计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.200900039

面向 NoSQL 数据库的 JSON 文档异常检测与语义消歧模型

刘立成　徐一凡　谢贵才　段　磊
四川大学计算机学院　成都 610065
(liuli_cheng@qq.com)

摘　要　随着信息化技术的发展,面对材料等相关领域数据的多源异构、扩展性强、爆炸增长等特点,传统关系数据库无法对数据进行存储,因此可利用 NoSQL 的无模式存储、高扩展性等特性来解决这一难题。作为 NoSQL 数据库常用的数据存储格式,JSON 因简单性和灵活性备受欢迎。然而,NoSQL 数据库缺乏模式信息,在 JSON 文档存入数据库之前,需要对其进行数据验证与分析。目前,大多数方法是基于 JSON schema 对 JSON 文档格式的规范性进行校验,无法有效解决 JSON 文档的异常检测以及语义歧义问题。为此,文中提出了面向 NoSQL 数据库的 JSON 文档异常检测与语义消歧模型 doctorJSON。该模型基于 JSON schema 对存入的 JSON 文档分别设计了异常检测算法 deoutJSON 和语义消歧算法 disemaJSON,以检测 JSON 文档存在的异常和歧义。在真实数据集与合成数据集上的实验验证了所提模型的有效性和执行效率。

关键词:NoSQL 数据库;JSON schema;JSON 文档;异常检测;语义消歧
中图法分类号　TP311

Outlier Detection and Semantic Disambiguation of JSON Document for NoSQL Database

LIU Li-cheng,XU Yi-fan,XIE Gui-cai and DUAN Lei
School of Computer Science,Sichuan University,Chengdu 610065,China

Abstract　With the development of information technology,traditional relational database cannot be used for storage due to multi-source heterogeneity,strong scalability and explosive growth of data in materials and other related fields. Therefore,NoSQL can be used with the charactersitics of schemaless storage and high scalability to solve this problem. As a common data storage format for NoSQL databases,JSON is popular for its simplicity and flexibility. However,NoSQL databases lack schema information,and JSON documents need to be validated and analyzed before being stored in the database. At present,most methods verify the normalization of JSON document format based on JSON schema,which cannot effectively solve the problem of exception detection and semantic ambiguity of JSON document. Therefore,a JSON document outlier detection and semantic disambiguating model for NoSQL database is proposed,named doctorJSON. Based on JSON schema,the model designs outlier detection algorithm deout JSON and semantic disambiguation algorithm disemaJSON to detect the outlier and disambiguation in JSON documents. The validity and efficiency of the model are verified by experiments on the real and synthetic datasets.

Keywords　NoSQL database,JSON schema,JSON document,Outlier detection,Semantic disambiguation

1　引言

随着信息化技术的快速发展,围绕数据驱动的数据库发展创新在诸多领域都有显著作用,如材料领域[1-2]、生物医用领域[3]。但是,面对海量多源异构数据的存储难题,传统关系型数据库明显难以处理。例如,在材料领域,由于材料学科的多样性,各种材料之间没有统一的数据表达和记录方式,材料数据的应用目的和需求也各不相同,数据的存储愈加复杂。而 NoSQL 数据库的无模式存储和高可扩展性可以有效解决这些问题和挑战。随着面向文档的 NoSQL 数据库越来越流行,JSON (JavaScript Object Notation)[1)] 正逐渐成为一种常见的数据表示格式,能够在缺乏显式数据模式的情况下处理大量数据。

尽管可以使用 JSON 描述各类数据,但 JSON 文档的分类标签和结构在不同的数据源中会存在差异,如在政府、高校中,各部门制定的数据标准不统一,各类数据的缺失值较多,进行数据清洗、数据整合的难度较大;在知识图谱构建中,数据多源异构,描述同一实体的数据可能存在差异;材料基因工

1) https://www.json.org/json-en.html

到稿日期:2020-08-04　返修日期:2020-09-25　本文已加入开放科学计划(OSID),请扫描上方二维码获取补充信息。
基金项目:国家自然科学基金(61972268)
This work was supported by the National Natural Science Foundation of China(61972268).
通信作者:段磊(leiduan@scu.edu.cn)

程专用数据库(Material Genome Engineering Databases)[1]使用基于JSON的材料描述语言,尽管可以使用这种统一的语法实现JSON模板文档的定义,但不同领域材料专家对材料研究的侧重点不同,各数据模板之间存在较大的差异。因此,有必要对这些异构JSON文档进行数据格式验证。由于人工审核数据存在成本过高的问题,对此本文提出了JSON文档检测模型doctorJSON,以检测文档中的异常数据并进行语义消歧。图1给出了数据校验的流程。

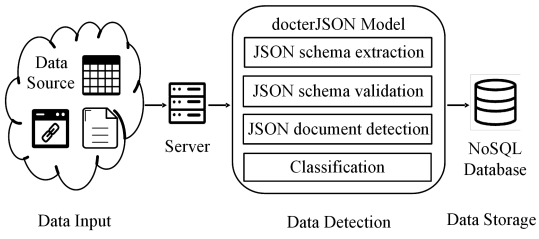

图1 JSON数据的校验流程

Fig. 1 JSON data validation flow

对JSON文档进行检测需要克服以下技术困难:1)NoSQL没有标准的查询语言并且缺少模式信息,因此难以执行数据检索、验证和分析;2)对JSON数据的检测研究主要基于JSON schema,这些研究大部分只能进行文档格式的校验,无法对不同JSON文档中的异常数据进行检测;3)对JSON数据格式的检测主要集中在JSON结构分析,缺少对JSON文档的语义分析。

在这样的背景下,本文基于JSON schema提出了面向NoSQL数据库的JSON文档检测模型doctorJSON,包括JSON schema抽取、JSON schema校验、JSON文档检测及分类。本文的主要贡献如下:1)提出了JSON文档检测模型doctorJSON;2)针对JSON文档检测,提出了JSON异常数据检测算法deoutJSON,可以对JSON文档中多种不同类型的异常进行有效检测;3)对不同JSON文档中具有相似语义的字段进行了语义消歧,提出了语义消歧算法disemaJSON,可提高JSON文档的分类准确率;4)在真实数据集上进行了大量实验,验证了算法的有效性。与其他方法相比,本文方法在异常检测效率、检测结果及分类准确率上都具有优势。

本文第2节介绍了相关工作;第3节定义了本文的研究问题;第4节对本文基于JSON schema提出的JSON文档检测模型docterJSON展开了叙述;第5节使用真实数据集和合成数据集进行实验验证与分析;最后总结全文,并对未来工作进行了展望。

2 相关工作

对于复杂的JSON文档,需要在结构级别和语义级别分别进行验证。对JSON的结构分析主要包含3种方法。1)基于JSON schema检测。JSON schema[4]首先被Json-schema组织提出,为默认的模式规范,目前已有大量程序语言支持这种规范,但这个定义并未成为一种标准,并且缺乏细节。在此基础上,Pezoa等[5]为JSON模式的规范提供了正式的语法,并对JSON schema的表达能力进行了理论研究。Bourhis等[6]对JSON模式和查询语言的逻辑表达和形式化定义进行了奠基性研究。Wang等[7]提出了称为skeleton的JSON模式以及有效管理JSON模式存储的框架。文献[8]介绍了用于高效解析和导入JSON数据以完成分析任务的JSON解析器,能动态地推断数据的结构信息。文献[9]介绍了一种从存储在NoSQL数据库中的JSON提取模式的方法,基于层次化数据结构对模式进行分组以生成全局模式。Meike等[10]提出了模式提取算法,基于所提取的模式信息,使用相似性度量捕获JSON的异构程度,揭示数据中的异常值。2)正则表达式匹配。Raihan等[11]提出了基于正则表达式的JSON模式匹配方式,更适合定义大型和复杂的模式。3)模式映射。文献[12]对JSON异构数据源的上下文生成模式映射关系,能够捕获JSON文档中的数据异常。

目前,对于JSON文档的语义分析并未有太多的相关工作,但对于这类半结构化的数据已经有类似方法进行语义处理。Jan等[13]基于XML文档中的语义信息,利用相似度分析算法辅助识别有问题的文本。Chen等[14]提出了半结构化数据的相似度度量方法,基于这个相似度进行食谱数据的分类。

JSON文档检测问题的核心在于检测数据异常与语义消歧,但上述方案均不能完美解决这个问题。因此,本文提出基于JSON schema的文档检测模型,重点关注JSON文档中的异常检测和语义消歧,有效解决了JSON文档检测问题。本文提出的异常数据检测算法和语义消歧算法分别在检测速度和分类准确率上具有一定的优势。

3 问题定义

定义模板 T 为标准的JSON文档,S_T 为对应的标准模式,待检测JSON文档记为文档 D,S_D 为对应的模式。存储在JSON文档中的数据包含以下7种基本类型:string,number,integer,boolean,null,array,object。文档 T 和 D 均为形如(*key*:*value*)的键值对 kv 的集合。$kv.key$ 表示关键字,具有唯一值,数据类型只能为string;$kv.vlaue$ 表示原子值,可以是7种基本数据类型中的任意一种。

例1 对于模板{"元素信息":{"名称":"铁","化学式":"Fe"}},存在三组键值对 kv_1=("元素信息":{"名称":"铁","化学式":"Fe"}),kv_2=("名称":"铁"),kv_3=("化学式":"Fe")。$kv_1.value$ 的类型为object,$kv_2.value$ 与 $kv_3.value$ 的类型均为string。

JSON结构具有灵活性[15],对于同一概念可以用多个JSON文档来表述,如图2(a)与图2(b)均可表示为铜元素的两个JSON文档 D_1 与 D_2。基于JSON schema S_T(见图2(c))对 D_1 与 D_2 进行验证时,只有文档 D_2 可以通过验证。基于此,本文给出了具体的问题研究,并进行了问题定义。

将模板 T 与文档 D 进行对比检测,检测出文档中的异常数据,异常数据可归纳为如下3种类型:冗余异常、缺失异常以及类型异常。

[1] https://www.mgedata.cn/

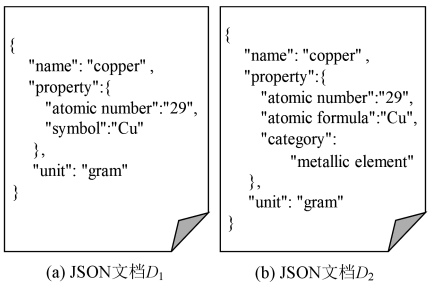

图 2 JSON 示例

Fig. 2 JSON example

例 2 对于模板 T={“元素信息”:{“名称”:“铁”,“化学式”:“Fe”}},文档 D={“元素信息”:{“元素名称”:“铁”,“化学式”:null}},文档 D 中存在如下异常数据:$kv_2^D.key$=“元素名称”冗余错误、$kv_2^T.key$=“名称”缺失错误、$kv_2^D.value$=null 类型错误。

同时,可以发现部分关键字 $kv.key$ 存在语义歧义。对于文档 D 在数据检测时出现的异常数据 $kv^D.key$,能够在模板 T 中找到与之相似的语义关键字 $kv^T.key$。对文档 D 中的关键字进行语义消歧,可以提高与模板 T 的语义相似度,减少数据库模板的冗余数量。

例 3 对于模板 T={“元素信息”:{“名称”:“铁”,“化学式”:“Fe”}},文档 D={“元素信息”:{“元素名称”:“铁”,“分子式”:“Fe”},文档 D 中 $kv_2^D.key$=“元素名称”、$kv_3^D.key$=“分子式”与模板 T 中 $kv_2^T.key$=“名称”、$kv_2^T.key$=“化学式”有极强的语义相似性。将文档 D 中的 $kv_i^D.key$ 替换为模板 T 中的关键字 $kv_j^T.key$ 进行语义消歧,可以提高与模板 T 的语义相似度。

模板 T 与文档 D 的相似度计算式如式(1)所示:

$$sim(T,D)=sim(\boldsymbol{M}_T,\boldsymbol{M}_D)=\frac{\boldsymbol{M}_T\cdot\boldsymbol{M}_D}{\|\boldsymbol{M}_T\|\ \|\boldsymbol{M}_D\|} \tag{1}$$

其中,$\boldsymbol{M}_T$ 和 $\boldsymbol{M}_D$ 分别为模板 T 与文档 D 中的关键字 $kv.key$ 经过 word embedding 后的词向量矩阵。为了方便描述,表 1 列出了本文常用的符号及定义。

表 1 符号汇总

Table 1 Summary of notations

符号	含义
T	标准 JSON 文档
D	待检测 JSON 文档
S_T,S_D	模板 T 和文档 D 分别对应的模式
kv	键值对(key:$value$)
C_T,C_D	键值对关键字 $kv.key$ 集合
$\boldsymbol{M}_T$,$\boldsymbol{M}_D$	关键字 $kv_i^T.key$ 和 $kv_j^D.key$ 经 word embedding 后的词向量矩阵
$sim(T, D)$	模板 T 与文档 D 的相似度

定义 1(异常检测问题) 给定 JSON 模板 T 和待检测 JSON 文档 D,对文档 D 中键值对 kv 的 3 种异常数据进行检测,即 $kv.key$ 冗余异常、$kv.key$ 缺失异常、$kv.value$ 类型异常。

定义 2(语义消歧问题) 给定 JSON 模板 T 和待检测 JSON 文档 D,对文档 D 中键值对 kv 的关键字 $kv.key$ 进行语义消歧。

4 JSON 文档检测模型

为了能够快速检测大规模 JSON 文档中的异常数据并进行语义消歧工作,本文基于 JSON schema 提出了 JSON 文档检测模型 doctorJSON,该模型的要点在于对未通过 JSON schema 校验的 JSON 文档分别进行异常检测以及语义消歧,以完成对待检测 JSON 文档的校验。

4.1 模型框架

doctorJSON 模型包含 JSON schema 抽取模块、JSON schema 校验模块、JSON 文档检测模块以及分类模块。图 3 给出了 JSON 文档检测模型 doctorJSON 的框架。

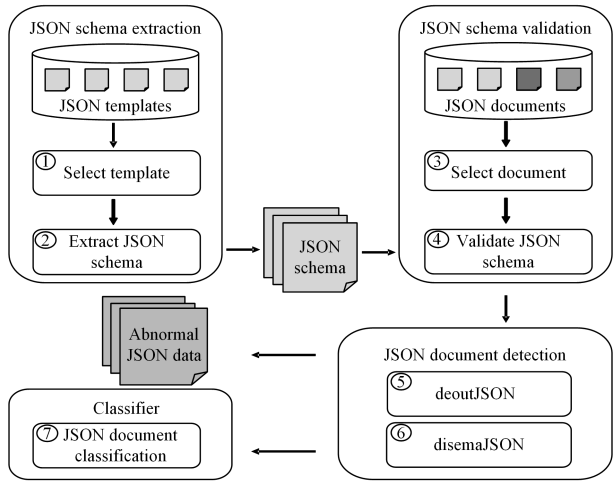

图 3 doctorJSON 模型框架

Fig. 3 Framework of doctorJSON model

(1)JSON schema 抽取模块。从标准 JSON 模板库中选择模板,利用 generate-schema 算法[16]生成 JSON schema S_T。

(2)JSON schema 校验模块。从待检测 JSON 文档库中选择文档,用 S_T 对选中的文档使用 jsonschema[17] 进行 schema 校验。若校验成功,则可证明该文档符合模板 T 的格式,执行模块(4);若校验失败,则证明该文档存在与模板 T 不匹配的字段,即存在异常数据,执行模块(3)。

(3)JSON 文档检测模块。该模块包含 JSON 异常数据检测算法 deoutJSON 和语义消歧算法 disemaJSON。输入模板 T 与检测文档 D,对文档 D 进行异常检测,输出待检测 JSON 文档中含有的异常类型信息;对文档 D 进行语义消歧时,根据语义相似度计算对 D 中的 $kv.key$ 进行关键字替换,完成后执行模块(4)。

(4)分类模块。将完成检测的 JSON 文档输入分类器,进行 JSON 文档分类。

在本文的模型框架中,模块(2)可以筛除大部分与模板 T 存在相同结构的待检测文档。对于未通过 JSON 校验的文

档，通过模块(3)可识别出具有不同类型异常的 JSON 文档，并且能够对与模板 T 中语义相似度较高的 $kv.key$ 进行关键字替换以实现语义消歧。模块(3)是 doctorJSON 模型的核心部分。下面将分别介绍模块(3)中的 deoutJSON 异常检测以及 disemaJSON 语义消歧算法。

4.2 异常检测算法 deoutJSON

目前，JSON 文档检测包含以下 3 种不同层次的检测策略：1)检测 JSON 的数据格式异常；2)检测 JSON 关键字 *key* 的异常；3)检测 JSON 所有数据信息的异常。deoutJSON 算法采用策略 2)和 3)，专注于检测键值对 kv 中 *key* 的冗余、缺失异常和 *value* 的类型异常。算法 1 给出了 deoutJSON 算法的伪代码，通过分别提取模板 T 与文档 D 中的键值对 $kv.key$ 生成关键字候选集合，在对比检测中移除具有相同关键字($kv_j^D.key=kv_i^T.key$)的候选项。其中，步骤 4－步骤 8、步骤 11－步骤 13 对候选集中剩余候选项进行关键字 *key* 和数据异常的检测，并将检测结果作为新的键值对加入集合$\mathbb{C}_D$中。步骤 10 返回利用算法 1 生成的异常数据集合$\mathbb{C}_D$。

算法 1 deoutJSON(T,D)

输入：模板 T=(kv_1^T,kv_2^T,…,kv_n^T)，提取模板 T 中所有关键字 kv_i^T.key 为集合 C_T；文档 D=(kv_1^D,kv_2^D,…,kv_m^D)，提取文档 D 中所有关键字 kv_j^D.key 为集合 C_D

输出：文档 D 中所有异常数据的 kv_j^D.key 的集合$\mathbb{C}_D$

1. $\mathbb{C}_D$←∅;//文档 D 中检测出的异常数据集合
2. C_T←{kv_i.key,kv_i∈T},C_D←{kv_j.key,kv_j∈D}
3. For kv_j^D.key∈C_D do
4. If kv_j^D.key∉C_T then
5. $\mathbb{C}_D$←(kv_j^D.key:“冗余异常”)
6. Else if typeof(D[kv_j^D].value)≠ typeof(T[kv_j^D].value) then
7. $\mathbb{C}_D$←(kv_j^D.value:“类型异常”)
8. Endif
9. Endfor
10. For kv_i^T∈C_T do
11. If kv_i^T∉C_D then
12. $\mathbb{C}_D$←(kv_i^T.key:“缺失异常”)
13. Endif
14. Endfor
15. Return $\mathbb{C}_D$

算法 1 生成关键字 $kv.key$ 的候选集合，能够有效避免不必要的计算，提升算法的执行效率。同时，针对模型关注的 3 类异常进行单独检测，可避免其他异常数据项的干扰，缩短数据检测的时间。

算法 1 中步骤 1－步骤 2 读入模板和文档并将数据初始化，时间复杂度为 $O(|C_T|+|C_D|)$。步骤 3—步骤 9 为一个迭代过程，每次迭代计算长度为 n 的模板 T 的关键字候选集合，并对比长度为 m 的文档 D 的关键字候选集合。每次迭代在最坏情况下的复杂度为 $O(m)$，当候选集合 C_T 为空时，算法结束，算法复杂度为 $O(m\times n+|C_T|+|C_D|)$。在实际对比检测中，C_T 中一旦出现异常项，就会直接查找 C_D 中对应的异常项，以提高执行效率。另外，doctorJSON 模型提供了文本或网页的形式数据检测结果，检测结果的对比更加明显。

4.3 语义消歧算法 disemaJSON

disemaJSON 算法首先对模板 T 及文档 D 进行关键字提取，生成关键字候选集合，然后输入至 embedding 模型得到词向量表达，对输出的词向量使用余弦距离计算两者间的相似度。相似度计算式如式(2)所示：

$$sim(\boldsymbol{t}_i,\boldsymbol{d}_j)=\frac{\boldsymbol{t}_i\cdot\boldsymbol{d}_j}{\|\boldsymbol{t}_i\|\ \|\boldsymbol{d}_j\|} \tag{2}$$

当相似度大于相似度阈值 γ 时，可认为这两个关键词的语义相似度较高，可以将模板 T 中的关键字 $kv_i^T.key$ 加入到 $kv_j^D.key$ 的关键字替换候选集合中，最后对候选集中的所有关键字进行相似度计算，选出与 $kv_j^D.key$ 相似度最高的关键字，并进行关键字替换以实现语义消歧。图 4 给出了 disemaJSON 的算法框架。

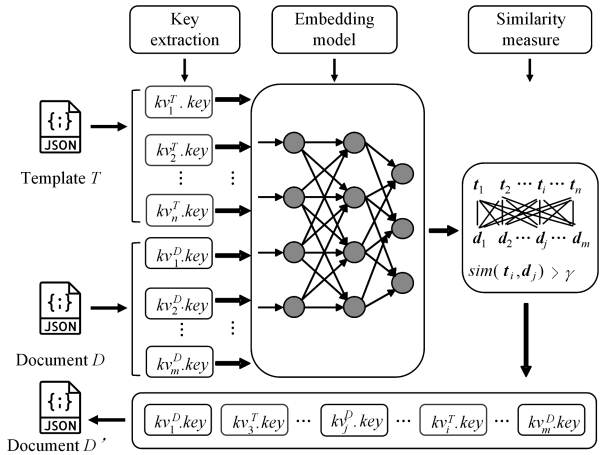

图 4 disemaJSON 算法框架

Fig. 4 Framework of disemaJSON algorithm

在上述计算过程中，对于单词 embedding 的模型，本文在实验中采用了目前使用较广的预训练模型[18]。该模型通过综合百度百科、中文维基百科等共 1 065 万个单词的语料库训练得到，具有良好的语义表达以及相似语义搜索能力。算法 2 给出了语义消歧算法 disemaJSON 的伪代码。

算法 2 disemaJSON(C_T,C_D)

输入：模板 T 中所有关键字 kv_i^T.key 的集合 C_T，经 embedding 模型处理后生成的词矩阵 $\mathbf{M}_T=\{\mathbf{t}_1,\mathbf{t}_2,\cdots,\mathbf{t}_n\}$；文档 D 中所有关键字 kv_j^D.key 的集合 C_D，经 embedding 模型处理后生成的词矩阵 $\mathbf{M}_D=\{\mathbf{d}_1,\mathbf{d}_2,\cdots,\mathbf{d}_m\}$；词向量相似度阈值 γ

输出：关键字替换后(语义消歧)的文档 D′

1. D′←∅
2. C_T←{kv_i.key,kv_i∈T},C_D←{kv_j.key,kv_j∈D}
3. $\mathbf{t}_i$←{embedding(kv_i^T.key),kv_i^T∈T}
4. $\mathbf{d}_j$←{embedding(kv_j^D.key),kv_j^D∈D}
5. For kv_j^D.key∈C_D do
6. If kv_j^D.key∉C_T and sim($\mathbf{t}_i$,$\mathbf{d}_j$)>γ then
7. D′←{kv_j^D←kv_k^T,max(sim($\mathbf{t}_k$,$\mathbf{d}_j$))} //关键字替换
8. Endif
9. Endfor
10. Return D′

算法 2 首先生成模板关键字集合 C_T、文档关键字集合 C_D，能够避免冗余计算，缩短检测时间。在算法执行过程中，

相似度阈值 γ 能有效避免不必要的关键字替换，使语义消歧结果更加合理，提高算法的执行效率。步骤 1－步骤 2 读入模板和文档并将数据初始化，步骤 3－步骤 4 将关键字输入 embedding 模型进行向量化处理，其时间复杂度为 $O(|C_T|+|C_D|+\sum|E|^{C_T}+\sum|E|^{C_D})$，$\sum|E|^{C_T}$ 和 $\sum|E|^{C_D}$ 为 embedding 模型处理词向量的时间。步骤 5—步骤 9 为一个迭代过程，每次迭代计算长度为 m 的文档 D 关键字集合 C_D 与长度为 n 的模板关键字集合 C_T 中关键字的相似度。每次迭代在最坏情况下的复杂度为 $O(m\times n)$，当文档关键字集合 C_D 为空时，算法结束。disemaJSON(C_T,C_D) 算法的复杂度为 $O(m\times n+|C_T|+|C_D|+\sum|E|^{C_T}+\sum|E|^{C_D})$。

5 实验

5.1 实验环境

本文在真实数据集和合成数据集上进行了实验，以验证算法的有效性和执行效率。数据集分别来自于材料基因工程专用数据库(MGE)[1)]提供的材料数据和四川公共数据平台[2)]提供的政府机构数据。同时，针对材料数据库中标准模板数量不足的问题，本文以不同策略从材料数据集中合成得到数据集 Dataset3，用于进行语义消歧实验。表 2 列出了实验中所选用数据集的特征。

表 2 数据集特征

Table 2 Characteristics of datasets

Dataset	Label	Category	Templates	Documents
Material	Dataset1	11	57	4 704
Organization	Dataset2	6	27	60 144
Generated data	Dataset3	11	57	436

deoutJSON 算法和 disemaJSON 算法均用 Python 编程实现，实验环境配置为 Intel(R) Xeon(R) CPU E5-2698 v3 @ 2.30 GHz CPU，Ubuntu 20.04 LTS 操作系统的服务器。

5.2 异常检测算法实验

因为本文检测的异常检测目标与 Jsondiff 方法[19]和 JsonCompare 方法[20]类似，所以本文选用这两种方法进行对比，以验证 deoutJSON 算法的有效性。本文在 Dataset1 和 Dataset2 上分别运行 JsonCompare，Jsondiff 与 deoutJSON 算法，对 JSON 文档异常检测的时间进行比较。

图 5 和图 6 给出了不同算法在两个真实数据集上的检测效率。可以看出，随着数据规模的扩大，deoutJSON 的检测时间呈线性增长，而 Jsondiff 和 JsonCompare 的检测时间不稳定，执行效率出现了波动，本文方法有更好的稳定性。同时，deoutJSON 算法在不同规模的数据中均能以最小消耗时间检测出 JSON 文档的异常，检测时间远短于 JsonCompare 方法，也短于 Jsondiff 方法，实验结果表明 deoutJSON 算法异常检测的效率更高。

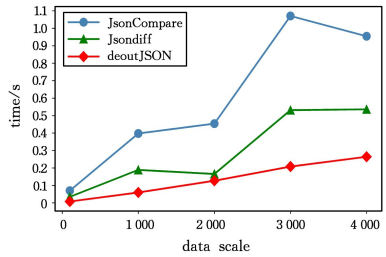

图 5 数据集 1 的异常检测

Fig. 5 Outlier detection on Dataset1

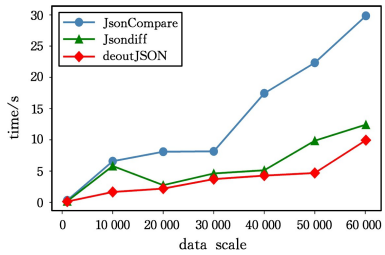

图 6 数据集 2 的异常检测

Fig. 6 Outlier detection on Dataset2

为了验证 deoutJSON 算法的有效性，我们随机选取了数据集 Dataset1 和 Dataset2 中的 100 个文档，并在其中添加了 3 类异常项($kv.key$ 冗余、$kv.key$ 缺失、$kv.value$ 类型异常)，构建的数据集为 Dataset1* 和 Dataset2*。使用 Jsondiff，JsonCompare，deoutJSON 方法进行异常项检测，从异常项检测的精确度、召回率、F1 值 3 方面对比 3 种方法，结果如表 3 所列。在精确度上，deoutJSON 都优于另外两种方法；在召回率上，deoutJSON 提升了 20%～40%；在 F1 值的评价上，deoutJSON 在不同数据集中都取得了最好的表现。

表 3 deoutJSON 实验结果

Table 3 deoutJSON experimental results

Methods	*Accuracy*		*Recall*		*F1*	
	Dataset1*	Dataset2*	Dataset1*	Dataset2*	Dataset1*	Dataset2*
Jsondiff	0.787	0.911	0.934	**0.982**	0.854	0.945
JsonCompare	0.798	1.000	0.497	0.743	0.612	0.852
deoutJSON	**0.991**	**1.000**	**0.970**	0.929	**0.981**	**0.963**

本文选取了真实数据集 Dataset1 中的模板 T 与文档 D 进行实例分析，结果如图 7 所示。对 JSON 模板 T(图 7(a))对和 JSON 文档 D(图 7(b))分别使用 deoutJSON，Jsondiff，JsonCompare 进行异常检测，可以得到图 7(c)－图 7(e)所示的输出结果。deoutJSON 算法对于模型关注的 3 种异常类型均能进行有效检测(异常数据如图 7(b)中红色部分所示)，如 $kv_1^T.key$＝“时效时间(h)”缺失异常、$kv_2^D.key$＝“密度”冗余异常、$kv_2^D.value$＝null 类型异常。而 Jsondiff 和 JsonCompare 方法无法有效分辨出 $kv_1^D.key$＝“时效时间(h)”缺失异常和 $kv_2^D.key$＝“密度”冗余异常，并且 Jsondiff 方法对于 $kv.value$

1) http://www.mgedata.cn

2) http://www.scdata.net.cn/odweb

也进行了异常检测，如 $kv_2^D.value$="1 428"，为错误检测项。

实例分析结果验证了 deoutJSON 算法的有效性。

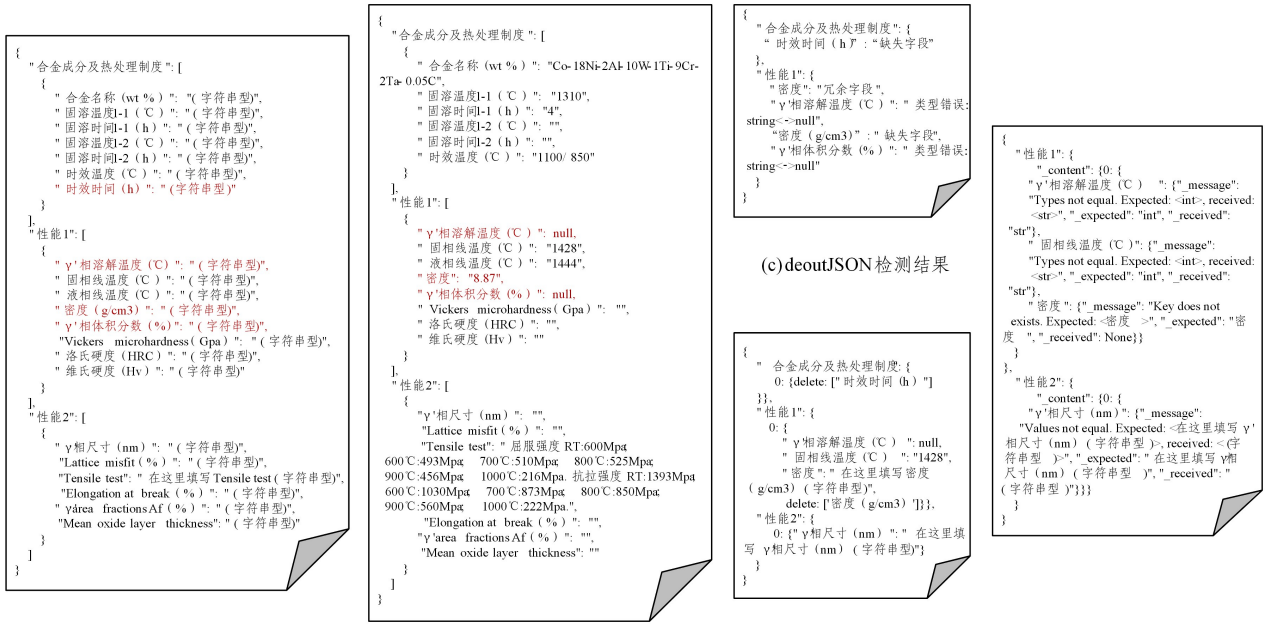

图 7 实例分析(电子版为彩色)

Fig. 7 Case study

5.3 语义消歧算法实验

为了验证语义消歧算法 disemaJSON 的实验效果，本文基于 3 个数据集对 JSON 文档进行语义消歧后基于相似度量 $sim(T,D)$ 进行分类实验，最后对语义消歧前后的分类准确率进行了对比。

本文选用了 5 种不同的分类算法，其中方法 W. Chen 是文献[14]提出的基于相似度的结构化文档分类方法。所有的实验结果都是在相似度阈值 γ 为 0.65(由 5.4 节可知，当 γ=0.65 时，模型的结果最好)时，采用十折交叉验证得到的，如表 4 所列。带有"+"号的结果为经过 disemaJSON 算法进行语义消歧之后的分类结果，其中较好的结果加粗表示。

从实验结果可以看出，使用 disemaJSON 算法进行语义消歧后，在大多数分类器模型上的效果均有提升，这可以证明语义消歧算法 disemaJSON 的有效性。

对于 disemaJSON 算法的实验结果(见表 4)中出现的异常结果，我们认为原因如下：

(1)分类器算法不同导致 disemaJSON 在不同分类器中的提升效果有差异，根据实验结果，disemaJSON 在随机森林中的提升效果最明显。

(2)数据集质量。对于消歧实验，在数据集 2 中，disemaJSON 算法在一些分类器下的效果反而下降，其原因是使用的机构数据集更加规范化，存在的需要语义消歧项较少，因此语义消歧后可能会破坏原有模板的规范性，使得分类效果反而下降。

表 4 disemaJSON 实验结果

Table 4 disemaJSON experimental results

Methods	*Macro-F*1 Score			*Micro-F*1 Score		
	Dataset1	Dataset2	Dataset3	Dataset1	Dataset2	Dataset3
W. Chen[14]	0.123	0.120	0.304	0.217	0.273	0.355
W. Chen +[14]	**0.138**	**0.143**	**0.351**	**0.239**	**0.273**	**0.410**
KNN	0.238	0.357	0.469	0.343	0.467	0.551
KNN+	**0.254**	**0.431**	**0.805**	**0.387**	**0.550**	**0.847**
SVM	0.235	**0.672**	0.661	0.283	**0.750**	0.717
SVM+	**0.259**	0.661	**0.848**	**0.393**	0.733	**0.893**
Decision Tree	0.191	**0.678**	0.352	0.283	**0.733**	0.505
Decision Tree+	**0.227**	0.644	**0.737**	**0.300**	0.633	**0.806**
Random Forest	0.183	0.575	0.365	0.280	**0.700**	0.495
Random Forest+	**0.211**	**0.578**	**0.767**	**0.320**	0.667	**0.793**

5.4 参数 γ 分析实验

相似度阈值 γ 是 disemaJSON 算法衡量不同关键词语义相似度的指标。如果 γ=0，表示待检测 JSON 文档与模板 T 中的 *key* 完全没有语义相似性，无需进行语义消歧；如果 γ=1，则待检测 JSON 文档与模板 T 相同，无需进行异常检测。

图 8、图 9 给出了 disemaJSON 算法在 Dataset3 上参数取值在 0.60～0.95 之间变化时 *Macro-F*1 和 *Micro-F*1 的变化。可以看到，当 γ 参数增加到一定阈值时，*Macro-F*1 与 *Micro-F*1 的分数会停止提升甚至开始下降，这说明在模型中可以根据适当的相似度阈值 γ 来平衡语义消歧的结果。

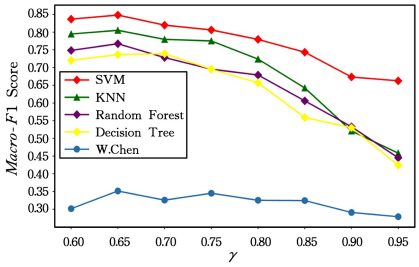

图 8 不同阈值 γ 在数据集 3 上的 *Macro-F*1 值

Fig. 8 *Macro-F*1 score on Dataset3 with different γ

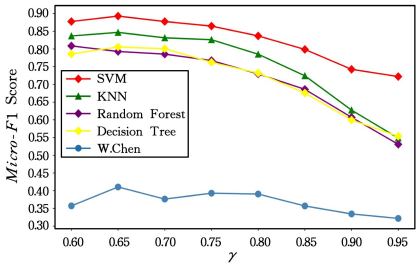

图 9 不同阈值 γ 在数据集 3 上的 *Micro-F*1 值

Fig. 9 *Micro-F*1 score on Dataset3 with different γ

结束语 本文对 JSON 数据检测进行了研究,提出了基于 JSON schema 的数据检测模型。针对 JSON schema 无法有效检测出异常数据的问题,本文提出了 deoutJSON 算法进行异常数据检测。同时,针对 JSON 文档中关键字语义性模糊的问题,使用 disemaJSON 算法进行语义消歧。最后在真实数据集和合成数据集上验证了 deoutJSON 算法和 disemaJSON 算法的有效性和执行效率。

在下一阶段的工作中,我们考虑部署分布式集群,使 deoutJSON 算法适用于更大规模的数据分析,并在更多实际场景中验证算法的可靠性。针对 disemaJSON 算法,利用 JSON 结构相似性结合语义相似性进行算法优化,考虑将 JSON 转化为树结构,计算结构相似性;对于异常项的语义相似性计算,通过加入向量权重计算词向量的相似度,提高异常项相似度的准确性。

参 考 文 献

[1] KAUFMAN J, BEGLEY E. MatML: A Data Interchange Markup Language[C]// Advanced Materials & Processes/November. 2003: 35-36.

[2] FRENKEL M, CHIRICO R, DIKY V, et al. ThermoML: XML-based IUPAC Standard for Experimental, Predicted, and Critically Evaluated Thermodynamic Property Data Storage and Capture[J]. IUPAC Recommendations, 2006, 78(3): 541-612.

[3] LAKIOTAKI K, VORNIOTAKIS N, TSAGRIS M, et al. BioDataome: a collection of uniformly preprocessed and automatically annotated datasets for data-driven biology[J]. Database the Journal of Biological Databases & Curation, 2018, 2018: bay011.

[4] JSON schema language[OL]. http://json-schema.org.

[5] PEZOA F, REUTTER J, SUAREZ F, et al. Foundations of JSON Schema[C]// Proceedings of the 25th International Conference on World Wide Web(WWW '16). 2016: 263-273.

[6] BOURHIS P, REUTTER J, SUÁREZ F, et al. JSON: data model, query languages and schema specification[C]// PODS '17. 2017: 123-135.

[7] WANG L, ZHANG S, SHI J, et al. Schema management for document stores[J]. Proc. VLDB Endow, 2015, 8(9): 922-933.

[8] LI Y, KATSIPOULAKIS N, CHANDRAMOULI B, et al. Mison: a fast JSON parser for data analytics[J]. PVLDB, 2017, 10(10): 1118-1129.

[9] FROZZA A, MELLO R, COSTA F. An Approach for Schema Extraction of JSON and Extended JSON Document Collections [C]// IRI '18. 2018: 356-363.

[10] MEIKE K, UTA S, STEFANIE S. Schema Extraction and Structural Outlier Detection for JSON-based NoSQL Data Stores[C]// BTW '15. 2015: 425-444.

[11] RAIHAN R, MALEEHA N, HAFIZ F, et al. A novel JSON based regular expression language for pattern matching in the internet of things[J]. Journal of Ambient Intelligence and Humanized Computing, 2019, 10: 1463-1481.

[12] HAI R, QUI X C, KENSCHE D. Nested Schema Mappings for Integrating JSON [C]// Conceptual Modeling. ER 2018. 2018, 11157: 397-405.

[13] JAN O, CHRISTOPH L. Semantically Weighted Similarity Analysis for XML-based Content Components[C]// DocEng. 2018, 20: 1-4.

[14] CHEN W, ZHAO X. Similarity-Based Classification for Big Non-Structured and Semi-Structured Recipe Data[C]// Database Systems for Advanced Applications. 2016: 57-64.

[15] BRAY T. The JavaScript Object Notation (JSON) Data Interchange Format[J]. RFC, 2014, 8259: 1-16.

[16] Nigikokun. Generate-schema[OL]. https://github.com/nijikokun/generate-schema.

[17] Julian. Jsonschema[OL]. https://github.com/Julian/jsonschema.

[18] LI S, ZHAO Z, HU R F, et al. Analogical Reasoning on Chinese Morphological and Semantic Relations[C]// ACL. 2018: 138-143.

[19] Fzumstein. Jsondiff: Diff JSON and JSON-like structures in Python[OL]. https://github.com/fzumstein/jsondiff.

[20] Rugleb. JsonCompare: compare two objects with a JSON-like structure and data types[OL]. https://github.com/rugleb/JsonCompare.

LIU Li-cheng, born in 1995, master candidate, is a student member of China Computer Federation. His main research interests include data mining and knowledge engineering.

DUAN Lei, born in 1981, Ph.D, professor, Ph.D supervisor, is a senior member of China Computer Federation. His main research interests include data mining, health-informatics and evolutionary computation.