DEIM Forum 2023 1b-9-1

Fine-grained Column Type Annotation Using Multiple Knowledge Graphs

Shen SHUYANG† and Mizuho IWAIHARA‡

† ‡Graduate School of Information, Production and Systems, Waseda University
2–7 Hibikino, Wakamatsu-ku, Kitakyushu, Fukuoka 808–0135 Japan
E-mail: †shua@akane.waseda.jp, ‡iwaihara@waseda.jp

Abstract Tabular data contain rich semantic information. However, identifying and interpreting tables are still challenging. Particularly, we look into the column type annotation task as part of table interpretation. Although deep-learning models based on pretrained language models have been achieving state of the art performance in recent years, these column typing results are still not successful in predicting specific types, because of the limitation on re-lying only on pretrained language models, which can be improved by incorporating multiple knowledge graphs with extended ontology. In this paper, we propose a hybrid method which combines the advantage of deep-learning and linguistic feature approaches over knowledge graphs. We utilize large-scale knowledge graph CaLiGraph over Wikipedia lists and categories to create a fine-grained mapping to the target knowledge graph of DBPedia. Our results were evaluated on the fine-grained subset of WikiTable dataset, and shows advantages on granularity over the state-of-art model.

Key words Column Type Annotation, Deep-learning, Knowledge Graph, Fine-grain

1 Introduction

Web tables (relational HTML table from the Web) are a kind of tabular data source which can be accessed over the Internet. Twelve years ago, Caferella et al. [4] estimated that the Web contains around 14.1 billion HTML tables. It is convincing that an abundant source of tabular knowledge is currently available on the Semantic Web. To fully understand tabular data, table interpretation solutions with LOD (Linked Open Data) are necessary.

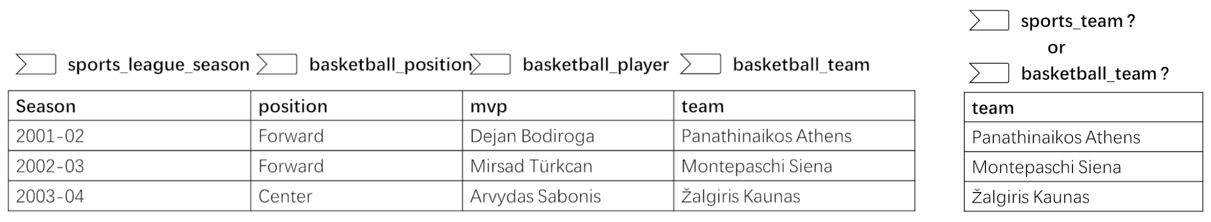
Column type annotation [1] [7] [5] [23], as an important task of table understanding, is the process of identifying and annotating the data type of each column in the given tabular dataset. It is important in data processing because correctly identifying the data type of each column is crucial for many downstream tasks, such as triplet extraction and database update [22]. If a column is mistakenly annotated with another type, any statistical analysis or machine learning models that are applied to the data may produce incorrect or biased results. Therefore, accurately identifying the data types of columns is an important step in preparing and cleaning data for further processing.

LOD such as DBpedia [15], YAGO [13] creates an ontology by defining a series of categories, properties, and relations between the concepts, data, and entities to establish a cross-domain knowledge graph, which is used as background knowledge inventory for a wide range of applications, including web search, natural language understanding, data integration, and data mining. It is also requested to be as complete, correct, and up-to-date as possible to finish a mapping from the external contents to the concepts or entities.

The English version of DBpedia [15] describes 6.2M entities using 1.1B triples, including 1.6M persons,800K places, 480K works (e.g., films, music albums), 267K organizations, 293K species, and 5K diseases. Besides, the DBpedia Ontology is a shallow, cross-domain ontology, which has been manually created based on the most commonly used infoboxes within Wikipedia. The ontology currently covers 685 classes which form a subsumption hierarchy and are described by 2,795 different properties, which maintains hierarchical structure at the semantic level, and has been widely used in a variety of tasks and fields.

CaLiGraph [10] [9] [11] is a knowledge graph that uses the encyclopedic structure of Wikipedia to derive its fine-grained type ontology and extract huge amounts of novel entities. Additionally, it provides restrictions for many of the derived types to further refine the contained entities. The ontology currently covers 1,061,598 types, which offer way too much detailed information than DBpedia [15], acting as a reliable external knowledge graph other than DBpedia.

Column type annotation can be done manually, but it is often time-consuming and error-prone. As a result, automated



sports_league_season	basketball_position	basketball_player	basketball_team
Season	position	mvp	team
2001-02	Forward	Dejan Bodiroga	Panathinaikos Athens
2002-03	Forward	Mirsad Türkcan	Montepaschi Siena
2003-04	Center	Arvydas Sabonis	Žalgiris Kaunas

(a)

sports_team ?
or
basketball_team ?

team
Panathinaikos Athens
Montepaschi Siena
Žalgiris Kaunas

(b)

Figure 1: Two examples of vertical tables annotated semantic types above each column. (a) is a vertical table with multiple columns, and (b) is one single column with two different types annotation.

approaches have been developed to accurately and efficiently annotate column types. These approaches can range from simple rule-based systems to more complex machine learning based methods. Rule-based systems [1] [17] [12] [21] [24] often utilize information from various knowledge bases and make the final ranking and predicting part through ensemble of features. On the other hand, machine learning based methods [28] [5] [25] [7] [23] focus on extracting semantics with limited table data (like only the surface contents of tables given). Besides, most datasets [7] [29] [6] for column type annotation have overlap among the given types (like *Director* and *Artist*).

We focus on the WikiTable dataset [7], which is a collection of tables collected from Wikipedia. According to [7], the type and relation labels are from FreeBase [2] and are obtained by aggregating entity links attached to the original tables. However, FreeBase has been shut down years ago, and the API would not be supported anymore, which makes it difficult to access the metadata with FreeBase schema to obtain fine-grained typing results. To this end, we propose to utilize multiple knowledge graphs like DBpedia [15] and CaLiGraph [10] [9] [11] to harvest fine-grained types.

This paper attempts to show a hybrid model which is designed to exploit the advantages of deep-learning and knowledge graphs to perform column type annotation on tabular data. As shown in Figure 1, there is one column in table (b) that has values identical to table (a), and we consider the task to assign the column type of (b) as fine-grained as possible to capture the specific semantics of the target table. It is difficult to improve granularity of column and obtain fine-grained semantic types in (a). However, the entities in cells of each column can bring extended information from corresponding knowledge graphs, thus leading to more fine-grained column type annotation in (b) (in this case, basketball_team is more specific and thus appropriate than sports_team). The main idea is to retrieve fine-grained types from multiple knowledge graphs by utilizing deep representations of the contents of tables. We choose the backbone of DODUO [23], which proposes a shared parameter encoder, and extend the deep-learning part by a hybrid model and improve its performance with a contrastive learning approach.It should be noted that our approach is agnostic to pretrained language models, so different pre-trained language models can be utilized in the deep learing part and knowledge graph part for fine-tuning and embedding generation, respectively. Our main contributions can be summarized below:

1. Improved the deep-learning part [23] by adding a contrastive learning module whose training samples are generated according to the results on confident samples in each epoch, which can separate types having overlapping semantics with other types.
2. Use an unsupervised method on embeddings over type texts and interpretable linguistic features to accomplish mapping between ontologies of multiple knowledge graphs. Combine the advantage of deep-learning and knowledge graphs, building a more interpretable, robust, and fine-grained method to cope with the challenge of column type annotation.
3. Create a fine-grained WikiTables dataset. Since TURL WikiTables adopts both fine-grained and coarse-grained types, which is a multi-label dataset. To further evaluate the perform on column type annotation, we manually curate part of WikiTable dataset and choose one most specific type out of the given multiple types as fine-grained WikiTables dataset. Evaluation is performed on both TURL WikiTables and fine-grained WikiTables dataset.

The remainder of this paper is organized as follows. In Section 2 we introduce the related work we investigated. The problem is formulated in Section 3 and the method proposed to annotate column type is specifically explained in Section 4. Then, we evaluate our hybrid model with a subset of TURL WikiTables in Section 5 and make a conclusion in Section 6.

2 RELATED WORK

Column type annotation is a kind of named-entity recognition, which is a subtask of table interpretation that aims to locate or categorize named entities in tabular data into pre-defined concepts. While column type annotation focuses on annotating the header of the table, entity linking is aiming to link all the mentions in columns (except the headers) of a table to or are instantiated from ground-truth concepts in a knowledge graph.

Approaches based on Markov random field model [17] [1] [14] are first introduced into semantic table interpretation by Limaye et al. [17]. In this method, Iterative Classification Algorithm (ICA) [8] and knowledge graphs are used with latent variables and some manually constructed potential functions based on the structural features of tables. These potentials could connect the components in the tables, jointly constructing a maximal objective function to describe the table interpretation problem. Such methods based on the Markov random field model were improved by a method proposed by C.S. Bhagavatula et al. [1] mainly by introducing more potential functions and more prior knowledge. Accordingly, the improved method which exhausts the background knowledge is computationally expensive to realize better performance. Let alone the memory limit for storing that amount of knowledge. Embedding was later introduced by Takeoka et al. [25] to exploit the semantic information with the texts and terms, which addressed the issue of computational costs by translating the knowledge in knowledge graph into vector representation instead of searching through the knowledge.

Compared with the table interpretation method based on Markov random field, which disambiguates all background knowledge on the knowledge graph at once, the methods based on traditional feature engineering and queries are more robust and applicable. Among them, the most important feature engineering and query methods are the candidate entity search that relies on a database, the context modeling of table contents, and extension of information sources [3]. G. Hignette et al. [12] interprets tables through feature construction based on similarity, scoring function and database search. Several researches [21] [24] [26] emphasize the influence of database background knowledge on table interpretation, trying to interpret tables by exploiting the ontology and search methods of LOD. Table augmentation method is used in [12], [16] to improve the quality of table interpretation for these small web tables. Several researches have improved the processing flow [30], [31], using a part of the contents of a table as a supplementary explanation to the rest, making full use of the characteristics of the disordered structure of tables.

There are also works focused on the improvement of word embedding methods [18] [5] [7] [23], and a number of novel word embedding processes have been proposed, and state-of-art results have been achieved in the evaluation of Chen et al.'s work [5]. Among these methods, the word embedding model based on Word2Vec [19] is the most commonly used. But recent work [7] [23] are adopting transformer [27] based models.TURL [7] is a transformer-based pretraining framework for table interpretation. Pre-training contextualized representations of table contents are learned in an unsupervised way during pre-training and later applied to different tasks during finetuning phase. Moreover, Suhara et al. [23] propose shared parameter model DODUO, which allows different training tasks to fine-tune the same encoder together, capturing various levels of semantics and generalizing the model. This framework also allows different datasets to be applied to fine-tune the same pre-trained language model,extending the range of knowledge.

3 PROBLEM FORMULATION

This section gives mathematical formulation of our task. Assuming a target table gives column names and cell values that can be linked to its metadata, but without other information such as the table title, caption and contexts. With this restriction, the goal of the proposed task is to predict semantic types of the table columns and make them as fine-grained as possible. Here the target semantic types are chosen from the DBPedia ontology. With the construction schema of the WikiTable dataset, the problem is therefore formulated as a classification task, annotating one of the predefined semantic types on the target column.

Assume that a table T which contains r rows of content cells (excluding the row of table headers) and c columns are to be interpreted. Symbol m_{ij} is called a mention, which denotes the cell content of row i and; column j (suppose one cell contains only one mention). T_i is the i^{th} row of table T, T_j is the j^{th} column of table T. DBpedia knowledge graph (KG) in this paper can be described by $(\mathcal{E}, \mathcal{T}, \mathcal{P}, \mathcal{F})$, where $\mathcal{E} = \{e_1, ..., e_{|\mathcal{E}|}\}$ is a set that consists of all the entities in the KG. $\mathcal{T} = \{t_1, ..., t_{|\mathcal{T}|}\}$is the set of entity types in the KG, and the types are connected with the relations to form the ontology of KG. $\mathcal{P} = \{p_1, ..., p_{|\mathcal{P}|}\}$is the set of properties (which is used to describe the entity). $\mathcal{F} = \{f_1, ..., f_{|\mathcal{F}|}\}$ contains all the facts where each fact is denoted by a unique RDF triplet in the KG (includes the triplets defined by ontology). For every table T in the dataset, column type annotation is to assign a type $\mathcal{M}(T, T_j) \in \mathcal{T}$ to each column in T.

4 METHODS

In this section, we first introduce the deep-learning part of our hybrid model, describing in detail how contrastive learning with hard samples are achieved. Then, the features generated by multiple knowledge graphs are listed individually, followed by the feature combination method.

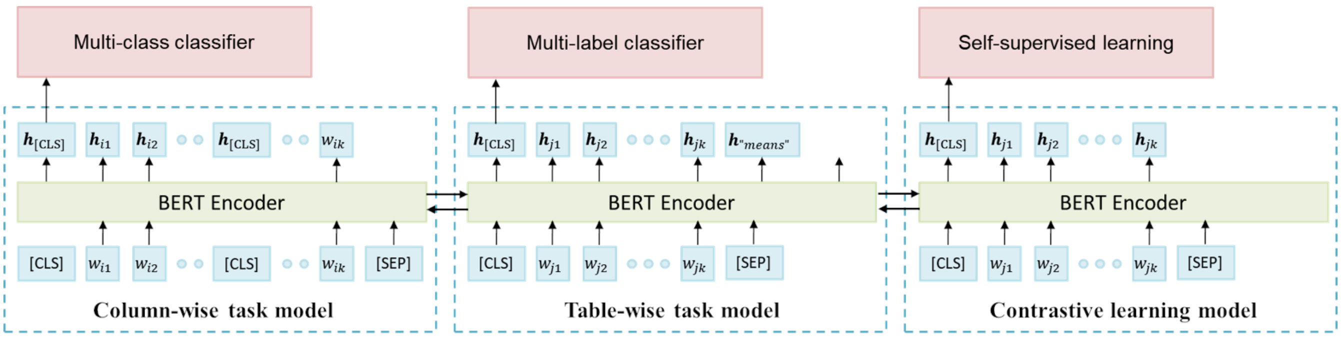

Multi-class classifier
Multi-label classifier
Self-supervised learning
$h_{[CLS]}$ h_{i1} h_{i2} $h_{[CLS]}$ w_{ik}
BERT Encoder
[CLS] w_{i1} w_{i2} [CLS] w_{ik} [SEP]
Column-wise task model
$h_{[CLS]}$ h_{j1} h_{j2} h_{jk} $h_{\text{"means"}}$
[CLS] w_{j1} w_{j2} w_{jk} [SEP]
Table-wise task model
$h_{[CLS]}$ h_{j1} h_{j2} h_{jk}
Contrastive learning model

Figure 2: This figure shows how the deep-learning part o our model is trained using the same BERT encoder layers. Every training task would contribute to the fine-tuning of the same encoder. Weight of every task could be contralled by adjusting number of training data.

4.1 Deep-learning Prediction

The training part of our model is shown in Figure 2, where the first two tasks represent the tasks described in Suhara et al.'s [23] work. The last task is contrastive learning task we add to improve the generality of model. It should be noted that contrastive learning is one of the training progress as is shown in the figure instead of fine-tuning before training. Training methods and contrastive learning methods will be introduced specifically.

4.1.1 Model Training

As mentioned before, the backbone of the deep-learning part of our model follows the idea proposed by Suhara et al. [23], where the column serialization is implemented in a very simple way by concatenating the content in the cell of corresponding column in order. That is, suppose a column T_j has values T_{1j}, T_{2j},..., T_{mj}, the column could be serialized to the sequence as follows

$$Sequence_{col}(T_j) ::= [CLS]\ m_{1j}\ ...\ m_{rj}\ [SEP] \tag{1}$$

Where [CLS] (classification) and [SEP] (separation) are special tokens that are used to represent the beginning and end of a sentence, respectively. They are used to mark the boundary between individual sentences or document segments in the input text. For example, the column in Figure 1(b) would be serialized to $[CLS]$ *panathinaikos athens montepaschi siena Zalgiris kaunas* $[SEP]$. In this way, the problem is converted into a sequence classification task. Using training data to fine-tune the BERT model is therefore straightforward.

When it comes to table-wise serialization, the idea is almost th same. Take the entire table T as the input, column sequences are concatenated in order, constructing the table sequence

$$\begin{aligned} Sequence_{table}(T) ::= [CLS]\ m_{11}\ ...\ m_{r1}\ [CLS] \\ ...\ [SEP]\ m_{1c}\ ...\ m_{rc}\ [SEP] \end{aligned} \tag{2}$$

In this case, the column sequences are connected with $[CLS]$ instead of $[SEP]$ to calculate an embedding vector for every classification token based on surrounding tokens, which allows the table-wise semantics to be involved in each $[CLS]$ token of corresponding column. For the first two tasks in Figure 2, table sequence input is encoded to corresponding embedding $\boldsymbol{h}$. The embedding which represents $[CLS]$ tokens in the sequence is extracted for type classification.

Let LM donates the pre-trained language model of the encoder while $|C_{type}|$ is the number of types in the dataset. Let $\boldsymbol{g}$ be the dense layer of dimension $d \times |C_{type}|$ for the column type prediction. The deep-learning model would compute

$$softmax(LM(Sequence_{table}(T)) \cdot g) \tag{3}$$

Since the Wikitable dataset is a multi-label dataset, binary cross-entropy loss is applied instead of cross entropy loss to properly calculate the loss.

4.1.2 Contrastive learning

Contrastive learning [20] is a machine learning technique that involves training a model to recognize the difference between two or more input samples. In contrastive learning, the model is presented with two or more input samples and is trained to maximize the difference between the representations of these samples in the learned feature space.

In the module we propose, every batch for contrastive learning is composed of one target column, one positive column, and several negative columns. They are all serialized to a corresponding column sequence and sent into the encoder to obtain the embedding $\boldsymbol{h}$. The embedding of the target embedding $\boldsymbol{h}$ and the positive embedding $\boldsymbol{h}^+$ should be pulled together. Also, $\boldsymbol{h}$ and negative column embeddings $\boldsymbol{h}^-$ should be pushed apart. Accordingly, the loss function of contrastive learning loss is

$$L_{cl} = -\log \frac{e^{\frac{cos(h,h^{+})}{\tau}}}{\sum_{i=1}^{N^{-}} e^{\frac{cos(h,h_i^{-})}{\tau}}} \quad (4)$$

where N^{-} is the number of negative samples.

The generation of negative samples is performed by analyzing confusion matrix which is produced in each epoch over the validation dataset. Confusion matrix compares the predicted class labels of the model with the true class labels to determine the number of true positives, false positives, false negatives, and true negatives. The precision of each type in the confusion matrix could vary differently and some types could reach a precision of 1. Those types should not be considered as the types to be optimized. Let the threshold factor be u. Every type whose precision is smaller than u will be selected as the target type, And the column with corresponding ground truth target type will be collected as target columns. Negative samples are also generated according to row where target types are in the confusion matrix. Let u^{-} be the threshold for negative sample selection. The false positive types whose ratio in the row exceeds u^{-} will be collected as the negative samples for the target type.

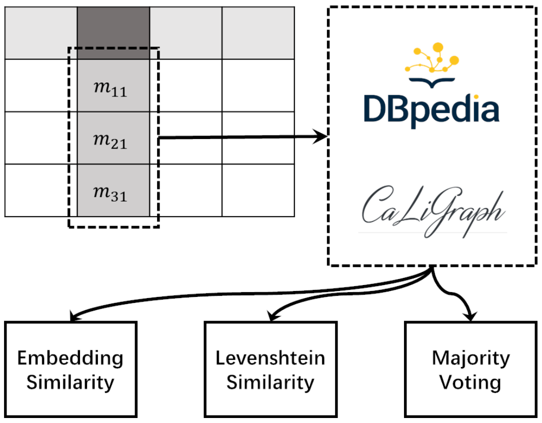


Figure 3: The features of the surfaces are extracted in different tasks. Weight of every task could be contralled by adjusting number of training data. m_{11}, m_{21} and m_{31} are mentions in table T

In TURL WikiTables, for example, one column could have two ground truth types *sport.athlete* and *basketball.basketball_player*. If *sport.athlete* is classified to *basketball.basketball_player*, the positive sample is the column with type *sport.athlete*, and the negative sample should include type *basketball.basketball_player*. Mutual inclusion case will not be further processed because the batch is constructed randomly and the quantity of TURL WikiTables is large enough to avoid the duplicates. Besides, since the training is accomplished on TURL WikiTables, where the misclassifications mostly exist between non-overlapped types, the semantically overlapped type pair will not be taken into consideration.

Contrastive learning dataset is organized with respect to the proportion of corresponding types in the validation dataset. For example, there are two target types *Artist* and *PLayer*, where *Artist* accounts for 0.1 in the validation set while *Player* accounts for 0.4 in the validation set. Let N be the total number of contrastive learning samples. Then the number of samples in a batch whose target is *Artist* will be $0.2N$, and the number of samples in a batch whose target is *Player* will be $0.8N$.

4.2 Knowledge Graph Prediction

We utilize Mmultiple knowledge graphs for finding fine-grained column types from predicted types by the deep-learning model. Acting as an alternative to FreeBase which is not under maintenance anymore, which makes the type identificatin very hard. And such problem can be converted to a mapping or alignment task to other trusted knowledge graphs, which exists ubiquitously in many fields. In our model, we try to create a mapping from ontologies of DBpedia and CaLiGraph to the ontology of FreeBase using ranking result based on linguistic features, which calculates the confidence of mapping between each type pair. The features we select to perform the mapping and the feature combination methods will be shown.

4.2.1 Entity Linking

As is shown in Figure 3, we find meta URL e (entity page) of every mention in the column according to the hints in the dataset. After a SPARQL query over these entities is executed on DBpedia and CaLiGraph, statistics is collected on these queried types, recording their count for the following feature computation.

Below are the main SPARQL queries used in our model

```
SELECT DISTINCT str(?mtype) as ?mtype
WHERE {
        entity rdf:type ?mtype.
}
FILTER (strstarts(str(?mtype),
        'http://caligraph.org/ontology/'))
```

```
SELECT DISTINCT str(?mtype) as ?mtype
WHERE {
        entity rdf:type ?mtype.
}
FILTER (strstarts(str(?mtype),
        'http://dbpedia.org/ontology/'))
```

4.2.2 Feature Selection

In order to pass the information from ontologies of CaLiGraph and DBpedia to FreeBase, several features need to be designed to achieve the mapping. Since DBpedia ontology well maintains semantic hierarchy and relationship among entities in the KB, types in this ontology form a tree structure according to the thickness of the granularity. This means that relying on a pure voting mechanism can not select the optimal column type. This is because certain types are superior in numbers, but their coverage is too wide to describe the common characteristics of the content in a column, thus several features need to be constructed manually to deliver information. As Figure 3 shows, we select three features to capture the information of a knowledge graph: embedding similarity, levenshtein similarity, and majority voting.

Similarity between strings: Given two plain texts s_1 and s_2, we define the string similarity $SSim(s_1, s_2)$ as follows:

$$SSim(s_1, s_2) = 1 - \frac{LevenshteinDistance(s_1, s_2))}{sum\left\{length(s_1), length(s_2)\right\}} \tag{5}$$

It should be noted that the Levenshtein distance calculation method here uses the calculation method in python-levenshtein(注1), which treats the "replace" edit operation differently than the other operations (i.e. with a cost of 2). We think such change is more reasonable. For example, the *StringSimilarity* between "abcd" and "dcba" should be 0.25 instead of 0 (if the traditional Levenshtein Distance method is adopted). Which could retain the relevance of certain texts when the string order changes。

Levenshtein Similarity: In our model, tasks are accomplished using multiple knowledge graphs. Most of the types in DBPedia have standard format labels which describe the general information. We can easily access the type of each entity on local dump and SPARQL endpoint. However, in FreeBase, the type is organized by the category (in type *tume.event*, time is the category and event is the type), which makes it hard to perform similarity function on the type phrase. Hence we define the similarity between type phrases as follows:

$$LSim(s, \boldsymbol{a}) = \max_{a_i \in \boldsymbol{a}} SSim(s, a_i) \tag{6}$$

where $\boldsymbol{a} = \{a_1, a_2, ..., a_n\}$ is the preprocessed type phrase with the removal of stop words and $n = |\boldsymbol{a}|$ is the length of preprocessed type phrase. After removing the stop words, we tokenize the type phrase to retain the main information of it. In this case, the similarity between phrases can avoid the interference of invalid information as much as possible. If the number of terms in s is greater than one, the results will be averaged.

(注1)：https://github.com/ztane/python-Levenshtein

For every type collected by querying DBpedia and CaLiGraph, levenshtein similarity is performed on all the FreeBase types provided by the WikiTable dataset. Let the output be $\boldsymbol{ls}$, which is of dimension $|\boldsymbol{t^h}|$ by $|C_{type}|$, where $|\boldsymbol{t^h}|$ is total number of queried type.

Embedding Similarity: :

In our proposed method, the fine-tuned deep-learning model is used to predict column types. At the same time, we also use other plain pre-trained language models to directly produce the embeddings of types and compared with all the types in FreeBase using cosine similarity to generate confidence.

$$cosSim(\boldsymbol{h_1}, \boldsymbol{h_2}) = \frac{\boldsymbol{h_1} \cdot \boldsymbol{h_2}}{||\boldsymbol{h_1}|| \cdot ||\boldsymbol{h_2}||} \tag{7}$$

For every type collected by querying DBpedia and CaLiGraph, embedding similarity is performed on all the FreeBase types provided by the WikiTable dataset. Let the output be $\boldsymbol{es}$, which is of dimension $|\boldsymbol{t^h}|$ by $|C_{type}|$.

4. 2. 3 Feature Combination

The ontology of CaLiGraph is rich but complicated, which indicates that types with few occurences should be ignored. Compared to CaLigraph, DBPedia has fewer samples, but the ontology is more organized. Most of the types in DBpedia have standard format labels which describe their general information. Therefore, a reasonable weighting strategy is required.

Let the count statistics vector produced in *Entity Linking* be $\boldsymbol{c}$. The final confidence is shown as follows

$$softmax((\boldsymbol{es} + \boldsymbol{ls}) \cdot \frac{\boldsymbol{c}}{|\boldsymbol{c}|}) \tag{8}$$

Where $|\boldsymbol{c}|$ is the total count of queried types. The application of softmax could neglect the type with little count.

4. 3 Column Type Annotation

Assuming the outputs of deep-learning model and knowledge graph method are $\boldsymbol{D}$ and; $\boldsymbol{K}$ respectively (of dimension $|C_{type}|$). The final ranking function is

$$argmax_{idx \in 1,...,|\boldsymbol{D}|}((\alpha \boldsymbol{D} + \beta \boldsymbol{K})_{idx}) \tag{9}$$

where $\alpha, \beta \in [0, 1]$ with $\alpha + \beta = 1$ are the weight parameters. The type with the highest score will be selected as the output result. Since the range of these scores is between 0 and 1, it can also be regarded as the confidence of corresponding type as the column type.

	DODUO				TURL				Proposed method (dl only)			
	R	P	Micro F1	Macro F1	R	P	Micro F1	Macro F1	R	P	Micro F1	Macro F1
TURL WikiTables	92.69	**92.21**	92.45	-	90.54	87.23	88.86	-	**93.12**	90.57	**92.51**	-
Fine-grained WikiTables w/o threshold	22.07	22.07	20.89	26.81	20.05	20.05	13.89	18.97	**23.11**	**23.11**	**23.11**	**28.59**
Fine-grained WikiTables w threshold	**38.90**	87.63	**54.15**	**77.01**	33.43	**91.93**	49.03	-	38.82	86.96	53.68	74.74

Table 1: Performance on WikiTable Dataset with only deep-learning module of the proposed method.

5 EVALUATION

5.1 Dataset

As the benchmark dataset used in the evaluation, TURL WikiTable [7] is a collection of tables collected from Wikipedia, which consists of 580,171 tables in total. The dataset provides both annotated column types and relations for training and evaluation. For column type prediction, the dataset provides 628,254 columns from 397,098 tables annotated by 255 column types. For column relations, the dataset provides 62,954 column pairs annotated with 121 relation types from 52,943 tables for training. According to [7], the type and relation labels are from FreeBase [2] and are obtained by aggregating entity links attached to the original tables. For both tasks, we used the same train/valid/test splits as TURL [7]. Each column/column-pair allows to have more than one annotation, and thus, the task is a multi-label classification task.

During the curation on TURL WikiTables dataset, we check the header text, table name and section title in order. For example, given two types *film.actor* and *tv.tv_actor* and the table name is *american television star*, then*tv.tv_actor* will be chosen as the fine-grained type. Our fine-grained WikiTables dataset includes 617 tables containing 1350 columns. Our fine-grained WikiTables dataset it's only used as test dataset in evaluation.

5.2 Baselines

TURL [7] is a Transformer-based language model (LM), in which a pre-trained LM is further pre-trained using table data, making it suitable for tabular data.TURL proposes a novel approach for table understanding, which involves learning a low-dimensional representation of tabular data using deep neural networks.

DODUO [23] proposes a method for automatically annotating tabular data columns with semantic types using pre-trained LMs. The proposed method leverages the contextual informgtiation captured by encoders which share parameters among different tasks to improve the accuracy of column annotation, even when dealing with complex and noisy datasets.

5.3 Experimental Setting

We use the bert-base-uncased as default encoder, and AdamW optimizer with learning rate as 5e-5, and train epochs set to be 20, input max length as 32, and batch size as 16. For the combination part α is set to 0.8, and β is correspondingly 0.2. For every training epoch in the shared parameter model, $\frac{|\mathcal{T}|}{10}$ samples is offered by the contrastive learning step.

5.4 Main Result

	full	
	Micro F1	Macro F1
DODDUO	20.89	28.61
TURL	13.89	18.97
Proposed Method	**41.19**	**46.17**

Table 2: Performance on fine-grained WikiTable Dataset using top 1 prediction.

In Table 1, with and without threshold indicates different ways to deal with the confidence generated by each model. When the evaluation is performed without the threshold, only the type with the highest confidence will be selected as the result. When we evaluate with the threshold, for any type whose confidence is greater than a certain threshold will be select as the result, which makes the task multi-output task. As result showed in Table 1, with the contrastive learning module being added to the shared parameter model, recall increased and procession decreased. That might be caused by the additional module improving the generality of the model. The fine-grained WikiTables without threshold means we only take the top one type predicted by the model as the result type. the macro f1 score is remarkably improved. But if we use the evaluation with threshold which is designed for multi-label output, Doduo is better, which takes top K type as the predicted types where K equals to the number of types whose logit is greater than a specific threshold. In the experiment the threshold is set to $\log 0.5$.

As shown in Table 2, After integrating the knowledge graph module, performance for fine-grained WikiTables dataset without threshold on micro F1 and macro F1 score both extensively outperform DODUO, which means plain surface texts provided by knowledge graphs can show noise brought by embeddings generated by the pre-trained language model can be alleviated.

6 Conclusion

In this paper, we proposed a hybrid model that makes use of deep-learning and knowledge graph-based approaches for annotating table columns. The key contributions we introduce include 1) Used an unsupervised method on embeddings over type texts and interpretable linguistic features to accomplish mapping between ontologies of multiple knowledge graphs; 2) combining the advantage of deep-learning and knowledge graphs, to build a more interpretable, robust and fine-grained typing results to cope with the challenge of column type annotation.

In the future, we will pay more attention to the performances on macro F1 score of the model and finish the ablation experiments for knowledge graph module. Since the semantic hierarchy and relationship are well maintained in the ontology of DBPedia (which has a limited number of words and terms), the granularity of the result in task might also be improved by adding more interpretable features like the depth of DBPedia ontology, making the results more fine-grained. With embeddings, we can achieve better accuracy when it comes to correlations at the semantic level, which makes contrastive learning worth a try in the future. Furthermore, we will try to develop a real-time system based on the model proposed in this paper.

References

[1] Chandra Sekhar Bhagavatula, Thanapon Noraset, and Doug Downey. Tabel: Entity linking in web tables. In *International Semantic Web Conference*, pp. 425–441. Springer, 2015.

[2] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pp. 1247–1250, 2008.

[3] Michael J Cafarella, Alon Halevy, and Nodira Khoussainova. Data integration for the relational web. *Proceedings of the VLDB Endowment*, Vol. 2, No. 1, pp. 1090–1101, 2009.

[4] Michael J Cafarella, Alon Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. Webtables: exploring the power of tables on the web. *Proceedings of the VLDB Endowment*, Vol. 1, No. 1, pp. 538–549, 2008.

[5] Jiaoyan Chen, Ernesto Jiménez-Ruiz, Ian Horrocks, and Charles Sutton. Colnet: Embedding the semantics of web tables for column type prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33, pp. 29–36, 2019.

[6] Vincenzo Cutrona, Federico Bianchi, Ernesto Jiménez-Ruiz, and Matteo Palmonari. Tough tables: Carefully evaluating entity linking for tabular data. In *International Semantic Web Conference*, pp. 328–343. Springer, 2020.

[7] Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. Turl: Table understanding through representation learning. *arXiv preprint arXiv:2006.14806*, 2020.

[8] Lise Getoor. Link-based classification. In *Advanced methods for knowledge discovery from complex data*, pp. 189–207. Springer, 2005.

[9] Nicolas Heist and Heiko Paulheim. Uncovering the semantics of wikipedia categories. In *International semantic web conference*, pp. 219–236. Springer, 2019.

[10] Nicolas Heist and Heiko Paulheim. Entity extraction from wikipedia list pages. In *European Semantic Web Conference*, pp. 327–342. Springer, 2020.

[11] Nicolas Heist and Heiko Paulheim. Information extraction from co-occurring similar entities. In *Proceedings of the Web Conference 2021*, pp. 3999–4009, 2021.

[12] Gaëlle Hignette, Patrice Buche, Juliette Dibie-Barthélemy, and Ollivier Haemmerlé. Fuzzy annotation of web data tables driven by a domain ontology. In *European Semantic Web Conference*, pp. 638–653. Springer, 2009.

[13] Johannes Hoffart, Fabian M Suchanek, Klaus Berberich, and Gerhard Weikum. Yago2: A spatially and temporally enhanced knowledge base from wikipedia. *Artificial Intelligence*, Vol. 194, pp. 28–61, 2013.

[14] Yusra Ibrahim, Mirek Riedewald, and Gerhard Weikum. Making sense of entities and quantities in web tables. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pp. 1703–1712, 2016.

[15] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, Sören Auer, et al. Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic web*, Vol. 6, No. 2, pp. 167–195, 2015.

[16] Oliver Lehmberg and Christian Bizer. Stitching web tables for improving matching quality. *Proceedings of the VLDB Endowment*, Vol. 10, No. 11, pp. 1502–1513, 2017.

[17] Girija Limaye, Sunita Sarawagi, and Soumen Chakrabarti. Annotating and searching web tables using entities, types and relationships. *Proceedings of the VLDB Endowment*, Vol. 3, No. 1-2, pp. 1338–1347, 2010.

[18] Xusheng Luo, Kangqi Luo, Xianyang Chen, and Kenny Zhu. Cross-lingual entity linking for web tables. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32, 2018.

[19] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pp. 3111–3119, 2013.

[20] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, Vol. 26, , 2013.

[21] Varish Mulwad, Tim Finin, Zareen Syed, and Anupam Joshi. T2ld: Interpreting and representing tables as linked data. In *9th International Semantic Web Conference ISWC*, p. 25. Citeseer, 2010.

[22] Dominique Ritze, Oliver Lehmberg, Yaser Oulabi, and Christian Bizer. Profiling the potential of web tables for augmenting cross-domain knowledge bases. In *Proceedings*

of the 25th international conference on world wide web, pp. 251–261, 2016.

[23] Yoshihiko Suhara, Jinfeng Li, Yuliang Li, Dan Zhang, Çağatay Demiralp, Chen Chen, and Wang-Chiew Tan. Annotating columns with pre-trained language models. In *Proceedings of the 2022 International Conference on Management of Data*, pp. 1493–1503, 2022.

[24] Zareen Syed, Tim Finin, Varish Mulwad, Anupam Joshi, et al. Exploiting a web of semantic data for interpreting tables. In *Proceedings of the Second Web Science Conference*, 2010.

[25] Kunihiro Takeoka, Masafumi Oyamada, Shinji Nakadai, and Takeshi Okadome. Meimei: An efficient probabilistic approach for semantically annotating tables. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33, pp. 281–288, 2019.

[26] Zareen Syed Varish Mulwad, Tim Finin and Anupam Joshi. Using linked data to interpret tables. In *Proceedings of the the First International Workshop on Consuming Linked Data*, November 2010.

[27] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

[28] Daheng Wang, Prashant Shiralkar, Colin Lockard, Binxuan Huang, Xin Luna Dong, and Meng Jiang. Tcn: Table convolutional network for web table interpretation. In *Proceedings of the Web Conference 2021*, pp. 4020–4032, 2021.

[29] Dan Zhang, Yoshihiko Suhara, Jinfeng Li, Madelon Hulsebos, Çağatay Demiralp, and Wang-Chiew Tan. Sato: Contextual semantic type detection in tables. *arXiv preprint arXiv:1911.06311*, 2019.

[30] Ziqi Zhang. Towards efficient and effective semantic table interpretation. In *International Semantic Web Conference*, pp. 487–502. Springer, 2014.

[31] Ziqi Zhang. Effective and efficient semantic table interpretation using tableminer+. *Semantic Web*, Vol. 8, No. 6, pp. 921–957, 2017.