相关工作

目前, 数据清洗领域的研究工作按照采用的方法可分为 3 类: 基于规则的方法, 基于概率的方法以及规则与概率相结合的方法. 针对不一致错误, 现有方法按照对数据的操作可分为子集修复和更新修复, 其中子集修复[1−5]通过删除一些元组使得保留元组一致, 更新修复[4,5,11,12]通过修改一些元组的属性值使得修改后的数据一致.

● 基于规则的方法. 针对子集修复, 基于规则的方法通常将最小删除元组数量作为优化目标求解最优子集修复方案[11,13]. Arenas 等人[14]证明当数据中只有两条 FD 规则时, 求解最优子集修复方案是 NP-complete. Livshits 等人[4]证明大部分情况下最优子集修复问题是 APX-complete 的. Miao 等人[6]证明当关系模式和 FD 规则固定时, 最优子集修复问题在大多数情况下的近似比下界为 17/16, 并给出了近似比为 $(2-1/2^{|\Sigma|-1})$ 的近似算法. 针对更新修复, 基于规则的修复方法通常以最小化更新代价作为优化目标求解最优更新修复方案. 理论研究成果表明, 最优更新修复的计算复杂度同样是 NP-hard 和 APX-complete 的[4]. 为此, 研究人员提出了大量近似计算方法[3,15]以及启发式方法[16], 如 NADEEF[17]、BigDansing[18]、 Temporal[19]、LLUNATIC[20].

● 基于概率的方法. 针对更新修复, 基于概率的修复方法利用数据的概率统计特性检测错误并指导修复[7,8,21−24]. Berti-Équille 等人[21]提出迭代式修复框架 DEC, 利用联合概率分布修复数据错误. Dasu 等人[22]提出了“统计失真”定义修复后的数据分布与理想数据分布之间的距离. Hellerstein[23]提出了一种高效的异常值检测方法. Yakout 等人[7]提出了基于最大似然估计的错误修复框架 SCARE. Krishnan 等人[24]设计了增量式数据清洗框架 ActiveClean. Mayfield 等人[8]提出了基于信念传播和关系依赖网络的迭代修复框架 ERACER.

● 规则与概率结合的方法. 为了提高更新修复准确率, Prokoshyna 等人[25]最早将概率与规则相结合, 提出概率扰动最小的修复方案的求解方法. 随后, Ge 等人[9,26]提出了基于马尔可夫逻辑网络与拒绝约束规则的混合式数据清洗框架. Rekatsinas 等人[5]提出将完整性约束、知识库、统计学等若干修复方法相结合的 HoloClean 系统. 由于满足约束规则的方法不止一个, Yu 等人[27]提出选取概率较大的几种修复方案提交给用户, 让用户从中选择真实值, 然后根据反馈结果更新概率计算模型.

讨论: (1) 基于规则的修复方法通常以最小代价为优化目标求解最优修复方案, 已有研究工作可分为两类, 一类致力于提高近似算法的近似比, 以减小近似修复方案代价与最优修复方案代价之间的差距, 然而这类研究工作通常只考虑删除元组的数量, 而忽略了删除元组的质量, 导致修复准确性不高; 另一类研究工作致力于提高修复准确性, 提出了多种启发式修复方法, 但是修复结果没有近似比或概率保证, 修复可靠性较低. 本文提出的方法可以在保证近似比的前提下, 提高子集修复准确性.

(2) 基于概率的修复方法利用概率统计信息建模元组质量, 并依此检测和修复数据中的错误. 然而, 当数据中不一致元组的数量较多时, 基于概率的方法很难发现其中的错误, 进而无法将其修复, 修复结果仍然是不一致的. 不难发现, 将完整性约束规则与概率统计信息结合起来, 可有效提高错误检测与修复的准确性.

(3) 现有规则与概率相结合的方法都是针对更新修复的, 若使其适用于子集修复, 一种直观的想法是将更新修复方案中更新的元组删除, 从而得到一个子集修复方案, 然而现有规则与概率相结合的更新修复方法都是启发式的, 通过这种方式得到的子集修复方案没有任何近似比保证, 换言之, 这种方式得到的子集修复方案的删除代价与最优子集修复方案的删除代价的差距可能非常大. 此外, 现有规则与概率相结合的更新修复方法中的概率模型无法保证错误元组的平均概率小于正确元组的平均概率, 进而无法解决小分组中错误元组数量大于正确元组数量的问题 (详细分析见第 3.3 节), 从而导致修复准确性不高, 而本文提出的概率模型及子集修复方法可以保证删除代价近似比为 2 的前提下, 提高修复准确性.

2 预备知识及问题定义

本节首先介绍函数依赖与子集修复背景知识, 其次给出规则与概率相结合的最优子集修复问题定义, 最后介绍一种直观的求解方法并分析该方法的不足. 需要指出的是, 本文只考虑由拼写错误、替换错误等引发的数据不一致问题, 其他类型的错误 (如缺失值、重复值等) 不在本文研究范围内.

2.1 函数依赖与子集修复

函数依赖反映了不同属性之间多对一的约束关系, 可以保证数据一致性, 大量的数据质量管理工作都建立在其基础之上. 为了方便讨论, 本文采用函数依赖作为约束规则, 值得一提的是, 本文的研究成果通过一定的扩展可以适用于其他类型的完整性约束规则, 如拒绝约束、条件函数依赖、包含依赖等.

定义 1 (**函数依赖**). 给定属性集 U 上的关系模式 R, X 和 Y 是 U 的子集, 若对于 R 的任意一个可能的关系 r, r 中不存在两个元组在上的属性值相等, 而在 Y 上的属性值不等, 则称“X 函数确定 Y”或“Y 函数依赖于 X”, 记作: $X \to Y$, 称 X 为左部属性, Y 为右部属性.

根据 Armstrong 公理系统可知, 对于属性集 U 上的任意 FD: $X \to Y$, 若 $Y = \{A_1, A_2, \ldots, A_n\}$, A_i 是 U 上的一个属性, 则 $X \to A_i$ 一定成立. 不失一般性, 本文假设 FD 都是一元形式 $X \to A$, 即右部属性只有一个. 根据 FD, 我们可以识别数据中的不一致元组.

定义 2 (**不一致元组**). 给定关系实例 I 及 FD 集合 Σ, 对于 I 中的一个元组 t_i, 若存在 Σ 中的一条 FD $\phi: X \to A$

及元组 t_j, $j \neq i$, 使得 $t_i[X] = t_j[X]$, 但 $t_i[A] \neq t_j[A]$, 则 t_i 是不一致元组; 反之, t_i 是一致元组.

例如, 假设图 1(a) 中的订单信息表上只有一条 FD: ZIP → CT, 对于 t_1 元组, 由于存在 t_3 元组使得 t_1 [ZIP] = t_3 [ZIP], 但 t_1 [CT] ≠ t_3 [CT], 因此, t_1 是不一致元组; 而对于 t_{13} 元组, 其 ZIP 值为 2135x, CT 值为 Brighton, 由于表中不存另一个元组的 ZIP 值为 2135x 且 CT 值不等于 Brighton, 因此 t_{13} 是一致元组. 同理可知, 该表中除了 t_{13} 之外的其他元组都是不一致的.

定义 3 (不一致数据库). 给定数据库实例 I 及 FD 集合 Σ, 若 I 中所有元组都是一致元组, 则 I 是一致的, 记作 $I \vDash \Sigma$; 否则, I 不一致, 记作 $I \nvDash \Sigma$.

对于不一致数据库, 子集修复方法通过删除一些元组, 使得保留元组满足 FD, 此时保留元组是原数据集合的一致子集. 例如, 图 1(a) 中删除 t_1 和 t_6 后, $S_1 = \{t_2, t_3, t_4, t_5, t_7, t_8, t_9, t_{10}, t_{11}, t_{12}, t_{13}, t_{14}\}$ 中的元组不再违反 FD, 构成一致子集; 同理可知, 删除 t_1, t_6 和 t_{14} 后, $S_2 = \{t_2, t_3, t_4, t_5, t_7, t_8, t_9, t_{10}, t_{11}, t_{12}, t_{13}\}$ 也是一致子集. 不难发现, 删除 t_1, t_6 以及 S_1 的任何子集中的元组后, 剩余元组都是一致子集, 因此, 该表中存在指数多个一致子集.

为了尽可能多地保留原始数据中的信息, 子集修复方法通常保留极大的一致子集, 所谓极大一致子集是指向其中加入任何已删除元组都将导致其不再一致. 例如, 向 S_1 中加入删除的 t_1 将导致 t_3 和 t_9 不一致, 加入删除的 t_6 将导致 $t_2, t_4, t_5, t_7, t_8, t_{10}, t_{11}, t_{12}, t_{14}$ 不一致, 因此 S_1 是极大一致子集; 相反, 向 S_2 中加入删除的 t_{14} 不会导致 FD 违反, 因此, S_2 不是极大一致子集. 从形式上说, 极大一致子集不是任何其他一致子集的子集 (不难发现, S_2 是 S_1 的子集), 下面给出极大一致子集的形式化定义.

定义 4 (极大一致子集). 给定关系实例 I 及一组 FD 集合 Σ, $I \nvDash \Sigma$, 若 I 的子集 $J \vDash \Sigma$, 则称 J 为 I 的一致子集. 若对于 I 的任何其他一致子集 K, 都有 $J \cap K \neq J$, 则称 J 为实例 I 上关于 Σ 的极大一致子集.

图 1(a) 的极大一致子集有 4 个: $J_1 = \{t_2, t_3, t_4, t_5, t_7, t_8, t_9, t_{10}, t_{11}, t_{12}, t_{13}, t_{14}\}$, $J_2 = \{t_3, t_6, t_9, t_{13}\}$, $J_3 = \{t_1, t_6, t_{13}\}$, $J_4 = \{t_1, t_2, t_4, t_5, t_7, t_8, t_{10}, t_{11}, t_{12}, t_{13}, t_{14}\}$. 现有子集修复方法通常将删除元组数量最少的 J_1 作为最优子集修复方案, 然而, J_1 删除的元组中只有 t_6 是错误的, 而 J_1 保留的元组中存在 3 个错误元组 t_3, t_9, t_{13}. 其中, t_{13} 没有违反 FD, 因此子集修复方法无法将其删除, 为此, 我们可以在子集修复前进行异常分组检测, 将 t_{13} 所在的异常分组 ZIP=2135x 识别出来, 进而可以将 t_{13} 删除. 而对于错误元组 t_3 和 t_9, 它们所在的 ZIP=5039 分组中还有一个 t_1 元组 (t_1 的 CT 值与 t_3, t_9 不同), 该元组虽然是正确的, 但是数量较少, 根据前文分析可知, t_1 的平均概率大于 t_3 和 t_9 的平均概率, 因此我们可以将删除元组的数量与平均概率之积作为删除代价, 求解删除代价最小的子集修复方案.

2.2 问题定义

下面, 给出规则与概率相结合的最优子集修复问题的形式化定义.

输入: 经过异常分组检测后的关系实例 I 及一组 FD 集合 Σ.

输出: 规则与概率相结合的最优子集修复方案 J^*, J^* 是 I 中删除元组概率和最小的极大一致子集:

$$J^* = \arg\min\left\{\sum_{t_i \in I \setminus J} p(t_i) : J \in S_{I,\Sigma}\right\} \tag{1}$$

其中, $p(t_i)$ 是元组 t_i 的概率, 可以根据第 3.3 节的概率模型计算出来, $p(t_i) \in [0,1]$, $S_{I,\Sigma}$ 表示 I 的所有极大一致子集集合. 不难发现, 当所有元组概率都为相等时, 删除元组数量最少的极大一致子集即是规则与概率相结合的最优子集修复方案. 由此可见, 基于规则的子集修复问题是规则与概率相结合的子集修复问题的一种特殊情况. 当 FD 数量大于 1 时, 基于规则的子集修复问题是 NP-complete 的[14], 因此, 规则与概率相结合的子集修复问题也是 NP-complete.

2.3 规则与概率相结合的最优子集修复方法

下面我们通过一个示例介绍本文提出的规则与概率相结合的最优子集修复方法的核心思想. 首先, 对于修复代价为删除元组数量的最优子集修复问题, 现有方法通常将其转换为冲突图上的最小权顶点覆盖问题, 将顶点权赋为 1, 通过求出冲突图上最小权顶点覆盖, 将顶点覆盖中的顶点对应的元组删除即可得到删除元组数量最小的子集修复方案[4,6]. 不难发现, 对于修复代价为删除元组概率和的最优子集修复问题, 我们只需要将冲突图上的顶点权改为元组概率, 就可以利用最小权顶点覆盖方法求出删除元组概率和最小的子集修复方案, 具体求解过程如

下所示.

给定经过异常分组检测后的关系实例 I 及一组 FD 集合 Σ, 我们首先根据第 3.3 节的概率模型计算 I 中元组的概率 $p(t_i)$; 然后构建冲突图 $G=(V,E,W,L)$, 其中 V 是顶点集, E 是边集, W 是顶点的权重集合, L 上的标签集合. 冲突图 G 反映了 I 中元组违反 FD 的情况, 对于 I 中的一个元组 t_i, G 中有一个顶点 v_i 与之对应, v_i 的权重为 $p(t_i)$, 若 I 中有元组对 (t_i,t_j) 违反 $\varphi\in\Sigma$, 则 G 中 v_i 和 v_j 两个顶点之间有一条边 $(v_i,v_j)\in E$, 边上的标签为 φ. 冲突图 G 构建完成后, 我们计算 G 的最小权顶点覆盖 $C\subseteq V$, 定理 1 表明, C 中顶点对应的元组即是最优子集修复方案 J^* 中删除的元组, 换言之, 不在 C 中的顶点对应的元组即是 J^* 中保留的元组.

定理 1. 给定关系实例 I 及其冲突图 $G=(V,E,W,L)$, G 的最小权顶点覆盖 $C\subseteq V$ 中顶点对应的元组是 I 的最优子集修复方案中 J^* 中删除的元组.

证明: 首先, 将顶点覆盖 C 中的顶点删除后, 剩余顶点之间没有边, 由于冲突图上的边代表元组之间违反 FD 的情况, 因此, 将 C 中顶点对应的元组删除后, 剩余元组将不再违反 FD, 即剩余元组构成一致子集; 其次, 由于顶点覆盖 C 是一个极小顶点覆盖, 即去掉 C 中任意顶点都将导致有未被覆盖的边存在, 换言之, 将 C 中任意顶点对应的元组保留都将导致 FD 违反, 即剩余元组构成极大的一致子集; 最后, 由于 C 中顶点的权重之和是所有顶点覆盖中最小的, 因此 C 中顶点对应的元组的概率和是所有子集修复方案删除元组概率和最小的, 即 C 中顶点对应的元组是最优子集修复方案中删除的元组, 证毕.

例 1: 假设图 1(a) 中的订单信息表中只有一条 FD: ZIP→CT, 经过异常分组检测后, t_{13} 元组被删除, 剩余元组对应的冲突图如图 2 所示, 顶点 v_1–v_{12}, v_{14} 分别对应元组 t_1–t_{12}, t_{14}, 元组概率是根据第 3.3 节的概率模型计算得到的, 由于元组对 (t_1,t_3) 违反 φ_1, 因此图中存在一条边 (v_1,v_3), 标签为 φ_1, 同理可知图上的其他边, 图 2 的最小权顶点覆盖 $C=\{v_3,v_6,v_9\}$, 删除顶点的权重之和为 1.1, 因此, 最优子集修复方案 J^* 删除的元组为 t_3,t_6,t_9. 对比图 1(a) 中真实值可知, 该修复方案中删除元组的准确率为 1, 召回率也为 1.

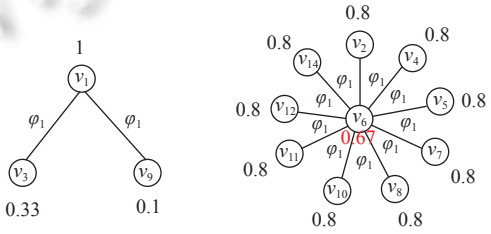


图 2 订单信息表上只有一条 FD: ZIP→CT 时对应的冲突图

讨论: 通过例 1 可以看出, 将规则与概率相结合可以有效解决一些分组中错误元组数量大于正确元组数量的问题, 修复准确性大大提高. 然而, 该方法需要计算全部不一致元组的概率, 当不一致元组数量较多时, 概率计算的时间开销不可小觑. 实际上, 根据绪论中的分析可知, 若能将大分组中的错误元组提前删除, 则可大大减小不一致元组规模, 进而减小概率计算开销.

3 规则与概率相结合的子集修复框架 HybridOSR

为了减小不一致元组规模, 本文提出规则与概率相结合的最优子集修复框架 HybridOSR, 总体流程如后文图 3 所示, 输入是关系实例 I 以及 FD 集合 Σ, 输出是最优子集修复方案保留的元组集合 J, HybridOSR 框架主要包括 3 个阶段: 错误检测、概率建模与计算以及最优子集修复求解.

3.1 HybridOSR 概述

● 错误检测阶段. 首先, 为 Σ 中的每条 FD 构建 FD 导出矩阵, 并将 I 中元组划分到 FD 导出矩阵中, FD 导出矩阵的构建过程见第 3.2 节. 接下来, 依次检测每个 FD 导出矩阵中的异常分组, 并将其中的错误元组在所有 FD 导出矩阵中删除; 然后, 检测每个 FD 导出矩阵中的大分组, 将其中的错误元组在所有 FD 导出矩阵中删除; 最后, 检

测每个 FD 导出矩阵的小分组, 将其中不一致的元组加入 I_n 中并标记其违反的 FD 的左部属性与右部属性. 所有 FD 导出矩阵检测完成后, 将不在 I_n 中的元组加入 J 中.

● 概率建模与计算阶段. 首先, 设计不一致元组概率模型, 将不一致元组概率建模为给定正确属性取值时, 左部属性取当前值的条件概率; 然后, 利用最大似然估计计算计算 I_n 中不一致元组的概率, 即利用频率估计概率.

● 最优子集修复求解阶段. 首先, 根据 I_n 中元组违反 FD 的情况以及元组的概率信息构造冲突图 $G_n=(V_n,E_n)$, 将最优子集修复问题转换为 G_n 上最小权顶点覆盖问题; 其次, 利用线性规划方法近似求出 G_n 上权值最小的顶点覆盖集合 $C_n \subseteq V_n$; 最后, 将 C_n 中顶点对应的元组删除, 将 $V_n \backslash C_n$ 中顶点对应的元组加入保留元组集合 J 中.

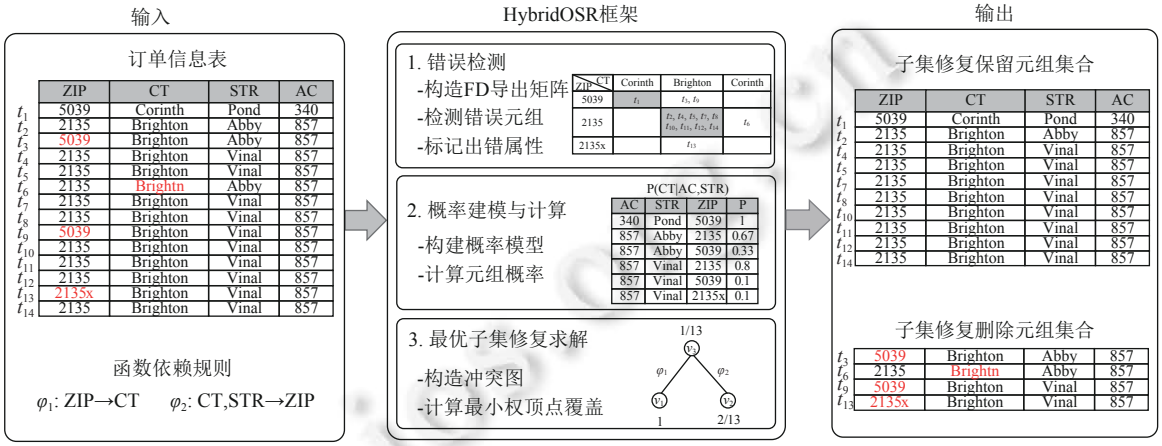


图 3 HybridOSR 总体框架

3.2 错误检测

本节首先给出 FD 导出矩阵的定义, 然后给出大分组和小分组的判断标准, 最后介绍基于 FD 导出矩阵的错误检测算法.

定义 5 (FD 导出矩阵). 给定实例 I 及 FD $\phi: X \to A$, 若 I 中元组在 X 属性集合上有 m 种可能取值 $x_1, x_2, \ldots, x_m$, 在 A 属性上有 n 种可能的取值 $a_1, a_2, \ldots, a_n$, 将 I 中元组按 X 与 A 的取值划分到 $m \times n$ 的导出矩阵 M_φ 中. 矩阵的行表示按 X 划分的分组, 第 i 行代表 $X=x_i$ 分组, 每个分组按照 A 属性值划分为不同格子, 位于第 j 列的格子称为 $A=a_j$ 格子, M_ϕ 反映了 I 中元组在 X 属性和 A 属性上的取值分布情况.

例 2: 假设图 1 中的订单信息表上共有两条 FD φ_1 : ZIP → CT 和 φ_2 : CT, STR → ZIP, 将所有元组按照 ZIP 和 CT 取值划分到 FD 导出矩阵 M_{φ_1} 中, 按照 CT, STR, ZIP 取值划分到 FD 导出矩阵 M_{φ_2} 中, 划分结果如图 4 所示.

ZIP \ CT	Corinth	Brighton	Corinth
5039	t_1	t_3, t_9	
2135		t_2, t_4, t_5, t_7, t_8 t_{10}, t_{11}, t_{12}, t_{14}	t_6
2135x		t_{13}	

(a) M_{φ_1}

CT,STR \ ZIP	5039	2135	2135x
Corinth,Pond	t_1		
Brighton,Abby	t_3	t_2	
Brightn,Abby		t_6	
Brighton,Vinal	t_9	t_4, t_5, t_7, t_8 t_{10}, t_{11}, t_{12}, t_{14}	t_{13}

(b) M_{φ_2}

图 4 FD 导出矩阵

对于 FD $\phi: X \to A$, 根据 FD 语义可知, X 值可以确定唯一的 A 值, 因此 FD 导出矩阵 M_φ 的每个分组中应有且仅有一个格子非空, 此时分组一致; 否则, 不一致. 假设 $X=x_i$ 时, 对应的正确的 $A=a_j$, 则称 $A=a_j$ 格子为 $X=x_i$ 分组的正确格子, 其余 $A \neq a_j$ 的格子为错误格子, 显然, 错误格子中的元组一定是错误的, 可能是左部属性出错, 也可能是右部属性出错. 例如, 图 4(a) 中 ZIP=5039 分组中 CT=Corinth 格子和 CT=Brighton 格子都有元组, 因此, 该

分组不一致, 由于 5039 是 Corinth 城市中 Pond 街道的邮编, 因此该分组中的正确格子为 CT=Corinth. 根据图 1(a)中的真实值可知, 错误格子 CT=Brighton 中的 t_3 和 t_9 元组都是由于左部属性 ZIP 出错进入该分组的, 其真实的 ZIP 值为 2135. 为了方便理解, 我们将每个分组的正确格子背景填充为灰色, 异常分组除外.

下面, 我们首先介绍如何利用 FD 导出矩阵识别异常分组并删除其中错误元组. 给定关系实例 I 以及 FD 集合 Σ, 为 Σ 中的每条 FD $\varphi_i: X \to A$ 构建 FD 导出矩阵 M_{φ_i}, 将 I 中元组按 X 与 A 的取值划分到每个 M_{φ_i} 中, 依次检测每个 FD 导出矩阵中的分组, 若分组规模小于给定阈值并且分组名字与其他分组名字的编辑距离也小于给定阈值, 则将该异常分组中的元组删除. 值得一提的是, 只有同时满足以上两个条件的分组才有可能是异常分组, 原因有两个方面: (1) 只考虑分组规模, 会将一些规模较小的正常分组误判为异常分组, 例如, 图 4(b) 中的 CT, STR=Corinth, Pond 的正常分组只有 1 个元组; (2) 只考虑编辑距离, 会将一些与异常分组名字相似的正常分组误判为异常分组, 例如图 1(b) 中的 ZIP=2135 的正常分组与 ZIP=2135x 的异常分组的编辑距离为 1.

经过异常分组检测后, 理想情况下 I 中应不存在左部属性出现拼写错误的元组, 此时左部属性上的错误只有替换错误. 基于前文分析可知, 当时数据干净时, 每个分组中只有正确格子中有元组, 此时正确格子的规模最大且正确元组数量最多; 发生替换错误时, 一些分组中的元组分布发生了改变, 正确格子中的元组数量可能不再是最多的. 然而, 根据文献 [28] 可知, 工业界中的数据错误率通常在 5% 左右, 学术界使用的测试数据集中的错误比例通常不超过 30%[9,29,30]. 换言之, 数据中的错误通常较少, 因此, 我们的一个直观猜想是, 少量的错误元组不会改变大分组中的元组分布, 大分组中正确格子的规模仍然是最大的. 引理 1 和定理 2 证实了我们的猜想, 根据定理 2 可知, 当错误率小于 1/3 时, 分组规模大于平均分组规模的 2/3 的分组中正确格子的元组数量一定是最多的. 基于此, 我们给出大分组和小分组的形式化定义.

引理 1. 假设错误率 $\varepsilon < \frac{1}{3}$ 且错误均匀随机发生在 FD 相关的属性上, 若数据干净时分组规模大于平均分组规模的 $1/2$, 则错误发生时正确格子中的元组数量大于错误格子中的元组数量.

证明: 由于篇幅有限, 详细证明过程见附录.

定理 2. 当错误率 $\varepsilon < \frac{1}{3}$ 且错误均匀随机发生在 FD 相关的属性上时, 对于分组规模大于平均分组规模 2/3 的分组, 其正确格子中的元组数量大于该分组中任意一个错误格子中的元组数量.

证明: 由于篇幅有限, 详细证明过程见附录.

定义 6 (大分组和小分组). 给定 FD 导出矩阵中的一个分组, 若其规模大于平均分组规模的 2/3, 则该分组为大分组; 反之为小分组.

不难发现, 由于大分组中规模最大的格子一定是正确格子, 因此, 对大分组进行错误检测时, 我们首先确定其正确格子, 然后可以直接删除所有不在正确格子中的错误元组. 而小分组中规模最大的格子不一定是正确格子, 因此, 对小分组错误检测时, 我们将该分组中所有元组加入噪声元组集 I_n 中等待后续子集修复. 下面, 我们给出基于 FD 导出矩阵的错误检测算法, 如算法 1 所示. 算法的输入是关系实例 I, FD 集合 Σ, 分组规模阈值 δ 以及编辑距离 θ, 输出是保留元组集合 J 和噪声元组集 I_n. 算法首先对 Σ 中的每条 FD φ_i 构造 FD 导出矩阵 M_{φ_i}, 然后检测 M_{φ_i} 中的异常分组并将其中错误元组删除 (第 1–5 行); 其次, 检测每个 M_{φ_i} 中的不一致大分组, 将不在正确格子中的错误元组删除 (第 6–10 行); 再次, 检测每个 M_{φ_i} 的不一致小分组, 将其中元组加入噪声元组集 I_n 中并标记出错属性位置 (第 11–16 行); 最后, 将不在 I_n 中的元组加入 J 中.

算法 1. 基于 FD 导出矩阵的错误检测算法.

输入: 关系实例 I 及 FD 集合 $\Sigma = \{\varphi_1, \varphi_2, \ldots, \varphi_{|\Sigma|}\}$, 分组规模阈值 δ, 编辑距离 θ;
输出: 保留元组集合 J, 噪声元组集 I_n.

1 **for** i from 1 to $|\Sigma|$ **do**
2 根据 φ_i 和 I 构建 FD 导出矩阵 M_{φ_i}, 并将其加入 M_Σ 中;
3 **for** M_{φ_i} 的每个分组 $X = x_j$ **do**

4 **if** (($|X = x_j| < \delta$) &($\exists X = x_j$ 分组使得 EditDist $(x_j, x_j) < \theta$)) **then**
5 将分组中每个元组从所有 FD 导出矩阵中删除;
6 **for** $M_{\varphi_i} \in M_\Sigma$ **do**
7 $\bar{n}_i \leftarrow M_{\varphi_i}$ 的平均分组数量;
8 **for** M_{φ_i} 的每个分组 $X = x_j$ **do**
9 **if** (($|X = x_j| > 2\bar{n}_i/3$) & ($X = x_j$ 分组中非空格子数量>1)) **then**
10 将错误格子中的元组从所有 FD 导出矩阵中删除;
11 **for** $M_{\varphi_i} \in M_\Sigma$ **do**
12 $\bar{n}_i \leftarrow M_{\varphi_i}$ 的平均分组数量;
13 **for** M_{φ_i} 的每个分组 $X = x_j$ **do**
14 **if** (($|X = x_j| < 2\bar{n}_i/3$) & ($X = x_j$ 分组中非空格子数量>1)) **then**
15 将分组中的元组加入噪声元组集 I_n 中;
16 将分组中每个元组的 X 属性标记为违反的左部属性, A 属性标记为右部属性;
17 将 $M_\Sigma - I_n$ 中的元组加入 J 中;

定理 3. 错误检测阶段的时间复杂度为 O ($|\Sigma|N$), N 为 I 中元组数量.

证明: 首先, 构建每个 FD 导出矩阵需要将 I 中元组按照 FD 左部属性和右部属性取值划分到各个格子中, 时间开销为 O(N) , 一共有 $|\Sigma|$ 个 FD 导出矩阵, 因此 M_Σ 的构建时间为 O($|\Sigma|N$); 其次, 检测每个 FD 导出矩阵检测异常分组中, 最坏情况下, 所有分组都是异常分组, 换言之, 需要将 I 中元组全部删除, 代价为 O(N) , 一共有 $|\Sigma|$ 个 FD 导出矩阵, 因此异常分组检测的时间开销为 O($|\Sigma|N$); 同理可知, 大分组检测和小分组检测的时间开销同样为 O($|\Sigma|N$). 因此, 总的时间开销为 O($|\Sigma|N$).

例 3: 接例 2, 我们首先对图 4 中的 FD 导出矩阵进行异常分组检测, 假设分组规模阈值和编辑距离阈值都为 1, 则 M_{φ_1} 中的 ZIP=2135x 分组以及 M_{φ_2} 中的 CT, STR=Brightn, Abby 分组满足条件, 将其中元组 t_6 和 t_{13} 从所有 FD 导出矩阵中删除后, FD 导出矩阵如图 5 所示.

ZIP \ CT	Corinth	Brighton	Corinth
5039	t_1	t_3, t_9	
2135		t_2, t_4, t_5, t_7, t_8 t_{10}, t_{11}, t_{12}, t_{14}	

(a) M_{φ_1}

CT,STR \ ZIP	5039	2135	2135x
Corinth,Pond	t_1		
Brighton,Abby	t_3	t_2	
Brighton,Vinal	t_9	t_4, t_5, t_7, t_8 t_{10}, t_{11}, t_{12}, t_{14}	

(b) M_{φ_2}

图 5 将异常分组中的错误元组删除后的 FD 导出矩阵

接下来检测图 5 中每个 FD 导出矩阵的大分组, 将大分组中的错误元组删除. 对于 M_{φ_1} , 其平均分组规模为 6, 规模大于 $6\times\frac{2}{3}=4$ 的分组只有 ZIP=2135, 然而该分组中只有 CT=Brighton 一个格子非空, 因此该分组一致, 不需要删除任何元组; 接下来, 我们检测 M_{φ_2} , 其平均分组规模为 4, 规模大于 $4\times\frac{2}{3}=\frac{8}{3}$ 的只有 CT, STR=Brighton, Vinal 分组, 该分组中有两个非空格子: ZIP=5039 和 ZIP=2135, 其中规模最大的是 ZIP=2135 格子, 因此该格子正确格子, 我们将 ZIP=5039 格子中的错误元组 t_9 删除后, FD 导出矩阵如图 6 所示.

最后, 我们对图 6 中的 FD 导出矩阵进行小分组检测, 将其中不一致的元组加入 I_n 中. 首先, 对于 M_{φ_1} , ZIP=5039 分组是小分组并且该分组不一致, 将 t_1 和 t_3 加入 I_n 中; 其次, 对于 M_{φ_2} , 该 FD 导出矩阵中有两个小分组, 其中 CT, STR=Corinth, Pond 小分组一致, 而 CT, STR=Brighton, Abby 不一致, 将其中元组 t_2 和 t_3 加入 I_n 中, 此时 $I_n = \{t_1, t_2, t_3\}$. 最后, 我们将不在 I_n 中的元组加入 J 中, $J = \{t_4, t_5, t_7, t_8, t_{10}, t_{11}, t_{12}, t_{14}\}$

ZIP \ CT	Corinth	Brighton	Corinth
5039	t_1	t_3	
2135		t_2, t_4, t_5, t_7, t_8 $t_{10}, t_{11}, t_{12}, t_{14}$	

(a) M_{φ_1}

CT,STR \ ZIP	5039	2135	2135x
Corinth,Pond	t_1		
Brighton,Abby	t_3	t_2	
Brighton,Vinal		t_4, t_5, t_7, t_8 $t_{10}, t_{11}, t_{12}, t_{14}$	

(b) M_{φ_2}

图 6 将大分组中的错误元组删除后的 FD 导出矩阵

3.3 概率建模与计算

经过错误检测后, 我们只需要对噪声元组集 I_n 中的元组进行子集修复, 通常情况下, I_n 中的元组数量远小于 I 中的元组数量. 接下来, 我们需要计算 I_n 中不一致元组的概率. 需要注意的是, I_n 中的元组都来自小分组, 基于前文分析可知, 小分组中可能会出现错误元组数量大于正确元组数量的情况. 因此, 我们设计的概率模型需要保证正确元组的平均概率大于错误元组的平均概率, 为后续子集修复提供概率保障.

为了方便讨论, 我们假设 Σ 中只有一条 FD, 给定 FD $\varphi: X \to A$ 及不一致元组 t, t 出错的属性可能是左部属性 X, 也可能是右部属性 A, 还有可能是 X 和 A 都出错, 令 $Y = U - X - A$, U 为全部属性集合, 则 Y 中属性没有错误. 此时, 一种直观的想法是将 t 的概率建模为给定正确属性 Y 的取值时, 可能出错的 X 属性和 A 属性取当前值的概率, 如公式 (2) 所示:

$$P(t) = P(X = t[X], A = t[A] | Y = t[Y]) \tag{2}$$

值得一提的是, 该模型常被用在数据清洗领域中, 如基于概率的修复方法[20,22]以及规则与概率相结合的更新修复方法[5]. 然而, 我们通过分析发现, 若 X 属性与 Y 属性之间存在相关性, 则该模型无法保证小分组中错误元组的平均概率小于正确元组的平均概率. 例如, 引理 2 表明当 X 属性与 Y 属性之间存在 $X \to Y$ 的函数依赖时, 若小分组中错误元组都位于同一错误格子中, 且数量大于正确元组数量, 则该模型下错误元组的平均概率大于正确元组的平均概率. 需要说明的是, 实际应用中, X 属性与 Y 属性之间通常存在相关性, 但不会是 $X \to Y$ 这么强相关, 我们在引理 2 中假设 $X \to Y$ 是为了简化证明过程, 更一般情况的证明可以参考定理 4.

引理 2. 给定关系实例 I 及 FD 集合 $\Sigma = \{\varphi_1: X \to A, \varphi_2: X \to Y\}$, 全部属性集 $U = X \cup A \cup Y$. 对于 M_{φ_1} 中的一个不一致小分组 $X = x_i$, 假设该分组中的元组只违反了 φ_1 而没有违反 φ_2, 若采用公式 (2) 中的概率模型, 则该分组中的元组概率为 $P(X = t[X], A = t[A] | Y = t[Y])$. 若该分组中的错误元组都位于同一错误格子中, 且数量大于正确元组数量, 则错误元组的平均概率大于正确元组的平均概率.

证明: 假设 $X = x_i$ 分组的正确格子为 $A = a_j$, 错误元组所在的格子为 $A = a_k$. 根据 $X \to Y$ 可知, 当 $X = x_i$ 时, Y 属性有唯一的取值, 假设是 y. 此时, 正确元组的平均概率为 $\bar{P}_1 = P(X = x_i, A = a_j | Y = y)$, 错误元组的平均概率为 $\bar{P}_2 = P(X = x_i, A = a_k | Y = y)$, 根据条件概率公式, 我们可以将 $\bar{P}_1$ 和 $\bar{P}_2$ 分别展开为公式 (3) 和公式 (4).

$$\bar{P}_1 = P(X = x_i) P(A = a_j | X = x_i) P(Y = y | X = x_i, A = a_j) / P(Y = y) \tag{3}$$

$$\bar{P}_2 = P(X = x_i) P(A = a_k | X = x_i) P(Y = y | X = x_i, A = a_k) / P(Y = y) \tag{4}$$

由于 $X = x_i$ 时, $Y = y$ 的概率为 1, 因此 $P(Y = y | X = x_i, A = a_j) = P(Y = y | X = x_i, A = a_k) = 1$, 对 $\bar{P}_1 - \bar{P}_2$ 提取公因式后得到公式 (5), 代证 $\bar{P}_1 - \bar{P}_2 < 0$.

$$\bar{P}_1 - \bar{P}_2 = \frac{P(X = x_i)}{P(Y = y)} (P(A = a_j | X = x_i) - P(A = a_k | X = x_i)) \tag{5}$$

由于 $X = x_i$ 分组中错误元组数量大于正确元组数量, 即 $X = x_i$ 时, $A = a_k$ 的元组数量大于 $A = a_j$ 的元组数量, 因此 $P(A = a_j | X = x_i) < P(A = a_k | X = x_i)$, 从而 $\bar{P}_1 - \bar{P}_2 < 0$, 证毕.

根据引理 2 的证明过程不难发现, 导致 $\bar{P}_1 < \bar{P}_2$ 的原因是由于 $P(A = a_j | X = x_i) < P(A = a_k | X = x_i)$, 因此我们将 $P(A = a_j | X = x_i)$ 和 $P(A = a_k | X = x_i)$ 分别从公式 (3) 和公式 (4) 中删除. 此外, 由于 $X \to Y$, 即给定 X 值时, Y 的值由 X 唯一确定, 与 A 的取值无关, 因此, $P(Y = y | X = x_i, A = a_j) = P(Y = y | X = x_i, A = a_k) = P(Y = y | X = x_i)$, 此时, 公式 (3)

和公式 (4) 经过化简变为公式 (6).

$$\bar{P}_1 = \bar{P}_2 = P(X = x_i|Y = y) \tag{6}$$

根据公式 (6) 可知, 当小分组中错误元组数量大于正确元组数量, 且都位于同一个格子中时, 我们将不一致元组概率建模为 $P(X = x_i|Y = y)$ 可以保证错误元组的平均概率等于正确元组的平均概率 (即 $\bar{P}_1 = \bar{P}_2$). 而实际应用中小分组中的错误元组通常不会位于同一个格子中, 因此, 错误元组的平均概率一定小于正确元组的概率 (即 $\bar{P}_1 > \bar{P}_2$).

基于上述分析, 我们将不一致元组 t 的概率建模为给定正确属性 Y 的取值时, 左部属性 X 取当前值的概率, 如公式 (7) 所示. 定理 4 表明该模型可以保证任何分组中 (无论是大分组还是小分组), 正确元组的概率期望都大于错误元组的概率期望. 根据中心极限定理可知, 当样本足够多时 (大于 30), 样本均值是总体期望的无偏估计, 因此, 正确元组的平均概率大于错误元组的平均概率. 值得一提的是, 定理 4 对 X 属性与 Y 属性之间的相关性不做任何假设, 因此该模型适用于任何场景.

$$P(t) = P(X = t[X]|Y = t[Y]) \tag{7}$$

定理 4. 假设 Σ 中只有一条 FD $\varphi: X \to A$, 其导出矩阵为 M_φ, 全部属性集 $U = X \cup A \cup Y$. 若不一致元组概率模型为公式 (7), 则不一致分组中正确元组的概率期望大于错误元组的概率期望.

证明: 不失一般性, 假设 $X = x_1$ 分组不一致且分组中的元组服从概率分布 F, 我们在 F 上均匀采样得到样本 $S = \{t_1, t_1, \ldots, t_k\}$, 由于 S 中元组独立同分布, 因此可利用 S 中元组的概率均值估计总体概率 $P(X = x_1|Y)$ 的期望. 若 S 中元组都是正确元组, 即 X 属性没有出错, 则概率期望如公式 (8) 所示:

$$E_1 = \sum_{i=1}^{k} P(X = x_1|Y = t_i[Y])P(Y = t_i[Y]|X = x_1) \tag{8}$$

若 S 中元组都是错误元组, X 属性的真实值不等于 x_1, 则概率期望如公式 (9) 所示:

$$E_2 = \sum_{i=1}^{k} P(X = x_1|Y = t_i[Y])P(Y = t_i[Y]|X \neq x_1) \tag{9}$$

待证 $E_1 - E_2 > 0$. 为方便讨论, 我们将 $P(Y = t_i[Y])$ 简记为 p_i, 则 $\sum_{i=1}^{k} p_i = 1$, 将 $P(Y = t_i[Y]|X = x_1)$ 简记为 q_i, 则 $\sum_{i=1}^{k} q_i = 1$, 经化简 $E_1 - E_2$ 满足公式 (10), 其中, $\frac{P(X = x_1)}{1 - P(X = x_1)} > 0$, 因此只需要证明 $\sum_{i=1}^{k} p_i \left(\frac{q_i}{p_i}\right)^2 - 1 > 0$.

$$E_1 - E_2 = \frac{P(X = x_1)}{1 - P(X = x_1)} \left(\sum_{i=1}^{k} p_i \left(\frac{q_i}{p_i}\right)^2 - 1\right) \tag{10}$$

由于 $\sum_{i=1}^{k} p_i = 1$, 我们将其中每个 p_i 乘以 $\frac{q_i}{p_i}$ 后, 有 $\sum_{i=1}^{k} p_i \times \frac{q_i}{p_i} = \sum_{i=1}^{k} q_i = 1$, 即概率和没有改变, 因此 p_i 乘以 $\frac{q_i}{p_i}$ 时, 一部分元组的概率增大, 一部分减小, 一部分保持不变. 我们将概率增大或不变的元组集合记为 S_1, 增大倍数记为 $\alpha_i = \frac{q_i}{p_i} \geqslant 1$, 将概率减小的元组集合记为 S_2, 缩小倍数记为 $\beta_i = \frac{q_i}{p_i} < 1$, 则 S_1 中元组概率增加之和等于 S_2 中元组概率减小之和, 如公式 (11) 所示.

$$\sum_{t_i \in S_1} (\alpha_i - 1) p_i = \sum_{t_i \in S_2} (\beta_i - 1) p_i \tag{11}$$

我们对公式 (11) 等式两边分别乘以 α_i 和 β_i, 由于 $\alpha_i \geqslant 1$, $\beta_i < 1$, 故公式 (12) 成立.

$$\sum_{t_i \in S_1} (\alpha_i - 1) p_i \alpha_i > \sum_{t_i \in S_2} (\beta_i - 1) p_i \beta_i \tag{12}$$

将 $\alpha_i = \beta_i = \frac{q_i}{p_i}$ 代入公式 (12) 并化简, 有公式 (13) 成立, 故 $\sum_{i=1}^{k} p_i \left(\frac{q_i}{p_i}\right)^2 - 1 > 0$, 因此, $E_1 > E_2$, 证毕.

$$\sum_{t_i \in S_1} \left(\frac{q_i}{p_i}\right)^2 + \sum_{t_i \in S_2} \left(\frac{q_i}{p_i}\right)^2 > \sum_{t_i \in S_1} q_i + \sum_{t_i \in S_2} q_i = 1 \tag{13}$$

下面, 我们将公式 (7) 中的概率模型推广到一般情况, 即 Σ 中有多条 FD 时. 给定 FD 集合 Σ 及一致元组 t, 由于 t 可能同时违反多条 FD, 我们将其违反的所有 FD 的左部属性加入查询集 $\mathcal{X}$ 中, 将不在其违反的 FD 中的属性加入证据集 $\mathcal{Y}$ 中, 建模元组 t 的概率为给定 $\mathcal{Y}$ 中属性取值时, $\mathcal{X}$ 中属性取当前值的条件概率, 如公式 (14) 所示.

$$p(t) = P(\mathcal{X} = t[\mathcal{X}]|\mathcal{Y} = t[\mathcal{Y}]) \tag{14}$$

最后, 我们采用极大似然估计方法估计 $p(t)$ 的值. 由于 I 中元组彼此独立, 因此可以看作总体上的一个随机样本, $p(t)$ 的极大似然估计为:

$$\hat{p}(t)=n(\mathcal{X}=t[X],\mathcal{Y}=t[Y])/n(\mathcal{Y}=t[Y]) \tag{15}$$

其中, $n(\mathcal{X}=t[X],\mathcal{Y}=t[Y])$ 为 I 中 $\mathcal{X}=t[X],\mathcal{Y}=t[Y]$ 的元组数量, $n(\mathcal{Y}=t[Y])$ 为 I 中 $\mathcal{Y}=t[Y]$ 的元组数量.

定理 5. 概率计算阶段的时间复杂度为 $O(nN)$, 其中 n 为 I_n 中元组数量, N 为 I 中元组数量.

证明: 对于噪声元组集 I_n 中的每一个不一致元组, 我们需要在全部元组集 I 上统计 $\mathcal{X}=t[X],\mathcal{Y}=t[Y]$ 和 $\mathcal{Y}=t[Y]$ 的频率, 因此时间复杂度为 $O(nN)$.

例 4: 接例 3, 经过错误检测后, 噪声元组集 $I_n=\{t_1,t_2,t_3\}$. 对于 t_1 元组, 其违反的 FD 为 φ_1 : ZIP → CT, φ_1 的左部属性为 ZIP, 不在 φ_1 中的属性有 AC 和 STR, 因此 $p(t_1)=P$(ZIP=5039|AC=340, STR=Pond)=1; 对于 t_2 元组, 其违反的 FD 为 φ_2 : CT, STR → ZIP, φ_2 的左部属性为 CT, STR, 不在 φ_2 中的属性有 AC, 因此 $p(t_2)=P$(CT=Brighton, STR=Abby|AC=857)=2/13; 对于 t_3 元组, 其违反的 FD 为 φ_1 : ZIP → CT 和 φ_2 : CT, STR → ZIP, φ_1 和 φ_2 中的左部属性有 ZIP, CT 和 STR, 不在 φ_1 和 φ_2 中的属性为 AC, 因此 $p(t_3)=P$(ZIP=5039, CT=Brighton, STR=Abby|AC=857)=1/13.

3.4 最优子集修复求解

经过错误检测和概率计算后, 我们已知噪声元组集 I_n 中元组的概率, 接下来, 我们需要对 I_n 进行子集修复, 即删除一些元组使得剩余元组满足 FD. 根据前文分析可知, I_n 中元组所在的小分组中, 正确元组的数量可能小于错误元组的数量, 但正确元组的平均概率一定大于错误元组的平均概率. 因此, 我们将删除元组概率和最小的极大一致子集加入保留元组集合 J 中.

我们为噪声数据集 I_n 构造冲突图 $G_n=(V_n,E_n)$, 并利用线性规划方法求解 G_n 上的最小权顶点覆盖, 过程如下: 为 V_n 中每个顶点 v_i 构造标识变量 x_i, x_i 表示顶点 v_i 是否在顶点覆盖 C 中, 若 v_i 在 C 中, 则 $x_i=1$, 否则 $x_i=0$. 此时, 每个顶点覆盖都对应以下整数线性规划问题的一个可行解, 且最小权顶点覆盖对应该整数规划的一个最优解:

$$\begin{cases}\min \sum_{v_i\in C} w_i x_i \\ \text{s.t. } x_i+x_j\geqslant 1,(v_i,v_j)\in E_n \\ x_i=0\text{或}1,\ i=1,2,\ldots,n\end{cases} \tag{16}$$

其中, 最优化目标表示 C 是所有顶点覆盖中权值最小的, 约束条件 $x_i+x_j\geqslant 1$ 表示 E_n 中任意一条边的两个顶点至少一个在 C 中. 由于整数线性规划是 NP 难问题, 因此将整数约束条件 $x_i=0$ 或 1 松弛为实数约束条件 $0\leqslant x_i\leqslant 1$, 从而可以在多项式时间内求出该线性规划的最优解 x_i^*, $i=1,2,\ldots,n$. 由于 x_i^* 可能是小数, 因此对 x_i^* 进行舍入求出整数近似解: 若 $x_i^*\geqslant\frac{1}{2}$, 则近似解 $x_i^A=1$; 反之, $x_i^A=0$.

最优子集修复算法如算法 2 所示. 算法输入为错误检测阶段输出的保留元组集 J, 噪声元组集 I_n 及 FD 集合 Σ, I_n 中元组的概率值已知, 输出是更新后的保留元组集 J. 算法首先为 I_n 构造冲突图 $G_n=(V_n,E_n)$ (第 1 行); 其次, 利用线性规划方法求出公式 (6) 的最优解 x_i^* (第 2 行); 然后遍历每个最优解对其舍入得到近似解 x_i^A (第 3–6 行); 最后, 将 $x_i^A=1$ 对应的元组删除, 将 $x_i^A=0$ 对应的元组加入 J 中 (第 7 行).

算法 2. 最优子集修复算法.

输入: 错误检测阶段输出的保留元组集 J, 噪声元组集 I_n, I_n 中每个元组概率, FD 集合 Σ;
输出: 更新后的保留元组集 J.

1 为 I_n 构造冲突图 $G_n=(V_n,E_n)$;
2 利用线性规划方法求出公式 (6) 的最优解 x_i^*, $i=1,2,\ldots,|I_n|$;
3 **for** i from 1 to $|I_n|$ **do**
4 **if** $x_i^*\geqslant\frac{1}{2}$ **then**

5 $x_i^A = 1$;

6 **else** $x_i^A = 0$;

7 将 $x_i^A = 1$ 对应的元组删除, 将 $x_i^A = 0$ 对应的元组加入 J;

定理 6. 子集修复阶段的时间复杂度为 $O(n^2)$, n 为 I_n 中元组的数量.

证明: 首先, 冲突图构建时, 需要检测 I_n 中违反 FD 的元组对, 并将其在冲突图中对应的边加入到 E_n 中, 最坏情况下, I_n 中的元组两两违反, 此时 E_n 中有 $n(n-1)/2$ 条边, 因此构建冲突图的时间开销为 $O(n^2)$. 接下来, 调用已有最快的线性规划方法求解 (14) 并进行舍入需要 $O(n)$ 时间[10], 因此总的时间复杂度为 $O(n^2)$.

定理 7. 算法 2 求出的近似修复方案的修复代价与最优子集修复方案的修复代价之间的近似比为 2.

证明: 由于子集修复方案的修复代价为删除元组的概率之和, 因此, 近似修复方案的代价为 $\sum_{i=1}^{|I_n|} x_i^A$, 最优子集修复方案的代价为 $\sum_{i=1}^{|I_n|} x_i^*$, 待证:

$$\sum_{i=1}^{|I_n|} x_i^A < 2\sum_{i=1}^{|I_n|} x_i^* \tag{17}$$

由于 $x_i^A = 1$ 时, $x_i^* \geqslant \frac{1}{2}$, 即 $2x_i^* \geqslant x_i^A$; $x_i^A = 0$ 时, $0 < x_i^* < \frac{1}{2}$, 即 $2x_i^* > x_i^A$, 因此有公式 (18) 成立, 证毕.

$$\sum_{i=1}^{|I_n|} x_i^A = \sum_{x_i^A=1} x_i^A + \sum_{x_i^A=0} x_i^A < 2\sum_{x_i^A=1} x_i^* + 2\sum_{x_i^A=0} x_i^* = 2\sum_{i=1}^{|I_n|} x_i^* \tag{18}$$

例 5: 接例 4, 错误检测后 $J = \{t_4, t_5, t_7, t_8, t_{10}, t_{11}, t_{12}, t_{14}\}$, 噪声元组集 $I_n = \{t_1, t_2, t_3\}$, 其中 $p(t_1) = 1$, $p(t_2) = 2/13$, $p(t_3) = 1/13$. 为 I_n 构造冲突图如图 7 所示, 顶点 v_1, v_2, v_3 分别代表 t_1, t_2, t_3, 由于 t_1 和 t_3 违反 φ_1, 因此 v_1 和 v_3 之间存在边, 标签为 φ_1, 同理可知, v_2 和 v_3 之间存在边, 标签为 φ_2. 该图的最小顶点覆盖 $C = \{v_3\}$, 因此, 我们将 t_3 删除, 将剩余元组 t_1 和 t_2 加入 J 中, 最终保留的元组集 $J = \{t_1, t_2, t_4, t_5, t_7, t_8, t_{10}, t_{11}, t_{12}, t_{14}\}$, 删除的元组集为 $\{t_3, t_6, t_9, t_{13}\}$, 对比图 1(a) 中的真实值可知, 删除错误元组的准确率为 1, 召回率也为 1.

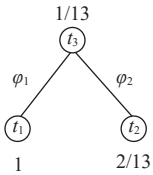


图 7 噪声元组集 I_n 对应的冲突图

定理 8. HybridOSR 的时间复杂度为 $O(Nn)$, 最坏情况下为 $O(N^2)$, 其中 n 为 I_n 中元组数量, N 为 I 中元组数量.

证明: HybridOSR 各阶段时间复杂度如表 1 所示, 由于 $|\Sigma|$ 是常数且 $n < N$, 因此总的时间复杂度为 $O(Nn)$. 最坏情况下, 错误检测阶段没有删除任何元组, 此时 $n = N$, 时间复杂度为 $O(N^2)$.

表 1 HybridOSR 各阶段时间复杂度

阶段	错误检测阶段	概率计算阶段	子集修复阶段	所有阶段
时间复杂度	$O(\|\Sigma\|N)$	$O(nN)$	$O(n^2)$	$O(Nn)$

4 实验分析

为了验证本文方法修复的准确性, 我们在真实数据集和合成数据集上对 HybridOSR 及 5 种对比方法在不同参数下进行测试, 测试参数包括错误率、拼写错误比例及小分组比例, 测试内容包括: (1) 将 HybridOSR 与 4 种基于规则的子集修复方法对比, 验证 HybridOSR 修复的准确性, 4 种方法分别是 BL-LP, TE-LP , et-TE-LP 和 Baseline, 其中 TE-LP 和 et-TE-LP 是当前近似比最好的子集修复方法, 近似比分别为 $2 - 1/2^{|\Sigma|-1}$ 和 $2 - 1/2^{|\Sigma|-1} - \varepsilon$,

而 BL-LP 的近似比为 $2-1/n$, Baseline 是一种基本的子集修复求解方法, 通过将子集修复问题转换为图上的最小权顶点覆盖问题 (权为 1), 利用公式 (16) 对应的线性规划求出近似解, 近似比为 2; (2) 将 HybridOSR 与规则与概率相结合的更新修复方法 Holoclean[5]对比, 验证 HybridOSR 修复的准确性. 由于 Holoclean 是更新修复方法, 我们将 Holoclean 中更新的元组删除, 得到对应的子集修复方案, 值得一提的是, Holoclean 是数据清洗领域最常使用的基准系统之一.

4.1 实验设置

本文提出的 HybridOSR 以及对比方法中 Baseline 和 Holoclean 均采用 Python 实现, 线性规划采用 Python 库函数 scipy.optimize.linprog 求解, Holoclean 的 GitHub 链接为 https://github.com/HoloClean/holoclean. 对比方法中 BL-LP, TE-LP, et-TE-LP 采用 C++实现, 其中的线性规划采用 C++库函数 GLPK-LP/MIP-Solver 求解, 其 GitHub 链接为 https://github.com/gaoxy914/Computing-OSR. 实验运行环境配置为 Intel(R) Core(TM) i7-7700HQ CPU @ 2.80 GHz 处理器, 8 GB 内存, 运行 Windows 10 操作系统.

实验采用的数据集有 3 个: (1) ORDER 数据集是从 Amazon 网站 (https://data.world/datafiniti/consumer-reviews-ofamazon-products) 爬取的在线商品评价信息, 包含 700 条元组, 我们随机复制一些元组使其规模达到 1k, 属性有购买者姓名, 区域号码, 电话号码, 街道名称, 国家, 城市, 州名, 邮政编码等, 该数据集上的 FD 有 3 条: {邮政编码} → {城市}, {邮政编码} → {州名}, {城市, 街道名称} → {邮政编码}; (2) DBLP 数据集是从 DBLP 网站 (https://dblp.org/xml/) 上爬取的计算机类文献信息, 包含 1k 元组, 属性有文献题目, 作者, 发表年限, 出版商, 页码, 网页链接等, 该数据集上的 FD 有 3 条: {网页链接} → {文献题目}, {文献题目} → {作者}, {DOI} → {文献题目}; (3) ORDER-λ 数据集, 为了测试小分组对修复准确性的影响, 我们首先从 ORDER 数据集的 FD 规则中选择一条 φ: {邮政编码} → {城市}, 假设其对应的 FD 导出矩阵为 M_φ, 然后从 M_φ 的分组中随机选择 10 个分组, 通过复制或删除元组使得分组规模之和为 1k, 且其中 $\lambda\times 10$ 个分组为小分组 (即规模小于 $2N/3m$, $N=1\text{k}$, $m=10$), 此时小分组占全部分组的比例为 λ.

需要说明的是, 为了衡量修复准确性, 我们使用的原始数据集都是干净的, 并将其作为真实值与修复结果比较. 实验时, 我们采用均匀随机注错的方式向原始数据集中注入错误得到脏数据, 注入的错误类型包括拼写错误和替换错误, 对于一个属性值, 若注入拼写错误, 则在该属性值中随机添加一个字母; 若注入替换错误, 则从该属性值域中选取一个值替换 .

实验相关的参数设置与默认值选取如下: (1) 错误率 δ 表示数据中错误属性值的比例, $\delta\in[0,1]$, 默认值为 0.2; (2) 拼写错误比例 σ 表示数据中拼写错误占全部错误的比例, $\sigma\in[0,1]$, 默认值为 0.5; (3) 小分组比例 λ 表示 ORDER-λ 数据集中小分组的比例, $\lambda\in[0,1]$, 默认值 0.3; (4) 分组规模阈值 α 表示异常分组检测时分组规模的上限, 默认值为 1; (5) 编辑距离阈值 ω 表示异常分组检测时分组名字之间的编辑距离上限, 默认值为 1. 需要说明的是, 由于本文采用的注错机制已知 (拼写错误是在原属性值的基础上随机添加一个字母, 因此编辑距离和分组规模通常都为 1), 因此我们没有测试分组规模阈值和编辑距离阈值对本文方法的影响.

实验采用精确率、召回率及 F-score 评估修复结果的准确性, 其中, 精确率 *Precision* 是指子集修复删除的元组中错误元组的比例, 召回率 *Recall* 是指子集修复删除的错误元组占全部错误元组的比例. F-score 的计算公式如公式 (19) 所示.

$$F\text{-score}=\frac{2\times Precision\times Recall}{Precision+Recall} \tag{19}$$

4.2 准确性对比

本节测试 HybridOSR 方法与对比方法在不同参数下的精确率、召回率及 F-score, 并分析实验结果.

● 错误率的影响. 为了测试错误率对每种方法修复准确性的影响, 我们在 ORDER 数据集和 DBLP 数据集上随机注入错误, 错误率 δ 从 0.05 提升到 0.3 (如前文所述, 实际应用中的错误率通常在 5% 左右, 最多不会超过 30%[24,27]), 其中拼写错误比例 σ 采用默认值 0.5, 分组规模阈值 α 和编辑距离阈值 ω 采用默认值 1, 实验结果如图 8 和图 9 所示. 根据实验结果可知, HybridOSR 方法的精确率、召回率以及 F-score 均高于对比方法, 且受错误

率影响较小. 接下来, 我们分析基于规则的子集修复方法以及规则与概率相结合的更新修复方法准确率和召回率较低的原因.

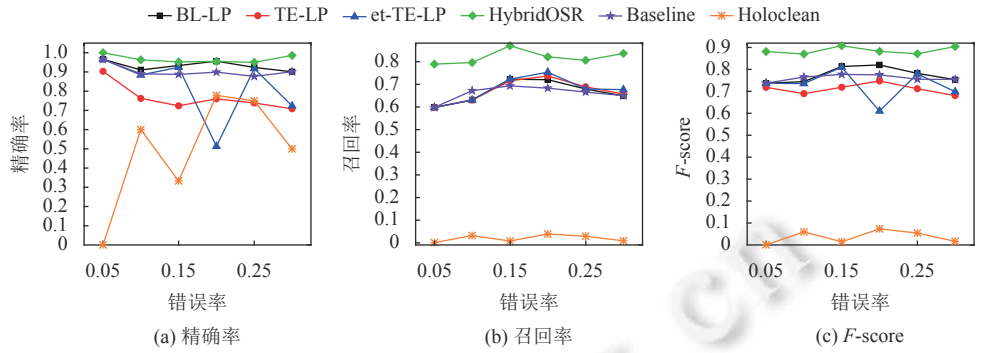


图 8 ORDER 数据集上不同错误率对修复准确性的影响

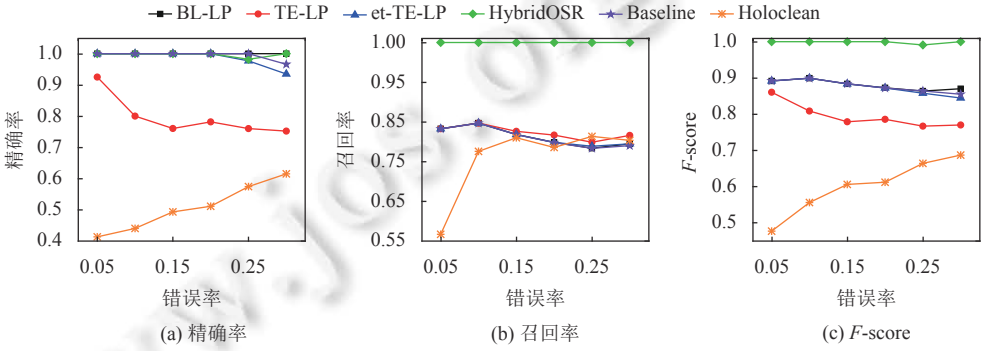


图 9 DBLP 数据集上不同错误率对修复准确性的影响

首先, 我们分析基于规则的子集修复方法在 ORDER 数据集上准确率和召回率较低的原因, DBLP 数据集上的情况与之类似, 不再赘述. (1) 在精确率方面, HybridOSR 的平均精确率分别为 0.96, BL-LP 和 Baseline 的平均精确率分别为 0.93 和 0.9, 而 TE-LP 和 et-TE-LP 的平均精确率分别仅为 0.76 和 0.82. 经分析, 造成 TE-LP 和 et-TE-LP 精确率较低的主要原因是其修复结果不是极大一致子集, 删除的元组中存在大量正确元组, 这与其调用的线性规划求解器没有找到最优解有关. 而 BL-LP 和 Baseline 的精确率低于 HybridOSR 的原因是它们没有考虑小分组问题, 当小分组中错误元组数量大于正确元组数量时, 将删除数量较少的正确元组. (2) 在召回率方面, HybridOSR 方法的平均召回率为 0.82, 而 BL-LP、TE-LP、et-TE-LP 以及 Baseline 的平均召回率分别为 0.66、0.67、0.67、0.66. 经分析, 造成基于规则的方法召回率低的原因主要有两个方面: 一方面, 当小分组中错误元组数量大于正确元组数量时, 基于规则的方法没有删除数量较多的错误元组; 另一方面对于由拼写错误导致的异常分组, 由于这些元组没有违反 FD, 因此基于规则的方法无法将其删除. (3) 在 F-score 方面, HybridOSR 的平均 F-score 为 0.89, 而 BL-LP、TE-LP、et-TE-LP 以及 Baseline 的平均 F-score 分别为 0.77、0.71、0.73、0.76. 由此可见, HybridOSR 的修复准确性最好. 此外, 近似比较好的 TE-LP 和 et-TE-LP 的 F-score 反而低于近似比较差的 BL-LP 和 Baseline, 由此可见, 在实际应用中我们可以牺牲一些近似比, 以换取更高的修复准确性.

其次, 我们分析规则与概率相结合的 Holoclean 方法在 ORDER 数据集和 DBLP 数据集上准确率和召回率较低的原因. (1) 在 ORDRE 数据集上, HybridOSR 的平均精确率和召回率 0.96 和 0.82, 而 Holoclean 的仅为 0.49 和 0.02. 经分析, 造成 Holoclean 准确率和召回率低的主要原因是其错误检测阶段将很多大分组中的正确元组识别为错误元组, 并加入到噪声集 D_n 中, 而将错误元组识别为正确元组加入到干净集 D_c 中, 在错误修复阶段, Holoclean 按照 D_c 中元组的属性值修改 D_n 中元组的属性值得到错误的修复结果 (需要说明的是, Holoclean 将错误检测看做

黑盒, 可以支持任意错误检测方法, 因此将本文所提的错误检测方法用在 Holoclean 系统中可以有效提高修复准确性). 另外, 我们发现 Holoclean 修复后的结果仍然是不一致的, 存在大量错误元组没有被删除. 换言之, Holoclean 的修复结果不是极大一致子集, 因此无法用在子集修复问题中. (2) 在 DBLP 数据集上, HybridOSR 的平均精确率和召回率为 0.99 和 1, 而 Holoclean 的为 0.5 和 0.73. 不难发现, Holoclean 的修复准确性在 DBLP 数据集上有一定提升, 这是由于 DBLP 中没有重复元组, ORDER 数据集中有 30% 的重复元组, 而 Holoclean 系统中所使用的错误检测方法在重复数据上的准确性较低, 进而导致后续修复的准确性低.

● 拼写错误比例的影响. 为了测试不同错误类型对每种方法修复准确性的影响, 我们在 ORDER 数据集和 DBLP 数据集上随机注入拼写错误和替换错误, 其中拼写错误比例 σ 从 0 提升到 1, 错误率采用默认值 0.2, 分组规模阈值 α 和编辑距离阈值 ω 采用默认值 1, 实验结果如图 10 和图 11 所示. 根据实验结果可知, HybridOSR 方法的精确率、召回率以及 F-score 均高于对比方法, 并且 HybridOSR 方法的各项指标随着拼写错误的比例增大而呈上升或水平趋势, 对比方法呈下降趋势. 接下来, 我们分析每种方法准确率和召回率随拼写错误比例变化的原因.

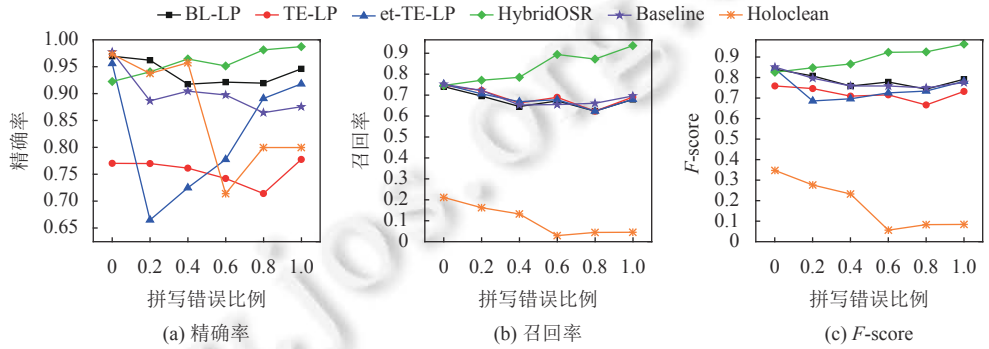


图 10 ORDER 数据集上不同拼写错误比例对修复准确性的影响

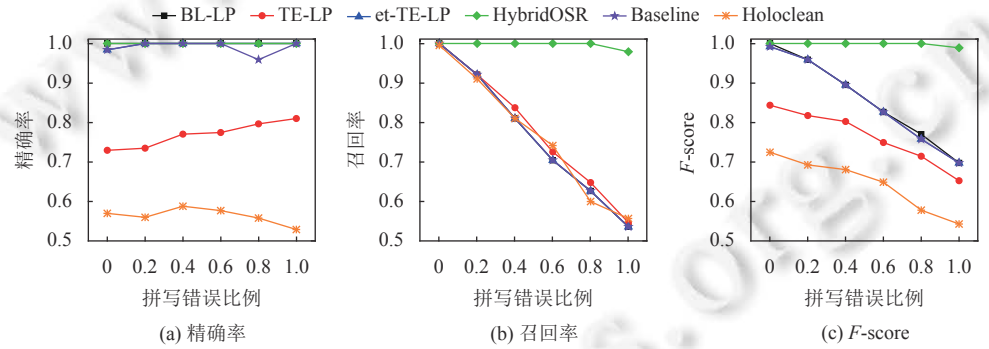


图 11 DBLP 数据集上不同拼写错误比例对修复准确性的影响

首先, 我们分析每种方法在 ORDER 数据集上准确率和召回率随拼写错误比例变化的原因. (1) 在精确率方面, 随着拼写错误比例的增大, HybridOSR 方法的精确率呈上升趋势, 而基于规则的子集修复方法 (BL-LP、TE-LP、et-TE-LP 及 Baseline) 及规则与概率相结合的更新修复方法 Holoclean 的精确性整体呈下降趋势. 经分析, 当数据中拼写错误增多时, ORDER 数据集上出现很多分组中正确格子与错误格子的规模相同的情况, 此时, 基于规则的子集修复方法删除正确格子的代价和删除错误格子的代价相同, 因此会随机选择一个格子删除, 导致精确率下降, 而 HybridOSR 方法采用最小化删除元组的概率和原则, 因此将删除概率和较小的错误格子, 精确率较高. Holoclean 的精确率下降的原因是其错误检测阶段无法有效识别拼写错误, 导致出现拼写错误的元组被加入干净集 D_c 中指导噪声集 D_n 修复. (2) 在召回率方面, 随着拼写错误比例的增大, HybridOSR 方法的召回率呈水平趋势且平均召回率接近于 1, 而对比方法整体呈下降趋势, 这是由于 HybridOSR 在错误检测阶段能够有效识别由拼写错误引发的

异常分组, 进而可以将发生拼写错误的错误元组删除, 而这些拼写错误没有违反 FD, 因此基于规则的子集修复方法无法将其删除. 对于 Holoclean, 由于其错误检测阶段将出现拼写错误的元组识别为正确元组, 因此召回率较低, 仅为 0.1.

其次, 我们分析每种方法在 DBLP 数据集上准确率和召回率随拼写错误比例变化的原因. (1) 在精确率方面, 随着拼写错误比例的增大, HybridOSR、BL-LP、et-TE-LP 及 Baseline 方法的精确率整体呈水平趋势且平均精确率接近于 1, 而 TE-LP 的精确性整体呈上升趋势, 平均精确率仅为 0.77, Holoclean 的精确性整体呈下降趋势, 精确率仅为 0.56. 经分析, 由于 DBLP 数据集中重复元组较少, 当数据中拼写错误增多时, 没有出现分组中正确格子与错误格子的规模相同的情况, 因此, 大部分基于规则的子集修复方法的精确率都很高, TE-LP 的精确性较低的原因是由于其修复结果不是极大一致子集, 与调用的线性规划求解器没有找到最优解有关. Holoclean 的精确率降低原因与 ORDER 数据集类似, 不再赘述. (2) 在召回率方面, 各方法的召回率变化原因与 ORDER 数据集类似, 不再赘述.

● 小分组比率的影响. 根据错误率影响实验的结果可知, 对比方法精确率和召回率较低的一个主要原因是小分组问题, 因此我们在本组实验中进一步测试小分组比例对每种方法的影响. 我们在 ORDER-λ 数据集上随机注入替换错误, 错误率采用默认值 0.2, 分组规模阈值 α 和编辑距离阈值 ω 采用默认值 1, 实验结果如图 12 所示. 根据实验结果可知, HybridOSR 方法的精确率、召回率以及 F-score 均高于对比方法且受小分组比例的影响较小. 接下来, 我们分析每种方法准确率和召回率随小分组比例变化的原因.

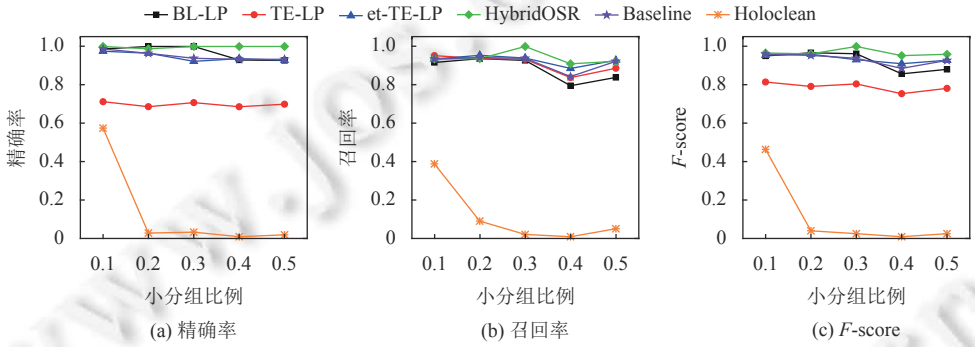


图 12 ORDER-λ 数据集上不同小分组比例对修复准确性的影响

随着小分组比例的增大, HybridOSR 方法的精确率和召回率呈水平变化趋势, 而基于规则的子集修复方法 (BL-LP、TE-LP、et-TE-LP 及 Baseline) 及规则与概率相结合的更新修复方法 Holoclean 的精确性和召回率整体呈下降趋势. 经分析, 当小分组比例增大时, 此时错误元组数量大于正确元组数量的分组数量增多, 而基于规则的子集修复方法只考虑删除元组数量最小化, 因此将删除正确元组而保留错误元组, 导致删除元组的精确率和召回率下降, 而 HybridOSR 同时考虑删除元组的数量与质量可以较好地解决小分组中错误元组数量较多的问题, 因此修复准确性不受小分组比例影响. 对于规则与概率相结合的更新修复方法 Holoclean, 其精确率和召回率随着小分组比例增大而大大降低, 当小分组比例超过 0.2 时, 精确率和召回率降至 0.1 以下, 这是由于 ORDER-λ 数据集中的重复元组比例非常大, 基于前文分析可知, Holoclean 系统中所使用的错误检测方法在重复数据上的准确性较低, 会将错误元组识别为正确的, 而正确元组识别为错误的.

5 结 论

本文研究了不一致数据子集修复问题, 提出了规则与概率相结合的子集修复方法, 通过设计元组概率模型使得正确元组的平均概率大于错误元组的平均概率, 并给出近似比为 2 的近似算法求解删除元组概率和最小的子集修复方案. 此外, 本文提出基于 FD 导出矩阵的错误检测方法, 可有效检测异常分组和大分组中的错误元组, 减小不一致元组规模, 提高子集修复效率. 真实数据上的实验结果验证了提出方法的准确性优于现有最好的子集修复方法.

References:

[1] Bertossi L, Kolahi S, Lakshmanan LVS. Data cleaning and query answering with matching dependencies and matching functions. Theory of Computing Systems, 2013, 52(3): 441–482. [doi: 10.1007/s00224-012-9402-7]

[2] Chu X, Ilyas UF, Papotti P. Holistic data cleaning: Putting violations into context. In: Proc. of the 29th IEEE Int'l Conf. on Data Engineering. Brisbane: IEEE, 2013. 458–469. [doi: 10.1109/ICDE.2013.6544847]

[3] Kolahi S, Lakshmanan LVS. On approximating optimum repairs for functional dependency violations. In: Proc. of the 12th Int'l Conf. on Database Theory. St. Petersburg: ACM, 2009. 53–62. [doi: 10.1145/1514894.1514901]

[4] Livshits E, Kimelfeld B, Roy S. Computing optimal repairs for functional dependencies. ACM Trans. on Database Systems, 2020, 45(1): 4. [doi: 10.1145/3360904]

[5] Rekatsinas T, Chu X, Ilyas IF, Ré C. Holoclean: Holistic data repairs with probabilistic inference. Proc. of the VLDB Endowment, 2017, 10(11): 1190–1201. [doi: 10.14778/3137628.3137631]

[6] Miao DJ, Cai ZP, Li JZ, Gao XY, Liu XM. The computation of optimal subset repairs. Proc. of the VLDB Endowment, 2020, 13(12): 2061–2074. [doi: 10.14778/3407790.3407809]

[7] Yakout M, Berti-Équille L, Elmagarmid AK. Don't be SCAREd: Use scalable automatic repairing with maximal likelihood and bounded changes. In: Proc. of the 2013 ACM SIGMOD Int'l Conf. on Management of Data. New York: ACM, 2013. 553–564. [doi: 10.1145/2463676.2463706]

[8] Mayfield C, Neville J, Prabhakar S. ERACER: A database approach for statistical inference and data cleaning. In: Proc. of the 2010 ACM SIGMOD Int'l Conf. on Management of Data. Indiana: ACM, 2010. 75–86. [doi: 10.1145/1807167.1807178]

[9] Ge CC, Gao YJ, Miao XY, Yao B, Wang HB. A hybrid data cleaning framework using Markov logic networks. IEEE Trans. on Knowledge and Data Engineering, 2022, 34(5): 2048–2062. [doi: 10.1109/TKDE.2020.3012472]

[10] Fan W, Geerts F. Foundations of data quality management. Synthesis Lectures on Data Management, 2012, 4(5): 1–217.

[11] Chomicki J, Marcinkowski J. Minimal-change integrity maintenance using tuple deletions. Information and Computation, 2005, 197(1–2): 90–121. [doi: 10.1016/j.ic.2004.04.007]

[12] Lopatenko A, Bertossi L. Complexity of consistent query answering in databases under cardinality-based and incremental repair semantics. In: Proc. of the 11th Int'l Conf. on Database Theory. Barcelona: Springer, 2007. 179–193. [doi: 10.1007/11965893_13]

[13] Afrati FN, Kolaitis PG. Repair checking in inconsistent databases: Algorithms and complexity. In: Proc. of the 12th Int'l Conf. on Database Theory. St. Petersburg: ACM, 2009. 31–41. [doi: 10.1145/1514894.1514899]

[14] Arenas M, Bertossi L, Chomicki J, He X, Raghavan V, Spinrad J. Scalar aggregation in inconsistent databases. Theoretical Computer Science, 2003, 296(3): 405–434. [doi: 10.1016/S0304-3975(02)00737-5]

[15] Lopatenko A, Bravo L. Efficient approximation algorithms for repairing inconsistent databases. In: Proc. of the 23rd IEEE Int'l Conf. on Data Engineering. Istanbul: IEEE, 2007. 216–225. [doi: 10.1109/ICDE.2007.367867]

[16] Bohannon P, Fan WF, Flaster M, Rastogi R. A cost-based model and effective heuristic for repairing constraints by value modification. In: Proc. of the 2005 ACM SIGMOD Int'l Conf. on Management of Data. Baltimore: ACM, 2005. 143–154. [doi: 10.1145/1066157.1066175]

[17] Dallachiesa M, Ebaid A, Eldawy A, Elmagarmid A, Ilyas IF, Ouzzani M, Tang N. NADEEF: A commodity data cleaning system. In: Proc. of the 2013 ACM SIGMOD Int'l Conf. on Management of Data. New York: ACM, 2013. 541–552. [doi: 10.1145/2463676.2465327]

[18] Khayyat Z, Ilyas IF, Jindal A, Madden S, Ouzzani M, Papotti P, Quiané-Ruiz JA, Tang N, Yin S. BigDansing: A system for big data cleansing. In: Proc. of the 2015 ACM SIGMOD Int'l Conf. on Management of Data. Melbourne: ACM, 2015. 1215–1230. [doi: 10.1145/2723372.2747646]

[19] Abedjan Z, Akcora CG, Ouzzani M, Papotti P, Stonebraker M. Temporal rules discovery for Web data cleaning. Proc. of the VLDB Endowment, 2015, 9(4): 336–347. [doi: 10.14778/2856318.2856328]

[20] Geerts F, Mecca G, Papotti P, Santoro D. The LLUNATIC data-cleaning framework. Proc. of the VLDB Endowment, 2013, 6(9): 625–636. [doi: 10.14778/2536360.2536363]

[21] Berti-Équille L, Dasu T, Srivastava D. Discovery of complex glitch patterns: A novel approach to quantitative data cleaning. In: Proc. of the 27th IEEE Int'l Conf. on Data Engineering. Hannover: IEEE, 2011. 733–744. [doi: 10.1109/ICDE.2011.5767864]

[22] Dasu T, Loh JM. Statistical distortion: Consequences of data cleaning. Proc. of the VLDB Endowment, 2012, 5(11): 1674–1683. [doi: 10.14778/2350229.2350279]

[23] Hellerstein JM. Quantitative data cleaning for large databases. United Nations Economic Commission for Europe, 2008, 25: 1–42.

[24] Krishnan S, Wang JN, Wu E, Franklin MJ, Goldberg K. ActiveClean: Interactive data cleaning for statistical modeling. Proc. of the

VLDB Endowment, 2016, 9(12): 948–959. [doi: 10.14778/2994509.2994514]
[25] Prokoshyna N, Szlichta J, Chiang F, Miller RJ, Srivastava D. Combining quantitative and logical data cleaning. Proc. of the VLDB Endowment, 2015, 9(4): 300–311. [doi: 10.14778/2856318.2856325]
[26] Ge CC, Gao YJ, Miao XY, Chen L, Jensen CS, Zhu ZY. IHCS: An integrated hybrid cleaning system. Proc. of the VLDB Endowment, 2019, 12(12): 1874–1877. [doi: 10.14778/3352063.3352088]
[27] Yu ZR, Chu X. PIClean: A probabilistic and interactive data cleaning system. In: Proc. of the 2019 Int'l Conf. on Management of Data. Amsterdam: ACM, 2019. 2021–2024. [doi: 10.1145/3299869.3320214]
[28] Li JZ, Liu XM. An important aspect of big data: Data usability. Journal of Computer Research and Development, 2013, 50(6): 1147–1162 (in Chinese with English abstract). [doi: 10.7544/issn1000-1239.2013.20130646]
[29] Redman TC. The impact of poor data quality on the typical enterprise. Communications of the ACM, 1998, 41(2): 79–82. [doi: 10.1145/269012.269025]
[30] Pham M, Knoblock CA, Chen MH, Vu B, Pujara J. SPADE: A semi-supervised probabilistic approach for detecting errors in tables. In: Proc. of the 30th Int'l Joint Conf. on Artificial Intelligence. Montreal: IJCAI, 2021. 3543–3551.

附中文参考文献:

[28] 李建中, 刘显敏. 大数据的一个重要方面: 数据可用性. 计算机研究与发展, 2013, 50(6): 1147–1162. [doi: 10.7544/issn1000-1239.2013.20130646]

附录 A

我们首先通过引理 A1 证明当数据干净时, 若一个分组的规模大于平均分组规模的 1/2, 则错误发生时, 该分组的正确格子中的元组数量大于错误格子中的元组数量; 然后, 基于引理 A1, 我们通过定理 A1 证明, 当数据中有错误时, 若一个分组的规模大于平均分组规模的 2/3, 则该分组的正确格子中的元组数量一定大于错误格子中的元组数量.

引理 A1. 假设错误率 $\varepsilon<\frac{1}{3}$ 且错误均匀随机发生在 FD 相关的属性上, 若数据干净时分组规模大于平均分组规模的 1/2, 则错误发生时正确格子中的元组数量大于错误格子中的元组数量.

证明: 给定 FD $\varphi: X\to A$ 及其导出矩阵 M_φ, 假设 M_φ 有 m 个分组 (即 X 属性有 m 种可能的取值), N 个元组, 平均分组数量 $\bar{n}=N/m$, 当数据干净时, 将每个 $X=x_i$ 分组中的元组数量记为 n_i.

假设 $X=x_i$ 分组的正确格子是 $A=a_j$, 则数据干净时, $A=a_j$ 格子中有 n_i 个元组. 发生错误时, $X=x_i$ 分组中将有 $n_i\varepsilon$ 个错误元组, $n_i(1-\varepsilon)$ 个正确元组. 由于错误均匀发生在 $X\cup A$ 的属性中 (A 中属性数量为 1, X 中属性数量为 $|X|$), 因此 A 属性出错的元组有 $n_i\varepsilon\frac{1}{|X|+1}$ 个, 这些元组的 X 属性值没有发生改变, 因此它们仍然在 $X=x_i$ 分组中; 而 X 属性出错的元组有 $n_i\varepsilon\frac{|X|}{|X|+1}$ 个, 这些元组的 X 属性值发生了改变, 因此不再属于 $X=x_i$ 分组. 此时, $X=x_i$ 分组中有 $n_i(1-\varepsilon)$ 个正确元组 (正确元组的 $A=a_j$), $n_i\varepsilon\frac{1}{|X|+1}$ 个错误元组 (错误元组的 $A\neq a_j$).

与此同时, $X\neq x_i$ 的分组中会有一些元组由于 X 属性出错而进入 $X=x_i$ 分组, 即出错后的 X 值等于 x_i. 接下来, 我们计算从其他分组进入 $X=x_i$ 分组的元组数量, 由于所有 $X\neq x_i$ 的分组中共有 $N-n_i$ 个元组, 其中 X 属性出错的有 $(N-n_i)\varepsilon\frac{|X|}{|X|+1}$ 个. 由于经过异常分组检测后, 左部属性上的错误只有替换错误, 而 X 属性共有 m 种可能的取值 ($X=x_1, X=x_2, \ldots, X=x_m$), 因此, $X\neq x_i$ 的分组中由于 X 属性出错而进入 $X=x_i$ 分组的元组数量为 $(N-n_i)\varepsilon\frac{|X|}{|X|+1}\frac{1}{m}$.

我们将错误发生后, $X=x_i$ 分组中的元组总数记为 n_i^*, 此时, $X=x_i$ 分组中的元组可分为 3 种: (1) 来自 $X=x_i$ 分组的 $n_i(1-\varepsilon)$ 个正确元组, 这些元组都在正确格子 $A=a_j$ 中; (2) 来自 $X=x_i$ 分组的 A 属性出错的 $\frac{n_i\varepsilon}{|X|+1}$ 个错误元组, 这些元组都在错误格子中; (3) 来自 $X\neq x_i$ 分组中的 X 属性出错的 $\frac{(N-n_i)\varepsilon|X|}{m(|X|+1)}$ 个错误元组, 其中 $A=a_j$ 的在正确格子中, $A\neq a_j$ 的在错误格子中, 最坏情况下, 这些元组都在错误格子中. 我们假设所有错误元组都在同一个

错误格子中, 此时, 错误格子的规模最大, 假设是 $A=a_k$ 格子. 接下来, 我们需要分析当 $n_i>\frac{N}{2m}$ 时, 正确格子 $A=a_j$ 中的元组数量是否大于错误格子 $A=a_k$ 中的元组数量.

我们首先将正确格子 $A=a_j$ 中的元组数量记为 $N_1=n_i(1-\varepsilon)$, 将错误格子 $A=a_k$ 中的元组数量记为 $N_2=\frac{n_i\varepsilon}{|X|+1}+\frac{(N-n_i)\varepsilon|X|}{m(|X|+1)}$. 通过对 n_i 合并同类项, 得到 N_1-N_2 的表达式如公式 (A1) 所示.

$$N_1-N_2=n_i\left[1-\varepsilon\left(1+\frac{1}{|X|+1}-\frac{|X|}{m(|X|+1)}\right)\right]-\frac{N\varepsilon|X|}{m(|X|+1)} \tag{A1}$$

然后, 判断 n_i 的系数是否大于 0, 即判断 $\varepsilon\left(1+\frac{1}{|X|+1}-\frac{|X|}{m(|X|+1)}\right)$ 是否大于 1. 不难发现, 当 $|X|$ 取最小值 1 时, $\frac{1}{|X|+1}$ 的值最大, 等于 $\frac{1}{2}$; 当 m 取无穷大, $|X|$ 取最小值 1 时, $\frac{|X|}{m(|X|+1)}$ 的值最小, 约等于 0; 因此 ε 的系数最大值小于 $\frac{3}{2}$, 又由于 ε 的最大值为 $\frac{1}{3}$, 因此, 两者乘积小于 $\frac{1}{2}$, 进而可知 n_i 的系数一定大于 0, 换言之, N_1-N_2 的值随着 n_i 增大而单调递增. 因此, 当 $n_i>\frac{N}{2m}$ 时, 有公式 (A2) 成立:

$$N_1-N_2>\frac{N}{2M}\left[1-\varepsilon\left(1+\frac{1}{|X|+1}-\frac{|X|}{m(|X|+1)}\right)-\frac{2\varepsilon|X|}{|X+1|}\right] \tag{A2}$$

对公式 (A2) 中的 ε 合并同类项得到公式 (A3). 接下来, 我们分析不等式右侧表达式的最小值, 不难发现, 当 ε 取最大值 $\frac{1}{3}$, ε 系数也取最大值时, 右侧表达式的值最小. 与前文分析类似, 当 $|X|$ 和 m 都取最大值无穷大时, ε 的系数最大, 最大值小于 3, 因此 ε 与系数的乘积小于 1, 进而可知 $1-\varepsilon\left(3-\frac{1}{|X|+1}-\frac{|X|}{m(|X|+1)}\right)>0$, 又由于 $\frac{N}{2M}>0$, 故右侧表达式的最小值大于 0, 因此 $N_1>N_2$, 证毕.

$$N_1-N_2>\frac{N}{2M}\left[1-\varepsilon\left(3-\frac{1}{|X|+1}-\frac{1}{m+m/|X|}\right)\right] \tag{A3}$$

引理 A1 给出了一种根据数据干净时分组规模判断大分组的方法. 然而, 在实际应用中干净数据通常是未知的, 下面定理 A1 给出了一种根据错误发生后分组规模判断大分组的方法.

定理 A1. 假设错误率 $\varepsilon<\frac{1}{3}$ 且错误均匀随机发生在 FD 相关的属性上, 若发生错误时分组规模大于平均分组规模的 2/3, 则该分组的正确格子中的元组数量大于错误格子中的元组数量.

证明: 给定 FD $\varphi: X\to A$ 及其导出矩阵 M_φ, 假设 M_φ 有 m 个分组 (即 X 属性有 m 种可能的取值), N 个元组, 平均分组数量 $\bar{n}=N/m$, 当数据干净时, 将每个 $X=x_i$ 分组中的元组数量记为 n_i, 错误发生后, $X=x_i$ 分组中的元组数量记为 n_i^*. 假设 $n_i^*>\frac{2N}{3m}$ 时, $X=x_i$ 分组中正确格子中的元组数量小于错误格子中的元组数量, 则根据引理 A1 可知, n_i 一定小于 $\frac{N}{2m}$. 根据引理 A1 的证明过程可知, 发生错误后, $X=x_i$ 分组中正确格子 $A=a_j$ 中的元组数量为 $N_1=n_i(1-\varepsilon)$, 错误格子 $A=a_k$ 中的元组数量为 $N_2=\frac{n_i\varepsilon}{|X|+1}+\frac{(N-n_i)\varepsilon|X|}{m(|X|+1)}$. 由于 $n_i^*=N_1+N_2$, 因此公式 (A4) 成立.

$$n_i^*=n_i(1+\varepsilon)+\frac{n_i\varepsilon}{|X|+1}+\frac{(N-n_i)\varepsilon|X|}{m(|X|+1)} \tag{A4}$$

对公式 (A4) 中的 n_i 合并同类项后, 再对 ε 合并同类项, 可得到公式 (A5).

$$n_i^*=n_i\left[1-\varepsilon\left(1-\frac{1}{|X|+1}+\frac{|X|}{m(|X|+1)}\right)\right]+\frac{N\varepsilon|X|}{m(|X|+1)} \tag{A5}$$

接下来, 我们需要证明当 $n_i<\frac{N}{2m}$ 时, n_i^* 一定小于 $\frac{2N}{3m}$, 换言之, 假设不成立, 从而当 $n_i^*>\frac{2N}{3m}$ 时, $X=x_i$ 分组中正确格子中的元组数量一定大于错误格子中的元组数量.

我们首先分析 n_i 的系数是否大于 0, 即判断 $\varepsilon\left(1-\frac{1}{|X|+1}+\frac{|X|}{m(|X|+1)}\right)$ 是否大于 1. 与前文分析类似, 当 $|X|$ 取无穷大时, $\frac{1}{|X|+1}$ 的值最小, 约等于 0; 当 m 取最小值 1, $|X|$ 取无穷大时, $\frac{|X|}{m(|X|+1)}$ 的值最大, 最大值为 1, 因此, ε 的

系数的最大值小于 2, 由于 ε 的最大值为 $\frac{1}{3}$, 因此, 两者乘积小于 $\frac{2}{3}$, 进而可知 n_i 的系数一定大于 0. 换言之, n_i^* 的值随着 n_i 增大而单调递增. 因此, 当 n_i 取最大值时, n_i^* 的值最大. 由于 $n_i < \frac{N}{2m}$, 因此, n_i^* 满足公式 (A6).

$$n_i^* < \frac{N}{2m}\left[1-\varepsilon\left(1-\frac{1}{|X|+1}+\frac{|X|}{m(|X|+1)}\right)\right]+\frac{N\varepsilon|X|}{m(|X|+1)} \tag{A6}$$

对公式 (A6) 中的 ε 合并同类项可得到公式 (A7).

$$n_i^* < \frac{N}{2m}\left[1+\varepsilon\left(\frac{|X|(m-1)}{m(|X|+1)}\right)\right] \tag{A7}$$

接下来, 我们分析公式 (A7) 右侧表达式的最大值, 首先分析 ε 的系数是否大于 0. 由于分组数 $m \geqslant 1$, 分组内元组数量 $|X| \geqslant 1$, 因此 ε 的系数大于 0, 并且当 ε 取最大值, 系数也取最大值时, 右式值最大. 我们首先将系数 $\frac{|X|(m-1)}{m(|X|+1)}$ 展开为公式 (A8).

$$\frac{|X|(m-1)}{m(|X|+1)} = 1-\frac{1}{|X|+1}-\frac{1}{m\left(1+\frac{1}{|X|}\right)} \tag{A8}$$

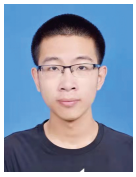
不难发现, 当 $|X|$ 和 m 取无穷大时, 公式 (A8) 值最大, 因此 ε 系数的最大值为 1, 又因为 ε 的最大值为 $\frac{1}{3}$, 因此, 公式 (A7) 的最大值为 $\frac{2N}{3m}$, 换言之, $n_i^* < \frac{2N}{3m}$, 这与假设中的 $n_i^* > \frac{2N}{3m}$ 矛盾, 因此假设不成立, 正确格子中的元组数量最多, 证毕.

张安珍(1990－), 女, 博士, 讲师, CCF 专业会员, 主要研究领域为大数据质量管理, 近似查询处理库.

朱睿(1982－), 男, 博士, 副教授, CCF 高级会员, 主要研究领域为流数据管理, 查询处理与优化.

司佳宇(1997－), 女, 硕士, 主要研究领域为数据质量管理.

邱涛(1989－), 男, 博士, 讲师, CCF 专业会员, 主要研究领域文本数据管理, 查询优化处理.



梁天宇(2001－), 男, 学士, 主要研究领域为数据质量管理.