DynoClass: A Dynamic Table-Class Detection System Without the Need for Predefined Ontologies

Anonymous Author(s)
Affiliation
Address
email

Abstract

Table-class detection plays a crucial role in various data tasks. Traditional approaches typically depend on predefined ontologies such as DBpedia[1], but these are often insufficient for domain-specific or evolving datasets. In response, we present DynoClass, a novel table-class detection system that leverages the power of large language models (LLMs) and eliminates the reliance on external ontologies. DynoClass uses LLMs to generate table classes and descriptions directly from sample data and existing documentation, dynamically constructing hierarchical ontology classes. This approach matches the performance of traditional methods while eliminating the need for predefined ontologies.

1 Introduction

Common data tasks, such as machine learning, often require integrating datasets from multiple sources using unions and joins. One of the key challenges in this process is table-class detection[2], which involves identifying the semantic structure of tables. Tables with similar semantic structures share commonalities in integration; for example, they may have the same join condition in data integration. This property can be leveraged to simplify tasks in data integration, such as schema matching[3, 4], entity resolution[5, 6], dataset search[7], and the creation of dataset catalogs and knowledge bases.

Recent advancements in large language models (LLMs) have significantly expanded their applicability across numerous table-related tasks. These models, trained on vast datasets of natural language, are capable of handling tasks beyond their initial training objectives with minimal fine-tuning. Due to their ability to interpret both structured data and natural language, LLMs are particularly well-suited for handling complex table-related tasks[8, 9, 10], including table-class detection. For instance, Kayali et al. (2023) [11] introduced a novel approach to table-class detection by leveraging predefined ontology classes. They utilize LLMs to analyze sample table data and contextual information, then select the most appropriate class from a subset of DBpedia [1] ontology classes.

However, relying on an external ontology like DBpedia is undesirable in practice because ontologies are often incomplete. This is because domain-specific or enterprise tables will not be represented in an open-domain ontology, or data simply evolves overtime. Additionally, creating and maintaining such ontologies within a domain requires considerable effort. As a result, a table can be misclassified into a table class simply because it's the closest available match. Such misclassifications can significantly increase the effort required to perform downstream tasks efficiently.

For example, Table 1 is a table contains information specifically about electric vehicles (EVs), including their battery capacity, range, and charging time. Due to the absence of an `Electric Vehicle` ontology class, Table 1 is mapped to the `Automobile`[1] class.

[1] `http://dbpedia.org/ontology/Automobile`

Meanwhile, a data analyst studying the average charging efficiency across different EV models would like to search for relevant datasets from a dataset search system based on table-class detection results from DBpedia. However, due to the incompleteness of DBpedia ontology, the system cannot distinguish between EVs and non-EVs within the `Automobile` class, handing the effort of differentiating datasets about EVs to the data analyst.

Brand	Model	Battery Capacity (kWh)	Range (miles)	Charging Time (hours)
Tesla	Model 3	75	353	8.5
Nissan	Leaf	40	149	6.0
Chevrolet	Bolt EV	66	259	9.5
BMW	i3	42	153	7.2

Table 1: Example table for electric vehicle (EV) data

To address these challenges, we propose an approach for table-class detection that leverages LLMs without relying on external ontologies. Our approach uses LLMs to generate a rich description of a table from a sample of the table and any available documentation. The LLM also generates a small ontologies specific to the table. We then scan the tables and merge each table-specific ontology into a global set of hierarchical ontologies. We show that this paradigm can generate ontologies of similar quality to those defined by experts on certain benchmarks.

2 Background

2.1 Problem Definition

Table-class Detection: Given a table T_i, determine its appropriate class C_j, such that each row $r_k \in \{r_1, r_2, \ldots, r_n\}$ represents an instance of the class C_j.

This definition, as presented by Kayali et al. [11], encapsulates the fundamental goal of table class detection: identifying a semantic class that accurately represents the common type embodied by all rows within a given table. For example, consider Table 1. The appropriate class for this table would be *ElectricVehicle*, as each row represents a specific electric vehicle model with attributes commonly associated with electric vehicles, such as battery capacity, range, and charging time.

2.2 Related works

Table representation learning has shown significant potential in table-class detection. Methods like DoDuo [12], TaBERT [13], and TURL [14] convert tables into token sequences and use pre-trained Transformer-based language models (LMs) to encode the serialized data. Notably, TaBERT [13] enables joint understanding of both natural language (NL) and tabular data, allowing for table-class detection based on the generated embeddings.

Chorus [11] proposes using LLMs to directly select the table class from a set of predefined DBpedia [1] ontologies, based on sample data and documentation text.

Shen et al. [15] also use LLMs to improve entity set and taxonomy expansion. Their approach uses two main steps: "find siblings" and "find parents," by creating a fine-tuned training set to figure out where to place new entities. While they focus on keywords (e.g., products), we focus on tables with detailed documentation, which makes better use of the LLMs' ability to understand natural language.

3 Methods

In this work, we propose a novel approach for table-class detection and ontology construction that operates independently of prior knowledge or external ontologies by leveraging the power of LLMs and embedding-based similarity measures. Our method classifies tables into relevant classes, uses LLMs to generate rich descriptions, and iteratively generates nodes that represent the table's class, and inserts them into existing hierarchical ontology classes. Figure 1 illustrates an example workflow for our methods.

3.1 Table Preprocessing

The first step of our algorithm leverages LLMs to generate a comprehensive description of each input table. For each table, we sample k rows, combining this with any available table documentation, and

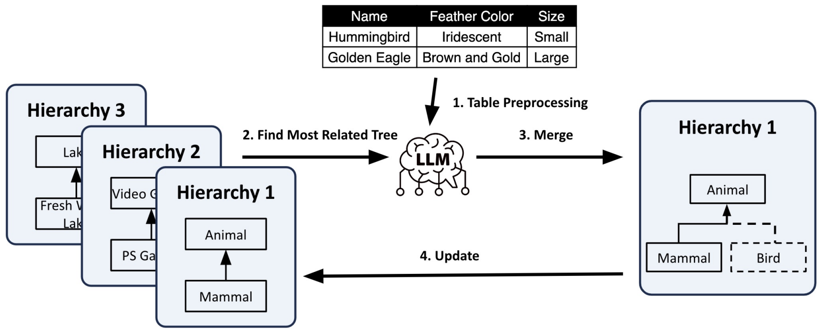

Figure 1: Example Workflow for DynoClass

use LLMs to produce a detailed and context-rich description. This description includes several key elements: the specific entity type represented by the table (e.g., a person, product, or event), possible parent entity types (reflecting hierarchical ontologies), the table's ontology class name (which situates the table within the broader ontology), a brief summary of the table's purpose, the general entity type (to capture higher-level conceptual groupings like object or event), and potential sibling entity types within the same domain (which may indicate related entities under the same or adjacent ontology classes). This initial classification provides a rich, contextual understanding of each table, forming the basis for our hierarchical tree construction.

3.2 Hierarchical Ontology Classes Construction Algorithm

After generating detailed descriptions and initial classifications for each table, we proceed to construct hierarchical ontology classes. This process involves inserting and merging each table-specific ontology into a set of hierarchical ontology classes.

Algorithm 1 Hierarchical Ontology Classes Construction for Table-class Detection

Require: Set $I = \{(R_i, P_i, C_i, S_i, E_i, B_i)\}$ where R_i: Root entity, P_i: Possible parents, C_i: Class, S_i: Description, E_i: Entity type, B_i: Possible siblings

```
Initialize 𝒪 ← ∅, ℛ ← ∅                                        ▷ Processed nodes and root elements
for each x = (R_i, P_i, C_i, S_i, E_i, B_i) ∈ I do
    new_node ← CreateNode(R_i, C_i, E_i, S_i)
    related_tree ← FindMostRelatedTree(R_i, P_i, B_i)           ▷ Use embeddings
    if related_tree = null then
        ℛ ← ℛ ∪ {new_node}; 𝒪 ← 𝒪 ∪ {new_node}
    else
        decision, (parent, child) ← FindPos(new_node, related_tree)   ▷ LLM-based
        if decision = "merge" then
            merged_node ← MergeNodes(parent, child)             ▷ Nodes represent same concept
            𝒪 ← 𝒪 ∪ {merged_node}
        else if decision = "sibling" then
            new_parent ← CreateParent(parent, child)            ▷ Create new sibling for root
            𝒪 ← 𝒪 ∪ {new_parent, parent, child}
        else
            InsertNode(parent, child)                           ▷ Insert node into tree
            𝒪 ← 𝒪 ∪ {parent, child}
        end if
    end if
    AddNodeAndEmbedding(new_node, R_i)                          ▷ Update embeddings
end for
return ℛ
```

The algorithm utilizes several key functions to build and maintain the hierarchical structure:

- **FindMostRelatedTree**(R_i, P_i, B_i): Use embedding-based cosine similarity to identify the top-k related hierarchical ontology classes and let LLMs select the most relevant one from the candidates.
- **FindPos**$(new\_node, related\_tree)$: Utilizes LLMs to determine the optimal position of the new node within the related tree. It returns a decision (merge/sibling/insert) along with the relevant parent-child pair. For large trees, we break them down into individual root-to-leaf paths to ensure the entire path fits within the LLM's context window.
- **MergeNodes**$(parent, child)$: Combines nodes representing the same concept, consolidating information and updating the tree structure.
- **InsertNode**$(parent, child)$: Adds the new node to the appropriate position in the existing tree, as determined by the result of **FindPos**.
- **AddNodeAndEmbedding**(new_node, R_i): Generates and associates an embedding with the newly created or updated node, facilitating future similarity comparisons.

4 Experiments

Data and Model For the dataset, we evaluate the same subset as Kayali et al. [11], consisting of 237 tables from the T2Dv2 dataset, and compare our results against the baselines DoDuo[12], TaBERT[13], and Chorus[11]. For the benchmark models, we follow the same experimental settings for DoDuo and TaBERT as outlined by Kayali et al. We use the Bedrock anthropic.claude-3-5-sonnet-20240620-v1:0 model for both CHORUS and our model, which supports a 200k token context window.

Evaluation Setting For each node in the hierarchical tree of class C_i, classifying any descendant of it as class C_i is *consistent* to the hierarchical tree. Following strategies in Kayali et al. [11], we evaluate all classifications that are *consistent* to the generated hierarchical tree, compute the precision, recall and F1 score with respect to each table class, and report the best weighted average based on sizes of each class.

Evaluation Results As shown in Table 2, our model achieves the highest F1 score of 0.930, outperforming all other baselines, including Chorus, which uses LLMs with predefined ontologies. This demonstrates that leveraging an LLM without relying on a predefined ontology can still achieve very high performance.

	F-1 Score	**Precision**	**Recall**
DoDuo-Viz	0.654	66.8%	68.3%
DoDuo-Wiki	0.757	78.6%	76.9%
TaBERT	0.746	76.3%	76.8%
Chorus	0.922	89.9%	**94.6%**
DynoClass	**0.930**	**93.0%**	91.2%

Table 2: Performance comparison of different models.

Error Analysis One of the main errors occurs when a table labeled as 'nursing school' actually refers to a 'university'. Another frequent error arises when the system struggles to clearly differentiate between 'political party' and 'election'. These issues illustrate cases where a single real-world table can be associated with multiple nodes within the same hierarchical class, or even across different hierarchical ontology trees, which can lead to misclassifications in our current methods.

5 Conclusion

In conclusion, we presented DynoClass, a novel approach to table-class detection that leverages large language models to generate dynamic, hierarchical ontologies without relying on predefined classes. Our method addresses the limitations of traditional approaches while outperforms existing methods on the T2Dv2 dataset. Looking ahead, we plan to enhance the scalability of DynoClass, optimize the system to reduce computational costs, and improve its ability to handle overlapping or ambiguous classifications within hierarchical ontology trees.

References

[1] Sören Auer et al. "DBpedia: a nucleus for a web of open data". In: *Proceedings of the 6th International The Semantic Web and 2nd Asian Conference on Asian Semantic Web Conference*. ISWC'07/ASWC'07. Busan, Korea: Springer-Verlag, 2007, pp. 722–735. ISBN: 3540762973.

[2] Michael J Cafarella et al. "Webtables: exploring the power of tables on the web". In: *Proceedings of the VLDB Endowment* 1.1 (2008), pp. 538–549.

[3] Philip A. Bernstein, Jayant Madhavan, and Erhard Rahm. "Generic schema matching, ten years later". In: *Proc. VLDB Endow.* 4.11 (Aug. 2011), pp. 695–701. ISSN: 2150-8097. DOI: `10.14778/3402707.3402710`. URL: `https://doi.org/10.14778/3402707.3402710`.

[4] Roee Shraga, Avigdor Gal, and Haggai Roitman. "ADnEV: cross-domain schema matching using deep similarity matrix adjustment and evaluation". In: *Proc. VLDB Endow.* 13.9 (May 2020), pp. 1401–1415. ISSN: 2150-8097. DOI: `10.14778/3397230.3397237`. URL: `https://doi.org/10.14778/3397230.3397237`.

[5] Alexandros Zeakis et al. "Pre-Trained Embeddings for Entity Resolution: An Experimental Analysis". In: *Proc. VLDB Endow.* 16.9 (May 2023), pp. 2225–2238. ISSN: 2150-8097. DOI: `10.14778/3598581.3598594`. URL: `https://doi.org/10.14778/3598581.3598594`.

[6] Giovanni Simonini et al. "Entity resolution on-demand". In: *Proc. VLDB Endow.* 15.7 (Mar. 2022), pp. 1506–1518. ISSN: 2150-8097. DOI: `10.14778/3523210.3523226`. URL: `https://doi.org/10.14778/3523210.3523226`.

[7] Sonia Castelo et al. "Auctus: a dataset search engine for data discovery and augmentation". In: *Proc. VLDB Endow.* 14.12 (July 2021), pp. 2791–2794. ISSN: 2150-8097. DOI: `10.14778/3476311.3476346`. URL: `https://doi.org/10.14778/3476311.3476346`.

[8] Ralph Peeters and Christian Bizer. "Entity matching using large language models". In: *arXiv preprint arXiv:2310.11244* (2023).

[9] Zezhou Huang et al. "The Fast and the Private: Task-based Dataset Search". In: *arXiv preprint arXiv:2308.05637* (2023).

[10] Zezhou Huang and Eugene Wu. "Cocoon: Semantic Table Profiling Using Large Language Models". In: *Proceedings of the 2024 Workshop on Human-In-the-Loop Data Analytics*. 2024, pp. 1–7.

[11] Moe Kayali et al. "CHORUS: foundation models for unified data discovery and exploration". In: *arXiv preprint arXiv:2306.09610* (2023).

[12] Yoshihiko Suhara et al. "Annotating Columns with Pre-trained Language Models". In: *Proceedings of the 2022 International Conference on Management of Data*. Association for Computing Machinery, 2022. ISBN: 9781450392495. URL: `https://doi.org/10.1145/3514221.3517906`.

[13] Pengcheng Yin et al. "TaBERT: Pretraining for joint understanding of textual and tabular data". In: *arXiv preprint arXiv:2005.08314* (2020).

[14] Xiang Deng et al. "Turl: Table understanding through representation learning". In: *ACM SIGMOD Record* 51.1 (2022), pp. 33–40.

[15] Yanzhen Shen et al. "A Unified Taxonomy-Guided Instruction Tuning Framework for Entity Set Expansion and Taxonomy Expansion". In: *arXiv preprint arXiv:2402.13405* (2024).