

Annotating Column Type Utilizing BERT and Knowledge Graph Over Wikipedia Categories and Lists

Jiixin QIN[†] and Mizuho IWAIHARA[‡]

[†] [‡] Graduate School of Information, Production and Systems, Waseda University 2-7 Hibikino, Wakamatsu-ku, Kitakyushu-shi, Fukuoka, 808-0135 Japan

[†] School of Mechatronic Engineering and Automation, Shanghai University 333 Nanchen, Dachang, Baoshan, Shanghai, China

E-mail: [†]jiaxinqin@fuji.waseda.jp, [‡]iwaihara@waseda.jp

Abstract Automatically annotating semantic type of table column task plays a vital role in the process of information retrieval and NLP tasks. In this paper, given an entity column of a table without a header, we study the problem of predicting its column type using both fine-tuning on pre-trained BERT model and knowledge graph lookup, integrating the two methods to complement each other's shortcomings. A data augmentation method utilizing similar entities in Wikipedia categories and lists is proposed to fine-tune the BERT model for small datasets. To uncover the semantics of Wikipedia categories and lists, a root phrase-based method is proposed as a key in data augmentation. We evaluate our approach using two small datasets (T2Dv2 and Limaye) and we demonstrate substantial improvements in terms of evaluation metrics over baseline models.

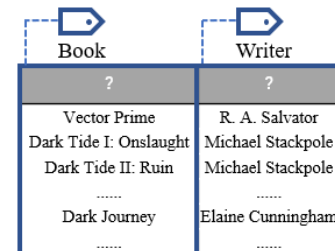
Keyword Column semantics, Knowledge Graph, Tabular data

1. Introduction

Tabular data constitute the main form of datasets in the web and enterprises: web tables, spreadsheets, CSV files and database relations, which represents a typical type of structured information and a highly valuable knowledge resource. Annotating semantic types of table columns refers to the task of identifying the real-world concepts that capture the semantics of the data, which plays a vital role in a number of applications [11][12][25]. For instance, dataset discovery automatically determines whether two columns (or even entire tables) are joinable or not, which requires accurate identification of column semantic types [12]. Figure 1 depicts semantic type annotation for columns with missing column types of a given table.

In this paper, we study the problem of predicting semantic type of a table column without header. With the aim of addressing the limitations of prior techniques, we make the following contributions: (1) We propose a knowledge graph (KG) lookup strategy based on CaLiGraph, a novel knowledge graph extracting entities from Wikipedia categories and lists, to narrow knowledge gap. Its KG ontology class hierarchy is fully utilized to predicting the most specific class(es); (2) A data augmentation strategy utilizing co-occurrence entities under Wikipedia categories and lists is proposed for finetuning a BERT multi-class classification model to deal with data deficiency and fit it

on small datasets. To uncover the semantics of Wikipedia categories and lists, a root phrase-based method is proposed as a key in data augmentation; (3) We integrate the two methods, KG lookup and finetuning of BERT, to complement each other's shortcomings.



Book	Writer
?	?
Vector Prime	R. A. Salvator
Dark Tide I: Onslaught	Michael Stackpole
Dark Tide II: Ruin	Michael Stackpole
.....
Dark Journey	Elaine Cunningham
.....

Figure 1 Example of annotating semantic type of columns

2. Related work

2.1 Knowledge graph lookup-based methods

Typically, the knowledge graph (KG) lookup approach [6] [13][14] [22] for matching tabular data to a KG, is a highly efficient way for inference. Nguyen et al. [14] presented MTab based on an aggregation of multiple cross-lingual lookup services and probabilistic graphical models. Limaye et al. [13] use a probabilistic graphical model to represent different matchings and collectively determine annotations. These techniques have been observed to be sensitive to noisy values, not capturing semantically similar values, which are successfully captured by recent word embedding-based techniques. Knowledge gap is

another inevitable problem, such that KG resources are missing due to the inconsistencies between KG updates and emergence of new entities.

2.2 Neural Language Models-based methods

Based on the type of data used for training and type of concepts identified, there are two main approaches for column type prediction: (1) KG-based methods [3][6][18] use external KGs on top of machine learning models to improve column type prediction, which are sensitive to noisy values and knowledge gap, and do not capture semantically similar values. Chen et al. [3] present ColNet, which integrates KG lookup and a CNN-based approach for classification. It constructs synthetic column samples by looking up the cell values on DBpedia. Word2Vec embeddings of these samples are used to train a CNN to build context among different cells of the column. (2) Data-based approaches [4][10][17][20][26][27] treat semantic annotation task as a multi-class classification problem training classifiers on large datasets without any external KG. Recently, pre-trained transformer-based language Models (LMs) such as BERT, which were originally designed for NLP tasks, have shown the power of contextualized representations in table semantic annotation task [4][17][20]. These transformer-based models serialize the columns or entire table into a sequence of tokens and obtain table context through contextualized column embeddings using the transformer-architecture. The table information was fully learned by such deep contextualized language models. Since KG ontologies are not referred in their systems, the type concepts depend on labeled dataset which are not defined by standardized format. Moreover, they rely on a large gold standardized labeled training dataset, which means manual data labeling increases the cost. A given small dataset makes the model unstable [27], and long-tail types have few samples. Type hierarchy is another challenge. For overlapping types like ‘Person’ and ‘Monarch’, the most specific type ‘Monarch’ is harder to be predicted than general type ‘Person,’ due to imbalanced training samples.

2.3 Data augmentation

Data Augmentation (DA) refers to the approach used to increase the amount of data by adding slightly modified copies of already existing data or newly created synthetic data from existing data [16]. Chen et al. [3] proposed a KG-based synthetic sampling augmentation strategy including two major types of samples: General samples from all KG instances under candidate classes for CNN classifiers training and particular samples from matched KG instances

for finetuning the neural network.

3. Background

3.1 Basic assumptions

In this paper, we assume that (i) each cell in the target entity column contains entity names, and numeric data are not within the discussion scope; (ii) the semantic type of cells in the same column should be consistent and there exist at least one common semantic type for a column; (iii) table titles and column headers are not available. To be more specific, we do not utilize textual descriptions that can be used to infer the column type; (iv) relations between different columns in a table are ignored, so we do not rely on the relations modeled in the knowledge graph or other columns as target column’s context. A column has only one best type. Here, “best type” refers to the most specific type rather than general types.

3.2 Problem formulation

Table 1 lists our notations. Our goal task is column type prediction, i.e., to classify each column to its only one best *semantic type*. The best semantic type of a target in column set column is from a subset of all DBpedia ontology classes in English language.

Table 1 Notation

Symbol	Description
$\mathbf{K} = (E, P, C)$	Knowledge graph A knowledge graph
$\mathbf{E} = \{e_1, e_2, \dots, e_w\}$	A collection of all resources in a knowledge graph
$\mathbf{P} = \{p_1, p_2, \dots, p_x\}$	A collection of all properties in a knowledge graph
$\mathbf{C} = \{c_1, c_2, \dots, c_y\}$	A collection of all classes in a knowledge graph
$\mathbf{L} = \{l_1, l_2, \dots, l_z\}$	A collection of resource labels in a knowledge graph
$\{(e, \text{rdfs:label}, l) e \in \mathbf{E}, l \in \mathbf{L}\} \in \mathbf{K}$	A collection of resource-label pairs in a knowledge graph
$\{(e, \text{rdf:type}, c) e \in \mathbf{E}, c \in \mathbf{C}\} \in \mathbf{K}$	A collection of resource-class pairs in a knowledge graph
$\mathbf{T} = \{T_1, T_2, \dots, T_n\}$	Table A set of tables
$\mathbf{I} = \{I_1, I_2, \dots, I_n\}$	A set of entity columns
$I_i = \{v'_1, v'_2, \dots, v'_n\}$	Cell entity set in Target Column
$\mathbf{C}^* \subset \mathbf{C}$	Candidate class set of a given column set

PROBLEM (COLUMN TYPE PREDICTION). Given an entity column I_i in a set of entity columns \mathbf{I} and a vocabulary \mathbf{C}^* of column types for \mathbf{I} , the column type prediction problem is to determine a column type $M(I_i, c_i) \in \mathbf{C}^*$ that best describes the semantics of c_i .

3.3 External resources

A knowledge graph (KG) is comprised of a large set of assertions about the world to reflect how humans organize information [2]. DBpedia is a common knowledge graph that is derived by automatically extracting structured data from Wikipedia infoboxes [1]. Figure 2 shows the DBpedia ontology class hierarchy extracted by SPARQL querying all parent-child classes relationships. Based on DBpedia and Wikipedia, CaLiGraph [7][8][9] is a large semantic knowledge graph extracted from Wikipedia categories and lists. Figure 3 shows CaLiGraph ontology schema. Table 2

shows that CaLiGraph contains much more classes and instances than DBpedia, that is helpful for narrowing the knowledge gap between tabular entity and KGs. Table 3 shows Main URI namespace used in DBpedia and CaLiGraph.

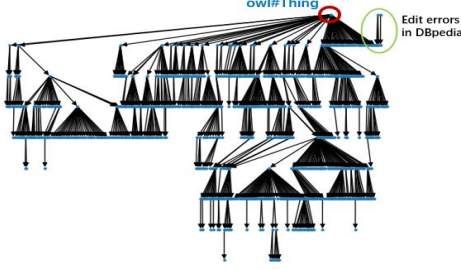


Figure 2 DBpedia ontology class hierarchy. Almost all of DBpedia ontology classes starting from the top node “owl:Thing”. There are 4 nodes (‘dbo:Infrastructure’, ‘dbo:ElectricalSubstation’, ‘dbo:TimeInterval’ and ‘dbo:Holiday’) whose ancestor node is not “owl:Thing”, which are edit errors caused by misspelling and cycle.

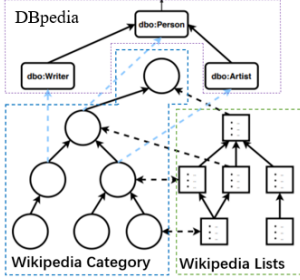


Figure 3 CaLiGraph ontology schema. Source: [13]
Table 2 Statistic of DBpedia and CaLiGraph

	DBpedia	CaLiGraph
Classes	781	1,061,598
Instances	4,828,418	14,452,393

Table 3 Main URI namespace used in Wikipedia, DBpedia and CaLiGraph

Prefix	URI	Description
DBpedia namespace		
dbp	http://dbpedia.org/resource/	One-to-one mapping between Wikipedia articles and DBpedia resources
dbp	http://dbpedia.org/property/	Properties from raw infobox extraction
dbo	http://dbpedia.org/ontology/	DBpedia Ontology
CaLiGraph namespace		
clgr	http://caligraph.org/resource/	One-to-one mapping between Wikipedia articles and CaLiGraph resources
clgo	http://caligraph.org/ontology/	CaLiGraph Ontology
External namespaces		
owl	http://www.w3.org/2002/07/owl#	W3C web ontology language
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#	Standard W3C RDF vocabulary
rdfs	http://www.w3.org/2000/01/rdf-schema#	Extension of the basic RDF vocabulary

3.4 Pre-trained BERT Model

Transfer learning [19] mechanism and transformer [21] architecture are creating the latest real boost in NLP tasks, in which BERT [5] is one of the most successful models. Context-aware word embedding based on the self-attention mechanism is a special component of pre-trained transformer-based models. Different from other word embedding methods like word2vec [15] or FastText [28], a word is embedded into a numeric vector based on its context and the same word has different vectors if it

appears in different sentences. Such a contextualized embedding method can discern polysemy and deal with synonyms well [17]. As shown in Figure 4, we treat column annotation task as a multi-class classification problem, serializing entities in a target column as a sequence input and utilize the BERT model with an added single linear layer on top as a sentence classifier to predict the target column type.

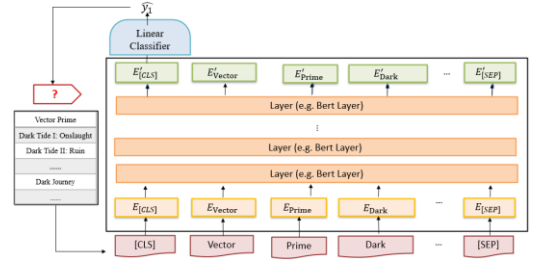


Figure 4 BERT model for column type prediction.

4. Methodology

Our framework utilizes CaLiGraph and the pretrained BERT model to train a multi-class classification model for annotation types of target entity columns. As shown in Figure 5, it mainly includes three steps: *KG lookup*, *model training* and *prediction*.

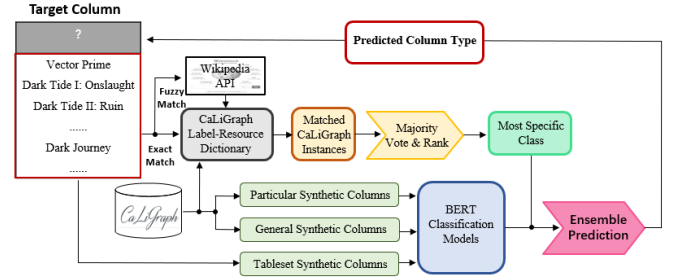


Figure 5 Proposed Framework

4.1 KG lookup

For a table cell entity, there are two functions of KG lookup: (i) retrieving all possible classes including ambiguous classes due to entity ambiguity; (ii) retrieving all other entities grouped together in a same KG class.

Cell entity class lookup. The process is divided into three steps: (1) Matching each cell entity to a certain KG label. Firstly, a given cell entity is tried to exactly match with a KG label from a collection of all KG labels. If it fails, fuzzy match is processed by the search and suggestion function of Wikipedia API [29], in which we can utilize the configuration that KG labels and KG instances in CaLiGraph are identically named with corresponding Wikipedia articles. (2) Retrieving all KG instances with the matched KG label. The following SPARQL query retrieves all instances with a given label l . Even though entity disambiguation has been done in KGs, which KG instance corresponds to the entity in the table has not yet been

determined. We will use majority voting and model training to tackle it.

```
SELECT DISTINCT ?r
WHERE {
  {?r rdfs:label ?l}
  UNION
  {?r skos:altLabel ?l}
}
```

(c) Retrieving all KG classes for each retrieved KG instance. The following SPARQL query retrieves all parent (including ancestor) classes of a given instance r . Then we filter out DBpedia classes among all retrieved classes according to URI namespace shown in Table 3. Finally, all possible classes of a cell entity are successful retrieved.

```
SELECT DISTINCT ?c
WHERE {
  r rdf:type ?e.
  ?e rdfs:subClassOf* ?c
}
```

KG class entity extraction. To construct augmented columns for model training, we are interested in two kinds of KG instances:

(1) General KG instances, which are under the candidate classes for a given table dataset. For example, if a given table dataset has a candidate class ‘Person’, then all KG instances under ‘Person’ class belong to general KG instances with training label ‘Person’. Given a set of candidate classes C^* for target table dataset, we extract all general KG instances of each candidate classes.

(2) Particular KG instances, which are under the same Wikipedia categories and lists (i.e., under the same CaLiGraph class) with each KG instance of a given cell entity. As shown in Figure 6, the cell entity ‘Dark Journey’ has multiple corresponding CaLiGraph instances; a matched CaLiGraph instance belong to multiple groups (CaLiGraph classes); and a group contains multiple entities. Grouping by classes would generate overlapping groups. The more specific CaLiGraph class is, the closer similar entities distribution under the class are.

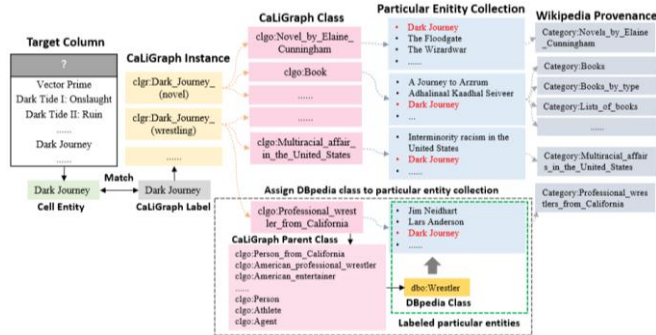


Figure 6 Illustration of particular KG instances extraction utilizing CaLiGraph class. The goal is to extract particular KG instances related to ‘Dark Journey’.

To fit for supervised training, assigning a DBpedia class to each group is necessary. Our proposed root-phrase method (Section 4.3) requires each group’s CaLiGraph class provenance and its CaLiGraph parent classes to infer a

DBpedia class. Given a cell entity, we extract particular entities under each corresponding CaLiGraph class.

4.2 KG lookup majority voting and ranking

A widely used method for column type annotation is KG lookup majority voting. We make the following contributions: (1) Majority voting and class specific degree are both considered; (2) Specific class selecting algorithm is proposed to select the most specific DBpedia class(es) among a list of classes.

Majority voting. Given a target entity column set with its candidate class set C^* , utilizing cell entity class lookup method (section 4.1) for each cell entity v_j^i of a target entity column $I_i = \{v_1^i, v_2^i, \dots, v_m^i\}$, all possible KG classes $C_j^i = \{c_{j1}^i, c_{j2}^i, \dots, c_{jo}^i\}$ of v_j^i is retrieved; then the retrieved KG classes set of target column I_i is $\{C_1^i, C_2^i, \dots, C_m^i\}$. The candidate class set of I_i is $(C_1^i \cup C_2^i \cup \dots \cup C_m^i) \cap C^*$, denoted as $C_{candidate}^i$. Each entity v in I_i votes a class from C^* by 1 if the class is present in the class set associated with v . Then each class’s voting score is simply equal to the number of votes divided by the number of rows in the column and multiply by 100, indicating the percentage of supporting cell entities among all cells.

Ranking and selecting the top class(es). For candidate class ranking, there are two factors that need to be considered: (i) Frequency of occurrence. The voting score of a candidate class c indicates the confidence of c . If the voting score of a candidate class is higher than a high threshold θ (e.g., 90), it implies very high confidence and requires to be selected. (ii) Specific degree. When the same voting score occurs in the $C_{candidate}^i$, we want to pick more specific classes (e.g., Actor) over general classes (e.g., Person) among these classes with the same voting score based on algorithm 1.

Algorithm 1: Specific class(es) selection algorithm

Input: $DBO_{candidate} = \{dbo_1, dbo_2, \dots, dbo_n\}$,

$G \leftarrow$ Direct graph of DBpedia ontology class hierarchy*

Output: $DBO_{most\ specific} = \{dbo_{s1}, dbo_{s2}, \dots, dbo_{sm}\}$

Process:

- 1: $H \leftarrow$ Subgraph by mapping $DBO_{candidate}$ to G
- 2: $Roots, Leaves, DBO_{most\ specific} \leftarrow \emptyset$
- 3: For node in H :
- 4: If zero edge pointing node:
- 5: $Roots \leftarrow Roots \cup node$
- 6: If zero edge pointing out of node:
- 7: $Leaves \leftarrow Leaves \cup node$
- 8: For root in $Roots$:
- 9: For leaf in $Leaves$:
- 10: $Paths \leftarrow \emptyset$
- 11: $Paths \leftarrow$ all simple paths in G from root to leaf
- 12: For path $Paths$:
- 13: If path not \emptyset :
- 14: $DBO_{most\ specific} \leftarrow DBO_{most\ specific} \cup leaf$
- 15: Return $DBO_{most\ specific}$

* As shown in Figure 2 (a), the extraction of DBpedia ontology class hierarchy is a preliminary work.

We first rank candidate classes $C_{candidate}^i$ of target column I_i by voting score from high to low. If there are classes with voting score higher than θ , we select the most specific class(es) among them. If not, we select the most specific class(es) of the top one score class(es).

4.3 Model training

Assign DBpedia class to particular entity collection. Due to the result (Table 4) of evaluating exactkt matching DBpedia ontology class to CaLiGraph ontology class by randomly sampled 2000 is not ideal, we propose a supplementary method to it.

Table 4 CaLiGraph Ontology Class Matching Evaluation

Situation	Number	Ratio
Correct	978	48.90%
Wrong	Not specific	187
	Others	22
Missing	813	40.65%

For a given CaLiGraph class name, head word [28] (i.e., *root word*) is statistically predicted by a dependency parser (Figure 7). In this paper, dependency parser of spaCy transformer-based pre-trained pipeline is utilized to parse the grammatical structure and relationships between root and modifiers. The root of a sentence is a verb (POS: VERB), while the root of a noun phrase is supposed to be a noun (POS: NOUN). A judgment statement is added to prevent the dependency parser from mistaking the noun phrase for a sentence (e.g., “Publication established in 1806”). Then a set of pseudo compound-root phrases defined as root with modifier n -words before the root treated as bag of words. For example, the root of the CaLiGraph class named “20th-century American male actor” is predicted as “actor”; then the modifier words before “actor”, which are “20th-century”, “American” and “male”, are treated as bag of words; then all of modifier word permutations with root are generated as pseudo compound-root phrases, i.e., “20th-century actor”, “American actor”, “male actor”, “20th-century American actor”, “20th-century male actor”, “American male actor” and “20th-century American male actor”. We use target CaLiGraph class name, its root and pseudo compound-root phrases, and its parent CaLiGraph classes to assign a DBpedia class.

Exact match is a high confident method given the top priority. Firstly, the target CaLiGraph class is exactly matched with DBpedia classes, like ‘clgo:Person’ is matched with ‘dbo:Person’. If the matching fails, then its parent CaLiGraph classes are exactly matched with DBpedia classes and select the most specific one by Algorithm 1. If it fails, then we use oracle matching rules provided manually. For example, when matching ‘clgo:YEAR_birth’ to ‘dbo:Person’ failed, then its pseudo

compound-root phrases and its root are exactly matched with DBpedia classes and the most specific one is selected by Algorithm 1. If it fails, then we use oracle matching rules (Table 5) for the root constructed manually. For example, when matching those CaLiGraph classes with root ‘Alumni’ to ‘dbo:Person’. Essentially, our oracle matching rules are intended to complement incompleteness of our current matching algorithm. We also further introduce a semantic matching method for enhancement. However, our current semantic similarity matching methods may sometimes reduce the accuracy, requiring further research in future.

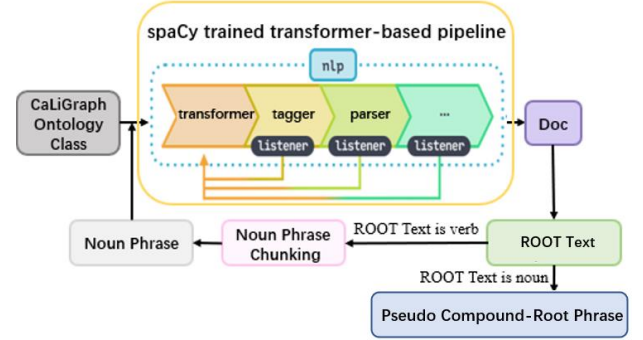


Figure 7 Pipeline of pseudo compound-root text prediction

Table 5 Part of oracle matching rules used in this work

No.	Matched text	dbo
1	Vehicle	MeanOfTransportation
2	Publication	WrittenWork
3	Program	Work
4	Amusement ride	AmusementParkAttraction
5	Walt Disney Parks and Resorts attraction	AmusementParkAttraction
6	Tourist attraction	Place
7	Baseball pitcher	BaseballPlayer
8	Creative work	Work
9	Office-holder	OfficeHolder
10	Song recording	Song
...

Synthetic Columns. Inspired by ColNet [3], we construct synthetic columns as a data augmentation strategy for model training and validation by concatenating h number of entities. A training/validation sample is a synthetic column I_s with its standard golden class c . There are three kinds of synthetic columns: (i) General synthetic columns. A general synthetic column concatenates h general KG instances (Section 4.1, KG class entity extraction); (ii) Particular synthetic columns. A particular synthetic column concatenates h (or less) of particular KG instances from the same CaLiGraph class and related to given column sets; (iii) table dataset synthetic columns. They concatenate h given target column entities. The order of entities in a synthetic column is randomly shuffled. Different from ColNet, we utilize Wikipedia categories and lists entities (i.e., CaLiGraph class entities) to augment data which are more

relevant for the model to learn word co-occurrence features. **Column serialization and BERT Fine-Tuning.** Before we finetune on a pretrained BERT model, we are supposed to satisfy a few formatting requirements about BERT input [5], which are: (1) Serialization and tokenization. A column is required to be converted into a sequence, which can be easily realized by concatenating column values. The tokenization of the sequence is performed by a proper BERT tokenizer. Each column in our task is treated as a long sentence for BERT. (2) Special tokens. At the beginning and end of each column sequence, the special tokens [CLS] and [SEP] are required to be appended, respectively. (3) Sentence length. All sentences must be padded or truncated to a single, fixed length. The sentence cannot exceed a length of 512 tokens. The serialized sequence of a column I with column values $\{v_1, v_2, \dots, v_m\}$ is encoded in to the sequence:

$$[\text{CLS}] v_1 v_2 \dots v_m [\text{SEP}]$$

The category of column type prediction by a BERT model is a sequence classification task. We use the normal BERT model with an added single linear layer on top as a sentence classifier. As we feed input our above-mentioned synthetic columns, the entire pre-trained BERT model and the additional untrained classification layer are trained on our column type annotation task.

4.4 Ensemble prediction

To improve performance of column type prediction, we use a customized rule that can utilize the advantage of both KG lookup-majority voting and BERT classifier (Algorithm 2).

Algorithm 2: Ensemble algorithm	
Input: $DBO_{KG \text{ Lookup}} = \{dbo_1, dbo_2, \dots, dbo_m\}$, dbo_{BERT} , $G \leftarrow$ Direct graph of DBpedia ontology class hierarchy*	
Output: $dbo_{Ensemble}$	
Process:	
1:	$dbo_{Ensemble} \leftarrow \emptyset$
2:	For dbo_i in $DBO_{KG \text{ Lookup}}$:
3:	$Paths \leftarrow \emptyset$
4:	$Paths \leftarrow$ all simple paths in G from dbo_{BERT} to dbo_i
5:	If $Paths$ not \emptyset :
6:	$dbo_{Ensemble} \leftarrow dbo_i$
7:	If $dbo_{Ensemble}$ is \emptyset :
8:	$dbo_{Ensemble} \leftarrow dbo_{BERT}$
9:	Return $dbo_{Ensemble}$

The KG lookup-majority voting method can predict the most frequent and special class(es) as the top. But when classes with the same voting score and being sibling relationships in the class hierarchy, more than one class would be the top classes. For example, ‘Book’ and ‘Film’ are tied for the top when they have the same voting score

and they are siblings in the hierarchy, and there is no more evidence of which one can be removed, although we assume there is only one best class. The BERT classifier predicts only one best class but it does not consider the class hierarchy. A general class like ‘Person’ might be predicted even though the more specific one ‘Monarch’ is the ground truth. If the top one predicted class by the BERT classifier has its child in the top predicted class(es) of KG lookup-majority voting, that means the prediction of BERT classifier is not the most specific one. Therefore, we choose the corresponding child in the top result of KG lookup-majority voting as the final prediction. For the rest of cases, we prefer the prediction of the BERT classifier as the final prediction.

5. Evaluations

5.1 Datasets

We evaluate our proposed method on two web table datasets T2Dv2 and Limaye with a golden standard DBpedia class released by Chen et al. [3]. The statistics of T2Dv2 and Limaye are shown in Table 6. The knowledge gap means that the cell in the table cannot be associated with the corresponding entity in the knowledge base or with the wrong entity. Structuredness represents the proportion of empty grids in the table. It is obvious that knowledge gap in Limaye is larger and it includes more empty cells, which increases the difficulty of prediction compared with T2Dv2 benchmark. Both two datasets contain no more than 500 columns.

Table 6 Some statistics of two table sets.

Name	T2Dv2	Limaye
Table source	Web table	Wikipedia table
Columns	411	428
Avg.cell	124	23
Classes	29	18
Knowledge gap (DBpedia)	13.5%	34%
Knowledge gap (CaLiGraph)	7%	24%
Structuredness	0.03	0.48

5.2 Baselines

We compare our model with DBpedia Lookup-Majority Voting and T2K Match [30] whose authors developed T2Dv2, Efthymiou17-Vote [31] whose authors developed Limaye, and ColNet [3] whose authors further extended T2Dv2 and Limaye.

5.3 Experimental Settings

For BERT model training, synthetic columns size h is set to 5, i.e., each synthetic column concatenates 5 KG instances by random sampling without replacement. The number of general synthetic columns for each class is 50,000. The statistics of classes of particular instance collection is shown in Table 7. Table 8 shows the statistics of the synthetic columns for Limaye and T2Dv2. We set our learning rate is 1e-05, max sequence length is 85, epoch is

Table 7 Statistics of class of particular instance collection

Dataset	No. CaLiGraph class	No. CaLiGraph class with assigned DBpedia class	Ratio of assigned DBpedia class
Limaye	18,318	15,136	82.63%
T2Dv2	240,673	184,258	76.56%

Table 8 Statistics of synthetic columns for Limaye and T2Dv2

Dataset	No. general synthetic column	No. particular synthetic column	No. table set synthetic column
Limaye	900,000	796,567	2862
T2Dv2	1,450,000	761,786	7378

5.4 Experimental Results

Table 9 shows our results on the Limaye dataset. By comparing model No.1 and No.6, CaLiGraph shows its ability of novel entity retrieval better than DBpedia. By comparing model No.2, No.3, No.7 and No.8, even though we only use general synthetic columns or particular synthetic columns for training, BERT has shown better than the CNN-based ColNet model. Compared with model No.9, the result of No.10 is enhanced significantly because table set synthetic columns are more specific than general and particular synthetic columns. The ensemble prediction (model No.11) is the best among other models, which shows that the ensemble model takes the advantages of KG lookup and BERT models.

Table 9 Results (precision, recall, F1 score) on Limaye

No.	Model	Micro-P	Micro-R	Micro-F1
1	DBpedia Lookup-Majority Voting	0.571	0.447	0.501
2	ColNet	0.576	0.619	0.597
3	ColNet _{Ensemble}	0.602	0.639	0.620
4	T2K Match	0.453	0.330	0.382
5	Efthymiou17-Vote	0.626	0.357	0.454
6	CaLiGraph Lookup-Majority Voting, with oracle rules	0.920	0.980	0.950
7	BERT _{general synthetic columns}	0.738	0.738	0.738
8	BERT _{particular synthetic columns}	0.750	0.750	0.750
9	BERT _{general and particular synthetic columns}	0.833	0.833	0.833
10	BERT _{general and particular and table set synthetic columns}	0.952	0.952	0.952
11	Ensemble 6,10	0.980	0.980	0.980

Table 10 shows our results on the T2Dv2 dataset. Compared with model No.1, the result of model No.6 is worse because the entity in T2Dv2 is common and CaLiGraph gives more noisy redundant classes. Also, compared with model No.7, the result of model No.8 infers general synthetic columns have little effect on the prediction of the TDv2 dataset due to very common entities in T2Dv2. By comparing model No.2, No.6 and No.7, BERT shows its strengths on column type prediction task better than CNN. The ensemble prediction (model No.10)

also shows best than other models.

Table 10 Results (precision, recall, F1 score) on T2Dv2

No.	Model	Micro-P	Micro-R	Micro-F1
1	DBpedia Lookup-Majority Voting	0.862	0.821	0.841
2	ColNet	0.765	0.811	0.787
3	ColNet _{Ensemble}	0.853	0.846	0.849
4	T2K Match	0.624	0.727	0.671
5	CaLiGraph Lookup-Majority Voting, with oracle rules	0.850	0.720	0.780
6	BERT _{general synthetic columns}	0.790	0.790	0.790
7	BERT _{particular synthetic columns}	0.840	0.840	0.840
8	BERT _{general and particular synthetic columns}	0.840	0.840	0.840
9	BERT _{general and particular and table set synthetic columns}	0.930	0.930	0.930
10	Ensemble of 5,9	0.950	0.950	0.950

6. Conclusion

This paper presents a column type prediction model based on BERT and knowledge graph over Wikipedia categories and lists. Different from existing methods, it (i) utilizes CaLiGraph as an external source for table cell class lookup and data augmentation, (ii) trains a multi-class classifier based on the pretrained BERT model with a set of synthetic columns for small datasets, and (iii) takes class hierarchy into consideration when KG lookup is processed and combined with the BERT classifier, achieving better results.

In the future, there are still much work need to be done. First, semantic similarity method for assigning DBpedia classes onto CaLiGraph classes should be finished to replace the manually constructed oracle matching rules. Second, hierarchical multi-label classification should be considered in the column type prediction task, because there are more than one class for a column in the real world and there also exists hierarchal relationships between classes. Third, table-level input is supposed to be considered including the column relationship in the same column, the attribute context of a cell entity, etc.

References

- [1] S. Auer, C. Bizer and G. Kobilarov, etc. Dbpedia: A nucleus for a web of open data. The semantic web. pp.722-735. 2007.
- [2] K. Balog. Entity-Oriented Search. The Information Retrieval Series, vol. 39. 2018.
- [3] J.Y. Chen, E. Jiménez-Ruiz, and I. Horrocks, etc. Colnet: Embedding the semantics of web tables for column type prediction. In AAAI, vol. 33. pp. 29–36. 2019.
- [4] X. Deng, H. Sun, and A. Lees, etc. Turl: Table

- understanding through representation learning. arXiv:2006.14806. 2020.
- [5] J. Devlin, M.-W. Chang and K. Lee, etc. BERT: Pre-training of deep bidirectional transformers for language understanding. In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, vol. 1, pp. 4171–4186. 2019.
 - [6] V. Efthymiou, O. Hassanzadeh, and M. Rodriguez-Muro etc. Matching web tables with knowledge base entities: from entity lookups to entity embeddings. In International Semantic Web Conference, pp. 260–277. Springer. 2017.
 - [7] N. Heist, and P. Heiko. Entity extraction from Wikipedia list pages. European Semantic Web Conference. Springer, Cham, 2020.
 - [8] N. Heist, and P. Heiko. Uncovering the semantics of Wikipedia categories. International semantic web conference. Springer, Cham, 2019.
 - [9] N. Heist, and P. Heiko. Information extraction from co-occurring similar entities. Proceedings of the Web Conference 2021. 2021.
 - [10] M. Hulsebos, K. Hu, and M. Bakker, etc. Sherlock: A Deep Learning Approach to Semantic Data Type Detection. ACM SIGKDD. 2019.
 - [11] U. Khurana and S. Galhotra. Semantic Concept Annotation for Tabular Data. Proceedings of the 30th ACM International Conference on Information & Knowledge Management. pp. 844–853. 2021.
 - [12] C. Koutras, G. Siachamis and A. Ionescu, etc. Valentine: Evaluating Matching Techniques for Dataset Discovery. 2021 IEEE 37th International Conference on Data Engineering (ICDE). IEEE, 2021: 468–479.
 - [13] G. Limaye, S. Sarawagi and S. Chakrabarti. Annotating and searching web tables using entities, types and relationships. Proceedings of the VLDB Endowment, vol. 3, pp. 1338–1347, 2010.
 - [14] P. Nguyen, etc. MTab: matching tabular data to knowledge graph using probability models. arXiv preprint arXiv:1910.00246.2019.
 - [15] X. Rong. word2vec parameter learning explained. arXiv:1411.2738, 2014.
 - [16] C. Shorten, T. M. Khoshgoftaar and B. Furht. Text data augmentation for deep learning. Journal of big Data, vol. 8, pp. 1–34. 2021.
 - [17] Y. Suhara, J. Li and Y. Li, etc. Annotating Columns with Pre-trained Language Models. arXiv:2104.01785, 2021.
 - [18] K. Takeoka, M. Oyamada and S. Nakadai, etc. Meimei: An Efficient Probabilistic Approach for Semantically Annotating Tables. Proceedings of the AAAI Conference on Artificial Intelligence 33, vol. 1, pp. 281–288. 2019.
 - [19] C. Tan, F. Sun and T. Kong, etc. A survey on deep transfer learning. International conference on artificial neural networks. pp. 270–279. 2018:
 - [20] M. Trabelsi, J. Cao and J. Heflin. SeLaB: Semantic Labeling with BERT. 2021 International Joint Conference on Neural Networks (IJCNN). pp. 1–8. 2021.
 - [21] A. Vaswani, N. Shazeer and N. Parmar, etc. Attention is all you need. Advances in neural information processing systems. pp. 5998–6008. 2017.
 - [22] P. Venetis. A. Halevy, and J. Madhavan, etc. Recovering semantics of tables on the web. Proceedings of the VLDB Endowment, vol. 4, pp. 528–538. 2011.
 - [23] Z. Wang, H. Dong and R. Jia, etc. TUTA: Tree-based Transformers for Generally Structured Table Pre-training. Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining. pp. 1780–1790. 2021.
 - [24] E. Williams. On the notions “Lexically related” and “Head of a word” Linguistic inquiry, vol.12, pp.245–274. 1981.
 - [25] Y. Xian, H. Zhao and T. Y. Lee, etc. EXACTA: Explainable Column Annotation. Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining. pp. 3775–3785. 2021.
 - [26] D. Zhang, Y. Suhara, and J. Li, etc. Sato: Contextual Semantic Type Detection in Tables. VLDB. arXiv:1911.06311. 2020.
 - [27] T. Zhang, F. Wu and A. Katiyar. Revisiting Few-sample BERT Fine-tuning. arXiv: 2006.05987. 2020.
 - [28] <http://fasttext.cc/>
 - [29] <http://pypi.org/project/wikipedia/>