Mapping Wikipedia Categories and Lists to DBPedia Ontology Based on Structural and Semantic Features

Zhenyang ZHANG † Zhaoyi WANG ‡ and Mizuho IWAIHARA ‡

† Graduate School of Information, Production and Systems, Waseda University

2-7 Hibikino, Wakamatsu-ku, Kitakyushu, Fukuoka 808-0135 Japan

E-mail: † zhangzhenyang@fuji.waseda.jp, ‡ wangzy-joey@akane.waseda.jp, ‡ iwaihara@waseda.jp

Abstract Ontology mapping plays an important role in the integration of knowledge resources and many down-stream tasks that include entity recognition, such as entity linking, entity typing, table column typing, relation extraction, question answering, and other knowledge graph-related tasks. With the development of deep leaning and its successful application to various domains, the deep learning approach has been applied in ontology mapping. However, existing methods rarely focus on the hierarchy of the knowledge graph and the syntactic structures of the ontology class names. In this paper, we propose a novel ontology mapping method based on structural and semantic features. It contains two main parts: (i) Type inheritance based on structural features and (ii) Fine-tuning a classifier based on pre-trained language model BERT for capturing semantic features. For the first part, we design heuristic rules for assigning and propagating DBPedia types over Wikipedia list and category nodes. For the second part, we utilize POS tagging and dependency parsing graphs to find root phrases of Wikipedia category names, then use the unsupervised SimCSE model to generate embedding of each category name. Then the cosine similarities between these embeddings are used to fine-tune the BERT model. We also search sibling nodes which are likely to share the same DBPedia type, for augmenting the training set. Finally, we use Random Forests to select the most specific node among the results generated by two parts. Experimental results show that our method achieves superior results than baselines.

Keyword Knowledge graph, Ontology mapping, Wikipedia categories and lists, Distant supervision

1. Introduction

Ontology mapping attempts to match entities from different ontologies that have semantically similar properties. A mapping is a connection between two matched entities, where each mapping is often categorized as equivalence and subsumption.

Wikipedia is the world's largest and free encyclopedia maintaining high-quality trusted information. Most of its entries (i.e., Wikipedia articles) can be considered as (semi-structured) representations of entities. Wikipedia offers three complementary ways to group related entries together: Categories, lists, and navigation templates [21]. Categories are organized in a hierarchy and each Wikipedia article is assigned to at least one category. Lists provide a means for manual categorization of articles and can include entities that do not have a Wikipedia page yet. Lists are more difficult to process automatically due to informal construction.

DBPedia [22] is a large knowledge graph which leverages gigantic source of knowledge by extracting structured information from Wikipedia. The DBPedia ontology is the heart of DBPedia. The ontology (at the time of 2022-08-29) covers 768 classes which form a subsumption hierarchy as Figure 2 (a) shows and are described by around 3000 different properties. The DBPedia ontology (at the time of 2022-08-29) currently contains about 4,828,418 instances.

CaLiGraph [8] is a large semantic knowledge graph with a rich ontology compiled from the DBPedia ontology and Wikipedia categories and list pages, as Figure 3 (a) shows. The ontology is enriched with fine-grained value restrictions on its classes that are discovered with the Cat2Ax [9] approach. A large number of CaLiGraph's entities is extracted from Wikipedia listings through a combination of the ontological information and transformer-based extractors.

Mapping Wikipedia categories and lists to DBPedia Ontology plays a critical role in many downstream tasks that include entity recognition, such as entity linking, entity typing, table column typing, relation extraction, question answering, and other KG-related tasks. For example, for the Wikipedia category "1999_science_fiction_novels", if we assign the DBPedia type "Novel" to this category, then, given a table column of novel names such as "Vector_Prime" as Figure 1 shows, we can search the entities of the given column under the Wikipedia category or search the entity resources in the CaLiGraph ontology and assign the mapped DBPedia type to the matched entities as the table column typing.

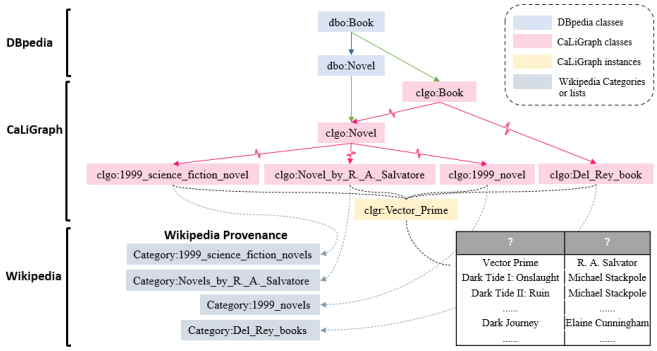

Figure 1 Illustration of matching column to CaLiGraph (Wikipedia category). [13]

Lexical matching serves as the foundation for traditional ontology mapping solutions, which is often combined with structural matching. This gave rise to various traditional systems such as Cat2Ax [9]. Their lexical matching approach, however, only focuses on the text's surface form, such as overlapping sub-strings and sharing a textual pattern, which is unable to capture word semantics. Lexical and structural matchings have recently been suggested to be replaced by machine learning; for instance, DeepAlignment [11] and OntoEmma [16] use word embeddings to represent classes and compute the similarity of two classes according to the Euclidean distance between their word vectors. However, these approaches either require extensive feature engineering that is ad-hoc and relies on great amounts of annotated examples for training, or they use classic non-contextual word embedding models like Word2Vec, which only learns a global (context-free) embedding for each word. Contrarily, pre-trained transformer-based language models such as BERT [2] and SimCSE [8] can learn robust contextualized text embeddings, and often just need a little amount of training resources to be fine-tuned. Although these approaches excel at many NLP tasks, ontology mapping has not yet received enough research attention from them.

In this paper, we propose a new ontology mapping method that utilizes hierarchies of two knowledge graphs to do type inheritance and exploits semantic embeddings to construct training data for BERT fine-tuning to perform type prediction. Specifically, our method includes the following main steps: (1) Type inheritance, where we predict the DBPedia type based on heuristic rules, which provides training samples for distantly supervised learning. (2) Finding root words. In this paper, a root word is a single noun word which holds the most basic meaning of a long noun phrase. For example, for a given CaLiGraph class name "Opera house in Puerto Rico", its root word is "house". Here we do the POS tagging to find root words of CaLiGraph class names. (3) Finding root phrases. In this paper, a root phrase is defined as a root word with a prefix of n-words which have "compound", "modifier of nominal" or "appositional modifier" dependency relations with the root word, and the part of speech (POS) of the words in root phrase must be "NOUN". For example, for a given CaLiGraph class name "Opera house in Puerto Rico", its root phrase is "Opera house". A root phrase is a part of a CaLiGraph class name. Here we extend a root word to a root phrase utilizing Level Order Traversal on dependency parsing graph. (4) Root path search. In this paper, a root path is defined as the concatenation of the ancestor nodes' root word which appears in the path from the given CaLiGraph class name node to the top of CaLiGraph's hierarchy. For example, for a given CaLiGraph class name "Opera house in Puerto Rico", its root path is "Venue Theatre House". Here we search the ancestor nodes in the hierarchy of CaLiGraph and find their root words to generate the root path. (5) Semantic embedding generation for computing cosine similarities, where we utilize SimCSE to generate embeddings for constructing training data. (6) Training data augmentation, where we augment the training data for BERT fine-tuning by searching sibling nodes in the hierarchy of CaLiGraph. (7) BERT fine-tuning, using the training data, where a suitable pre-trained BERT model is chosen and fine-tuned. (8) Combining results, where Random Forests is utilized to select the most plausible type.

We evaluate our method on the CaLiGraph-DBPedia mapping task, and experimental results show that our method achieves superior results than the baseline model Cat2Ax in terms of Macro-Averaged F1 scores.

2. Related Work

2.1 CaLiGraph and Cat2Ax

CaLiGraph is a large semantic knowledge graph with a rich ontology compiled from the DBPedia ontology and Wikipedia categories and list pages [8]. Since the first version released in Oct. 14, 2019, the latest (19th) version was released in Sept. 21, 2021. The ontology is enriched with fine-grained value restrictions on its classes that are discovered with the Cat2Ax approach. Through a combination of the ontological information and transformer-based extractors, many CaLiGraph instances are retrieved from Wikipedia categories and listings.

Classes in CaLiGraph are derived from lists and categories in Wikipedia [13].

The Cat2Ax [9] approach has four major steps: (1) Identify candidate category sets that share a textual pattern. (2) Find characteristic properties and types for candidate sets and combine them to patterns. (3) Apply patterns to all categories to extract axioms. (4) Apply axioms to their respective categories to extract assertions.

2.2 Distant Supervision

Machine learning methods basically need a collection of training data. Manually assigning labels to a collection of documents is a standard method for constructing training data. This method is time- and money-consuming, and if the corpus is vast, produced data are not for models to be trained.

Another method to generating training data is distant supervision. Distant supervision is a learning scheme in which training samples are labeled automatically based on certain rules, suitable for situations where training data construction is costly. Distant supervision's assumption for relation typing is that any statement that contains two entities that are involved in a relationship may refer to that relationship [12].

Distant supervision for semantic typing is an extension of the paradigm used by [18] for utilizing WordNet to uncover hypernym (is-a) relations between entities, and is analogous to the application of poorly labeled data in bioinformatics [19][20], and in relation extraction which has no labeled data [12]. In our method for mapping entities to knowledge graph types, we apply distant supervision to discover mapping between CaLiGraph ontology classes and DBPedia types, based on manually constructed rules to generate initial mappings, then extend the mappings to siblings and descendants in the CaLiGraph hierarchy, that are predicted to share the same DBPedia type through dense representations. Based on the observations on Wikipedia category/list name structure and the hierarchy of CaLiGraph classes, we generate four rules for entity mapping between CaLiGraph nodes and DBPedia types.

2.3 BERT pretraining and finetuning

BERT is a contextualized pretrained language model built on bidirectional transformer encoders [15]. Both pretraining and finetuning are part of its training paradigm. In pretraining, the input consists of a sequence that includes a special token [CLS], tokens from one sentence A, another special token [SEP], tokens from another phrase B that follows A. Every token's first embedding encodes its content, place in the sequence, and the phrase it belongs to (A or B). The model's architecture consists of several sequential layers with the same design. The multi-head self-attention block, which is its core part, computes a contextual hidden representation of each token by taking into account the whole output of the preceding layer's sequence. The embeddings of the tokens from the final layer can be used as the input for a customized downstream layer. Pretraining is conducted by minimizing losses on two tasks: Next sentence prediction and masked language model. Contrary to traditional non-contextual word embedding techniques, which only give each token one embedding, BERT may identify many instances of the same token. In finetuning, customized downstream layers are fine-tuned based on a pretrained BERT model. For finetuning, it normally needs a relatively small number of epochs and data samples [2].

2.4 Sentence embedding

Sentence embedding techniques represent entire sentences and their semantic information as low-dimensional vectors. This helps the model in understanding the context, intention, and other nuances in the entire text. Sent2vec [17] utilizes word n-gram features to produce sentence embeddings. Arora et al. propose SIF in which sentences are represented as the weighted average of the word embeddings.

Recently, Siamese networks and contrastive learning framework have been proposed for sentence embedding models. SimCSE [8] is a simple contrastive learning framework that advanced SOTA records on sentence embeddings. SimCSE employs unsupervised and supervised approaches. The unsupervised SimCSE takes an input sentence and predicts itself in a contrastive objective, with only standard dropout used as noise. In our work, we utilize SimCSE for calculating cosine similarities between candidate root phrases (or root path) and DBPedia types.

3. Problem definition

Our goal is to predict DBPedia types of Wikipedia category and list names that are included in CaLigraph. We formulate this task as a multi-class classification task.

Given a pair of ontologies of two knowledge graphs K and K' , O and O', whose named class sets are C and C', respectively, and the ontology class hierarchies H and H' of two knowledge graphs. Then the ontology mapping

problem is to determine a mapping M from C to C' where M maps each class $c \in C$ to a target class $c' \in C$ such that c' represents the minimum concept that subsumes c. Table 1 shows the symbols used in our method.

Table 1. Symbol list.

Symbol	Description
K	Knowledge graphs
O	Ontology of knowledge graphs
$C = \{c_1, c_2, \ldots, c_m\}$	The set of named classes of O
H	Ontology class hierarchy
$T = \{t_1, t_2, \ldots, t_m\}$	The set of DBPedia types

4. Methodology

4.1. Model Structure

In this paper, we propose a new ontology mapping method, which can predict the DBPedia type for a given CaLiGraph ontology class based on structural and semantic features. Figure 2 shows the framework of our method.

Our model mainly contains two parts: (i) Type inheritance based on structural features and (ii) Classifier based on fine-tuning the pre-trained model BERT on semantic features. The training process mainly consists of the following five steps: (1) Do POS tagging of CaLiGraph ontology class names to find root words. (2) Extend each root word to a root phrase utilizing dependency parsing graph. (3) Search the ancestor nodes in the hierarchy of CaLiGraph and use ancestor nodes' root words to generate root paths. (4) Utilize SimCSE to embed the root phrase, root path, DBPedia type into the same space. (5) Calculate the cosine similarity between the root phrase, root path and DBPedia type, respectively, and select the most similar DBPedia type according to cosine similarities as training data for BERT fine-tuning. (6) Augment the training data for BERT fine-tuning utilizing sibling nodes which share the same root phrase. (7) Use the augmented training data to fine-tune the BERT classifier.

Finally, we utilize Random Forests to select the most specific DBPedia type among two results generated by Type Inheritance and BERT classifier as the final mapping.

4.2. Type Inheritance

Heuristic rules. We assume there exists a relation that there is a unique DBPedia specific type for each CaLiGraph node, thus this relation is a mapping. Based on this assumption, we construct the following four heuristic rules to do type inheritance.

Rule 1: A DBPedia class node in the hierarchy of DBPedia may have an exactly identical class name in the hierarchy of CaLiGraph. CaLiGraph uses DBPedia as an upper-level taxonomy and categorizes these rather general

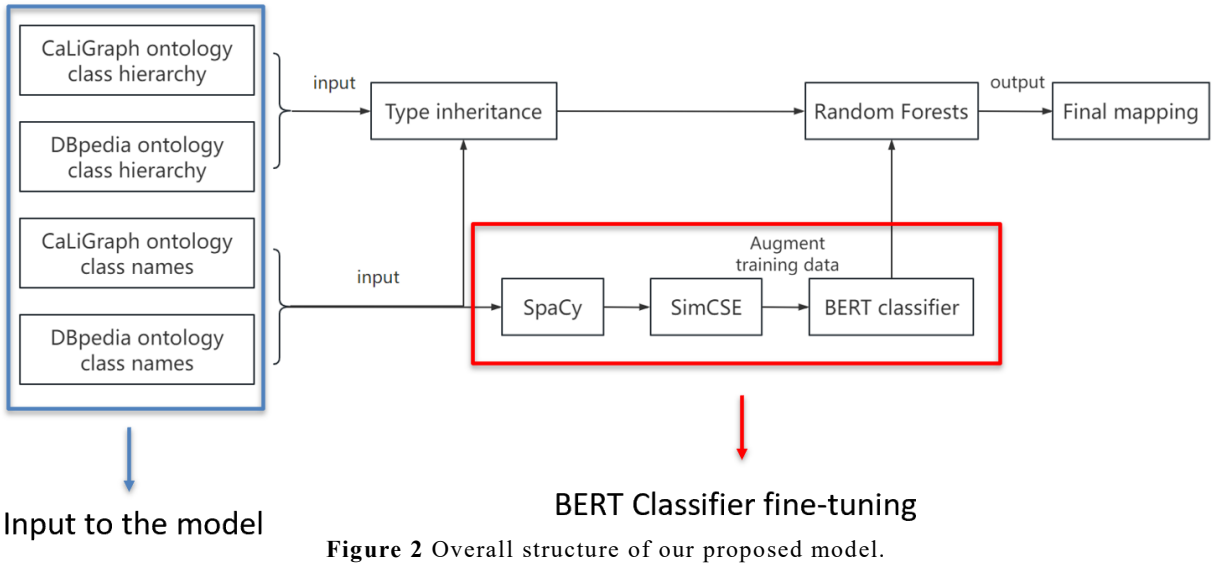

Figure 2 Overall structure of our proposed model.

types in DBPedia into more specific types. We check an exactly matching CaLiGraph node in the hierarchy for every DBPedia node. Figure 3 shows an example of Rule 1.

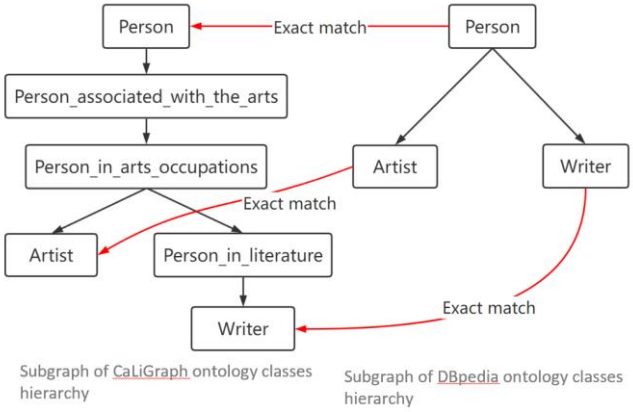

Figure 3. Example of Rule 1

Rule 2: Inheritance. If a CaLiGraph node *c* is mapped to a DBPedia type *t*, then *c*'s descendants are also mapped to the same type *t*. So, if a DBPedia node has an exactly matching CaLiGraph node, then its descendant CaLiGraph nodes can also be mapped to the DBPedia node. For example, as Figure 4 shows, the CaLiGraph node "20th-century_American_women_politician" has an ancestor node "Politician", and this ancestor node has an exactly matching DBPedia node "Politician", so we can assign this DBPedia type "Politician" to the CaLiGraph node "20th-century_American_women_politician". However, a descendant may be inheriting from multiple ancestor nodes, so this rule will give candidate types but not a unique type, as Figure 5 shows.

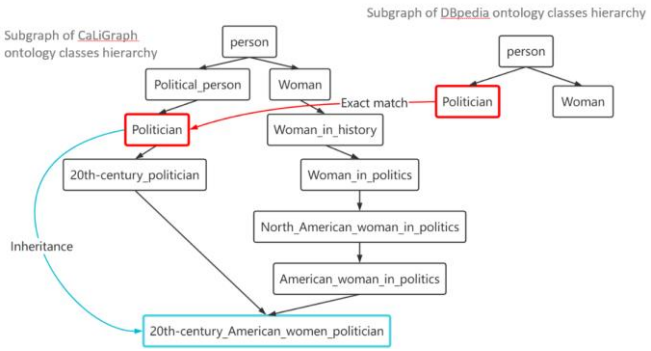

Figure 4. Example of Rule 2

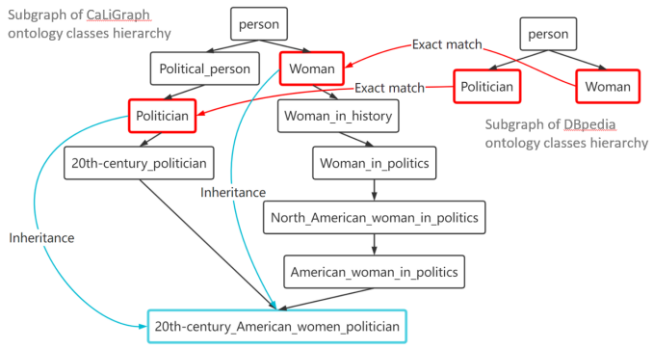

Figure 5. Example of multiple inheritance

Rule 3: The depth of DBPedia node in the hierarchy of DBPedia reflects the specificity degree of the DBPedia node. For example, as Figure 6 shows, the depth of DBPedia node "Politician" is larger than the depth of DBPedia node "Person", and we can say the type "Politician" is more specific than the type "Person".

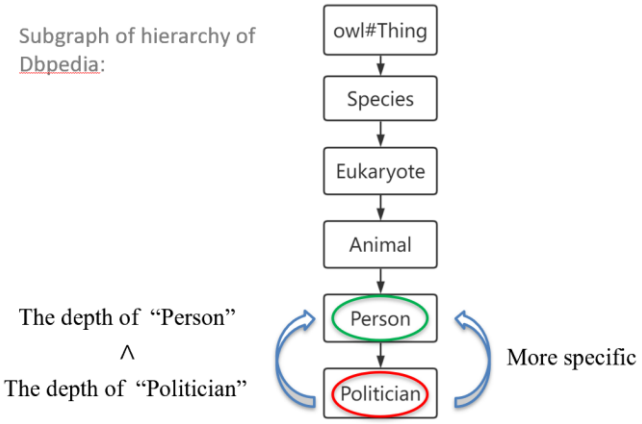

Figure 6. Example of Rule 3

Rule 4: The root word of a CaLiGraph node has significant semantic similarity with a DBPedia type. For example, as Figure 7 shows, the root word of the CaLiGraph node "Cuban expatriate sportsperson in the United States" is "sportsperson" which is semantically similar with its DBPedia type "Athlete".

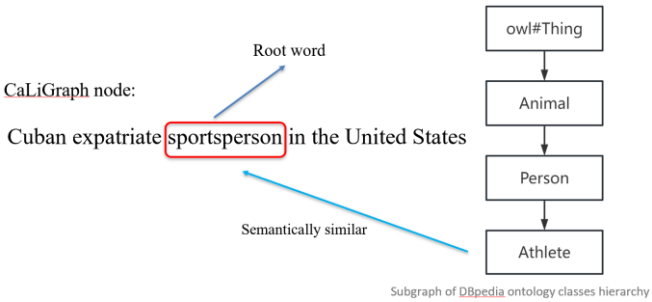

Figure 7. Example of Rule 4

Type Inheritance, Ranking, Selecting.

Given a CaLiGraph node, we first judge whether there is a path between the CaLiGraph node and also judge

whether there exists a CaLiGraph node in the hierarchy of CaLigraph exactly matching with the DBPedia node. If the path exists, we add this DBPedia node to the candidate type list of the given CaLiGraph node.

Then we define a scoring function for ranking candidate types: if the candidate type has a score proportional to the depth in the CaLiGraph hierarchy, denoted as S_depth, and if the candidate type contains the root word of the given CaLiGraph node, we give it a score denoted as S_root, and we rank the candidate types by the combination of two scores as follows:

$$S_{com} = \alpha * S_{depth} + (1 - \alpha) S_{root}$$

Here, $0 \leq \alpha \leq 1$ is a hyperparameter. Finally, we select the DBPedia type having the highest score as the result to be assigned to the given CaLiGraph node.

4.3. Training data construction

POS tagging. SpaCy is a POS tagging and syntactic dependency parse tool [24]. We utilize the spaCy processing pipeline to do POS tagging to find the root word of a CaLiGraph node. For a long noun phrase, the root word holds the most basic meaning of the whole phrase.

Firstly, to generate a Doc object, spaCy tokenizes the text when we call nlp on a paragraph of text. Next, the Doc object is processed by a few steps which can be seen as the processing pipeline. Utilized by the trained pipelines, the pipeline commonly contains a tagger, a parser, a lemmatizer and an entity recognizer. Every component in the pipeline returns the processed Doc, which is then forwarded to the following component. Figure 8 shows the example of POS tagging.

Token	Relation	Head	Children
Sports	compound	organization	[]
organization	ROOT	organization	[Sports, established]
established	acl	organization	[in]
in	prep	established	[1985]
1985	pobj	in	[]

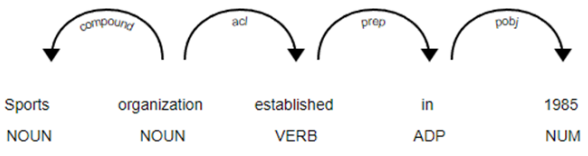

Figure 8. Example of POS tagging

Extend the root word to root phrase

Dependency parsing, one of the views of linguistic structure, refers to examining the dependencies between the words of a sentence to analyze its grammatical structure. Based on this, we can extract a dependency of a sentence that represents its grammatical structure and defines the relationships between "head" words and words, which modify those heads.

Sometimes only the root word cannot fully represent the semantic features of CaLiGraph class names. For example, for the CaLiGraph class name “South Korean football club season”, the root word is “season”, but it is hard for us to assign a plausible DBPedia type to the CaLiGraph node only according to the root word. However, for the root phrase "football club season”, we can easily find a plausible DBPedia type “SoccerClubSeason”.

For the above reason, we extend a root word to a root phrase utilizing Level Order Traversal on the dependency parsing graph. The process includes the following main six steps:

Step1: Tokenize the CaLiGraph class name and create an index for each token.

Step2: Find a root word based on POS Tagging.

Step3: Draw the dependency parsing graph based on the root word.

Step4: Do level order traversal on the dependency parsing graph and judge if the token satisfies a condition. If it satisfies the condition, add this token to the candidate_token_list.

Step5: After the level order traversal, sort the candidate_token_list according to the index created before.

Step6: Concatenate the sorted tokens in the candidate_token_list as the root phrase.

Figure 9 Shows an example of Level order traversal algorithm using FIFO queue on a dependency analysis graph.

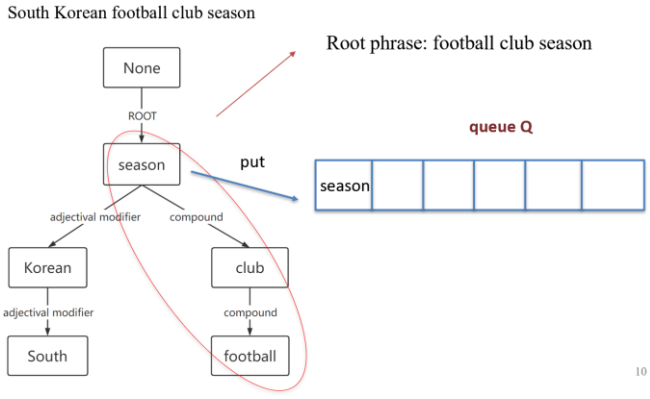

Figure 9. Example of dependency analysis graph

Another Root phrase extraction strategy

However, while enriching semantic information, root phrases sometimes bring noise. Therefore, we propose a different strategy for root phrase extraction. We extract different lengths of a root phrase as root phrase

candidates.

For example, “American football team in Finland” is a class in CaLiGraph. After we use dependency parsing, we can obtain the dependency relationship in Figure 10.

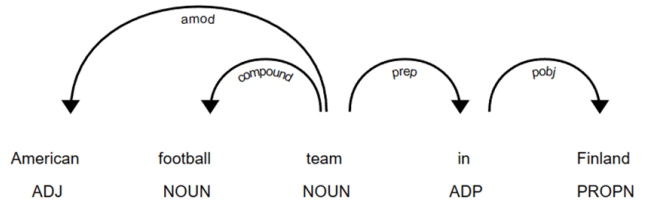

Figure 10. “American football team in Finland” after dependency parsing.

In the diagram, the “team” forms the head of the above sentence. The dependency relationship between any two words is represented with arrows. For instance, “American” is an adjectival modifier of root word “team”. In this paper, we start from the root word, then traverses the left and right subtrees of the root word according to the dependency tree. The words in the left and right subtrees are added to the root phrase candidate set with the root word. In the above sentence, the left subtrees of the root “team” are “American” and “football”, and the right subtree is the preposition “in”. We first add the root word “team” in the root phrase candidate set. Then we combine “American” and “team” and add the phrase “American team” to the candidate set, as is “football”. Since “American” and “football” are directly modifying “team”, we should combine these three words (“American football team”) as a root phrase. For the left subtree, “in” is a preposition which has no sense, and we should not add it to the root phrase candidate set, but “in” and the following object “Finland” together form a phrase which modifies “team”. Thus, “team in Finland” should be added in the root phrase candidate set. Finally, for this sentence, the root phrase candidate set contains “team”, “American team”, “football team”, “American football team” and “team in Finland”. For each CaLiGraph class, we extract root phrases as described above.

Using SimCSE to generate node representation.

Sentence embedding models capture semantic relatedness via the distances between the corresponding vector representations within the shared vector space. Because there is a high semantic similarity between root words (phrases) and DBPedia types, we use a sentence embedding model to rank the candidate types by measuring their distance to the root words (phrases). Recently, two sentence embedding models BERT and SimCSE catch attention because of their good performance.

However, using non-finetuned BERT sentence vectors directly for unsupervised semantic similarity calculation has poor performance. The similarity of BERT sentence vectors of any two sentences is quite high, where one of the reasons is the nonlinearity and singularity of its vector distribution.

SimCSE reflects the semantic relatedness between phrases when using similarity measures on the corresponding vectors. The unsupervised SimCSE model is greatly outperforming previous SOTA models.

By observing CaLiGraph node class names, we find that the root word of a class name has a high semantic similarity to its corresponding DBPedia type. However, just a root word is not enough, since the root word only is lacking contextual information. So, we search the path from CaLiGraph node to the top of the hierarchy and do POS tagging for these ancestor nodes in the path to generate a root path to capture contexts. However, some ancestor nodes in the top of hierarchy are too general, so that making them meaningless. We set a maximum token number to drop meaningless root words. In this paper we set the maximum token number as 3 as a boundary. If the length of a root path is larger than this boundary, we drop the root word beyond the boundary. If the length of the root path is smaller than the boundary, keep it as it is. As Figure 11 shows, the root path of the CaLiGraph path is “Venue Theatre House”, the tokens “Thing” and “Building” are dropped because they are beyond the boundary.

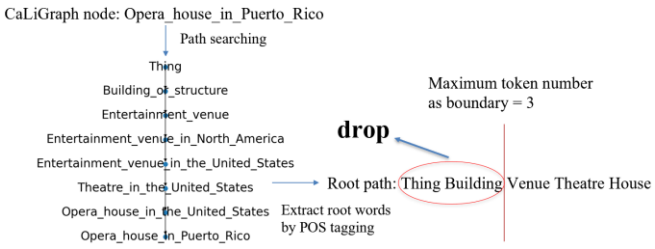

Figure 11. Example of root path generating

In this paper, we utilize the unsupervised SimCSE model to generate vectors for both root phrases and root paths of CaLiGraph nodes and DBPedia types.

Cosine similarity computing. The cosine similarity between a root phrase of CaLiGraph node embedding V_{root_phrase} and a DBPedia node embedding V_{DBP} is

calculated for ranking, as follows:

$$Sim(V_{root\_phrase}, V_{DBP}) = Cos(V_{root\_phrase}, V_{DBP}) = \frac{\overrightarrow{V_{root\_phrase}} \times \overrightarrow{V_{DBP}}}{|\overrightarrow{V_{root\_phrase}}| \times |\overrightarrow{V_{DBP}}|}$$

The cosine similarity between the root path of CaLiGraph node embedding V_{root_path} and DBPedia node embedding V_{DBP} is calculated in the same way. The cosine similarity between root phrase and DBPedia node is $Sim(V_{root\_phrase}, V_{DBP})$, the cosine similarity between a root path and DBPedia node is denoted as $Sim(V_{root\_path}, V_{DBP})$. We calculate the weighted cosine similarity, with weighting factor β, as:

$$Sim_{weighted} = \beta * Sim(V_{root\_phrase}, V_{DBP}) + (1-\beta) * Sim(V_{root\_path}, V_{DBP})$$

We rank DBPedia types according to the weighted cosine similarity scores and select the DBPedia type which has the highest weighted cosine similarity to the given CaLiGraph node.

4.4. Training data augmentation

In the hierarchy of CaLiGraph, nodes sharing the same parent node are regarded as sibling nodes. The sibling nodes which share the same root phrase are likely to have the same DBPedia type, therefore we search sibling nodes of the CaLiGraph nodes in the training data constructed before, to do training data augmentation. The augmentation mainly contains the following three steps: (1) Search all sibling nodes of a CaLiGraph node in the hierarchy of CaLiGraph. (2) For each sibling node, judge whether it contains the root phrase of the given CaLiGraph node. If so, we assign the same DBPedia type to the sibling node. (3) Add the sibling node-DBPedia type pairs to the training data as data augmentation. Figure 14 shows an example of augmentation: We have assigned the DBPedia type “BasketballPlayer” to the CaLiGraph class name “Virginia Tech Hokies women’s basketball player” utilizing the methods introduced before, then we search sibling nodes in the hierarchy and find three sibling nodes as shown in Figure 14. The root phrase of sibling node “Penn State Nittany Lions softball player” is “softball player” which is different from the root phrase of “Virginia Tech Hokies women’s basketball player”, so we drop it, and other two sibling nodes share the same root phrase “basketball player” with CaLiGraph class name “Virginia Tech Hokies women’s basketball player.” Thus we assign the same DBPedia type “BasketballPlayer” to the sibling nodes as augmentation.

4.5. BERT Fine-tuning

Given sets of pairs of CaLiGraph nodes and DBPedia type nodes generated by the previous method based on semantic features, a pre-trained BERT model with a downstream classifier is finetuned on the objective of the cross-entropy loss. A pair of a tokenized CaLiGraph node and DBPedia node label with a maximum length of 85 is an input sequence for the BERT model. The classifier has a linear layer (with dropout) that receives the embedding of the [CLS] token from the outputs of the final layer of BERT as input, and before applying the output softmax layer, and the output is convertd into a 2-dimensional vector. The Adam algorithm is used to perform the optimization. The final output is vectors of a probability distribution over the DBPedia types for each CaLiGraph node.

4.6. Result combination

We can obtain two DBPedia type candidates generated by Type Inheritance and BERT Classifier introduced before. We utilize the machine learning method Random Forests to select an optimum one.

Numerous classification trees are grown in Random Forests. Here we denote a CaLiGraph node embedding representation as A, the predicted DBPedia type embedding representation by Type Inheritance as B, the predicted DBPedia type representation embedding by the BERT classifier as C. Then We utilize SimCSE to do the concatenation of the embedding $[A; B]$ as E_1 and $[A; C]$ as E_2, and use E_1 and E_2 as the input vectors. We refer to each tree's categorization as a “vote” for that class. The categorization with the highest votes is selected by the forest (over all the trees in the forest). Each tree is grown as follows:

(1) If the number of cases in the training set is N, sample N cases at random but with replacement, from the original data. This sample will be the training set for growing the tree.

(2) We use the dimension of the embeddings as M input variables, a number m << M is specified such that at each node, select m variables randomly out of M and the best split on these m is utilized to split the nodes. Throughout the growth of the forest, m is maintained at a fixed value.

(3) Every tree is developed to its full potential. Pruning is not done.

To construct training data for Random Forests training,

we annotate labels to a small subset of CaLiGraph nodes by human annotation. Finally, we enter the training set into Random Forests to obtain the final prediction.

5. Experiments

In this section, we first introduce datasets used in our work, followed by evaluation metrics. Then, we present and discuss our experimental results on the CaLiGraph-DBPedia mapping task.

5.1. Datasets

Dataset construction. For BERT classifier training, we first randomly sample a small subset of all CaLiGraph nodes as the test set, and we annotate the golden label to the CaLiGraph nodes in the test set by human annotation as the ground truth. Then we randomly sample a subset of all CaLiGraph nodes as the training dataset, and we checked whether the CaLiGraph node in the test set appear in the training dataset. If it appears, we remove it from the training dataset to make sure that the CaLiGraph nodes in the training dataset are disjoint with the CaLiGraph nodes in the test set. Then we split the training dataset to 80% as a training set and 20% as a validation set.

For Random Forests training, we use the same test set constructed above as the test set for Random Forests, and we randomly sample a small subset of CaLiGraph nodes as the training set for Random Forests training. We also check whether each CaLiGraph node in the test set appears in the training set. If the node appears, we remove it from the training set.

Table 2. shows statistics of the dataset.

5.2. Evaluation Metrics

We evaluate the model performance on the CaLiGraph-DBPedia mapping task. We use Macro-Averaged Precision, Macro-averaged recall, and Macro-Averaged F1 as main evaluation metrics. The formulas for calculating these metrics are as follows:

$$P = \frac{|M_{out} \cap M_{test}|}{|M_{out}|} \quad (1)$$

$$R = \frac{|M_{out} \cap M_{test}|}{|M_{test}|} \quad (2)$$

$$F_1 = \frac{2 * P * R}{P + R} \quad (3)$$

5.3. Experimental Settings

For the BERT classifier, we choose the pretrained bert-base-multilingual-cased models. The details of training are that epoch = 10, the training batch size = 16, the input max length = 85, learning rate = 1e-5.

For the Random Forests, n_estimators (number of classification trees) = 2000, criterion as gini, max_features = sqrt(n_features).

5.4. Experimental Results and Analysis

Table 3 shows the results of the experiments.

From the results we can see that utilizing the combination of semantic feature and structural feature to predict DBPedia types for CaLiGraph nodes outperforms the method which only utilizes the semantic feature to fine-tune the BERT Classifier. The reason may be that we use root words to represent meaning of each CaliGraph node, but only the root word cannot accurately represent the semantic feature of the CaliGraph node and cause

Table 2 Statistics of dataset.

Dataset	For BERT Classifier		For Random Forests	Number of test set
	Number of training set	Number of validation set	Number of training set	500
CaLiGraph-DBPedia	142758	35689	5186	

Models	Macro-Averaged Precision	Macro-averaged recall	Macro-Averaged F1
Cat2Ax	0.593	0.607	0.599
SimCSE+BERT (root word)	0.589	0.608	0.588
SimCSE+BERT (root phrase)	0.649	0.621	0.635
SimCSE+BERT (root phrase with different length)	0.726	0.708	0.707
SimCSE+BERT (root phrase and sibling nodes)	0.665	0.633	0.649
Type Inheritance +SimCSE+BERT (root word)	0.835	0.826	0.830
Type Inheritance +SimCSE+BERT (root phrase and sibling nodes)	0.884	0.891	0.883

Table 3 Experiment results.

Model SimCSE+BERT (root word) uses the root word of CaLiGraph class names to compute cosine similarity for training data construction. Model SimCSE+BERT (root phrase) extends the root word to root phrase to compute cosine similarity for training data construction. Model SimCSE+BERT (root phrase and sibling nodes) uses the root phrase for training data construction and searches the sibling nodes for training data augmentation. Model Type Inheritance+SimCSE+BERT (root word) combines

misunderstanding and errors on predicting the DBPedia type. Therefore, to contain more useful information, we extend root words to root phrases to better represent the semantic feature of CaLigraph nodes. From the result we can see that the root phrase method outperforms the root word method. However, some root phrases will bring meaningless words which can be seen as noise. To reduce the influence of noise, we further tried different length of root phrases and select the optimum one.

We also do training data augmentation to increase training data. Although this augmentation can amplify noise in the orignal training data, the result shows an improvement on F1 score.

5.5. Analysis on Errors and Difficulties

After checking the experimental results, we find that our model's final prediction has about 10% rate of errors. We compared predicted DBPedia types with golden labels of the error CaLiGraph nodes, and find that there are mainly three kinds of CaLiGraph class names which are difficult to predict DBPedia types: (1) Proper noun. Certain CaLiGraph class names only have proper noun, this make us hard to find the corresponding DBPedia type. For example, for the CaLiGraph node "San_Francisco_Deltas_player", San Francisco Deltas is an American professional soccer team, but only by class name we cannot find this information, so the predicted result of our model is "AthleticsPlayer," which is not specific enough. (2) Class name is too short. A number of CaLiGraph class names are too short to have enough information for prediction. For example, the CaLiGraph class name "Molluscicide" is a kind of pesticide, but according to semantic similarity, the predicted result given by our model is "Mollusca", which is wrong. (3) Part of speech error. According to the semantic feature of class names, we sometimes obtain a semantically related prediction result but its part of speech is not matching with the node. For example, for the CaLiGraph node "1970s crime", the most semantically similar DBPedia type is "Criminal", but the crime is an event, while criminal is a person, which is obviously wrong.

Due to the above errors, we need to find more information such as Wikipedia category pages which can indicate corresponding DBPedia types.

6. Conclusion and Future Work

In this paper, we proposed a novel ontology mapping

method based on structural and semantic features. We construct four heuristic rules to do Type Inheritance based on the hierarchies of CaLiGraph and DBPedia. We use spaCy processing pipeline to do POS tagging to find root words of CaLiGraph nodes. Then we utilize a dependency parsing graph based on root words and do the Level Order Traversal by FIFO queue on the dependency parsing graph to extend the root word to a root phrase. We use SimCSE to generate embeddings of a root phrase and root path of CaLiGraph nodes and compute the cosine similarity between them and DBPedia types, and rank the DBPedia type candidates and select the most similar one to construct the training data for BERT Classifier finetuning. We augment the training data by searching sibling nodes in the hierarchy of CaLiGraph. We further combine the predictions of Type Inheritance and BERT Classifier to generate the final result by Random Forests. Our experiments show that our method is outperforming Cat2Ax on the CaLiGraph-DBPedia mapping task.

In future work, we intend to apply masked language model to predict phrases which indicate the DBPedia types and design a ranking method to rank the candidate DBPedia types. Furthermore, for a small part of CaLiGraph nodes, we cannot predict specific DBPedia types only by class names, because class names are sometimes too short to contain much useful information. Therefore, we try to find more information besides the class names, such as looking up useful keyphrases in Wikipedia categories and list pages.

References

[1] Biswas R, Sofronova R, Sack H, et al. Cat2type: Wikipedia category embeddings for entity typing in knowledge graphs[C]//Proceedings of the 11th on Knowledge Capture Conference. 2021: 81-88.

[2] Devlin J, Chang M W, Lee K, et al. Bert: Pre-training of deep bidirectional transformers for language understanding[J]. arXiv preprint arXiv:1810.04805, 2018.

[3] Gao T, Yao X, Chen D. Simcse: Simple contrastive learning of sentence embeddings[J]. arXiv preprint arXiv:2104.08821, 2021.

[4] He Y, Chen J, Antonyrajah D, et al. BERTMap: A BERT-based ontology alignment system[C]//Proceedings of the AAAI Conference on Artificial Intelligence. 2022, 36(5): 5684-5691.

[5] He Y, Chen J, Antonyrajah D, et al. Biomedical ontology alignment with BERT[J]. 2021.

[6] Heist N, Paulheim H. Entity extraction from Wikipedia list pages[C]//European Semantic Web Conference. Springer, Cham, 2020: 327-342.

[7] Heist N, Paulheim H. Information extraction from co-occurring similar entities[C]//Proceedings of the Web Conference 2021. 2021: 3999-4009.

[8] Heist N, Paulheim H. The CaLiGraph ontology as a challenge for OWL reasoners[J]. arXiv preprint arXiv:2110.05028, 2021.

[9] Heist N, Paulheim H. Uncovering the semantics of Wikipedia categories[C]//International semantic web conference. Springer, Cham, 2019: 219-236.

[10] Ho T K. Random decision forests[C]//Proceedings of 3rd international conference on document analysis and recognition. IEEE, 1995, 1: 278-282.

[11] Kolyvakis P, Kalousis A, Kiritsis D. Deepalignment: Unsupervised ontology matching with refined word vectors[C]//Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers). 2018: 787-798.

[12] Mintz, M., Bills, S., Snow, R., & Jurafsky, D. (2009). Distant supervision for relation extraction without labeled data.

[13] QIN J, IWAIHARA M. Annotating Column Type Utilizing BERT and Knowledge Graph Over Wikipedia Categories and Lists[J].

[14] Svetnik V, Liaw A, Tong C, et al. Random forest: a classification and regression tool for compound classification and QSAR modeling[J]. Journal of chemical information and computer sciences, 2003, 43(6): 1947-1958.

[15] Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need[J]. Advances in neural information processing systems, 2017, 30.

[16] Wang L L, Bhagavatula C, Neumann M, et al. Ontology alignment in the biomedical domain using entity definitions and context[J]. arXiv preprint arXiv:1806.07976, 2018.

[17] Moghadasi, M. N., & Zhuang, Y. (2020, December). Sent2vec: A new sentence embedding representation with sentimental semantic. In 2020 IEEE International Conference on Big Data (Big Data) (pp. 4672-4680). IEEE.

[18] Snow, R., Jurafsky, D., & Ng, A. (2004). Learning syntactic patterns for automatic hypernym discovery. Advances in neural information processing systems, 17.

[19] Craven, M., & Kumlien, J. (1999, August). Constructing biological knowledge bases by extracting information from text sources. In ISMB (Vol. 1999, pp. 77-86).

[20] Morgan, A. A., Hirschman, L., Colosimo, M., Yeh, A. S., & Colombe, J. B. (2004). Gene name identification and normalization using a model organism database. Journal of biomedical informatics, 37(6), 396-410.

[21] https://en.wikipedia.org/

[22] https://www.DBPedia.org/resources/ontology/

[23] http://caligraph.org/statistics.html

[24] https://spacy.io/usage/processing-pipelines