PARTNER: Human-in-the-Loop Entity Name Understanding with Deep Learning

Kun Qian, Poornima Chozhiyath Raman, Yunyao Li, Lucian Popa
IBM Research – Almaden
qian.kun@ibm.com, {pchozhi, yunyaoli, lpopa}@us.ibm.com

Abstract

Entity name disambiguation is an important task for many text-based AI tasks. Entity names usually have internal semantic structures that are useful for resolving different variations of the same entity. We present, **PARTNER**, a deep learning-based interactive system for entity name understanding. Powered by effective active learning and weak supervision, **PARTNER** can learn deep learning-based models for identifying entity name structure with low human effort. **PARTNER** also allows the user to design complex normalization and variant generation functions without coding skills.

Understanding entity name variations can be helpful for many text-based AI tasks such as knowledge base curation, data integration, and information retrieval. Text-based similarity functions are widely used in these applications to resolve entity names (e.g., (Qian, Popa, and Sen 2017)). However, entity names can be highly ambiguous and text-based similarity functions are not sophisticated enough for resolving complex entity variations. For example, "`General Electric (CN) Company`" and "`General Electric Company`" are textually similar but referring to different entities. On the other hand, "`GE Co.`" and "`General Electric Company`" are referring to the same entity even if they are textually dissimilar.

Entity names often have internal semantic structure consisting of various semantic units (see Figure 1), which are crucial for entity normalization and variant generation (Bhutani et al. 2018; Arasu and Kaushik 2009). Specifically, if we can identify that "`General Electric`" is the ⟨*corename*⟩ unit and "`Company`" is the ⟨*suffix*⟩ unit, we can perform unit-level transformations such as creating initials for ⟨*corename*⟩ (e.g., `General Electric` → `GE`) and abbreviating ⟨*suffix*⟩ (i.e., `Company` → `Co.`), and eventually we get "`GE Co.`". In this way, "`General Electric Company`" and "`GE Co.`" can be resolved as the same name even though they are textually dissimilar.

Although it is possible to let a programmer to manually write type-specific algorithms to identify semantic structures of entity names (e.g., (Arasu and Kaushik 2009)), it is labor-intensive, time-consuming, and does not scale. Therefore,

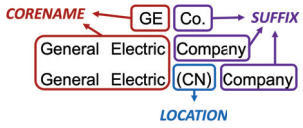

Figure 1: Structure of sample company entities

based on (Bhutani et al. 2018), an active learning based system called `LUSTRE` (Qian et al. 2018) has been developed to automatically generate rule-based models that identify the semantic structures of named entities. The rules learned by `LUSTRE` are sequences of regex-based matchers and dictionary-based matchers that define different sequence patterns. The availability of a list of comprehensive, accurate, and complete dictionaries is fundamental to the success of `LUSTRE`, otherwise the learned rules do not take into account any semantic information and are sensitive to noisy data (e.g., misspelling). Moreover, `LUSTRE` does not support incremental learning or transfer learning.

We present **PARTNER** (dee**P** le**A**rning fo**R** en**T**ity u**N**d**ER**standing), an interactive system that learns deep learning-based models for parsing the semantic structures of named entities. **PARTNER** has an intuitive user interface for specifying sophisticated entity normalization and variant generation functions with no coding skill needed.

Methodology

Learning framework. Parsing the semantic structures of entity names can be viewed as a sequence tagging problem (see Figure 1). Deep BiLSTM-CRF models have been shown to achieve state-of-the-art performance on sequence tagging problems (Huang, Xu, and Yu 2015). However, deep learning methods are known for data hungry. In our settings, we initially do not have any annotated examples. To address the data hungriness issue and minimize human effort, we use active learning to iteratively find the most "informative/uncertain" example to be labeled by the user; we also use weak supervision to augment the training data by automatically generate a number of high-quality pseudo labels. Our BiLSTM-CRF approach can use either use pretrained fastText embeddings or use pretrained BERT models to gen-

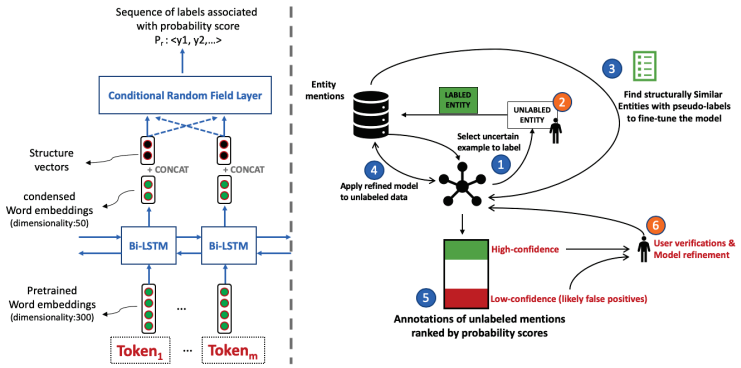

Figure 2: (left) BiLSTM-CRF models with fastText (BERT-based models has similar architecture). (right) User interaction.

erate contextual embeddings.

Figure 2(left) shows the architecture that **PARTNER** uses for entity structure parsing. Concretely, given an entity mention E we tokenize it into a set $T = \{\text{token}_1, \ldots, \text{token}_m\}$ of tokens. We then convert the sequence of tokens into a sequence $\{v_1, \ldots, v_m\}$ of high dimensional embedding vectors using some pretrained embeddings. **PARTNER** can use pretrained BERT to tokenize entity names and generate contextual token embeddings or use embeddings generated by fastText[1]. These embedding vectors are then fed into a BiLSTM layer. We set a low dimensionality (e.g., 50) for both hidden size and output size of the BiLSTM layer so that we can compress high-dimensional word embeddings into low-dimensional "*semantic*" vectors capturing contextual information of the input tokens. We also predefined a list of generic regex-based functions, such as *hasDigits(x)*, *allCaptial(x)*, *isInDictionary(x,'suffix')*, to capture various structure information of an input token and convert it into a "*structure*" vector. Then, we concatenate the corresponding "*structure*" and "*semantic*" vectors, and feed them, sequentially, to the conditional random field (CRF) layer to find out the most likely sequence labels.

Active learning and weak supervision. Figure 2(right) illustrates the flow of one active learning iteration, where only two places need human interaction (see more details from the submitted video demo). First, we apply the current candidate model (in first iteration, it will be a randomly initialized model) to annotate all unlabeled entity mentions, then we rank these annotations based on their *informative* score (details omitted due to limited space). We choose the most informative example e to be labeled by the user. After the user labels e, the system will find a set S of unlabeled entities that are structurally similar to e (i.e., examples have identical sequence of "*structure*" vectors). It follows that the labels of e will be used as pseudo labels for all examples in S. We then refine the candidate model with both human labels as well as pseudo labels. Next, we use the newly refined model to annotate all unlabeled entity mentions, and we then order these annotations based on their probability scores produced by the CRF layer. We choose the annotations with top-k highest (resp. bottom-k lowest) probability scores as high-confidence true positives (resp. likely false positives) and present them to the user for verification. In this step, the user just need to provide binary feedback to these annotations. The verified annotations are used in the subsequent iterations. Depending on labeling budget, high-confidence true positives can also be treated as weak labels to further minimize human effort (configurable from the UI).

Variant generation. During the learning, the user can test the parsing model by either uploading a test file or manually type in test cases. The user can also play with the parsing model by building normalization or variant generation functions on top of it. **PARTNER** provides a rich set of unit-level and pattern-level transformations (intuitively visualized in UI), which allows the user, without coding skills, to design complex normalization and variant generation functions.

Switching between learning and normalization is simple and flexible. Moreover, the design of **PARTNER** makes transfer learning and incremental learning easy. The user can simply adapt a pre-trained model to new data by fine-tuning it with the aforementioned active learning strategy over the new data. **PARTNER**[2] is developed in Angular, Django, PyTorch, and HuggingFace Pytorch-Transformers.

References

Arasu, A., and Kaushik, R. 2009. A grammar-based entity representation framework for data cleaning. In *SIGMOD'09*, 233–244. New York, NY, USA: ACM.

Bhutani, N.; Qian, K.; Li, Y.; Jagadish, H.; Hernandez, M.; and Vasa, M. 2018. Exploiting structure in representation of named entities using active learning. In *COLING 2018*, 687–699.

Huang, Z.; Xu, W.; and Yu, K. 2015. Bidirectional LSTM-CRF models for sequence tagging. *CoRR* abs/1508.01991.

Qian, K.; Bhutani, N.; Li, Y.; Jagadish, H.; and Hernandez, M. 2018. Lustre: An interactive system for entity structured representation and variant generation. In *ICDE*, 1613–1616.

Qian, K.; Popa, L.; and Sen, P. 2017. Active learning for large-scale entity resolution. In *CIKM*, 1379–1388. ACM.

[1]While the BERT-based version is potentially better than the fastText-based version, it is computationally more expensive.

[2]https://github.com/System-T/PARTNER