基于词嵌入的元组级数据溯源方法

杨 彬[1],高俊涛[1],王志宝[1],李 菲[2],马 强[2],江树涛[1]
(1. 东北石油大学 计算机与信息技术学院,黑龙江 大庆 163318;
2. 黑龙江八一农垦大学 信息与电气工程学院,黑龙江 大庆 163319)

摘 要:在信息爆炸时代,数据量与日剧增,使用数据挖掘技术可挖掘其内在联系,但前提是所使用的数据正确无误,否则其后的一切工作将毫无意义。数据溯源技术可帮助数据分析人员快速定位到错误数据的来源和加工过程,减少错误数据的分析时间和难度,对数据质量控制与可信管理具有重要价值。现有的元组级数据溯源方法存在存储开销大和溯源效率低的问题,文章使用词嵌入技术改进元组级数据溯源方法。首先,研究元组向量化编码机制,依据元组向量相似度识别元组溯源关系;其次,提出基于属性重要性的优化算法提高溯源的精确率;再次,引入近似最近邻搜索和元组过滤优化机制降低溯源时间复杂度;最后,采用有向无环图展示元组数据的溯源关系。实验结果表明,该方法精确率较高、时间复杂度较低、存储消耗较少,能够有效改进元组级数据溯源方法。

关键词:结构化数据;数据溯源;元组向量;相似度比较;词嵌入
中图分类号:TP311.13;TP391 **文献标识码**:A **文章编号**:1673-629X(2023)12-0049-09
doi:10.3969/j.issn.1673-629X.2023.12.007

A Tuple-level Data Lineage Approach Based on Word Embedding

YANG Bin[1],GAO Jun-tao[1],WANG Zhi-bao[1],LI Fei[2],MA Qiang[2],JIANG Shu-tao[1]
(1. School of Computer and Information Technology,Northeast Petroleum University,Daqing 163318,China;
2. School of Information and Electrical Engineering,Heilongjiang Bayi Agricultural University,
Daqing 163319,China)

Abstract:In the era of information explosion,the volume of data is increasing day by day,and data mining technology can be used to explore its inner connection,but only if the data used is correct,otherwise all the subsequent work will be meaningless. Data lineage technology can help data analysts quickly locate the source and processing process of erroneous data,reduce the time and difficulty of analyzing erroneous data,and has important value for data quality control and trustworthy management. The existing tuple-level data lineage methods have the problems of high storage overhead and low lineage efficiency,and we use word embedding technology to improve the tuple-level data lineage methods. Firstly,the tuple vectorization encoding mechanism is investigated and tuple lineage relationships based on the similarity of tuple vectors is identified. Secondly,an optimization algorithm based on attribute importance is proposed to improve the precision of lineage. Thirdly,the approximate nearest neighbor search and tuple filtering optimization mechanism is used to reduce the lineage time complexity. Finally,a directed acyclic graph is used to show the lineage relationships of tuple data. The experiment shows that the proposed method has higher lineage precision,lower time complexity and less storage consumption,and can effectively improve the tuple-level data lineage method.

Key words:structured data;data lineage;tuple vectors;similarity comparison;word embedding

0 引 言

在大数据时代下数据生成规模激增,原生数据经过多次复制、迁移、集成、抽取等操作后形成海量派生数据,使数据来源及衍生路径表现出多样化、复杂化的特点[1]。若原生数据的来源模糊不清,则会极大程度地影响派生数据的可靠性[2]。数据溯源技术能够监控与评估数据质量,有助于定位错误根因,追踪错误路径,还可以对数据进行安全管控,能够帮助企业确定字段敏感信息。数据的可靠性和安全性是有效决策的基础,为加强数据质量,由此产生了数据溯源技术[3]。

收稿日期:2023-01-18 **修回日期**:2023-05-23
基金项目:国家自然科学基金资助项目(61902222);东北石油大学优秀中青年科研创新团队培育基金(KYCXTDQ202101)
作者简介:杨 彬(1996-),女,硕士研究生,研究方向为软件工程、数据分析、数据治理等;通讯作者:高俊涛(1979-),男,博士,副教授,CCF会员(F4854M),软件工程专委委员,研究方向为软件工程、过程建模、自动机学习等。

目前数据溯源在溯源理论、溯源模型以及方法实践上都开展了研究工作,但仍有不足,标注法的实现需要为元组保存完整的半环多项式(即标注),由于通过查询产生的元组依赖于先前查询的元组,导致半环多项式的数量大量增长,存在存储空间爆炸的问题。因此,国外学者 Leybovich M 等[4]提出基于词嵌入的元组级数据溯源方法,该方法有效避免存储数据标注。文中主要贡献如下:

(1)在元组向量化编码机制的基础上给出属性重要性优化算法,解决词嵌入方法中溯源精确率低的问题。

(2)引入近似最近邻搜索算法后又给出元组过滤优化策略,解决时间消耗长溯源效率低的问题。

1 相关工作

目前在数据溯源研究中主要有以下几方面工作。从溯源概念上,Lanter[5]首次提出“Data Lineage”用以描述目标数据的来源转化过程。Cui 等[6]从关系代数角度出发,定义了 View Data Lineage 用来标识数据仓库视图中目标数据的源数项集。Buneman 等[7]进一步将数据溯源进行分类,提出了 why 溯源和 where 溯源。Green 等[8]以半环多项式的形式提出了 how 溯源。从溯源粒度上,数据溯源被分为 3 个不同层次,第 1 层次是表级的数据溯源,其目标是获得目标表与源表之间的转换,是一种粗粒度的数据溯源;第 2 层次是字段级(列)的数据溯源,其目标是获得源表字段和目标表字段之间的属性映射关系,它是表级溯源的细化;第 3 层次是元组级(行)的数据溯源,其目标是获得目标表元组的源元组集合,是一种细粒度的数据溯源[9]。

从溯源模型上,W3C 发布的 PROV[10]是目前为止最成功的模型,成为数据溯源史上的里程碑。此后,围绕 PROV,专家学者对各个领域进行深入更深层次的研究。Niu X 等[11]将 PROV 引入关系数据库,将溯源信息存储成 PROV-JSON 的形式进行数据溯源。燕杨月[12]将 PROV 应用到物联网数据场景,实现对物联网起源信息的描述。杨斐斐等[13]对 PROV 进行扩展,构建了面向数据融合的溯源模型—PROV-Semi。

从溯源方法上,林悦邦[14]和张苒[15]等对支持全特性查询语言的逆置函数溯源方法进行研究。逆置函数法是指在计算时通过逆向查询或构造逆向函数对查询求逆,求逆的结果就是目标数据的源数据。其优点是只需存储少量的元数据就可实现数据溯源;缺点是具有一定局限性,需要提供逆置函数(并不是所有的函数都具有可逆性)和相对应的验证函数。Leybovich M 等[4,16]和 Hofmann F A 等[17]提出一种数据溯源的近似总结方法,其优点是可应用于派生数据更为膨胀的海量数据场景;缺点是以丢失一些信息为代价压缩表示溯源信息。Pierre Senellar 等[18-19]开发的 ProvSQL 系统[20]利用标注法实现元组级数据溯源。标注法是指记录数据的出处、产生过程、流转信息等作为数据标注,通过查询目标数据的标注来获得数据的溯源信息。其优点是实现简单,且容易管理;缺点是需要详细记录所有的数据转换信息,会出现元数据多于原始数据的情况。在实际应用中,细粒度形式的溯源信息标注通常会产成大容量存储,如何优化关系数据库下的溯源方法,成为亟待破解的难题[21]。

2 溯源定义及问题描述

2.1 元组级数据溯源

定义 1(元组级数据溯源):令 $T = Q(T_1, T_2, \cdots, T_m)$ 为将查询(Q)应用于表 $T_1, T_2, \cdots, T_m$ 的查询结果元组集,对于 $\forall t \in T$,都有 $Q^{-1}_{<T_1,T_2,\cdots,T_m>}(t) = < T_1^*, T_2^*, \cdots, T_m^* >$,则称 $T_1^*, T_2^*, \cdots, T_m^*$ 为元组(t)的源数据,即对产生元组(t)有贡献的数据元组集合(S)[6]。其中,$T_1^*, T_2^*, \cdots, T_m^*$ 是 $T_1, T_2, \cdots, T_m$ 的最大子集,则有:

(1) $Q(T_1^*, T_2^*, \cdots, T_m^*) = \{t\}$;

(2) $\forall T_i^* : \forall t^* \in T_i^* : Q(T_1^*, \cdots, \{t^*\}, \cdots, T_m^*) \neq \varnothing$。

2.2 问题描述

该文的主要内容是:研究元组级数据溯源方法,为数据库管理系统(DBMS)中元组的存在提供解释。即给定一个查询(Q)和查询输出的数据元组集合(T),在来源表中找出对产生每个元组($t \in T$)有贡献的源数据元组集合(S)。

示例 1:SQL 语句示例用来帮助理解元组级数据溯源定义。

```
Q:  INSERT INTO results(title, rating, timestamp)
    SELECT m.title,r.rating,r.timestamp
    FROM ratings r,movies m
    WHERE m.movieId=r.movieId
    AND r.userId=4;
```

表 1(a)为执行语句 Q 后获得的查询结果表(results),每个元组包含 userId 为 4 的用户观看的电影名,所给评分及评价时间。表 1(b)为评价表(ratings),每个元组包含有关用户表达的电影偏好的信息(0~5 星评分)。表 1(c)为电影表(movies),每个元组包含有关电影的基本信息。

通过分析可以得到,表 1(a)中标注为 q_1 的元组由表 1(b)中标注为 r_1 的源元组和表 1(c)中标注为 m_1 的源元组结合形成,如图 1 所示,以此实现元组(q_1)的一次溯源。

表 1　数据统计表

表名	字段				
	title	rating	timestamp		
(a)查询结果表 (results)	3	1573944252	Toy Story (1995)		q_1
	3.5	1573938415	Star Wars: Episode IV - A New Hope (1977)		q_2
	4	1573938898	Pulp Fiction (1994)		q_3
	userId	movieId	timestamp	rating	
(b)用户评价表 (ratings)	4	1	1573944252	3	r_1
	4	260	1573938415	3.5	r_2
	4	296	1573938898	4	r_3
	movieId	title	genres		
(c)电影表 (movies)	1	Toy Story (1995)	Animation|Children|Comedy|Fantasy		m_1
	2	Jumanji (1995)	Adventure|Children|Fantasy		m_2
	3	Grumpier Old Men (1995)	Comedy|Romance		m_3

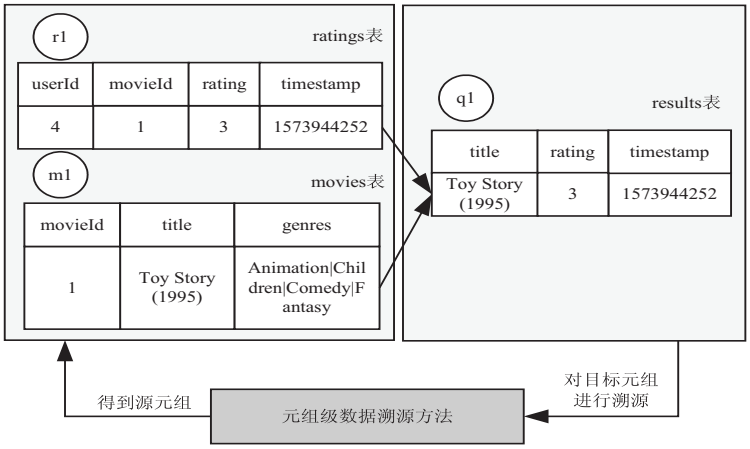


图 1　元组(q_1)的溯源过程描述

同理,可以对目标元组(q_2,q_3,q_4)直至 q_n进行溯源。并且可以对元组(r_1,m_1)向上溯源,最终形成一个有向无环图,即全链路数据溯源关系图。

3　元组级数据溯源方法

该文在 ProvEmb[4]的基础上增加了精确率优化机制和搜索效率优化机制,设计了一种基于词嵌入技术的元组级数据溯源改进方法(ProvEmb-X),其想法受生物学“基因”的启发,使寻找元组的源数据类似于通过 DNA 查找其前辈。

3.1　方法框架

图 2 为文中方法的研究框架,该方法支持范围广泛的非聚合 SQL 查询。输入数据是执行 SQL 后的查询结果元组集(即目标元组),输出数据是为该条元组返回其源元组 id,目的是对目标元组集中的每一条元组进行溯源。基本方法首先是通过一组向量来代表每个元组。其次,通过计算源表元组向量与目标元组向量的相似度,来识别溯源关系。再次,对仅依赖相似度比较导致相似的元组间可能没有溯源关系的问题,给

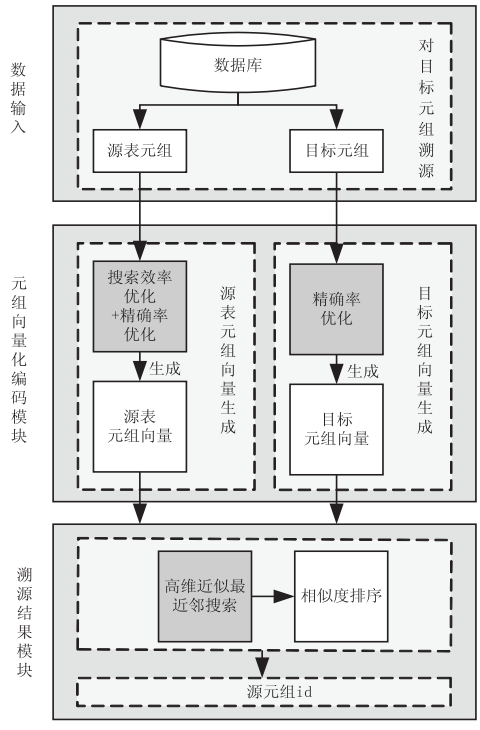


图 2　ProvEmb-X 方法框架

出优化方案提高精确率,并对溯源效率进行优化以减少时间消耗。最后,与基于词嵌入的数据溯源方法(ProvEmb)进行对比,验证文中方法的优化效果。该方法具有以下特点:

(1)该方法适合大规模数据集。通过分析 SQL 语句来精准定位所需要的数据库表,即便是派生数据更为膨胀的海量数据场景,仍可以快速过滤无用数据。

(2)该方法支持复杂的数据变换。可解决范围广泛的非聚合 SQL 查询的数据溯源问题,通过解析 SQL 语句中的连接、选择、分组、投影、集合和排序操作获取查询结果字段(目标字段)与数据库源表字段的对应关系,进而应用到方法中的属性重要性优化机制。

(3)该方法可实现自动化溯源。无论是纯手工收集的标注法,还是基于 SQL 的半自动标注法,都需要进行人工标注的工作。文中方法可实现自动化,提高溯源效率。

3.2 元组向量化编码机制

该文在元组向量化编码机制中进行目标元组向量和源表元组向量的两部分编码。该文使用 NLP 技术,词向量通过预训练的多语言词嵌入模型在基础文本上获得。文中元组向量化编码机制与 ProvEmb 不同的是,ProvEmb 是将元组中的每个列的文本组合起来形成一条没有语义的句子后进行词嵌入,该文是直接将每个元组中不同列的文本进行词嵌入后再组合,为后续属性重要性优化机制提高溯源精确率做准备。算法 1 为元组向量生成算法。

算法 1:元组向量生成算法(TPVEC)

输入:预训练的词嵌入模型(ST),输入数据表(T_i)

输出:为 T_i 中的每个元组(t)计算其元组向量(vector_t)并存在 tuplevector 中

```
1.  list vector_t   //存储每列文本的词向量
2.  list tuplevector   //存储生成的元组向量集
3.  for t ∈ T_i. tuples do   //遍历 T_i 中的每个元组
4.  vector_t = create(t)   //调用对 t 构造元组向量方法
5.  tuplevector . append(vector_t)
6.  end for
7.  return tuplevector
8.  function create( t )   //定义构造元组向量方法
9.      list wordvector
10.     list tuplevector
11. // w 是 t 中的每列文本,[ ]为构造 wordvector 列表
12.     wordvector = [ST(w) | w ∈ t]
13.     vector_t = weight(wordvector)
14.  return vector_t
```

示例 2:基于算法 1 的元组向量构造示例如下:设 $\overrightarrow{w_1}$,$\overrightarrow{w_2}$,$\overrightarrow{w_3}$,$\overrightarrow{w_4} \in \mathbb{R}^2$ 为元组(t)中的每列文本向量:

$$\overrightarrow{w_1}=\begin{bmatrix}1\\1\end{bmatrix}\quad \overrightarrow{w_2}=\begin{bmatrix}-0.5\\1\end{bmatrix}\quad \overrightarrow{w_3}=\begin{bmatrix}-1\\1\end{bmatrix}\quad \overrightarrow{w_4}=\begin{bmatrix}0.5\\-1\end{bmatrix}$$

通过算法 1 生成的元组向量($\vec{t}=\begin{bmatrix}0\\0.5\end{bmatrix}$),但此时 weight=1 生成的元组向量不具备元组特征,在精确率方面会有损失,下文将给出优化机制。

4 溯源方法优化机制

由于仅依赖目标元组向量与源元组向量的相似性,会出现两者间并无溯源关系,却还是错误地将该元组识别为溯源结果的情况。因此,该文提出了改进方案,通过以下 4 个步骤使算法在溯源精确率和效率上有较大的提升。(1)使用局部敏感哈希算法(LSH)提高溯源效率。(2)通过解析查询语句中 FROM 条件直接定位源表,初次过滤无关元组集合。(3)利用时间戳过滤目标元组的派生元组,再次过滤无关元组集合。(4)对于元组中的关键属性,采用加强其特征再组合的方式,形成独有的遗传密码“基因”,从而提高溯源精确率。

4.1 近似最近邻搜索

在进行实验时发现,暴力穷举式扫描对源表元组向量和目标元组向量进行相似度比较时,时间复杂度为 $O(dN)$,当数据的维度(d)以及数据的规模很大时,巨大的计算量与存储需求使得该搜索方式难以在效率上满足需求。

针对此问题,为满足大规模数据场景下的最近邻搜索任务需求,该文采用局部敏感度哈希算法(LSH)进行高维向量近似最近邻搜索(ANN),在损失一定精度的条件下,能够有效平衡精度与资源消耗,以更快的搜索速度和更少的内存负载得到查询项的近似精确甚至精确的搜索结果。LSH 的基本思想是将原高维空间的点都映射至 1 个或多个哈希表的不同位置(桶),原高维空间内距离较近的点会以较大概率映射至同一桶内,从而可直接在该桶内搜索元素,大大提高搜索效率。当哈希函数(h)满足以下两个条件,称 h 为局部敏感哈希函数[22]:

(1)如果 $L(q_1,q_2) < d_1$,则 $P[h(q_1)=h(q_2)] \geqslant p_1$;

(2)如果 $L(q_1,q_2) > d_2$,则 $P[h(q_1)=h(q_2)] \geqslant p_2$。

条件(1)保证 2 个相似点以较高概率被映射进同一个哈希桶;条件(2)保证 2 个不相似的点以较低概率映射进同一个哈希桶。其中,d_1,d_2,p_1,p_2 是给定的常数,$d_1 < d_2$;$h(q_1)$ 和 $h(q_2)$ 分别表示对 q_1 和 q_2 进行哈希变换;$P[h(q_1)=h(q_2)]$ 表示 q_1,q_2 两映射的哈希值相同的概率;$L(q_1,q_2)$ 表示 q_1 和 q_2 之间的相似

程度。该文使用基于余弦距离的局部敏感哈希函数,该算法主要通过计算 2 个向量之间夹角的余弦值来评估向量之间的相似性。其对应的哈希函数为:

$$H(V) = \text{sign}(\boldsymbol{V} \cdot \boldsymbol{R}) \quad (1)$$

其中,$\boldsymbol{R}$ 是一个随机向量,$\boldsymbol{V} \cdot \boldsymbol{R}$ 可以看作是将 $\boldsymbol{V}$ 向 $\boldsymbol{R}$ 上进行投影操作,其是 $(d_1, d_2, (180 - d_1)/180, (180 - d_2)/180)$ 敏感。

4.2　元组过滤优化机制

在对目标元组溯源时,源表中存在许多无关元组,若对这些无关元组也进行词嵌入生成元组向量,会浪费大量时间。因此,在本节中通过 2 个步骤过滤无关元组,加快搜索效率并提高溯源的精确率。

首先,解析 SQL 语句中的 FROM 条件进行筛选,因为 FROM 条件可以直接定位来源表,缩小查找范围。然后,在筛选后的元组中使用算法 2 元组过滤算法(TPFIL),通过元组创建时间戳过滤元组,能够帮助过滤掉非常相似但非源数据的元组,解决了完全依靠相似度的弊端,对查询远距离数据溯源尤其重要且加快了源元组的搜索效率。

算法 2:元组过滤算法(TPFIL)

输入:源数据表(T_i),查询结果表(R)

输出:经过 where 条件和时间戳过滤后的元组向量集(tuplevector)

```
1. list tuplevector
2. list vector_t
3. T_i' ← where(T_i)    //where 条件筛选得到相关元组集( T_i' )
4. //对每条查询结果元组在源表中进行时间戳过滤
5. for t ∈ R. tuples do
6.     for t' ∈ T_i'. tuples do
7.         if t'. timestamp > t. timestamp
8.             //调用构造元组向量方法,将 t 作为参数传入
9.             vector_t = create(t)
10.                tuplevector . append( vector_t )
11.        end if
12.    end for
13. end for
14. return tuplevector
```

(1)跟踪元组的创建时间戳。

①如果元组(t)被直接插入到 DB,则 t. timestamp 是其插入时间。

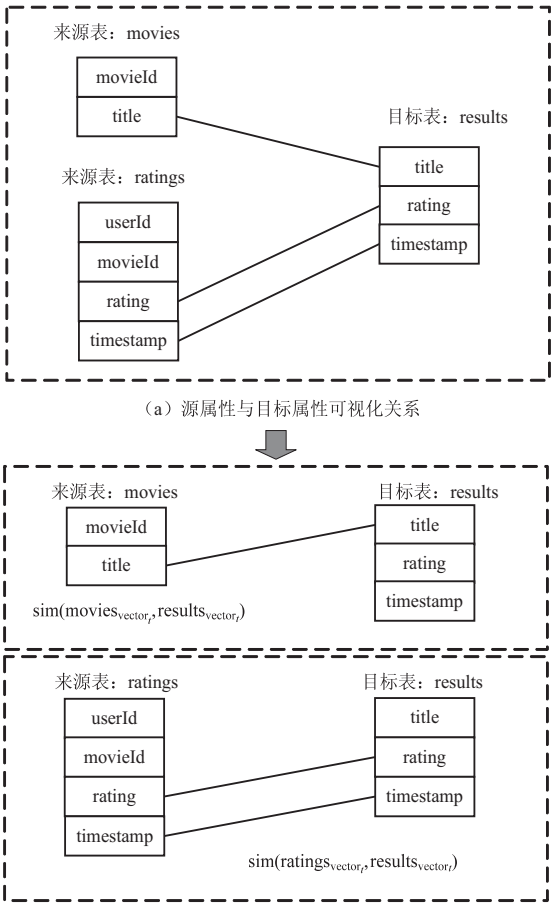
②如果元组(t)是通过查询计算的,则 t. timestamp 是查询的执行时间。

(2)当将一个元组(t)与一组其他元组(T')进行比较时,该文在计算元组向量生成之前进行判断:如果 t'. timestamp > t. timestamp ($t' \in T'$),则 t' 比 t 更新,并且不能成为其源元组的一部分。

4.3　属性重要性优化机制

由于 TPVEC 生成的元组向量不具备其本身特征,因此会导致溯源结果的精确率不高。为提高精确率,对数据库表结构进行研究发现,某些属性可能比其他属性更重要。例如,主键、外键或者是某些参与了数据库查询的重要属性。这意味着可以通过给这些属性的词向量赋予较高权重的方法来加强此元组向量的特征,进而提高相似度匹配的精确率。

该文通过解析 SQL 语句的方式寻找真正参与 SQL 运算的源属性和目标属性。解析 SQL 语句的主要步骤为:首先,对 SQL 查询语句进行词法分析和语法分析得到抽象语法树。然后,遍历以 Root 为节点的抽象语法树,得到 INSERT INTO target_table(target_attribute_list) 目标表(目标属性)、SelectList 源属性、FromClause 源表,再遍历 WhereClause 没有聚合函数的选择操作、GroupClause 分组操作、HavingClause 有聚合函数的选择操作、SortClause 排序操作等节点,从中获得目标属性和源属性的对应关系,以此来为目标属性和源属性的词向量加大权重。



(a)源属性与目标属性可视化关系

(b)得到加大权重的重要属性

图 3　属性权重获取流程

以示例 1 所示的查询语句为例,遍历抽象语法树后得到的可视化关系如图 3(a)所示。由图 3(b)可知结果表(results)中 title 来自于 movies 表,rating 和

timestamp 来自于 ratings 表。因此在 movies 表中元组向量生成时,应该为 title 赋予比 movies 表中其他属性要高的权重,说明 title 更能代表 movies 中元组的特征。同样,results 中的 title 权重也要加大。因此对每个属性进行词嵌入,将词向量加权平均生成元组向量后,再计算相似度从而提高匹配的精确率。权重计算公式如下所示(对于 $\forall w_i$ 都有 $0 \leqslant w_i \leqslant 1$):

(1) $\text{movies}_{\text{vector}_i} = w_1 \times \text{movieId} + w_2 \times \text{title}$ ($w_1 < w_2$ 且 $w_1 + w_2 = 1$)

(2) $\text{results}_{\text{vector}_i} = w_1 \times \text{title} + w_2 \times \text{rating} + w_3 \times \text{timestamp}$ ($w_1 > w_2, w_1 > w_3$ 且 $w_1 + w_2 + w_3 = 1$)

在 ratings 表中的元组向量生成时,需要为 rating 和 timestamp 赋予较高的权重,且 results 表中也要加大其对应的权重,权重计算公式如下所示(对于 $\forall w_i$ 都有 $0 \leqslant w_i \leqslant 1$):

(1) $\text{ratings}_{\text{vector}_i} = w_1 \times \text{userId} + w_2 \times \text{movieId} + w_3 \times \text{rating} + w_4 \times \text{timestamp}$ ($w_3 = w_4 > w_1, w_3 = w_4 > w_2$ 且 $w_1 + w_2 + w_3 + w_4 = 1$)

(2) $\text{results}_{\text{vector}_i} = w_1 \times \text{title} + w_2 \times \text{rating} + w_3 \times \text{timestamp}$ ($w_1 < w_2 = w_3$ 且 $w_1 + w_2 + w_3 = 1$)

算法 3 为属性权重获取算法,该算法结合深度优先搜索的思想,递归调用 visit 方法获取目标表属性和源表属性的对应关系用 attribute_r_list 存放,即应赋予高权重的重要属性。后执行算法 1(TPVEC),并更改由算法 3(PREUP)生成的重要属性的 weight 值。

算法 3:属性权重获取算法(PREUP)

输入:数据库语句(Q)

输出:目标表属性与源表属性的对应关系 attribute_r_list

```
1. Procedure AnalyzeDatalineage( Q )
2. List results_r_list  //存放目标表的表名和目标属性的关系
3. List source_r_list  //存放源表表名和源表属性的关系
4. List attribute_r_list  //存放目标表属性和源表属性的关系
5. QT ← generateSQLAST( Q );  //根据语句( Q ),生成抽象语法树(QT)
6. Function visit( r )  //对根节点为 r 的抽象语法树(QT)进行遍历
7. if R(r) ≠ ∅ then
8. String results_table  //初始化,定义变量 string 型
9. String source_table
10. String results_attribute
11.      for c in childs do //对节点进行类型判断
12.          //如果节点 c 是结果表类型
13.          If Type(c) = RESULTS TABLE then
14. results_table ← c //记录结果表表名
15.          Else if Type(c) = RESULTS ATTRIBUTE then
16.              //建立目标表表名与目标表属性的关系
17. results_r_list ← ([results_table, c])
18.          Else if Type(c) = SOURCE TABLE then
19. source_table ← c //记录源表表名
20.          Else if Type(c) = SOURCE ATTRIBUTE then
21.              //建立源表表名与源表属性的关系
22.              source_r_list ← ([source_table,c])
23.              //建立目标表属性与源表属性的关系
24. attribute_r_list ← ([results_attribute,c])
25.          Else if R(r) ≠ ∅ then
26. visit( c )
27.          End if
28.      End for
29. End if
30. Return attribute_r_list
```

综上所述,文中方法在第 1 阶段元组过滤优化机制时,FROM 过滤源表的时间复杂度为 $O(m)$,共 m 个源表,每个表有 k 个元组 n 个属性。时间戳过滤元组的时间复杂度为 $O(i \cdot k \cdot m')$,表示进行 $i \cdot k \cdot m'$ 次比较,i 为目标元组数,m' 为 FROM 过滤后的源表数量;在第 2 阶段属性重要性优化机制的时间复杂度为 $O(\max(m' \cdot n' + j' + 1))$,$n'$ 为遍历抽象语法树得到的源表的重要属性,j' 为遍历抽象语法树得到的目标表的重要属性(j 为目标表的所有属性);在第 3 阶段元组向量生成时的时间复杂度为 $O(i \cdot j' + k' \cdot n' \cdot m')$,$k'$ 为时间戳过滤后得到的元组;在第 4 阶段近似近邻搜索的时间复杂度为 $O(i \cdot j')$。因此,ProvEmb-X 的时间复杂度为 $O(\max(m' \cdot k' \cdot n' + i \cdot m' \cdot k))$,而 ProvEmb 为 $O(\max(m \cdot k \cdot n + i \cdot m \cdot k))$,$m' \cdot k' \cdot n'$ 远远小于 $m \cdot k \cdot n$。由此可知,文中方法能够有效降低时间复杂度。

5 实 验

基于提出的 ProvEmb-X 元组级数据溯源方法,在 3 种不同的数据集上进行实验,并与精确溯源系统 ProvSQL[20] 进行对比,验证 ProvEmb-X 的精确率;与 ProvEmb[4] 进行对比,验证 ProvEmb-X 的优化效果;最后展示对比实验结果。实验环境为:Intel(R) Core(TM)i7,16 GB 内存,Windows 10 操作系统。

5.1 实验数据

MovieLens 数据集描述 MovieLens 电影[23] 推荐系统网站中人们对电影的喜爱程度,数据集中的每部电影都有一个唯一标识符 movieId。固井作业数据集来自中国石油冀东油田公司实际固井数据,数据集中的每个井筒都有唯一标识符 wellbore_id。Olist 电子商务数据集[24] 来自巴西市场上最大的百货商店 Olist,数据集中的每个订单都有一个唯一标识符 order_id。

数据集统计信息如表 2 所示。

表 2　数据集统计信息

数据集	中文表名	英文表名	属性数量	元组数量
Movielens	电影表	movies	3	2 000
	评分表	ratings	4	10 000
	标签表	tags	4	1 048 575
固井作业	固井基础数据表	dr_ops_cement_info	48	22 440
	注水泥基本信息表	dr_ops_cement_info	32	21 683
	井筒表	cd_wellbore	3	5 424
Olist 电子商务	商品订单数据表	olist_order_items_dataset	7	112 652
	订单数据表	olist_orders_dataset	8	99 443
	客户数据表	olist_customers_dataset	5	99 443
	商品数据表	olist_products_dataset	9	32 953

5.2　实验结果及分析

5.2.1　精确率对比

(1)精确率评价指标。

ProvSQL 是基于标注的方法,由 Pierre Senellart 等人开发的开源项目,支持范围广泛的非聚合 SQL 查询,是目前为止较为精确的数据溯源系统。因此,该文使用公式(2)来计算实验结果的精确率,ApproxLineage(t)是该文实验结果元组的集合。ExactLineage(t)是由 ProvSQL 系统返回的关于 t 的精确源元组集合。Precision(t)代表返回的近似溯源结果与精确溯源结果的重合占比,该文将所得到的重合占比结果作为精确率的评价指标。

$$\text{Precision}(t)=\frac{|\ \text{ApproxLineage}(t,n)\cap\text{ExactLineage}(t)\ |}{\text{ApproxLineage}(t,n)} \tag{2}$$

(2)精确率优化消融实验。

在 3 种数据集上分别对 70 条 SQL 语句的查询结果进行溯源,将平均精确率作为实验结果。

由表 3 的精确率对比结果可知,ProvEmb-X 在 3 种数据集上的精确率均优于 ProvEmb(baseline)。在 Movielens 数据集和 Olist 电子商务数据集上的精确率优化效果不如固井作业数据集明显,是由于固井作业数据集中属性数量较多,由 w/o PREUP 与完整的 ProvEmb-X 相差 0.07 左右可以看出,PREUP 算法在极大程度地发挥作用提高精确率。由图 4(a)的增长情况也可得知 PREUP 算法表现较好。最终结果显示,在 Movielens 数据集上精确率整体提高了 2.35%,在固井作业数据集上精确率提高了 10.08%,在 Olist 电子商务数据集上精确率提高了 3.53%。

表 3　消融实验结果

	方法	Movielens 数据集	固井作业数据集	Olist 电子商务数据集
精确率对比	ProvEmb	0.852 40	0.763 51	0.834 15
	w/o FROM	0.864 13	0.835 63	0.842 78
	w/o TPFIL	0.865 32	0.843 18	0.851 86
	w/o PREUP	0.854 96	0.791 41	0.841 56
	ProvEmb-X	0.875 85	0.864 32	0.869 43
溯源效率对比	ProvEmb	2.97	4.35	3.07
	w/o FROM	2.76	4.27	2.93
	w/o TPFIL	2.61	3.84	2.69
	w/o LSH	2.64	3.92	2.84
	ProvEmb-X	2.58	3.77	2.66

5.2.2　溯源效率对比

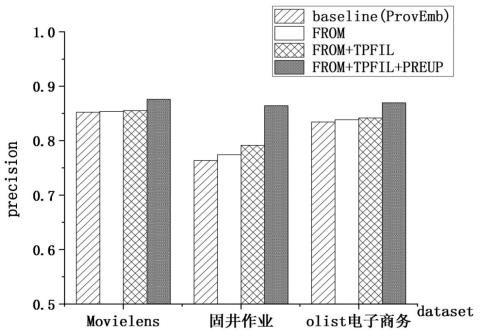
(1)溯源效率评价指标。

该文将单条元组的溯源消耗时间作为溯源效率的评价指标,单位为分钟。

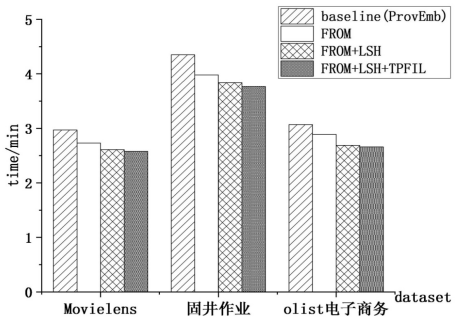
(2)溯源效率优化消融实验。

在 3 种数据集上分别对 70 条 SQL 语句的查询结果进行溯源,最后将单条平均耗时作为实验结果。

由表 3 溯源效率对比结果可知,ProvEmb-X 在 3 种数据集上的时间消耗均小于 ProvEmb。由 w/o FROM 与完整的 ProvEmb-X 差值得知“FROM 定位”表现最为突出,其次是 LSH 算法。



(a) 算法间精确率对比



(b) 算法间时间消耗对比

图 4　对比实验结果

由图 4(b)的 4 次对比实验可知,ProvEmb 的穷举式扫描对所有元组都进行元组向量生成,计算量巨大导致消耗时间也相对较长。因此,使用 FROM+LSH 算法将范围固定在少部分相关元组中。但是在实验中发现耗时仍然过大,继而使用 TPVEC+LSH+TPFIL 算法过滤非源数据的元组,再次缩短时间消耗。最终结果显示,在 Movielens 数据集上耗时减少了 13.13%,在固井作业数据集上减少了 13.33%,在 Olist 电子商务数据集上减少了 13.36%。

5.2.3 存储开销对比

该文采用计算 ProvSQL/ProvEmb(-X)相对比例的方式,对比标注法与词嵌入法的存储开销,等于 1 代表存储开销一样,小于 1 代表词嵌入法开销大,大于 1 代表标注法开销大,计算结果如表 4 所示。由结果可知标注法与词嵌入法的存储比较相差较小,是因为文中实验设备能力有限,实验数据体量相对较小且 SQL 语句数量较少,导致数据标注占用的存储空间不大,若是在大规模数据集上则会有较大的差异。且 ProvEmb-X 的存储开销比 ProvEmb 稍高,是由于在属性重要性优化机制阶段对解析 SQL 语句得到的 JSON 进行了存储。

表 4 存储开销相对比例

方法	Movielens 数据集	固井作业数据集	Olist 电子商务数据集
ProvEmb	1.14	1.22	1.35
ProvEmb-X	1.09	1.18	1.31

5.2.4 溯源结果展示

该文只对表 1(a)中 $t_{id}=8$ 的元组 results(Monty Python's Life of Brian (1979),3.5,1 573 944 005)进行溯源结果展示。由于文章篇幅限制,在相关表中各取相似度排名较高的前 7 个元组,并对其在 ProvSQL 中进行比较,最终实验结果如表 5 所示。其中,results 表中 $t_{id}=8$ 的元组的直系源数据是由 movies 表中 $t_{id}=1\ 053$ 的元组和 ratings 表中 $t_{id}=817$ 的元组组合而成。其他元组为 $t_{id}=8$ 的元组的间接源元组,如图 5 所示,可以清晰地展现 $t_{id}=8$ 的流转路径。

表 5 results 表中 $t_{id}=8$ 实验结果

n	来源表	t_{id}	相似度值排序	是否是源元组
1	movies	1 053	0.999 999 940 395 355 2	是
2	movies	6 684	0.784 715 116 024 017 3	是
3	movies	5 992	0.772 588 431 835 174 6	是
4	movies	1 108	0.714 576 542 377 471 9	是
5	movies	2 696	0.713 405 609 130 859 4	是
6	movies	5 014	0.702 540 755 271 911 6	是
7	movies	19 565	0.698 193 073 272 705 1	否
1	ratings	817	1.0	是
2	ratings	4 991	0.989 061 951 637 268 1	是
3	ratings	5 644	0.986 471 354 961 395 3	是
4	ratings	7 808	0.962 016 344 070 434 6	是
5	ratings	9 656	0.956 306 695 938 110 4	是
6	ratings	6 986	0.942 968 845 367 431 6	是
7	ratings	5 625	0.942 637 562 751 77	是

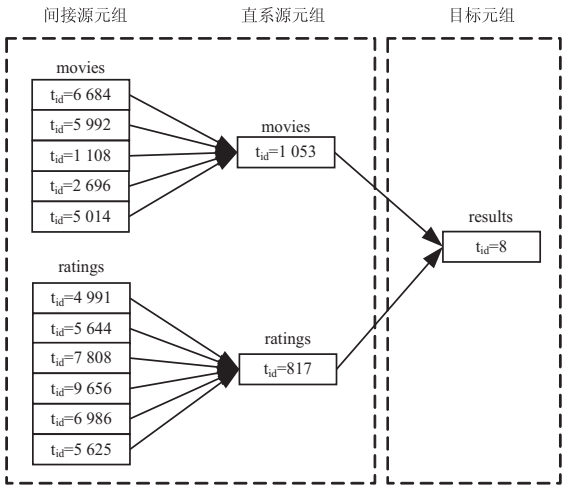


图 5 $t_{id}=8$ 的元组溯源关系

6 结束语

为解决传统方法中存储开销大的问题,该文提出了基于词嵌入的元组级数据溯源改进方法,并通过实验证明了该方法的有效性和普适性。该方法可应用于派生数据更为膨胀的海量数据场景,能够有效追溯元组级数据的来源,并通过溯源方法优化机制提高溯源精确率和时间效率,能够作为元组级数据溯源的有效方法。

目前数据溯源领域的研究成果相对较少,基于词嵌入的元组级数据溯源方法研究现仍然存在一些问题。通过计算相似度,以丢失一些信息为代价来压缩海量信息,最终实现元组级数据溯源。下一步研究内容为如何进一步提高精确率,以此来减少有用信息的丢失。并且目前在国内外关于元组级数据溯源方法的研究中,都没有解决聚合查询的溯源问题,下一步也可由此展开研究,从而提高方法的适用范围。

参考文献:

[1] 王 芳,赵 洪,马嘉悦,等. 数据科学视角下数据溯源研

究与实践进展[J]. 中国图书馆学报,2019,45(5):79-100.

[2] GLAVIC B. Data provenance[J]. Foundations and Trends® in Databases,2021,9(3-4):209-441.

[3] XIE Z. Tracer:a machine learning based data lineage solver with visualized metadata management[D]. Cambridg:Massachusetts Institute of Technology,2022.

[4] LEYBOVICH M,SHMUELI O. ML based provenance in databases[C]//Proceedings of the AIDB@ VLDB. Tokyo:VLDB,2020.

[5] LANTER D P. Design of a lineage-based meta-data base for GIS[J]. Cartography and Geographic Information Systems,1991,18(4):255-261.

[6] CUI Y, WIDOM J, WIENER J L. Tracing the lineage of view data in a warehousing environment[J]. ACM Transactions on Database Systems,2000,25(2):179-227.

[7] BUNEMAN P, KHANNA S, WANG-CHIEW T. Why and where:a characterization of data provenance[C]//Database theory—ICDT 2001:8th international conference. London:Springer,2001:316-330.

[8] GREEN T J,KARVOUNARAKIS G , TANNEN V. Provenance semirings[C]//Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems. Beijing:ACM,2007:31-40.

[9] 周　忠. 数据起源技术研究及其在 PostgreSQL 中的实现[D]. 广州:华南理工大学,2016.

[10] MISSIER P,BELHAJJAME K,CHENEY J. The W3C PROV family of specifications for modelling provenance metadata[C]//Proceedings of the 16th international conference on extending database technology. Genoa:EDBT, 2013:773-776.

[11] NIU X,GLAVIC B,GAWLICK D,et al. Interoperability for provenance-aware databases using PROV and JSON[C]//Proceedings of the 7th International Workshop on the Theory and Practice of Provenance(TaPP). Edinburgh, Scotland:In cooperation with USENIX and ACM SIGPLAN and SIGMOD,2015.

[12] 燕杨月. 物联网数据溯源模型研究与系统实现[D]. 北京:北京工业大学,2020.

[13] 杨斐斐,沈思妤,申德荣,等. 面向数据融合的多粒度数据溯源方法[J]. 计算机科学,2022,49(5):120-128.

[14] 林悦邦. 面向关系数据库的数据起源研究与设计[D]. 广州:华南理工大学,2017.

[15] 张　苒. 基于 PROV 的 ETL 溯源方法研究[D]. 长沙:国防科学技术大学,2017.

[16] LEYBOVICH M, SHMUELI O. Towards practical approximate lineage[C]//Proceedings of the 14th international workshop on the theory and practice of provenance(TaPP). Pennsylvania:[s. n.],2022:1-8.

[17] HOFMANN F A. Tracer:a machine learning approach to data lineage[D]. Cambridg:Massachusetts Institute of Technology,2020.

[18] SENELLART P. Provenance and probabilities in relational databases[J]. ACM SIGMOD Record,2018,46(4):5-15.

[19] SENELLART P, JACHIET L, MANIU S, et al. Provsql:provenance and probability management in postgresql[J]. Proceedings of the VLDB Endowment,2018,11(12):2034-2037.

[20] SENELLART P. ProvSQL on Github[EB/OL]. 2016-07-21. https://github.com/PierreSenellart/provsql.

[21] 王晓庆,孙战伟,吴军红,等. 基于数据要素流通视角的数据溯源研究进展[J]. 数据分析与知识发现,2022,6(1):43-54.

[22] 刘凤山. 近似最近邻搜索算法研究与应用[D]. 南京:南京大学,2021.

[23] HARPER F M,KONSTAN J A. The movielens datasets:history and context[J]. ACM Transactions on Interactive Intelligent Systems,2015,5(4):1-19.

[24] SIONEK A. Olist dataset[EB/OL]. 2021. https://www.kaggle.com/datasets/olistbr/brazilian-ecommerce.