ColNet: Embedding the Semantics of Web Tables for Column Type Prediction

Jiaoyan Chen,[1] Ernesto Jiménez-Ruiz,[2,4] Ian Horrocks,[1,2] Charles Sutton[2,3]
[1]Department of Computer Science, University of Oxford, UK
[2]The Alan Turing Institute, London, UK
[3]School of Informatics, The University of Edinburgh, UK
[4]Department of Informatics, University of Oslo, Norway

Abstract

Automatically annotating column types with knowledge base (KB) concepts is a critical task to gain a basic understanding of web tables. Current methods rely on either table metadata like column name or entity correspondences of cells in the KB, and may fail to deal with growing web tables with incomplete meta information. In this paper we propose a neural network based column type annotation framework named ColNet which is able to integrate KB reasoning and lookup with machine learning and can automatically train Convolutional Neural Networks for prediction. The prediction model not only considers the contextual semantics within a cell using word representation, but also embeds the semantics of a column by learning locality features from multiple cells. The method is evaluated with DBPedia and two different web table datasets, T2Dv2 from the general Web and Limaye from Wikipedia pages, and achieves higher performance than the state-of-the-art approaches.

Introduction

Tables on the Web, which often contain highly valuable data, are growing at an extremely fast speed. Their power has been explored in various applications including web search (Cafarella et al. 2008), question answering (Sun et al. 2016), knowledge base (KB) construction (Ritze et al. 2016) and so on. For most applications, web table annotation which is to gain a basic understanding of the structure and meaning of the content is critical. This however is often difficult in practice due to metadata (e.g., table and column names) being missing, incomplete or ambiguous.

An entity column is a table column whose cells are text phrases, i.e., mentions of entities. Type annotation of an entity column[1] means matching the common type of its cells with widely recognized concepts such as semantic classes of a KB. For example, a column composed of "Mute swan", "Yellow-billed duck" and "Wandering albatross" is annotated with dbo:Species and dbo:Bird, two classes of DBpedia (Auer et al. 2007). Column types not only enable understanding the meaning of cells but also form the base of other table annotation tasks such as property annotation (Pham et al. 2016) and foreign key discovery (Zhang et al. 2010).

[1]Note that data type prediction is not considered in this work.

Table annotation tasks are often transformed into matchings between the table and a KB, such as cell to entity matching, column to class matching and column pair to property matching. Traditional methods jointly solve all the matching tasks with their correlations considered using graphical models (Limaye, Sarawagi, and Chakrabarti 2010; Mulwad, Finin, and Joshi 2013; Bhagavatula, Noraset, and Downey 2015) or iterative procedures (Ritze, Lehmberg, and Bizer 2015; Zhang 2014; Zhang 2017). Considering the inter-matching correlation improves the disambiguation, but their metrics for computing the matchings *(i)* mostly adopt lexical comparisons which ignore the contextual semantics, and *(ii)* rely on metadata like column names and sometimes even external information like table description, both of which are often unavailable in real world applications.

Recent studies (Efthymiou et al. 2017) and (Luo et al. 2018) match column cells to KB entities with their contextual semantics considered using machine learning techniques like word embeddings and neural networks. With the matched entities, the column type can be further inferred using strategies like majority voting. However, such a cell to entity matching based column type annotation procedure assumes that the table cells have KB entity correspondences. Its performance will decrease when the column has a small number of cells, or there are missing or not accurate correspondences between the table cells and the KB entities, which we refer to as *knowledge gap*.

This study focuses on type annotation of entity columns, assuming table metadata like column names and table structures are unknown. We propose a neural network based framework named ColNet, as shown in Figure 1. It first embeds the overall semantics of columns into vector space and then predicts their types with a set of candidate KB classes, using machine learning techniques like Convolutional Neural Networks (CNNs) and an ensemble of results from lookup. To automatically train robust prediction models, we use the cells to retrieve the candidate KB classes, infer their entities to construct training samples, and deal with the challenge of sample shortage using transfer learning.

In summary, this study contributes a more accurate column type annotation framework by combining knowledge lookup and machine learning with the knowledge gap considered. As the framework does not assume any table metadata or table structure, it can be applied to not only web ta-

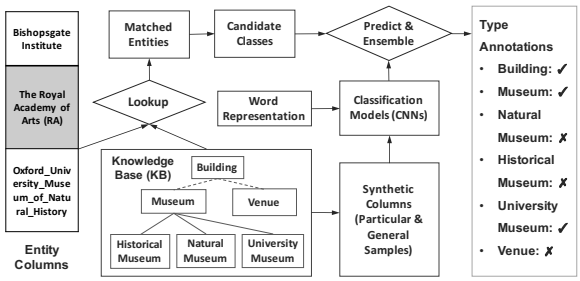

Figure 1: Column Type Annotation with ColNet

bles but also general tabular data. The study also provides a general approach that embeds the overall semantics of a column where the correlation between cells is incorporated. Our experiments with DBPedia and two different web table sets, T2Dv2 from the general Web (Lehmberg et al. 2016) and Limaye from Wikipedia pages (Limaye, Sarawagi, and Chakrabarti 2010) have shown that our method is effective and can outperform the state-of-the-art approaches.

Next section reviews the related work. Then we introduce the technical details of our approach. We finally present the evaluation, conclude the paper and discuss our future work.

Related Work

Annotating web tables with semantics from a KB has been studied for several years. It includes tasks of matching *(i)* table columns to KB classes, *(ii)* column pairs i.e., inter-column relationships to KB properties, and *(iii)* column cells to KB entities. More specifically, task *(i)* is equivalent to matching tables to KB classes while task *(iii)* is equivalent to matching rows to KB entities when one of the columns serves as a primary key (PK) which is defined as a column that can uniquely identify (most of) the table rows.

Collective Approaches

Collective approaches encompass multiple matching tasks and solve them together. We divide them into joint inference models and iterative approaches. (Limaye, Sarawagi, and Chakrabarti 2010) represents different matchings with a probabilistic graphical model and searches for value assignments of the variables that maximize the joint probability. (Mulwad, Finin, and Joshi 2013) extends this work with a more lightweight graphical model. TabEL (Bhagavatula, Noraset, and Downey 2015) weakens the assumption that columns and column pairs have KB correspondences, but assigns higher likelihood to entity sets that tend to co-occur in Wikipedia documents. (Venetis et al. 2011) proposes a maximum likelihood inference model that predicts the class of an column by maximizing the probability of all its cells. while (Chu et al. 2015) adopts a scoring model.

TableMiner+ (Zhang 2014; Zhang 2017) and T2K Match (Ritze, Lehmberg, and Bizer 2015) are two state-of-the-art iterative approaches. TableMiner adopts a bootstrapping pattern which first learns an initial interpretation with partial table data and then constrains the interpretation of the remaining data with the initial interpretation. T2K Match iteratively adjusts the weight of matchings until the overall similarity values converge. Early work (Syed et al. 2010) and (Mulwad et al. 2010) adopt a straightforward process which first determines the class of a column by matching its cells to KB entities and then refines the matchings with the column class.

These methods can achieve good performance on some datasets by jointly or iteratively determining multiple matchings with the inter-matching correlations modeled, but their performance will decrease on tabular data with cells provided alone, as they utilize some table metadata like column names and sometimes even external table information like table caption to calculate some correspondences. Meanwhile, these methods assume that all the table cells have corresponding entities in the KB, without considering the knowledge gap between them.

Column Type Annotation

Different from those collective work, some studies focus only on cell-to-entity matching (Zwicklbauer, Seifert, and Granitzer 2016; Efthymiou et al. 2017; Luo et al. 2018). The matched KB entities in turn can determine the column type with strategies like majority voting (Zwicklbauer et al. 2013). Such an approach can work with table contents alone, without relying on any metadata, but still ignoring the cases where a large part of cells have no entity correspondences.

A few studies take such knowledge gap cases into consideration. (Pham et al. 2016) utilizes machine learning and handcrafted features to predict the similarity between a target column and a seeding column whose type has been annotated. It actually transforms the gap between KB and target columns to the gap between seeding columns and target columns, with additional cost to annotate seeding columns. (Quercini and Reynaud 2013) does not match cells to entities but directly predicts the type of each cell by feeding the web page queried by the cell into a machine learning classifier. The idea is close to ours, but our method *(i) uses novel column semantic embedding and CNN-based locality feature learning for high accuracy*, and *(ii)* automatically trains machine learning models with samples inferred from a KB.

Semantic Embedding

Most table annotation methods calculate the degree of matchings with text comparison metrics like TF-IDF, Jaccard and so on, without considering the contextual semantics. The cell-to-entity matching by (Efthymiou et al. 2017) embeds cells and entities into vectors using word representations to introduce contextual semantics in prediction. It explores intra-cell semantic embedding (i.e., representing cells into vector space), but ignores inter-cell correlations.

As the locality correlation of tabular data is not as obvious as images and text, there are few studies learning inter-cell semantics. (Nishida et al. 2017) uses CNNs to learn locality features for table classification. The study presents that inter-cell correlations do exist and learning high level table features is meaningful. However, it differs from ours as it predicts a table structure type instead of semantic types, and uses manually labeled tables instead of automatic sample extraction from KBs to supervise the training.

Methodology

ColNet Framework

ColNet is a framework that utilizes a KB, word representations and machine learning to automatically train prediction models for annotating types of entity columns that are assumed to have no metadata. As shown in Figure 1, it mainly includes three steps, each of which will be explained in detail in the following subsections.

The first step is called *lookup*. Given an entity column, it retrieves column cells' corresponding entities in the KB and adopts the classes of the matched entities as a set of candidate classes for annotation. Meanwhile, this step generates labeled samples from the KB for model training, including particular samples which have a close data distribution to the column cells, and general samples that deal with the challenge of sample shortage caused by the big knowledge gap, small column size, etc.

The second step is called *prediction*, which calculates a score for each candidate class of a given column. For each candidate class, a customized binary CNN classifier which is able to learn both inter-cell and intra-cell locality features is trained and applied to predict whether cells of a column are of this class. In training, ColNet adopts word representations to incorporate contextual semantics and uses transfer learning to integrate particular and general samples.

The third step is called *ensemble*. Given a column and a candidate class, ColNet combines the vote from the matched entities of cells with the score predicted by the prediction model so as to keep the advantages of both. Cell to entity matching and voting with majority can contribute to a highly confident prediction, while prediction with CNNs, which considers the contextual semantics of words can deal with type disambiguation and recall cells missed by lookup.

Knowledge Lookup

In ColNet, we use a KB that is composed of a terminology and assertions. The former include classes (e.g., c_1 and c_2) and class relationships (e.g., $subClass(c_1, c_2)$), while the latter includes entities (e.g., e) and classification assertions (e.g., $c_1(e)$). The KB can support entity type inference and reasoning with the transitivity of $subClass$.

In sampling, we first retrieve a set of entities from the KB, by matching all the column cells with KB entities according to the entity label and entity anchor text using a lexical index. Those matched entities are called *particular entities*. The classes and super classes of each particular entity are inferred (via KB reasoning) and they are used as *candidate classes* for annotation, denoted as $\mathbb{C}$. The reason of selecting candidate classes instead of using all the KB classes is to avoid additional noise, thus reducing false positive predictions and computation. For each candidate class, we further infer all of its KB entities that are not matched. They are defined as *general entities*.

We also repeat the above lookup step with another round for refinement, inspired by (Syed et al. 2010; Mulwad et al. 2010). The second round uses each column's candidate classes from the first round to constrain the cell to entity matching, thus refining the entity suggestions. This step filters out some particular entities and candidate classes with limited matching confidence.

Synthetic Columns In training, ColNet automatically extracts labeled samples from the KB. A training sample $\boldsymbol{s} \doteq (\boldsymbol{e}, c)$ is composed of a synthetic column $\boldsymbol{e}$ and a class c in $\mathbb{C}$, while a synthetic column is constructed by concatenating a specific number of entities. This number is denoted as h. ColNet constructs a set of training samples by selecting different sets of entities with size h and concatenating entities in each set in different orders. In prediction, the input is a synthetic column that is constructed by concatenating table cells (cf. details in Prediction and Ensemble). Using the synthetic column as an input enables the neural network to learn the inter-cell correlation for some salient features like word co-occurrence, thus improving the prediction accuracy.

For example, given a column of IT Company that is composed of "Apple", "MS" and "Google", ColNet outputs a high score if the three cells are input as a synthetic column, but a low score if they are predicted individually and then averaged. This is because "Apple" and "MS" alone can be easily classified as other types like Fruit and Operating System, but will be correctly recognized as IT Company if their co-occurrence with "Google" is considered.

For another example, the combination of cell "Oxford University Museum of Natural History" and cell "British Museum" is more likely to be predicted as the right type Museum than the first cell alone, because the signal of Museum is augmented in the locality feature learned from the inter-cell word sequence "Museum", "of", "Natural", "History", "British" and "Museum".

Sampling For each candidate class c in $\mathbb{C}$, both positive and negative training samples are generated, where a sample is defined as positive if each entity in its synthetic column is inferred as an instance of c, and negative otherwise. To fully capture the hyperplane that can distinguish column cells for the class, we develop a table-adapted negative sampling approach. Given the candidate class c, ColNet first finds out its neighboring candidate classes, each of which is defined to be a specific column's candidate class co-occurring with c. Then ColNet uses the entities that are of a neighboring class of c but are not of class c to construct the synthetic column of negative samples.

Meanwhile, ColNet extracts two sample sets for each candidate class c. They are i) *particular samples* which are constructed with particular entities, denoted as $\boldsymbol{S}_p$, and ii) *general samples* which are constructed with general entities, denoted as $\boldsymbol{S}_g$. For example, given the above column of IT Company and its general entities retrieved from DBPedia "dbr:Google", "dbr:Apple", "dbr:Apple_Inc." and "dbr:Microsoft_Windows", synthetic columns constructed with "dbr:Google" and "dbr:Apple_Inc." (resp. "dbr:Apple" and "dbr:Microsoft_Windows") are particular positive (resp. negative) samples of IT Company, while synthetic columns constructed by general entities like "dbr:Amazon.com" and "dbr:Alibaba_Group" are general positive samples.

Compared with $\boldsymbol{S}_g$, $\boldsymbol{S}_p$ has a closer data distribution to the column cells, and therefore is able to make the models

adaptive to the prediction data. However, because of the big knowledge gap, short column size, ambiguous cell to entity matching, etc., $\boldsymbol{S}_p$ in many cases is too small to train robust classifiers, especially for those complex models like CNN. Consequently, we use transfer learning to incorporate both $\boldsymbol{S}_g$ and $\boldsymbol{S}_p$ in training (cf. details in the next subsection).

Model Training

Synthetic Column Embedding In model training, we first embed each synthetic column into a real valued matrix using word representation models like word2vec (Mikolov et al. 2013), so as to feed it into a machine learning algorithm with the contextual semantics of words incorporated.

The label of each entity is first cleaned (e.g., removing the punctuation) and split into a word sequence. Then the word sequences of all the entities of a synthetic column are concatenated into one. To align word sequences of different synthetic columns, their lengths are fixed to a specific value n which is set to the length of the longest word sequence. Those abnormally long sequences are not considered in setting n. A word sequence shorter than n is padded with "NULL" whose word representation is a zero vector, while those that are longer than n are cropped.

Briefly, the word sequence of a synthetic column $\boldsymbol{e}$ is denoted as $ws(\boldsymbol{e}) = [word_1, word_2, \cdots, word_n]$, and its embedded matrix is calculated as

$$\boldsymbol{x}(\boldsymbol{e}) = v(word_1) \oplus v(word_2) \cdots \oplus v(word_n) \quad (1)$$

where $v(\cdot)$ represents d dimension word representation and $\oplus$ represents stacking two vectors. For example, considering a synthetic column composed of DBpedia entities dbr:Bishopsgate_Institute and dbr:Royal_Academy_of_Arts, its matrix is the stack of the word vectors of "Bishopsgate", "Institute", "Royal", "Academy", "of", "Arts" and two zero vectors, where we assume n is fixed to 8.

Neural Network We use a CNN to predict the type of a synthetic column, inspired by its successful application in text classification (Kim 2014). For each candidate class c in $\mathbb{C}$, one binary CNN classifier $\mathcal{M}_c$ is trained.

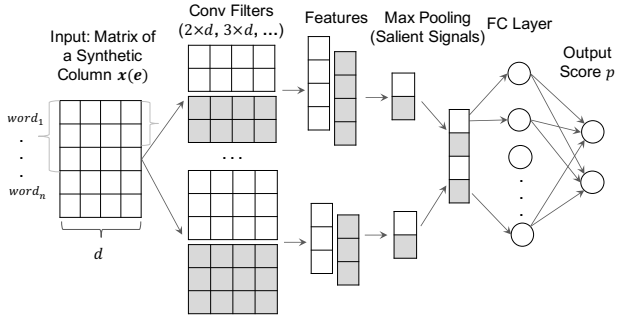

Figure 2: The CNN architecture used in ColNet.

As shown in Figure 2, the CNN architecture includes one convolutional (Conv) layer which is composed of multiple filters (i.e., convolution operations over the input matrix) with a fixed width (i.e., word vector dimension d) but different heights. For each Conv filter $\boldsymbol{w}$, one feature vector, with a specific granularity of locality, is learned:

$$\boldsymbol{f} = g(\boldsymbol{w} \otimes \boldsymbol{x}(\boldsymbol{e}) + \boldsymbol{b}) \quad (2)$$

where $\boldsymbol{b}$ is a bias vector, $\otimes$ represents the convolution operation and $g(\cdot)$ is an activation function (e.g., ReLU). Considering a convolution filter with size of $k \times d$, the dimension of its feature vector $\boldsymbol{f}$ is $n - k + 1$, and its i$^{\text{th}}$ element is calculated as $f_i = g(\boldsymbol{w} \cdot \boldsymbol{x}(\boldsymbol{e})_{i:i+k-1} + b_i)$, where $\cdot$ represents the element-wise matrix multiplication.

After the Conv layer, a max pooling layer which selects the maximum value of each feature vector and further concatenates all the maximum values is stacked:

$$\boldsymbol{f}' = \left[max(\boldsymbol{f}^1), max(\boldsymbol{f}^2), ..., max(\boldsymbol{f}^m)\right] \quad (3)$$

where m is the number of convolution filters and $\boldsymbol{f}^j$ is the feature vector of j$^{\text{th}}$ filter.

Intuitively, $\boldsymbol{f}'$ can be regarded as the salient signals of the learned features with regard to the specific classification task. For example, considering the input word sequence composed of "Oxford", "University", "Museum", "of", "Natural" and "History", the max pooling layer highlights the signal of "Museum" in training a binary CNN classifier for the class Museum and highlights the signal of "University" for the class Educational Institute. The max pooling layer also reduces the complexity of the network, playing a role of regularization.

A fully connected (FC) layer which learns the nonlinear relationship between the input and output is further stacked:

$$\boldsymbol{y} = g(\boldsymbol{f}' \times \boldsymbol{w}' + \boldsymbol{b}') \quad (4)$$

where $\boldsymbol{w}'$ and $\boldsymbol{b}'$ are the weight matrix and bias vector to learn, $\times$ represents the matrix multiplication operation. For binary classification, the dimensions of $\boldsymbol{w}'$ and $\boldsymbol{b}'$ are $m \times 2$ and 1×2 respectively. With $\boldsymbol{y}$, a Softmax layer is eventually added to calculate the output score p. No regularizations are added to the FC layer, as the max pooling is already able to prevent the network from over fitting.

Transfer Learning The training of each CNN classifier incorporates both general sample set $\boldsymbol{S}_g$ and particular sample set $\boldsymbol{S}_p$. ColNet first pre-trains the CNN with $\boldsymbol{S}_g$, and then fine tunes its parameters with $\boldsymbol{S}_p$. To make the model fully adapted to the table data, the iteration number of fine tuning is set to be inversely proportional to the size of $\boldsymbol{S}_p$. Specially, when there are no matched KB entities, we can reuse candidate classes from other columns in the table set and train the classifiers using $\boldsymbol{S}_g$ alone. For example, consider a column with (yet unknown) researcher names like "Ernesto Jimenez-Ruiz" and DBPedia as a KB. The particular sample set $\boldsymbol{S}_p$ is empty as there are not DBPedia correspondences for this column. ColNet, however, may still be able to predict a type for such a column, by relying on general entities extracted from other columns such as "dbr:Ernesto_Sabato" (of type dbo:Person).

Prediction

For each column, ColNet uses the trained CNNs of its candidate classes to predict its types. Synthetic columns are first extracted. However, traversing all the cell combinations costs exponential computation time, which is impractical. ColNet samples N testing synthetic columns by *(i)* sliding a window with size of h over the column, and *(ii)* randomly selecting ordered cell subsets with size of h from the column. N is often set to a large number for a high coverage and stable predictions. Each sampled synthetic column is embedded into a matrix by the same way used in model training, and then predicted by model $\mathcal{M}_c$ for each candidate class c of the column: $p_k^c \xleftarrow{\mathcal{M}_c} \boldsymbol{x}(\boldsymbol{e}_k)$, where $k = 1, \cdots, N$ and p_k^c is a score in $[0, 1]$. ColNet eventually averages all the scores as the prediction score: $p^c = \frac{1}{N}\sum_{k=1}^{N} p_k^c$.

Ensemble

We integrate the prediction from ColNet with the vote by KB entities that are retrieved by column cells. The latter method, denoted as Lookup-Vote is widely used for column type annotation (e.g., (Zwicklbauer et al. 2013)). Given a target column, it first matches cells to entities according to a lexical index, and then uses the rate of cells that have entity correspondences of class c as the score of annotating the column with c, denoted as v^c.

There have been many methods for combing multiple classifiers (Ponti Jr 2011). We use a customized rule that can utilize the advantage of both ColNet and Lookup-Vote. For a candidate class c, we combine p^c and v^c as follows:

$$s^c = \begin{cases} v^c, & \text{if } v^c \geq \sigma_1 \text{ or } v^c < \sigma_2 \\ p^c, & \text{otherwise} \end{cases} \quad (5)$$

where σ_1 and σ_2 are two hyper parameters in $[0, 1]$, and $\sigma_1 \geq \sigma_2$. The rule accepts classes supported by a large part of cells (i.e., $v^c \geq \sigma_1$) and rejects classes supported by few cells (i.e., $v^c < \sigma_2$). By setting an intermediate or high value (e.g., 0.5) to σ_1 and a small value (e.g., 0.1) to σ_2, the rule helps achieve a high precision. For the classes with less confidence from voting (i.e., $\sigma_2 \leq v^c < \sigma_1$), it adopts the predicted score, which helps recall some classes that have no entity correspondences. In final decision making, the column is annotated by class c if $s^c \geq \alpha$, and not otherwise, where α is a threshold hyper parameter in $[0, 1]$. The optimized setting of σ_1, σ_2 and α can be searched with small steps.

Evaluation

Experiment Settings

We use DBPedia (Auer et al. 2007) and two web table datasets T2Dv2 and Limaye for our experiments. T2Dv2 includes common tables from the Web,[2] with 237 PK entity columns, each of which is annotated by a fine-grained DBPedia class. We call such fine-grained classes as "best" classes, while their super classes which are right but not perfect as "okay" classes. We further extend T2Dv2 by *(i)* annotating its 174 non-PK entity columns with "best" classes and *(ii)* inferring "okay" classes of all the columns. Limaye contains tables from Wikipedia pages. We adopt the version published by (Efthymiou et al. 2017) with 428 PK entity columns, manually annotate these columns with "best" classes and infer the "okay" classes. Some statistics of T2Dv2 and Limaye are shown in Table 1.

Name	Columns	Avg. Cells	Different "Best" ("Okay") Classes
T2Dv2	411	124	56 (35)
Limaye	428	23	21 (24)

Table 1: Some statistics of the web table sets.

In the experiment, we adopt the DBpedia lookup service to retrieve particular entities. The service, which is based on an index of DBPedia Spotlight (Mendes et al. 2011), returns DBpedia entities that match a given text phrase. The DBPedia SPARQL endpoint is used to infer an entity's class and super classes. A word2vec model trained with the latest dump of Wikipedia articles is used. Each classifier is trained within 2 minutes on our workstation with Xeon CPU E5-2670, with our Tensforflow implementation[3]. Efficiency and scalability will be improved and evaluated in future work. We evaluate two aspects: *(i)* the overall performance of ColNet on column type annotation, and *(ii)* the impact of learning techniques on the prediction models (CNNs).

Overall Performance

We use precision, recall and F1 score to measure the overall performance of ColNet under both "strict" and "tolerant" models. Given a target column, the "tolerant" model equally counts each of its predictions, while the "strict" model counts its predictions if the "best" class is hit and directly regards all of them as false positives otherwise. On both table datasets, ColNet, with and without ensemble (i.e., s^c and p^c), is evaluated and compared with Lookup-Vote (i.e., v^c) and T2K Match[4] (Ritze, Lehmberg, and Bizer 2015) whose authors developed T2Dv2. On Limaye, ColNet is further compared with a voting method using entities matched by (Efthymiou et al. 2017), named Efthymiou17-Vote. On both table datasets, σ_1 and σ_2 are set to 0.5 and 0.08, while α has been adjusted and set to a value with the highest F1 score. α is set to 0.45, 0.55, 0.2, 0.2 and 0.1 for $\text{ColNet}_{\text{Ensemble}}$, ColNet, Lookup-Vote, T2K Match and Efthymiou17-Vote respectively. The results are shown in Table 2 and Table 3.[5]

Prediction Impact We first present the impact of prediction models by comparing $\text{ColNet}_{\text{Ensemble}}$ with Lookup-Vote. On T2Dv2, $\text{ColNet}_{\text{Ensemble}}$ has 2.3% and 0.8% higher F1 score under "tolerant" and "strict" models, while on Limaye, the corresponding improvements by integrating prediction are 15.0% and 23.8%. The comparison also verifies that the prediction can improves the recall as it can predict the type of columns that lack entity correspondences. The

[2] http://webdatacommons.org/webtables/goldstandardV2.html

[3] https://github.com/alan-turing-institute/SemAIDA

[4] Runnable system from https://goo.gl/AGj3dg

[5] Note that the results reported in (Ritze 2017) are different as they use a more tolerant calculation of precision and recall.

Models	Methods	All Columns	PK Columns
Tolerant	ColNet$_{\text{Ensemble}}$	0.917, 0.909, 0.913	0.967, 0.985, 0.976
	ColNet	0.845, 0.896, 0.870	0.927, 0.960, 0.943
	Lookup-Vote	0.909, 0.865, 0.886	0.965, 0.960, 0.962
	T2K Match	0.664, 0.773, 0.715	0.738, 0.895, 0.809
Strict	ColNet$_{\text{Ensemble}}$	0.853, 0.846, 0.849	0.941, 0.958, 0.949
	ColNet	0.765, 0.811, 0.787	0.868, 0.898, 0.882
	Lookup-Vote	0.862, 0.821, 0.841	0.946, 0.941, 0.943
	T2K Match	0.624, 0.727, 0.671	0.729, 0.884, 0.799

Table 2: Results (precision, recall, F1 score) on T2Dv2.

Models	Methods	PK Columns
Tolerant	ColNet$_{\text{Ensemble}}$	0.796, 0.799, 0.798
	ColNet	0.763, 0.820, 0.791
	Lookup-Vote	0.732, 0.660, 0.694
	T2K Match	0.560, 0.408, 0.472
	Efthymiou17-Vote	0.759, 0.414, 0.536
Strict	ColNet$_{\text{Ensemble}}$	0.602, 0.639, 0.620
	ColNet	0.576, 0.619, 0.597
	Lookup-Vote	0.571, 0.447, 0.501
	T2K Match	0.453, 0.330, 0.382
	Efthymiou17-Vote	0.626, 0.357, 0.454

Table 3: Results (precision, recall, F1 score) on Limaye.

average recall improvement is around 3.9% on T2Dv2 and around 32.0% on Limaye, each of which is much higher than the corresponding F1 score improvement. Meanwhile, we can also find ColNet (pure prediction) has higher F1 score, precision and recall than Lookup-Vote on Limaye which has a small average column size and is hard to be voted with entity correspondences. The F1 score outperforming is 14.0% and 19.2% under "tolerant" and "strict" models.

Ensemble Impact ColNet with an ensemble of lookup (ColNet$_{\text{Ensemble}}$) achieves higher F1 score than ColNet without ensemble on both table sets. For example, the ensemble benefits ColNet with 7.9% and 4.9% F1 score improvements on all columns of T2Dv2 under "strict" and "tolerant" models. Actually, ColNet$_{\text{Ensemble}}$ also outperforms ColNet on precision and recall in all the cases except for the recall on Limaye under "tolerant" model. Meanwhile, the results show that the improvement on precision is more significant than on recall. In the two cases of the above example, the precision (recall) improvements by ensemble are 11.5% and 5.4% (4.3% and 1.6%). This phenomena verifies that integrating the score from Lookup-Vote with our ensemble rule improves the precision.

Comparison with The State-of-the-art First, both ColNet and ColNet$_{\text{Ensemble}}$ outperforms T2K Match on precision, recall and F1 score in all the cases. For example, the F1 score outperforming by ColNet$_{\text{Ensemble}}$ is 27.7% (20.6%) on all (PK) columns of T2Dv2 under "tolerant" model. One potential reason is that the matchings in T2K Match, which ignore contextual semantics of words are ambiguous. Second, ColNet$_{\text{Ensemble}}$ and ColNet also outperform Efthymiou17-Vote. On Limaye, F1 score of ColNet$_{\text{Ensemble}}$ is 48.9% and 36.6% higher than Efthymiou17-Vote under "tolerant" and "strict" models respectively. Efthymiou17-Vote has competitive precision but much lower recall, because a large part of cells have no entity correspondences. In ColNet, for the cells without entity correspondences, the lookup part can get close entities with the same or overlapping classes, while the prediction part which considers the contextual semantics then accurately predict these cells' classes.

Column Size Impact By comparing Table 2 with Table 3 (results on two different data sets), we find that all the methods are more accurate on T2Dv2 than on Limaye, although the former has more "best" and "okay" ground truth classes. For example, ColNet$_{\text{Ensemble}}$ has 14.4% and 36.9% higher F1 score on Limaye under "tolerant" and "strict" models. One potential reason is that the average cell number per column in T2Dv2 is much larger than that in Limaye (i.e., 124 vs 23). This provides more evidence for prediction and voting. The results also show that the benefits of the prediction enhanced methods (ColNet and ColNet$_{\text{Ensemble}}$) over the cell to entity matching based methods (Lookup-Vote, T2K Match and Efthymiou17-Vote) are much more significant on Limaye than on T2Dv2. Considering F1 score under "tolerant" model, ColNet$_{\text{Ensemble}}$ outperforms Lookup-Vote by 3.0% on T2Dv2, but by 15.0% on Limaye.

The Prediction Models

We further evaluate the CNNs with the impact of synthetic columns, knowledge gap and transfer learning. To this end, we extract labeled synthetic columns as the testing samples, and divide the candidate classes into truly matched (TM) if they are among the ground truths, and falsely matched (FM) otherwise. For TM classes, we adopt Area Under ROC Curve (AUC) as the metric, while for FM classes, we use the average score (AS) of testing samples as only negative testing samples can be extracted. The higher AUC or the lower AS, the better performance. Results on four kinds of T2Dv2 columns are reported in Figure 3 and 4.

Synthetic Columns Figure 3 shows that the performance of CNNs with respect to both TM classes and FM classes mostly increases as the synthetic column size increases, especially from 1 to 4. For example, the average AUC of TM classes of Person increases from around 0.93 to 0.97, while the average AS of its FM classes drops from around 0.35 to 0.22. This phenomenon verifies that classification of synthetic columns is more accurate than classification of cells. The synthetic column on one hand provides more evidence, on the other hand enables the CNN to learn additional inter-cell locality features. Figure 3 also indicates that 4 is an optimized synthetic column size setting on T2Dv2.

Knowledge Gap Figure 4 shows that the performance of CNNs on both TM classes and FM classes significantly drops as the knowledge gap increases, especially when no transfer from general samples is conducted. For example, in the case without transfer, the average AUC of TM classes of Place, Person, Species and Organization drops by 7.2%, 4.4%, 4.8% and 8.1% respectively, when the ratio of particular entities drops from 1.0 to 0.1. When only 0.1 of particular entities are used, the average AS of FM classifiers

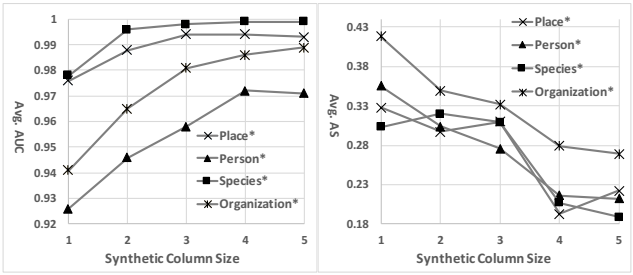

Figure 3: The performance of CNNs on TM classes [left] and FM classes [right] under different synthetic column sizes, trained by particular samples.

increases to higher than 0.5, which means the classifiers predict over half of the negative testing samples as positive. Such a performance drop is mainly caused by underfitting in training, due to particular sample shortage. The performance drop of CNNs on FM classes is more significant, because the FM classes, introduced by incorrect cell to entity matchings, have a smaller number of particular entities. These results support the fact that ColNet and $\text{ColNet}_{\text{Ensemble}}$ perform worse on Limaye than on T2Dv2, since the former has small column size with fewer entity correspondences.

Transfer Learning Figure 4 shows that transfer learning with general samples significantly benefits the CNNs, especially when the knowledge gap is large. When the ratio of particular entities used in training is set to 0.1, 0.25, 0.5, 0.75 and 1.0, the average improvements of CNNs of TM (FM) classes are 0.9%, 0.8%, 1.7%, 2.7% and 6.5% (56.4%, 68.7%, 70.2%, 71.7% and 77.7%) respectively. Figure 4 also shows that particular entities are essential in training. For example, considering Organization, training with both particular and general samples for FM classes achieves 25.8% lower average AS than training with general samples alone. Fine tuning with particular samples helps bridge the data distribution gap between column cells and KB entities.

Discussion

On the one hand, we analyze the impact of the knowledge gap. By comparing the performance on two different web table datasets that have a big gap in average column size, we find the shorter columns, which have less entity correspondences in average, are harder to be annotated. ColNet outperforms the latest collective approach T2K Match and two cell-to-entity matching based approaches Lookup-Vote and Efthymiou17-Vote, especially on shorter columns. This indicates that ColNet can be distinguished from the other methods by dealing with the knowledge gap for higher performance. It is further verified by the analysis on CNNs' performance under different simulated knowledge gaps. We could not compare our approach to TableMiner+ (Zhang 2017), T2K Match++ (Ritze 2017), and Efthymiou17-Vote on T2Dv2 (Efthymiou et al. 2017) as no runnable code was available at the time of conducting the evaluation.

On the other hand, we evaluate the impact of synthetic column size, which indicates that embedding the semantics

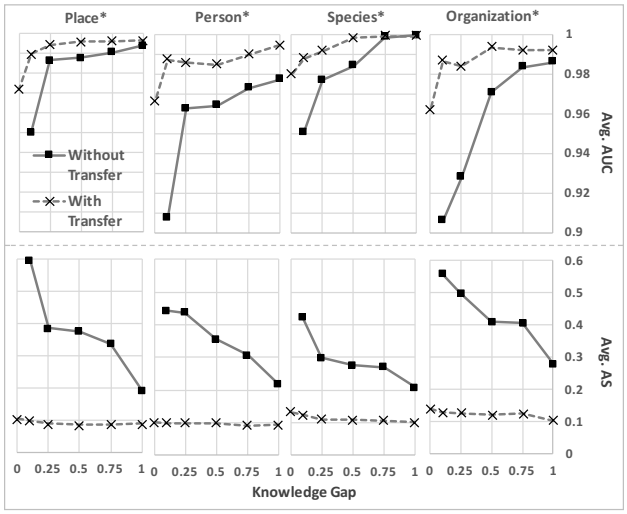

Figure 4: The performance of CNNs of TM classes [above] and FM classes [below], under different knowledge gaps, with and without transfer learning. Knowledge gap is simulated by randomly selecting a ratio of particular entities for training. The lower ratio, the larger gap.

of columns and learning locality features cross cells can improve prediction models' performance. We do not compare ColNet with (Nishida et al. 2017) and (Luo et al. 2018), although they also learn locality features of a table. There are two reasons. First, ColNet targets a different table annotation task and holds a different input assumption, where tabular data are provided column by column without any structure information. Second, ColNet automatically supervises the learning of prediction models by KB lookup and reasoning while those two studies use labeled tables. The former encounters some additional challenges like synthetic column construction, sample shortage and so on.

Conclusion and Outlook

The paper presents a neural network and semantic embedding based column type prediction framework named ColNet. Different from existing methods, it *(i)* utilizes column cells alone without assuming any table metadata or table structures, thus being able to be extended to any tabular data, *(ii)* learns both cell level and column level semantics with CNNs and word representation for high accuracy, *(iii)* automatically trains prediction models utilizing KB lookup and reasoning as well as machine learning methods like transfer learning, and *(iv)* takes the knowledge gap into consideration, thus being able to deal with growing web tables or be applied in populating KBs with new tabular data. The evaluation on two different web table sets T2Dv2 and Limaye under both "tolerant" and "strict" models verifies the effectiveness of ColNet and shows that it can outperform the state-of-the-art approaches.

In the future, we will extend column type annotation to other related tasks like property annotation and further study learning table locality features with semantic reasoning.

Acknowledgments

The work is supported by the AIDA project (UK Government's Defence & Security Programme in support of the Alan Turing Institute), the SIRIUS Centre for Scalable Data Access (Research Council of Norway, project 237889), the Royal Society, EPSRC projects DBOnto, MaSI^3 and ED^3.

References

Auer, S.; Bizer, C.; Kobilarov, G.; Lehmann, J.; Cyganiak, R.; and Ives, Z. 2007. Dbpedia: A nucleus for a web of open data. *The Semantic Web* 722–735.

Bhagavatula, C. S.; Noraset, T.; and Downey, D. 2015. Tabel: entity linking in web tables. In *International Semantic Web Conference*, 425–441. Springer.

Cafarella, M. J.; Halevy, A.; Wang, D. Z.; Wu, E.; and Zhang, Y. 2008. Webtables: exploring the power of tables on the web. *Proceedings of the VLDB Endowment* 1(1):538–549.

Chu, X.; Morcos, J.; Ilyas, I. F.; Ouzzani, M.; Papotti, P.; Tang, N.; and Ye, Y. 2015. Katara: A data cleaning system powered by knowledge bases and crowdsourcing. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, 1247–1261. ACM.

Efthymiou, V.; Hassanzadeh, O.; Rodriguez-Muro, M.; and Christophides, V. 2017. Matching web tables with knowledge base entities: from entity lookups to entity embeddings. In *International Semantic Web Conference*, 260–277. Springer.

Kim, Y. 2014. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 1746–1751.

Lehmberg, O.; Ritze, D.; Meusel, R.; and Bizer, C. 2016. A large public corpus of web tables containing time and context metadata. In *Proceedings of the 25th International Conference Companion on World Wide Web*, 75–76.

Limaye, G.; Sarawagi, S.; and Chakrabarti, S. 2010. Annotating and searching web tables using entities, types and relationships. *Proceedings of the VLDB Endowment* 3(1-2):1338–1347.

Luo, X.; Luo, K.; Chen, X.; and Q., Z. K. 2018. Cross-lingual entity linking for web tables. In *AAAI*, 362–369.

Mendes, P. N.; Jakob, M.; García-Silva, A.; and Bizer, C. 2011. Dbpedia spotlight: shedding light on the web of documents. In *Proceedings of the 7th international conference on semantic systems*, 1–8. ACM.

Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G. S.; and Dean, J. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, 3111–3119.

Mulwad, V.; Finin, T.; Syed, Z.; Joshi, A.; et al. 2010. Using linked data to interpret tables. In *Proceedings of the the First International Workshop on Consuming Linked Data*.

Mulwad, V.; Finin, T.; and Joshi, A. 2013. Semantic message passing for generating linked data from tables. In *International Semantic Web Conference*, 363–378. Springer.

Nishida, K.; Sadamitsu, K.; Higashinaka, R.; and Matsuo, Y. 2017. Understanding the semantic structures of tables with a hybrid deep neural network architecture. In *AAAI*, 168–174.

Pham, M.; Alse, S.; Knoblock, C. A.; and Szekely, P. 2016. Semantic labeling: a domain-independent approach. In *International Semantic Web Conference*, 446–462. Springer.

Ponti Jr, M. P. 2011. Combining classifiers: from the creation of ensembles to the decision fusion. In *Graphics, Patterns and Images Tutorials (SIBGRAPI-T), 2011 24th SIBGRAPI Conference on*, 1–10. IEEE.

Quercini, G., and Reynaud, C. 2013. Entity discovery and annotation in tables. In *Proceedings of the 16th International Conference on Extending Database Technology*, 693–704. ACM.

Ritze, D.; Lehmberg, O.; Oulabi, Y.; and Bizer, C. 2016. Profiling the potential of web tables for augmenting cross-domain knowledge bases. In *Proceedings of the 25th International Conference on World Wide Web*, 251–261.

Ritze, D.; Lehmberg, O.; and Bizer, C. 2015. Matching html tables to dbpedia. In *Proceedings of the 5th International Conference on Web Intelligence, Mining and Semantics*, 10. ACM.

Ritze, D. 2017. *Web-Scale Web Table to Knowledge Base Matching*. Ph.D. Dissertation, University of Mannheim, Germany.

Sun, H.; Ma, H.; He, X.; Yih, W.-t.; Su, Y.; and Yan, X. 2016. Table cell search for question answering. In *Proceedings of the 25th International Conference on World Wide Web*, 771–782.

Syed, Z.; Finin, T.; Mulwad, V.; Joshi, A.; et al. 2010. Exploiting a web of semantic data for interpreting tables. In *Proceedings of the Second Web Science Conference*.

Venetis, P.; Halevy, A.; Madhavan, J.; Paşca, M.; Shen, W.; Wu, F.; Miao, G.; and Wu, C. 2011. Recovering semantics of tables on the web. *Proceedings of the VLDB Endowment* 4(9):528–538.

Zhang, M.; Hadjieleftheriou, M.; Ooi, B. C.; Procopiuc, C. M.; and Srivastava, D. 2010. On multi-column foreign key discovery. *Proceedings of the VLDB Endowment* 3(1-2):805–814.

Zhang, Z. 2014. Towards efficient and effective semantic table interpretation. In *International Semantic Web Conference*, 487–502. Springer.

Zhang, Z. 2017. Effective and efficient semantic table interpretation using tableminer+. *Semantic Web* 8(6):921–957.

Zwicklbauer, S.; Einsiedler, C.; Granitzer, M.; and Seifert, C. 2013. Towards disambiguating web tables. In *International Semantic Web Conference*, 205–208.

Zwicklbauer, S.; Seifert, C.; and Granitzer, M. 2016. Doser-a knowledge-base-agnostic framework for entity disambiguation using semantic embeddings. In *European Semantic Web Conference*, 182–198. Springer.