Comprehending Semantic Types in JSON Data with Graph Neural Networks

Shaung Wei
sw2582@rit.edu
Rochester Institute of Technology
Rochester, New York, USA

Michael J. Mior
mmior@mail.rit.edu
Rochester Institute of Technology
Rochester, New York, USA

ABSTRACT

Semantic types are a more powerful and detailed way of describing data than atomic types such as strings or integers. They establish connections between columns and concepts from the real world, providing more nuanced and fine-grained information that can be useful for tasks such as automated data cleaning, schema matching, and data discovery. Existing deep learning models trained on large text corpora have been successful at performing single-column semantic type prediction for relational data. However, in this work, we propose an extension of the semantic type prediction problem to JSON data, labeling the types based on JSON Paths. Similar to columns in relational data, JSON Path is a query language that enables the navigation of complex JSON data structures by specifying the location and content of the elements. We use a graph neural network to comprehend the structural information within collections of JSON documents. Our model outperforms a state-of-the-art existing model in several cases. These results demonstrate the ability of our model to understand complex JSON data and its potential usage for JSON-related data processing tasks.

CCS CONCEPTS

• **Computing methodologies → Machine learning**.

KEYWORDS

JSON data, graph neural networks, semantic type detection, deep learning

1 INTRODUCTION

Detecting the semantic types of columns in a relational table can be useful for data preparation and information extraction tasks such as data cleaning and integration. For example, semantic type detection can help some rule-based automated data cleaning applications that are dependent on semantic data types [9, 16]. Schema matching tasks can also narrow down search spaces based on detected semantic types [15]. Furthermore, data discovery tasks can utilize detected semantic types to find semantically related results for input queries [3, 4].

Traditional systems only classify table columns into atomic types such as Boolean, integer, and string. Semantic types are a finer-grained classification of columns that provide richer information and a connection to real-world concepts. For example, a column containing values such as "Chicago", "Detroit" can be described using the type *location* rather than *string*. Prior work has attempted to predict semantic types using methods such as dictionary lookup and regular expression matching. In real-world cases, many tables are dirty with corrupted column names or missing values that these rule-based approaches do not correctly predict [19].

To solve this problem, a deep learning enabled model, Sherlock [8], was proposed to learn the semantic type based on column values. Sherlock is trained on huge table-based corpora [7]. It first extracts features from the values in each column and a deep learning model is trained on these features to perform semantic type detection. Although Sherlock outperforms traditional approaches, it is limited to relational data.

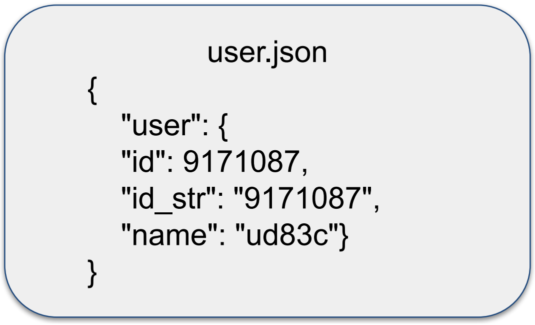
In this work, we propose a novel semantic type classification model for semi-structured JSON data. Unlike relational databases, JSON data is in the form of key-value pairs and uses a hierarchical structure. We annotate the semantic type of JSON data with its hierarchical structure and use a graph neural network to predict the semantic type with the same set of features extracted by Sherlock. The proposed model can achieve better accuracy and higher F1 scores than Sherlock on certain semantic types.

2 PROBLEM SETUP

Relational semantic type prediction is a multiclass classification problem that can be defined as follows. Given the columns $c_1, c_2, \ldots, c_m$ for a given table as the training dataset, where c_i is a vector of column values, their corresponding semantic types $t_1, t_2, \ldots, t_m \in \tau$ are considered as target values, where τ is a set of predefined semantic labels to be considered. We refer to this problem as *single-column prediction* where we use all values from each column to predict the semantic type of the column.

In our work, we extend this problem to include JSON data, which contains arbitrarily nested structures. To label JSON data, an example JSON document shown in Figure 1 could include a `user` key with the following key-value pairs: `"user": {"id":9171087, "id_str": "9171087", "name": ud83c}`. In this instance, the JSON document yields four distinct key-value pairs. The first key-value pair consists of the key `id` and its corresponding value of `9171087`. The second pair includes the key `id_str` with its corresponding value also being `9171087`. The third key-value pair comprises the key `name` with its corresponding value being `ud83c`. The fourth pair, has the key `user` and the value `{"id":9171087, "id_str": "9171087", and "name": ud83c}`.

In the context of relational data, the assignment of ground truth values for supervised semantic type prediction is often derived from columns, as they are indicative of the underlying meaning of the data within that column. In the case of JSON data, the semantic type is established by reference to the corresponding JSON Path[5] of the key. JSON Path is a query language that enables the navigation of complex JSON data structures by specifying the element location. The rationale is that values associated with the same path are likely to possess similar semantic meanings, and therefore can be grouped together under a common semantic type. For example, the path



user.json

```
{
  "user": {
  "id": 9171087,
  "id_str": "9171087",
  "name": "ud83c"}
}
```

Figure 1: Example of JSON Data

`$.user.username` refers to the "username" key within the top-level object nested under the key "user".

3 FEATURE SELECTION

In our work, we use the same set of features as Sherlock, which is a single-column prediction model that takes all values from a single column as input and outputs the predicted semantic type for the corresponding column. The extracted feature vectors are used to train a neural network for the detection of semantic types. In Sherlock, a total of 1,587 features are extracted from the column values across four different dimensions. These dimensions are described below.

(1) **Global statistics.** This category is a set of 27 hand-crafted features, typically some high-level statistical characteristics of the column. For example, column entropy describes the uniformity of the distribution of column values. Another example is the number of values that measures the number of unique values recorded in the column.
(2) **Character-level distributions.** This category contains simple statistical features of character distributions. Specifically, 10 statistical functions, i.e. any, all, mean, variance, min, max, median, sum, kurtosis, skewness, are applied on all 96 ASCII-printable characters plus the form feed character, resulting in 960 features. For example, the `any` function checks if any column value contains a specific character. `all` checks if all column values contain a character. Some other examples of features are the maximum number of appearances of a character in a single column value and the median value of appearance of a character for all column values.
(3) **Word embeddings.** Sherlock uses a pre-trained GloVe embedding [14] to characterize the semantic content of column values. The GloVe model contains a 50-dimensional embedding for 400,000 English words aggregated from 6,000,000,000 tokens. Similar to word2vec [13], GloVe embeddings can be used to measure semantic similarity between words. The advantage of GloVe compared to word2vec is that it does not rely simply on local information of words, but it also incorporates global statistics such as word co-occurrence. By calculating the mean, mode, median, and variance on the 50-dimensional GloVe feature vector for all column values, a 200-dimensional feature vector is produced in this category.
(4) **Paragraph vectors.** A distributed bag of words version of the paragraph vector (PV-DBOW), or the doc2vec model [12] is implemented to capture features at the "topic" level of the column. The doc2vec model forces the model to predict random words that are sampled from paragraphs in the output by ignoring context words from the input. The model is pre-trained in Sherlock using the Gensim library to extract a paragraph feature that has 400 dimensions.

4 PROPOSED GRAPH MODEL

Raw JSON data is annotated by treating each key-value pair as a data point. For instance, from the `user.json` file, four data points can be extracted as illustrated in Figure 1. For each key-value pair, the label is determined by annotating the JSON Path to represent the semantic meaning, while the features are extracted using Sherlock from the corresponding values at each path. As an example, consider the key-value pair `"user": {"id":9171087, "id_str": "9171087", "name": "ud83c"}`. Here, the label assigned is `user`, and the features are extracted from `{"id":9171087, "id_str": "9171087", "name": "ud83c"}`. At this stage, we can proceed to apply Sherlock and evaluate its performance for semantic type detection, using it as a baseline result.

The structure of our model is illustrated in Figure 2. Each JSON file is processed by first obtaining all key-value pairs, as described in the preceding section. Using the same example as in Figure 1, we can obtain four key-value pairs. Subsequently, the features for each key-value pair are computed based on their respective values, yielding features f_1, f_2, f_3, and f_4.

Following this, four graphs are generated, labeled as "id", "id_str", "name", and "user". The first three graphs, G_1, G_2, and G_3, each consist of a single-node, with the node features f_1, f_2, and f_3. The fourth graph, G_4, is a multi-node graph, with a root node having node features f_4 and three edges connecting to three other nodes, each having node features f_1, f_2, and f_3.

Once we obtain the graph representations, we use graph neural networks (GNNs)[11] to perform the classification task. Graph neural networks are well-suited for our problem, as JSON documents inherently possess a tree-like structure that can be represented as a graph. GNNs excel at capturing complex dependencies and relationships within graphs by leveraging structural information encoded in edges and node features. Using GNNs, we can effectively leverage the hierarchical relationships and dependencies present in JSON documents, enabling accurate analysis and prediction tasks such as semantic type prediction or classification. GNNs exploit the rich structural information of JSON documents, making them a powerful approach to understanding and extracting insights from this graph-based representation. In our study, we used the Spektral library[6] to implement our GNN. Specifically, we employed a two-layer GCN model, where the input graph was first fed into a GCN layer with 256 hidden units, followed by graph pooling and a dropout layer. The output was then forwarded to another GCN layer with 64 hidden units, before being connected to a dense layer for multi-class classification. To optimize the model, we adopted

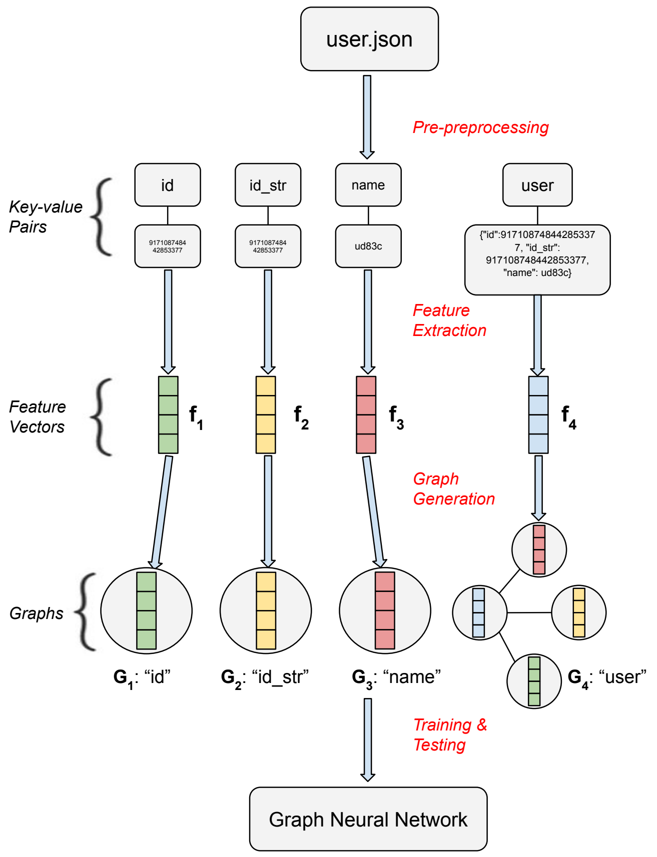


Figure 2: Proposed model architecture

Adam[10] as our optimizer and used categorical cross-entropy as our loss function. The learning rate was set to 2×10^{-4}.

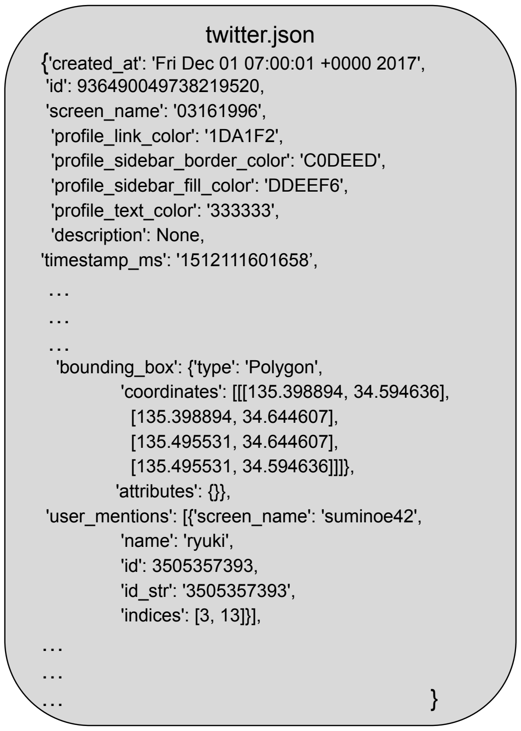
5 DATASET

Our study uses Twitter data and Meetup data available on push-shift.io [1], a large-scale archive of social media content. Due to the immense size of the dataset, we selected a representative subset of each dataset for our research. Specifically, we extracted all available data from a specific date, resulting in a total of 30,000 distinct JSON objects for Twitter data, and 20,000 distinct JSON objects for Meetup data.

Figure 3 shows an example of the Twitter dataset used in this study. The example contains labels for a single-node graph, including `created_at`, `id`, and `screen_name`. The `profile_link_color`, `profile_sidebar_border_color`, `profile_sidebar_fill_color`, and `profile_text_color` also belong to single-node graphs with label color. Additionally, the example comprises multi-node graphs with labels such as `bounding_box` and `user_mentions`, which are also illustrated in the figure. The above-mentioned nodes have a depth of 1 in the graph, while nodes such as `type` and `coordinate` in this example have a depth of 2. Table 1 presents the number of examples corresponding to each depth (level of nesting) for our two datasets.

Depth	Number of examples	
	Twitter	Meetup
1	1,205	191,884
2	32,158	304,750
3	74,187	101,930
4	3,907	0
5	124	0

Table 1: Number of examples for each depth


twitter.json

{'created_at': 'Fri Dec 01 07:00:01 +0000 2017',
 'id': 936490049738219520,
 'screen_name': '03161996',
  'profile_link_color': '1DA1F2',
  'profile_sidebar_border_color': 'C0DEED',
  'profile_sidebar_fill_color': 'DDEEF6',
  'profile_text_color': '333333',
  'description': None,
'timestamp_ms': '1512111601658',
 …
 …
 …
  'bounding_box': {'type': 'Polygon',
           'coordinates': [[[135.398894, 34.594636],
             [135.398894, 34.644607],
             [135.495531, 34.644607],
             [135.495531, 34.594636]]]},
          'attributes': {}},
 'user_mentions': [{'screen_name': 'suminoe42',
           'name': 'ryuki',
           'id': 3505357393,
           'id_str': '3505357393',
           'indices': [3, 13]}],
…
…
…   }

Figure 3: Example of Twitter JSON data

All key-value pairs with a null value or a type of Boolean are ignored since these values are highly repetitive in our dataset and do not contain useful semantic information. We then annotated each JSON Path with a class label, as discussed in the previous section, resulting in a label set comprising 43 distinct classes for the Twitter dataset and 32 distinct classes for the Meetup dataset.

To prepare the data for use in our model, we processed each JSON file into a graph structure, resulting in a total of around 110,000 distinct graphs for the Twitter dataset and 600,000 distinct graphs for the Meetup dataset. Each graph in our dataset is accompanied by a set of node features, an adjacency matrix, and a class label encoded in one-hot format. The dataset is partitioned into training, validation, and test sets in a 7:3:3 ratio.

6 EXPERIMENT METHOD

In our experiment, we initially preprocess the JSON file by extracting key-value pairs based on their corresponding JSON paths.

Figure 4 demonstrates the conversion of a JSON document into relational tables, where each key-value pair is represented as a separate column. Subsequently, a feature extraction process is applied to these values to obtain the feature vectors. This step enables us to utilize the Sherlock model for classification, allowing us to establish a baseline performance measure. We then proceed to process the data into graphs, following the procedures outlined in Figure 2. The classification task is then performed using a graph neural network model. We summarize the time spent on each preprocessing step in Table 2, providing an overview of the time requirements for these tasks on our two different datasets. The key-value pair extraction step takes 1,205s and 3,525s for the Twitter and Meetup datasets respectively, which is the most time-consuming preprocessing step. The feature extraction and graph processing steps take less time than the key-value pair extraction. The feature extraction step is the only preprocessing step required to train the Sherlock model.

Preprocessing steps	Twitter	Meetup
Key-value pair extraction	1,205s	3,525s
Feature extraction	785s	2,100s
Graph processing	152s	450s
Total time	2,142s	7,075s

Table 2: Time spent on preprocessing steps

7 EXPERIMENT RESULTS

Table 3: Comparison of Sherlock and the proposed model on Twitter dataset

		Sherlock		Proposed model	
	Label	F1 score	Accuracy	F1 score	Accuracy
Single-Node	screen_-name	0.92	0.93	**0.95**	0.93
	country_-code	0.80	**1.00**	**0.92**	0.85
	timestamp_-ms	**1.00**	**1.00**	0.97	0.96
	color	**0.99**	0.99	0.98	0.99
	description	**0.75**	**0.77**	0.67	0.60
Multi-Node	bounding_-box	0.83	1.00	**1.00**	1.00
	user_-men-tions	0.59	0.41	**0.84**	**0.97**
	retweet_-status	0.57	0.48	**0.82**	**0.86**
	hashtag	0.34	0.23	**0.40**	**0.80**
	full_-name	0.00	0.00	**0.22**	**0.97**
Average		0.82	0.84	**0.85**	**0.85**

Table 3 provides a comprehensive comparison between the performance of Sherlock and the proposed model in terms of the F1 score and accuracy metrics. The evaluation is conducted for all semantic classes, which are categorized into single-node and multi-node based on the number of nodes in the graph. The table presents some selected examples from each category, including `screen_name`, `country_code`, `timestamp_ms`, color, and description for single-node, and `bounding_box`, `user_mentions`, `retweet_-status`, `hashtag`, and `full_name` for multi-node. The average performance result is also presented at the bottom of the table.

In the single-node setting, the proposed model outperforms Sherlock on some labels, for example, with F1 scores of 0.95 for `screen_name`, 0.92 for `country_code`, compared to 0.92 and 0.80, respectively, for Sherlock. However, Sherlock achieves a perfect F1 score of 1.00 for the `timestamp_ms` label, while the proposed model only achieves a score of 0.97. The Sherlock model also exhibits higher accuracy levels for the semantic types of `country_code` and `timestamp_ms`. Sherlock also performs slightly better for the `description` label, with an F1 score of 0.75 compared to 0.67 for the proposed model. Based on the results, it can be suggested that the base Sherlock model might exhibit better performance when applied to single-node scenarios, which may be attributed to the fact that the Sherlock model utilizes a more complex neural network architecture compared to our network.

In the multi-node setting, the proposed model achieves a perfect F1 score of 1.00 for the `bounding_box` label, while Sherlock achieves a score of 0.83. The proposed model also outperforms Sherlock for the `user_mentions` and `retweet_status` labels, with F1 scores of 0.84 and 0.82 compared to 0.59 and 0.57, respectively. However, Sherlock achieves a slightly better F1 score of 0.40 for the hashtag label compared to 0.34 for the proposed model. The proposed model achieves a higher F1 score of 0.22 for the `full_-name` label compared to 0.00 for Sherlock. In terms of accuracy, our proposed model significantly outperforms Sherlock. Specifically, for the hashtag semantic type, our model achieves an accuracy of 0.80, while Sherlock only achieves an accuracy of 0.23. Similarly, for the `full_name` semantic type, our model achieves an accuracy of 0.50, while Sherlock fails to classify any instance correctly. These findings demonstrate that our proposed model is more adept at predicting complex structures, thus providing greater utility for practical applications.

Table 4 presents the comparison between Sherlock and our proposed model on the meetup dataset. The JSON files within the meetup dataset exhibit highly similar structures, resulting in higher overall prediction accuracy and F1 score. In cases involving multiple nodes, our model achieves perfect accuracy of 1.00, while Sherlock performs equally well on classes such as event and category, with F1 scores of 0.99 and 0.97 respectively for `group` and `group_photo`. For single-node classes, both models demonstrate similar performance levels. The Meetup dataset consists of numerous homogeneous JSON files, many of which exhibit similar hierarchical structures. As a result, the base model Sherlock can achieve good performance in the multi-node setup, indicating that our model does not outperform Sherlock on this dataset. The reason could be that the Meetup dataset exhibits a higher level of homogeneity and has a less complex hierarchical structure for our model to take advantage of.

Table 5 shows a comparison between the training time and model size of Sherlock and the proposed model. Our proposed model takes significantly longer to train but produces a much smaller model. We

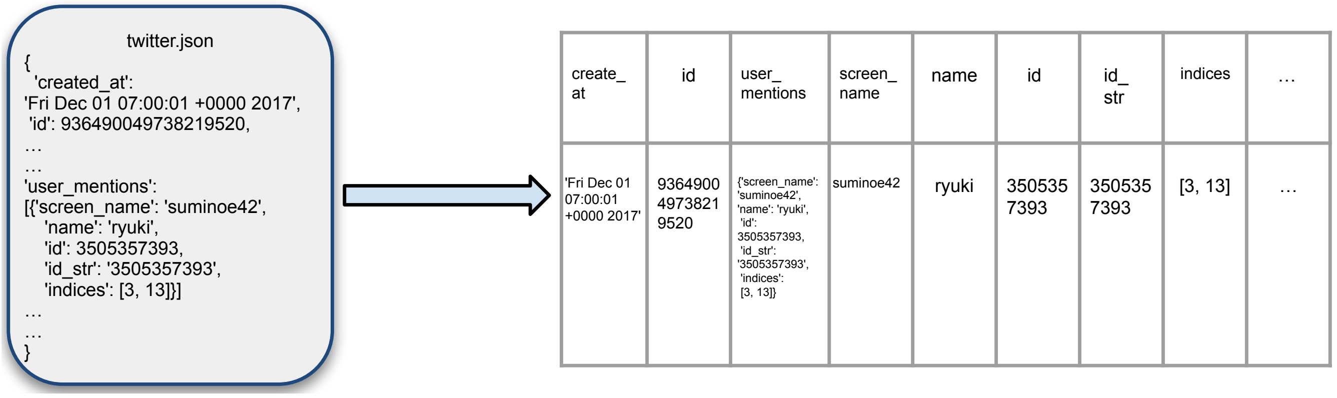


Figure 4: transformation of JSON document to relational table

Table 4: Comparison of Sherlock and the proposed model on the Meetup dataset

		Sherlock		Proposed model	
	Label	F1 score	Accuracy	F1 score	Accuracy
Single-Node	event_id	0.66	**0.64**	**0.68**	0.63
	id	0.86	0.85	**0.91**	**0.90**
	member_-name	**0.95**	**0.95**	**0.95**	0.94
	shortname	**0.99**	0.99	0.98	**1.00**
Multi-Node	event	**1.00**	**1.00**	**1.00**	**1.00**
	category	**1.00**	**1.00**	**1.00**	**1.00**
	group	0.99	0.97	**1.00**	**1.00**
	group_-photo	0.97	0.99	**1.00**	**1.00**
Average		0.89	0.89	**0.92**	**0.90**

Table 5: Training time and model size comparison

Model	Training time		Model size
	Twitter	Meetup	
Sherlock	252s	690s	5.9MB
Proposed model	1,230s	3,051s	1.9MB

expect that advances in graph neural network training will apply to our setting to further reduce the training time [2, 21]. Our proposed model exhibits a notable reduction in model size in comparison to Sherlock. This is mainly attributed to the fact that our model incorporates a smaller number of neural network layers. Consequently, this disparity may account for certain suboptimal predictions made by our model in comparison to Sherlock, particularly in single-node predictions.

Our proposed model achieves a higher average F1 score compared to Sherlock. These results suggest that our proposed model has a superior ability to learn structural information from the data.

8 FUTURE WORK

In our ongoing research, we aim to explore alternative subgraph representations to enhance the prediction of semantic types in our proposed model. The current subgraphs we utilize do not incorporate the parent node and its sibling nodes, thus missing out on valuable information present in those nodes. Sato [20] is a model that utilizes neighbor columns and table-level information for semantic prediction; however, it exhibits limited scalability in terms of training time. To address this limitation, we plan to construct subgraphs that include the parent and sibling nodes of the target node. In addition, we will assign edge weights to the graph, allowing us to emphasize the importance of the target node for prediction. By enabling additional edge features within each subgraph, we can leverage neural networks such as edge-conditioned GCN [17] that are capable of utilizing edge weight information. Subsequently, we intend to combine the outputs obtained from different subgraphs using a transformer [18]. The transformer architecture is suitable for this purpose because of its ability to capture global dependencies and model the relationships between the outputs of different subgraphs effectively.

In addition, we plan to extend our experimentation beyond the current dataset in order to enhance the robustness and generalizability of our model. Specifically, we will train and test our model on additional datasets that are representative of real-world scenarios. Our goal is to evaluate the effectiveness of the model's predictive capabilities on JSON structures that it has not yet encountered. Moreover, we aim to examine the impact of training set size on the performance of our model, with particular attention to the detection of less frequent semantic types. By conducting such analyses, we aim to better understand the limitations and strengths of our model and further refine it for real-world applications.

9 CONCLUSION

In this paper, we proposed a model for predicting semantic types in nested JSON data that can be used for various automated data processing tasks. Existing models either predict for a single set of values at a time or are limited to non-nested relational data.

To address this limitation, we proposed an extension of the semantic type prediction problem to semi-structured JSON data with types labeled based on JSON Paths. Our proposed model annotates the semantic type of JSON data with its hierarchical structure and employs a graph neural network to predict the semantic type using the same set of features extracted by Sherlock. We demonstrate several cases where our model outperforms Sherlock, indicating its ability to comprehend complex JSON data and its potential for semi-structured data processing tasks.

Our ongoing research focuses on enhancing the prediction of semantic types in our proposed model through alternative subgraph representations. By incorporating the parent and sibling nodes into subgraphs and assigning edge weights, we can capture valuable information for accurate predictions. Leveraging additional edge features will further improve our model's performance. We plan to validate our approach on diverse datasets, ensuring its robustness and generalizability to real-world scenarios. Furthermore, we plan to investigate the impact of the size of the training set on model performance, especially for detecting less frequent semantic types.

In conclusion, our work contributes to the development of deep learning models for predicting semantic types in JSON data. Our proposed model offers a better understanding of complex data and improved performance compared to Sherlock on semi-structured data.

REFERENCES

[1] Jason Baumgartner. 2021. *Push Shift Dataset*. https://files.pushshift.io/

[2] Ming Chen, Zhewei Wei, Bolin Ding, Yaliang Li, Ye Yuan, Xiaoyong Du, and Ji-Rong Wen. 2020. Scalable Graph Neural Networks via Bidirectional Propagation. In *Advances in Neural Information Processing Systems*, Vol. 33. 14556–14566.

[3] Raul Castro Fernandez, Ziawasch Abedjan, Famien Koko, Gina Yuan, Samuel Madden, and Michael Stonebraker. 2018. Aurum: A data discovery system. In *IEEE 34th International Conference on Data Engineering (ICDE)*. Paris, France, 1001–1012. https://doi.org/10.1109/ICDE.2018.00094

[4] Raul Castro Fernandez, Essam Mansour, Abdulhakim A Qahtan, Ahmed Elmagarmid, Ihab Ilyas, Samuel Madden, Mourad Ouzzani, Michael Stonebraker, and Nan Tang. 2018. Seeping semantics: Linking datasets using word embeddings for data discovery. In *IEEE 34th International Conference on Data Engineering (ICDE)*. Paris, France, 989–1000. https://doi.org/10.1109/ICDE.2018.00093

[5] Jeff Friesen. 2019. *Extracting JSON Values with JsonPath*. Apress, Berkeley, CA, 299–322. https://doi.org/10.1007/978-1-4842-4330-5_10

[6] Daniele Grattarola and Cesarei Alippi. 2021. Graph Neural Networks in TensorFlow and Keras with Spektral. *IEEE Computational Intelligence Magazine* 16 (2021), 99–106. https://doi.org/10.48550/arXiv.2006.12138

[7] Kevin Hu, Neil Gaikwad, Michiel Bakker, Madelon Hulsebos, Emanuel Zgraggen, César Hidalgo, Tim Kraska, Guoliang Li, Arvind Satyanarayan, and Çağatay Demiralp. 2019. Viznet: Towards a large-scale visualization learning and benchmarking repository. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. Glasgow, Scotland, UK, 1–12. https://doi.org/10.1145/3290605.3300892

[8] Madelon Hulsebos, Kevin Hu, Michiel Bakker, Emanuel Zgraggen, Arvind Satyanarayan, Tim Kraska, Çagatay Demiralp, and César Hidalgo. 2019. Sherlock: A deep learning approach to semantic data type detection. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. Anchorage, AL, USA, 1500–1508. https://doi.org/10.1145/3292500.3330993

[9] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. 2011. Wrangler: Interactive visual specification of data transformation scripts. In *Proceedings of the 29th SIGCHI Conference on Human Factors in Computing Systems*. Vancouver, BC, Canada, 3363–3372. https://doi.org/10.1145/1978942.1979444

[10] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, (ICLR)*. San Diego, CA, USA. http://arxiv.org/abs/1412.6980

[11] Thomas N. Kipf and Max Welling. 2016. Semi-Supervised Classification with Graph Convolutional Networks. In *5th International Conference on Learning Representations, (ICLR)*. http://arxiv.org/abs/1609.02907

[12] Quoc Le and Tomas Mikolov. 2014. Distributed Representations of Sentences and Documents. In *Proceedings of the 31st International Conference on Machine Learning (PMLR)*, Vol. 32. Bejing, China, 1188–1196. https://doi.org/10.5555/3044805.3045025

[13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. In *ICLR Workshop Papers*. International Conference on Learning Representations. https://arxiv.org/abs/1301.3781

[14] Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar, 1532–1543. https://doi.org/10.3115/v1/D14-1162

[15] Erhard Rahm and Philip A Bernstein. 2001. A survey of approaches to automatic schema matching. *the VLDB Journal* 10, 4 (2001), 334–350. https://doi.org/10.1007/s007780100057

[16] Vijayshankar Raman and Joseph M Hellerstein. 2001. Potter's wheel: An interactive data cleaning system. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB)*, Vol. 1. Roma, Italy, 381–390. https://doi.org/10.5555/645927.672045

[17] Martin Simonovsky and Nikos Komodakis. 2017. Dynamic Edge-Conditioned Filters in Convolutional Neural Networks on Graphs. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Hawaii, US, 29–38. http://arxiv.org/abs/1704.02901

[18] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, Vol. 30. Long Beach, CA, US. https://dl.acm.org/doi/10.5555/3295222.3295349

[19] Petros Venetis, Alon Halevy, Jayant Madhavan, Marius Paşca, Warren Shen, Fei Wu, Gengxin Miao, and Chung Wu. 2011. Recovering Semantics of Tables on the Web. In *Proceedings of the 37th International Conference on Very Large Data Bases (VLDB)*, Vol. 4. Seattle, WA, US, 528–538. https://doi.org/10.14778/2002938.2002939

[20] Dan Zhang, Yoshihiko Suhara, Jinfeng Li, Madelon Hulsebos, Çağatay Demiralp, and Wang-Chiew Tan. 2020. Sato: Contextual semantic type detection in tables. In *Proceedings of the 46th International Conference on Very Large Data Bases (VLDB)*, Vol. 13. Tokyo, Japan, 1835–1848. https://doi.org/10.14778/3407790.3407793

[21] Chenguang Zheng, Hongzhi Chen, Yuxuan Cheng, Zhezheng Song, Yifan Wu, Changji Li, James Cheng, Hao Yang, and Shuai Zhang. 2022. ByteGNN: Efficient Graph Neural Network Training at Large Scale. In *Proceedings of the 48th International Conference on Very Large Data Bases (VLDB)*, Vol. 15. Sydney, Australia, 1228–1242. https://doi.org/10.14778/3514061.3514069