Generating Table Vector Representations

Aneta Koleva[1,2], Martin Ringsquandl[1], Mitchell Joblin[1] and Volker Tresp[1,2]

[1]Siemens, Otto-Hahn-Ring 6, 81739 Munich, Germany
[2]Ludwig Maximilian University of Munich, Geschwister-Scholl-Platz 1, 80539 Munich, Germany

Abstract

High-quality Web tables are rich sources of information that can be used to populate Knowledge Graphs (KG). The focus of this paper is an evaluation of methods for table-to-class annotation, which is a sub-task of Table Interpretation (TI). We provide a formal definition for table classification as a machine learning task. We propose an experimental setup and we evaluate 5 fundamentally different approaches to find the best method for generating vector table representations. Our findings indicate that although transfer learning methods achieve high F1 score on the table classification task, dedicated table encoding models are a promising direction as they appear to capture richer semantics.

Keywords

table interpretation, table classification, representation learning.

1. Introduction

Tabular data is one of the most prevalent data representations. The effort by Cafarella [1], known as WebTables, identified and extracted more than 200 million high-quality tables from HTML pages. The availability of such large corpus of structured data initiated several directions of research related to the different applications of tabular data such as: table search [2], table improvement [3], question answering [4], and semantic annotation of columns [5]. As a result of the increasing adoption of KGs, which are often populated from tabular data, the task of aligning tables with KGs, also referred to as table interpretation (TI), has become a highly relevant task. In contrast to information extraction from unstructured documents, TI should leverage the explicit relational structure. The unique table structure with rows and columns of cells and other *metadata* can be exploited for discovery and disambiguation of the meaning captured in the table. The task of TI entails three different sub-tasks. The first sub-task, which is the focus in this paper, is the classification of tables according to classes in a given KG schema. The second sub-task is related to linking rows from tables to existing entities in the KG. The annotation of columns as entity attributes and the discovery of binary relations between columns is the third sub-task of TI. While there have been several works focusing on the row-to-entity [6, 7, 8], and column-to-attribute sub-tasks [5, 9], the task of linking a table to a class has been neglected. However, in the case of *entity tables*, where one column (the *core column*) is associated to the name of the entity and the remaining columns are attributes of this entity, discovering the class of the table as a first step can greatly improve the solving of the other two sub-tasks. It is often the case that the column names are missing or incorrect, therefore finding the name of the core

ISWC 2021 Workshop DL4KG
firstname.lastname@siemens.com

column does not imply finding the class of the table. Moreover, when two tables have the same column names and similar content (e.g., one table of class *Country* and one of class *City*), it is not trivial to disambiguate the entities and column types based only on the table content. Once a table has been *interpreted*, its content can be used for extracting new triples for enriching the KG, a task known as *KG completion*, or for extracting missing facts for the KG, which is the task of *slot-filling*.

Due to the inherent scarcity of labelled data for the first sub-task (class-annotated tables), a table classification model must either be of low complexity (few parameters) or leverage pre-trained models. Using pre-trained models in TI has been studied only to a very limited extend. Hence, we explore two promising directions for making learning-based approaches more efficient: (a) by using transfer learning, (b) by considering additional inductive biases that are unique to tabular data representations.

We propose an experimental setup with the intention of finding the best method for generating a representation which captures the information from the table but also the row and column structure, so that it can be later used towards solving the remaining sub-tasks of TI: row-to-entity linking, column type annotation and relation extraction. We are interested in understanding how pre-trained language models, such as BERT [10], and their dedicated table-based counterparts, for instance TaBERT [11], can be utilized for generating vector representation for table. Surprisingly, our experiments show that a transfer learning method with a rich vocabulary of pre-trained word embeddings achieves similar F1 score compared to more sophisticated pre-trained language models (LM). Another interesting finding is that the inductive bias for tabular structure in the LM pre-trained on tabular data does not bring beneficial impact to a text pre-trained LM. However, the classification confusion matrix for this method, gives an insight to the miss-classifications being justifiable and reasonable. Our main contributions are:

- A formal definition of table classification as a machine learning task and a protocol for evaluating performance on this task.
- A setup for table encoding using 5 fundamentally different approaches covering a spectrum of paradigms from general purpose document encoders to specialized pre-trained models designed for tabular data.
- An extensive empirical evaluation of the different approaches.

2. Background

In this section, we review prior work related to solving the different sub-tasks of TI. We also give a short overview of methods for generating vector representations of tables.

Table Interpretation The three sub-tasks of TI were first introduced in the paper by Ritze et al. [12]. That paper also introduced the T2K Matcher, a method for iterative value-based matching, which solves the TI tasks by matching values from the tables to values of retrieved candidates from the KG. More recent work by Limaye et al. [9] proposed a probabilistic graphical method which attempts to jointly solve the two sub-tasks of finding entity-to-row and column-to-attribute alignments. Deng et al. [13] exploited word embeddings for representing the contents of tables and utilized them for the discovery of new entities. The SemTab challenge

[14] has also motivated new approaches [15, 16]. However, the task of table-to-class annotation is not part of this challenge.

Table classification To the best of our knowledge, the T2K Matcher is the only existing method for solving the table-to-class task. Namely, the class of the table is chosen by ranking the sum of the similarity scores of the column-to-property correspondences aggregated per class. Since this method requires querying of the KG for candidate retrieval and first solving the column-to-property alignment in order to find the correct class of a table, we do not consider it during our experiments. In contrast to the T2K Matcher, we consider a *closed book* scenario, where the instances of the KG are not available, only the classes in the KG schema.

Representation Learning on Tables Based on powerful LM, dedicated *deep learning* models have recently been proposed to exploit tabular data structures, e.g., in table-based question answering [4, 17] and KG completion from tables [18]. One benefit from using pre-trained LM is that they can handle synonyms well, e.g., the abbreviation of New York as NY, which are frequently occurring in tables because of the innate limitation of the cells. The other benefit is that, due to the exposure to large textual corpora during the pre-training phase, the LM can *store* implicit information learned from the data whilst pre-training, in the form of model parameters [19]. TaBERT [11] by Yin et al. is a novel model which was pre-trained to jointly learn representation of a natural language question, called *utterance*, and tables. An example of utterance for the entity table shown in Figure 1 is the question: *How much is the population of New York?*. During encoding, instead of using the full table, TaBERT samples 1 or 3 rows, referred to as *content snapshot*. First, each row from the snapshot, concatenated with the utterance, is encoded by BERT [10]. Second, the encoding of the rows are stacked and in order to generate vector representations for each of the columns, a vertical self-attention mechanism is used. Finally, representation for the table is generated by pooling the column representations. Similar work is the method TAPAS by Herzig et al. [20], which is also pre-trained on tables and text segments. Ding et al. proposed TURL [17] as a framework for pre-training, also on tabular data, which uses the same objectives as TaBERT for learning representations of the content of the tables. Additionally, they proposed task-specific fine-tuning on the framework for solving the row-to-entity and column-to-attribute annotation. Wang et al. [21] presented a novel method which exploits information within one table but also aggregates the contextual information shared across similar tables in order to generate a vector representation that can be used for column-to-class annotation and relation prediction tasks.

3. Problem Description

We focus on the task of table-to-class annotation. The task has been introduced together with the two other TI sub-tasks in [12], however without a formal definition. The goal of the table-to-class annotation is to label a table with its corresponding class according to the given KG schema. We now provide a definition of this task as a machine learning task.

An entity table $\boldsymbol{T}_i$ is a $N_i \times M_i$ matrix where N_i and M_i are the number of rows and columns of the table $\boldsymbol{T}_i$. Each element of the matrix $\boldsymbol{T}_i$, $r^i_{n,m}$, contains one or more tokens, where each token

is a sequence of characters. We denote with $r^i_{n,*}$ and $r^i_{*,m}$ the n-th row and the m-th column of the matrix T_i respectively. The header of the table is the first row $H_i = r^i_{0,*}$. The content of the table are the rows $r^i_{1,*}, r^i_{2,*}, \dots, r^i_{N,*}$.

Let $\mathscr{D} = \{(T_1, c_i), \dots, (T_l, c_i)\}$ be the set of labeled tables with l number of tables, and each label $c_i \in C$ is in the set of classes defined in the KG schema $\mathscr{C} = \{c_1, \dots, c_k\}$. A table encoder E_{ω} is a model, with a parameter vector ω, which encodes each table $E_{\omega} : \{T_i\} \rightarrow \mathbb{R}^d$ to a vector $E_{\omega}(T_i) = x_i$ and $\mathscr{X} = \{x_0, x_1, \dots, x_l\}$ is the set of feature vectors for every $T_i \in \mathscr{D}$. The final task is to train a classification model $f_{\theta} : \mathbb{R}^d \rightarrow \mathscr{C}$ so that each table vector is assigned to one of the class labels. The problem is defined in the multi-class setting. Formally our setting is $f_{\theta} \circ E_{\omega} : \{T_i\} \rightarrow \mathscr{C}$, where only the parameters θ are trained on the table classification task, i.e., no gradient updates are performed on ω.

4. Experiments

Figure 1 shows the experimental setup for evaluating different table encoders. Given an entity table, a table encoder generates a high-dimensional vector representation of the table. We then train a classifier on the table-to-class task and evaluate the performance achieved by each of the table encoders. We experiment with different types of table encoders, a simple method such as document encoder, transfer learning methods with general-purpose pre-trained word embeddings (Figure 1 (a)) and more complex methods which include a LM pre-trained on large textual corpora and an approach for question-answering which has been pre-trained on tabular data (Figure 1 (b)). The code for the experiments is accessible online [1].

4.1. Dataset

For evaluation we used the second version of the T2D gold standard dataset [12], T2Dv2. To the best of our knowledge, the T2D sets are the only publicly available datasets which have been annotated with table-to-class correspondence. The second version of the dataset[2] contains 237 such annotations. In our experiments, we consider those classes which have at least two tables as representatives. The resulting dataset contains 223 tables, each labeled with one of the 27 unique classes. The mean of the number of rows in the dataset is 119.2 and the mean of the number of columns is 7.7.

4.2. Models compared

In the evaluation we used 5 different models as table encoders, varying from general purpose document encoders to more sophisticated LM, pre-trained on tabular data.

TF-IDF or term frequency-inverse document frequency, is a term weighting scheme which generates vector representation for a document based on the frequency of the words in the document. It is the simplest method which we used as a table encoder.

[1] https://github.com/anetakoleva/tableClassification

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

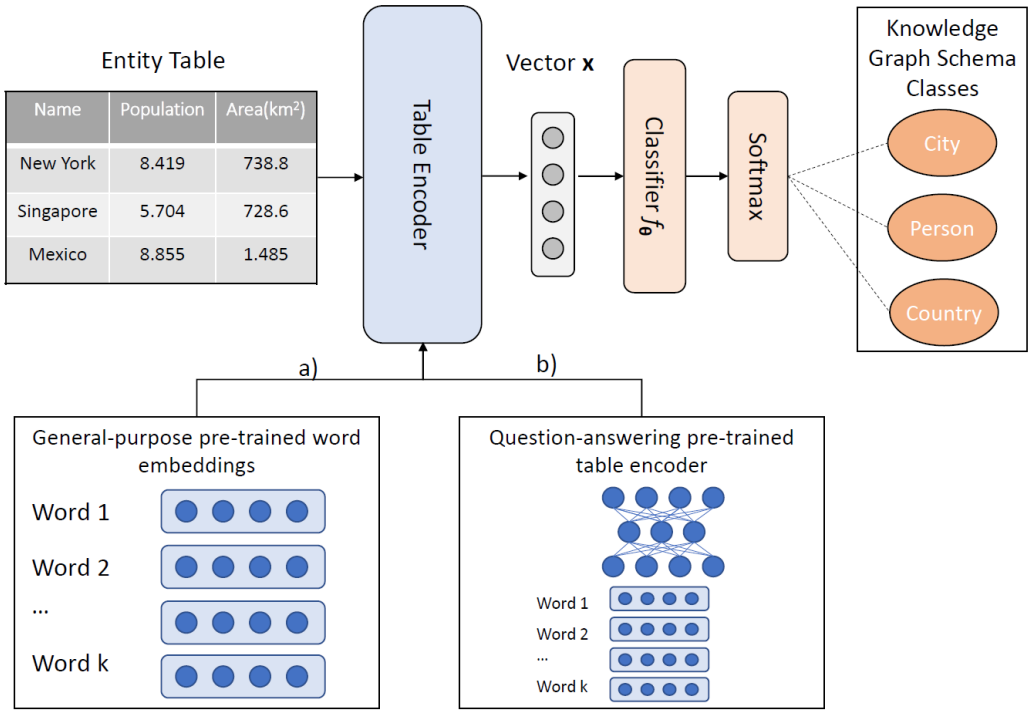

Figure 1: Experimental setup for evaluation of table encoders.

Spacy pre-trained word vectors on a text extracted from blogs, news and comments. We used the vectorizer from english-medium sized pipeline[3] which contains vocabulary of size 684830.

Word2Vec pre-trained word vectors trained with FastText [4] on a Wikipedia text corpus. The model used for the learning the vectors [22] is an extension of the original word2vec model. It is skip-gram based and trained to learn representations for character n-grams. This model consists of vocabulary of size 2.5 million.

BERT is a widely used, Transformer-based LM [10]. During the pre-training phase, the model has been exposed to a large corpus of unstructured text with the objective of predicting missing words and prediction of next sentence. This enables the model to learn the correlation of the words and to generate different vector representation for words depending on the context.

TaBERT is a table encoding method [11], pre-trained on Web tables with the objective to be used in question-answering tasks on tables. Since the model expects an *utterance*, i.e., a natural language question, as input together with a table, in our experiments we provided an empty space " ". We conducted more experiments to evaluate the influence of the utterance on the generated table representation and we discuss these results in Section 5.

[3]https://spacy.io/models/en#en_core_web_md
[4]https://fasttext.cc/docs/en/pretrained-vectors.html

4.3. Setup

To systematically evaluate the quality of the representations generated with the different table encoders, we compare their performance on the classification task under different scenarios. It is important to note that we did not train or fine-tune any of the methods for table encoding, i.e., we used them *off-the-shelf*. Since the tables can be large, in order to avoid scalability issues, we resort to sampling of rows. Namely, we first shuffle the rows in the tables and then we sample the first q rows. The shuffling of the rows is done only once. For the experiments, we sampled $q \in \{1, 3, 5, 7\}$ rows from each of the tables and used these sampled tables as input to the table encoders.

When using TF-IDF as table encoder, the input is a set of sequences, where each sequence corresponds to a table from the set of tables $\mathscr{D}$. More formally, a table sequence for table $\boldsymbol{T_i}$ is a sequence of rows $S_{T_i} = (r^i_{0,*}, r^i_{1,*}, \dots, r^i_{q,*})$, such that $q \in \{1, 3, 5, 7\}$, and the set of sequences is the set $I = \{S_{T_0}, \dots, S_{T_l}\}$. The table encoder TF-IDF transforms the set of table sequences to the set of feature vectors $E^{\text{tf-idf}}_{\omega} : I \rightarrow \mathscr{X}$.

Word2Vec and Spacy generate the vector representation for table $\boldsymbol{T_i}$ in 3 steps. First, the sequence S_{H_i}, representing the header of the table T_i, is encoded as the mean over the word vectors in the sequence S_{H_i}, represented as $\boldsymbol{x}^i_H$. Second, the content of the table, is transformed into a table sequence $S_{T_i} = (r^i_{1,*} \dots r^i_{q,*})$ and encoded as the vector $\boldsymbol{x}^i_B$, which represents the mean over all the word vectors in S_{T_i}. Finally, the vector representations for the header and for the table content are concatenated into one vector $\boldsymbol{x}_{T_i} = \boldsymbol{x}^i_H \| \boldsymbol{x}^i_B$.

Considering that there is a limit on the length of the sequence that BERT can encode in one step, we used different transformation for the last two methods. BERT encodes each table row by row, i.e, a sequence $S^i_{r_{z,*}}$ is generated for each of the rows $r^i_{z,*}$ of table $\boldsymbol{T_i}$, where $0 \leq z \leq q$. BERT generates row-wise vectors, so for each sequence $S^i_{r_{z,*}}$ the output is a vector $\boldsymbol{x}_{r_{z,*}}$. The vector representation for table $\boldsymbol{T_i}$ is the vector $\boldsymbol{x}_{T_i}$ which is the result of the mean-pooling over the set of the BERT's output vectors $\{\boldsymbol{x}_{r_{0,*}}, \dots, \boldsymbol{x}_{r_{q,*}}\}$ that correspond to the table rows. In the same manner, the TaBERT model also first generates an encoding for each of the rows of table $\boldsymbol{T_i}$ resulting in a set of vectors. This model uses vertical self-attention focused on the vertically stacked vectors, $\{\boldsymbol{x}_{r_{0,*}}, \dots, \boldsymbol{x}_{r_{q,*}}\}$. Because of the vertically aligned vectors, the output of the model is a column vector representation $\{\boldsymbol{x}_{r_{*,0}}, \dots, \boldsymbol{x}_{r_{*,M_i}}\}$ for each of the M_i columns in table $\boldsymbol{T_i}$. Finally, we do mean-pooling over the column representations to generate the table encoding $\boldsymbol{x}_{T_i}$.

We then use the Multi-layer Perceptron (MLP) with one hidden layer of size 500, the *tanh* activation function and *adam* optimizer as the classifier $f_{\boldsymbol{\theta}}$ from Figure 1. The hyper parameters are chosen after an extensive search and they are fixed for all of the experiments. Since the available dataset is small, instead of splitting it once into a training set and a test set, we use stratified K-fold validation with $K = 20$ splits. Considering that the dataset is imbalanced, we report the macro averaged F1 score. The reported scores are the average of the results on the test set after the cross validation. To explore the effect of the column names, we also encoded the tables with their column names masked. Specifically, for all of the tables, we substitute their column names with the token [UNK].

Table 1
Macro-averaged F1 score.

	Column names				**Masked column names**			
	$q = 1$	$q = 3$	$q = 5$	$q = 7$	$q = 1$	$q = 3$	$q = 5$	$q = 7$
tf-idf	0.56	0.56	0.54	0.54	0.41	0.45	0.51	0.55
spacy	0.64	0.69	0.74	0.73	0.48	0.58	0.61	0.63
word2vec	0.69	0.76	0.76	0.78	0.61	**0.77**	0.76	**0.80**
bert	**0.76**	**0.78**	**0.79**	**0.80**	**0.63**	0.75	**0.78**	0.78
tabert	0.75	0.77	0.77	0.78	0.61	0.71	0.71	0.74

5. Results

Table 1 shows the macro averaged F1 score for the 5 table encoders on the table classification task under two different settings: (1) given the input tables with the column names and (2) given the input tables with their column names masked ([UNK] token). We report the achieved F1 score for the different sizes of the input tables with the number of sampled rows q varying from 1 row to 7 rows. The simplest table encoder, TF-IDF achieves the lowest F1 score and the score only got lower when the column names of the tables were masked. For the two models with pre-trained word vectors, we observe that the model with the richer vocabulary has higher score. Indeed, the F1 score of Word2Vec is comparable with the scores achieved by BERT and TaBERT. In the first setting, when the column names of the tables are visible, there is no significant difference between the scores achieved by BERT and the scores of TaBERT. However, in the setting when the column names are masked, BERT consistently outperforms TaBERT. Interestingly, Word2Vec is the only table encoder that was not affected by the masking of the column names, on the contrary, it achieved better score in the case when $q = 3$ and $q = 7$ under the second setting compared to the setting when the column names are visible.

Figure 2 shows the row-normalized confusion matrix for the table classification task for Word2Vec and TaBERT across the different classes. The horizontal axis shows the predicted labels and the vertical axis shows the true labels. We observe the performance of the two models under the same scenario: the input tables are with $q = 7$ rows and the column names are masked. The classes are ordered by the number of instances assigned to them, *Country* is the class with the most instances, 33, while *Airline* has only 2 instances. From the confusion matrix for TaBERT (Figure 2 right) it can be observed that more miss-classifications are for the classes with a lower number of instances and they are not that unexpected. For instance, miss-classifying an instance of class *Person* as an instance of class *Scientist*, is an acceptable mistake. Similarly for the instances of classes *Academic Journal* and *Newspaper*, and *Political Party* and *Election*. On the other hand, the miss-classifications by Word2Vec for class *Wrestler* and class *Animal* as instances of class *Film* are much more unexpected and critical. Likewise, Word2Vec miss-classifies the tables of class *Scientist* and of class *Radio Station* as instances of the class *Country* which indicates a weak semantic structure in the vector representations. These results suggest that although Word2Vec achieves higher F1 score, the TaBERT vector representations capture semantics with a smoother transitions between classes.

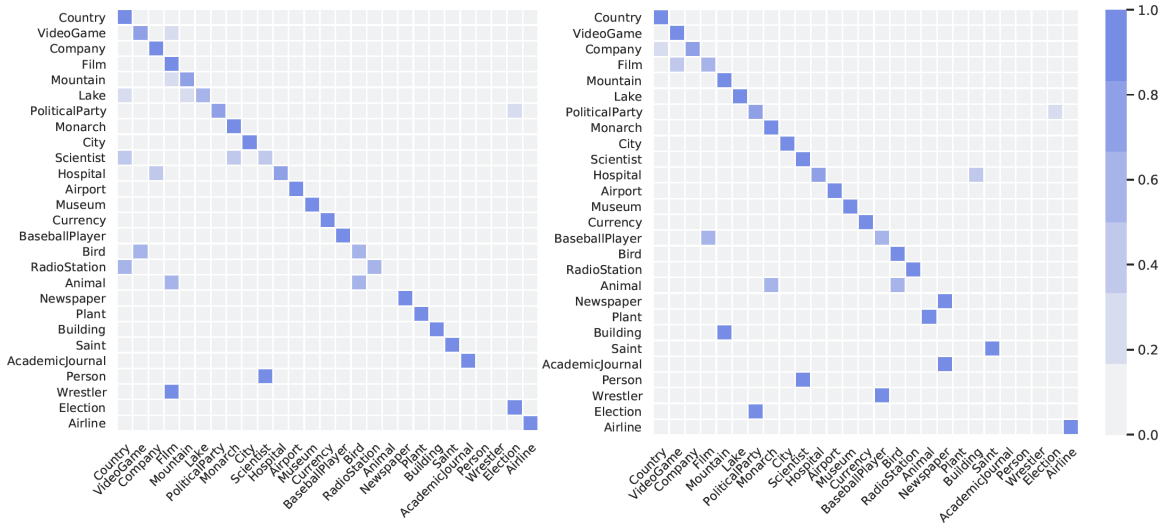

Figure 2: Classification confusion matrix for Word2Vec (left) and for TaBERT (right).

TaBERT Analysis To get a better understanding of the (under-) performance of TaBERT we analyse the influence of the *utterance* and its interplay with column names. In addition to the empty string " " used in previous experiments, we also used a randomly generated string with 10 characters (unique per table), and one constant string, *Thing*, for all tables. Moreover, we experimented with adding the correct class of the tables as utterance, as well as a wrong class (for instance, all the tables of class *Country* are encoded with the class *Plant* as utterance). Figure 3 shows the results of these experiments, where the input tables were with $q = 3$ rows. The horizontal axis shows the different options that we passed as utterance to the model and the vertical axis shows the achieved F1 score. The masking of column names has significant influence on the generated table representation. The reason for this might be in the way how a row is transformed into a string, i.e., the value of each table entry is concatenated with the column name of the entry and its value. Observing the results with the different utterance, we see that the choice of utterance does not affect the performance of the model when the column names are not masked. Nevertheless, when the column names are masked, the influence of the utterance is more significant. In both cases when the utterance is the wrong class or the correct class, the achieved score is much higher, which might be attributed to a class-wide shift in the vector space because of the grouping that these utterances cause.

6. Conclusion and Future work

In this paper we explored different types of table encoders for generating vector representations for tabular data. Specifically, we focused on evaluating different methods for table encoding on the sub-task for TI, table-to-class annotation. Despite the increasing interest in the problem of TI, so far, only one approach towards this specific sub-task has been proposed. In this direction, we provided a formal definition for the table-to-class annotation task as a machine learning task. We conduct an empirical study with five different methods for generating vector

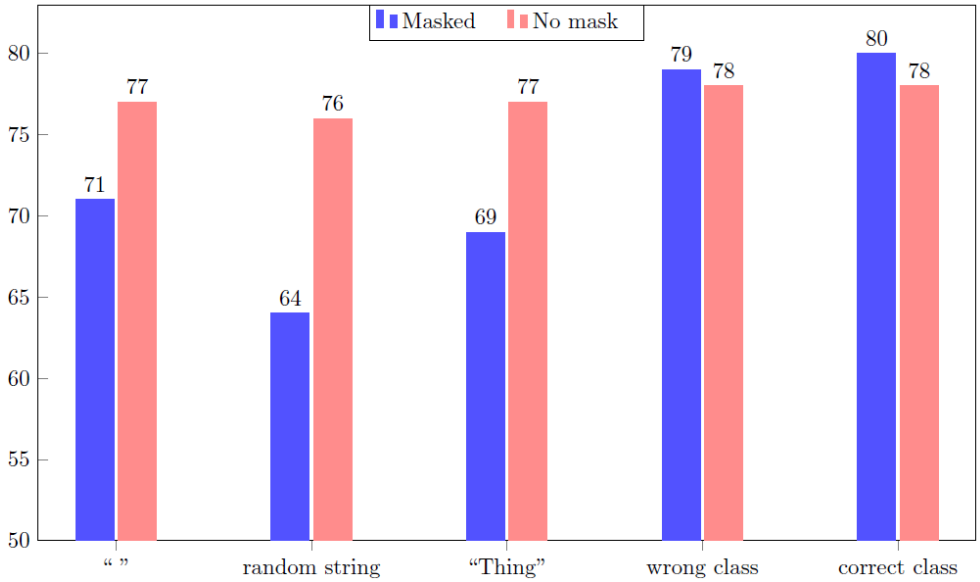

Figure 3: TaBERT performance with different utterances.

representation of a table and evaluate their performance on the table-to-class annotation task. The results from our experiments show that transfer learning methods with large vocabularies of pre-trained word embeddings perform on par with more complex and expensive modes such as LM pre-trained on tables. An interesting finding is that the inductive bias for tabular structure in TaBERT did not bring benefit to the performance of the BERT model. A possible explanation for this is the missing significant utterance that the TaBERT model expects as input. Nonetheless, the miss-classifications made by this model are reasonable, suggesting that the vector representations capture the semantics of the tables. Future work should target closing the gap between existing general-purpose models and model specific for encoding tabular data. To further our work we plan to explore other existing methods for table encoding for solving the table-to-class task, as well as for solving the entity-to-row and column-to-property tasks.

References

[1] M. J. Cafarella, A. Y. Halevy, D. Z. Wang, E. Wu, Y. Zhang, Webtables: exploring the power of tables on the web, VLDB (2008).

[2] P. Venetis, A. Y. Halevy, J. Madhavan, M. Pasca, W. Shen, F. Wu, G. Miao, C. Wu, Recovering semantics of tables on the web, VLDB (2011).

[3] M. Zhang, K. Chakrabarti, Infogather+: semantic matching and annotation of numeric and time-varying attributes in web tables, in: SIGMOD, 2013.

[4] H. Sun, H. Ma, X. He, W. Yih, Y. Su, X. Yan, Table cell search for question answering, in: WWW, 2016.

[5] J. Chen, E. Jiménez-Ruiz, I. Horrocks, C. Sutton, Learning semantic annotations for tabular

data, in: IJCAI, 2019.
[6] V. Efthymiou, O. Hassanzadeh, M. Rodriguez-Muro, V. Christophides, Matching web tables with knowledge base entities: From entity lookups to entity embeddings, in: ISWC, 2017.
[7] P. Nguyen, N. Kertkeidkachorn, R. Ichise, H. Takeda, Tabeano: Table to knowledge graph entity annotation, CoRR (2020). arXiv:2010.01829.
[8] S. Zhang, E. Meij, K. Balog, R. Reinanda, Novel entity discovery from web tables, in: WWW, 2020.
[9] G. Limaye, S. Sarawagi, S. Chakrabarti, Annotating and searching web tables using entities, types and relationships, VLDB (2010).
[10] J. Devlin, M. Chang, K. Lee, K. Toutanova, BERT: pre-training of deep bidirectional transformers for language understanding, in: NAACL-HLT, 2019.
[11] P. Yin, G. Neubig, W. Yih, S. Riedel, Tabert: Pretraining for joint understanding of textual and tabular data, in: ACL, 2020.
[12] D. Ritze, O. Lehmberg, C. Bizer, Matching HTML tables to dbpedia, in: WIMS, 2015.
[13] L. Zhang, S. Zhang, K. Balog, Table2vec: Neural word and entity embeddings for table population and retrieval, in: SIGIR, 2019.
[14] E. Jiménez-Ruiz, O. Hassanzadeh, V. Efthymiou, J. Chen, K. Srinivas, Semtab 2019: Resources to benchmark tabular data to knowledge graph matching systems, in: ESWC, 2020.
[15] S. Chen, A. Karaoglu, C. Negreanu, T. Ma, J. Yao, J. Williams, A. Gordon, C. Lin, Linkingpark: An integrated approach for semantic table interpretation, in: SemTab@ISWC, 2020.
[16] P. Nguyen, I. Yamada, N. Kertkeidkachorn, R. Ichise, H. Takeda, Mtab4wikidata at semtab 2020: Tabular data annotation with wikidata, in: SemTab@ISWC, 2020.
[17] X. Deng, H. Sun, A. Lees, Y. Wu, C. Yu, TURL: table understanding through representation learning, VLDB (2020).
[18] B. Kruit, P. A. Boncz, J. Urbani, Extracting novel facts from tables for knowledge graph completion, in: ISWC, 2019.
[19] A. Roberts, C. Raffel, N. Shazeer, How much knowledge can you pack into the parameters of a language model?, in: EMNLP, 2020.
[20] J. Herzig, P. K. Nowak, T. Müller, F. Piccinno, J. M. Eisenschlos, Tapas: Weakly supervised table parsing via pre-training, in: ACL, 2020.
[21] D. Wang, P. Shiralkar, C. Lockard, B. Huang, X. L. Dong, M. Jiang, TCN: table convolutional network for web table interpretation, in: WWW, 2021.
[22] P. Bojanowski, E. Grave, A. Joulin, T. Mikolov, Enriching word vectors with subword information, Transactions of the Association for Computational Linguistics (2017).