LUSTRE: An Interactive System for Entity Structured Representation and Variant Generation

Kun Qian*, Nikita Bhutani#, Yunyao Li*, H V Jagadish#, Mauricio A. Hernandez*
*IBM Research - Almaden, #University of Michigan, Ann Arbor
qian.kun@ibm.com, {yunyaoli, mahernan}@us.ibm.com, {nbhutani, jag}@umich.edu

***Abstract*—Many data analysis and data integration applications need to account for multiple representations of entities. The variations in entity mentions arise in complex ways that are hard to capture using a textual similarity function. More sophisticated functions require the knowledge of underlying structure in the representation of entities. People traditionally identify these structures manually and write programs to manipulate them: such work is tedious and cumbersome. We have built LUSTRE, an active learning based system that can learn the structured representations of entities interactively from a few labels. In the background, it automatically generates programs to map entity mentions to their representations and to standardize them to a unique representation. Furthermore, LUSTRE provides a user-friendly interface to allow user declaratively specify normalization and variant generation functions for downstream applications.**

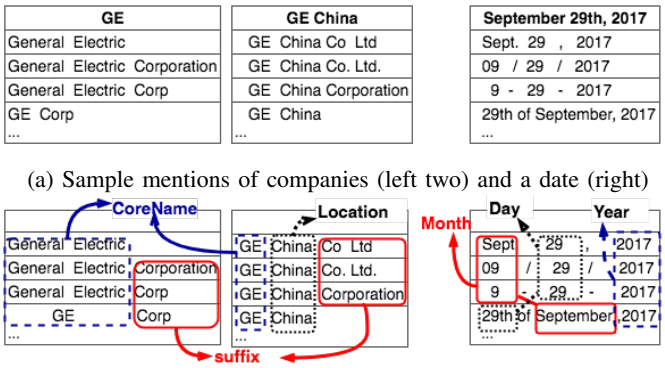
***Keywords*-entity standardization; active learning**

I. Introduction

Much of today's information is structured around and referenced by the properties of real-world entities. Entities, however, lack a unique representation. The problem of reconciling name variations of entities forms the core of many entity-centric applications such as data integration, knowledge base construction, and search. Reconciliation is challenging due to issues such as non-uniform variations in mentions of an entity and mention ambiguity [1] (a named entity mention is an entity mention in the form of a named entity text string that refers to a named entity). The variations arise from multiple factors, as illustrated in Fig 1. Firstly, variations arise from presence or absence of details that comprise an entity mention. For instance, mentions of `GE` could optionally include a suffix (e.g., `Corp`). Secondly, variations arise from different orderings of the details. For instance, a date mention could refer to the month before the day (as in `September 29th`) or vice-versa (as in `29th of September`). Moreover, variations can also arise from abbreviations and aliasing of the details (e.g. `9` for `September`, `GE` for `General Electric`).

Typically, name variations are handled using textual similarity functions ([2], [3], [4]), which ignore the underlying structure in representations and often lead to erroneous matches. Consider the following mentions:

(a) `General Electric Corporation`
(b) `General Electric China Corporation`
(c) `GE CO`



(a) Sample mentions of companies (left two) and a date (right)

(b) Structured representations of sample mentions

Figure 1: Structured representations of entity mentions

Mentions (a) and (b) are textually similar - they differ in just one token - but correspond to different entities, owing to the location detail `China`. Conversely, mentions (a) and (c) are textually dissimilar but refer to the same entity.

Our key observation is that an entity mention is not merely a sequence of characters. Instead, it has an internal structured representation comprised of semantic units [5]. Surface-form variations arise from variations in the structured representation. Furthermore, mentions of an entity type typically share semantic units in their representations (see Fig 1b). Knowledge of these representations for an entity type can enable effective reasoning to identify two textually dissimilar mentions as identical and two textually similar mentions as distinct.

While it is possible for programmers to write type-specific programs that can map mentions to their structured representations, the process is tedious and labor-intensive. For each entity type, they have to examine the mentions, identify the different structured representations and then write programs for each representation. This has to be repeated for each entity type. High skill and high effort limits the usability of such approaches (e.g., [5]).

To address these challenges, we have developed a web-based system, LUSTRE, that reduces the required skill and effort by (1) seeking only a few human-comprehensible labels from the user, (2) automatically generating programs from the labels that map mentions to their representations, (3) providing a user-friendly interface to declaratively configure and generate complex normalization and variant generation programs. The user can use these programs for

downstream applications (e.g., [6]).

II. System Overview

In this section, we give a brief overview of the system architecture and the user interaction model.

A. System Architecture

A high-level view of LUSTRE is shown in Fig 2. It has two main phases for (1) learning structured representations, and (2) synthesizing normalization and variant generation functions. Given a list of mentions of an entity type, the entity representation learner (ERL) discovers the structured representations iteratively. The output of ERL is a mapping model that can be applied to other mentions with similar structured representations. The *structured representation* of a mention is a sequence of semantic units where each semantic unit has a *label* (e.g. $\langle suffix \rangle$ in a company) and a pattern *matcher*. In each iteration, ERL would select a mention and ask the user to provide labels for its semantic units. It would infer the matchers for the units and generate a mapping program using these matchers.

Our system uses two types of pattern matchers for the semantic units: built-in regex-based matchers and custom dictionary-based matchers. Dictionary-based matchers can be provided by the user or learned using our domain vocabulary learner (DVL). Once the structured representations are learned, the user can use LUSTRE to declaratively define transformations over these structured representations to synthesize programs for normalization and variant generation. The synthesized programs and the mapping model are packaged as APIs for downstream applications.

B. User Interaction Model

LUSTRE expects the user to provide a set of unlabeled mentions of an entity type of interest. Our domain vocabulary and entity representation learners follow a similar iterative learning process, wherein a mention is selected for the user to label in each iteration. The user has to provide labels for the tokens in the selected mention, such as concept name for a dictionary or label for a semantic unit. Both the learners also allow the user to provide feedback on their intermediate predictions. This interaction model reduces the number of mentions the user has to label and hides the complexities of the underlying learning process. We provide an expressive, declarative framework for generating programs for normalization and variant generation wherein the user only has to configure a set of generic transformations over the learned structured representations.

III. Learning Structured Representations

In LUSTRE, we aim to learn a set of programs that map entity mentions to their structured representations. Each program has a mapping strategy, consisting of a set of matchers, that decides how a mention is mapped to the semantic units of a representation.

LUSTRE has several pre-defined generic regular expression matchers that capture a token with: *(1) uppercase alphabetic characters, (2) lowercase alphabetic characters, (3) mixture of uppercase and lowercase alphabetic characters, (4) numeric digits, (5) alphabetic and numeric characters, (6) special non-word characters* Additionally, it can use a set of dictionary-based matchers: *dictionary* of words or phrases for a domain-specific concept. Although crucial in learning good domain-specific representations, such dictionaries are often unavailable or cannot be easily generated with limited contextual information. The user can learn such dictionaries using our domain-vocabulary learner. Our system learns how to combine these different matchers for complex semantic units in the structured representations.

To reduce both the skill and effort required in the learning process, our system hides the details on how the mentions are selected for the user to label or how the learned model generalizes to unseen mentions. The user only has to provide human-comprehensible labels for a handful of mentions and optionally feedback on a few intermediate predictions, using a user-friendly, web-based interface.

Domain Vocabulary Learner (Optional)

To learn dictionaries for an entity type, the user has to provide a list of mentions to our domain-vocabulary learner (DVL). Textually similar mentions are grouped by DVL using Hierarchical clustering. It uses an active learning based algorithm, wherein it selects representative mentions from one of the clusters in each iteration and asks the user to label them. From the user labels, DVL identifies a set of preliminary, incomplete dictionaries and incrementally complete these dictionaries in the subsequent iterations. This iterative process continues until all the mentions are processed or the user is satisfied with the dictionaries.

The user-interface for providing labels for concepts in the selected mention is shown in Fig 3a. The user can click to select one or more tokens (with dotted line) in the selected mention. For the selected token(s), such as the highlighted `VARSARTIS`, the user can provide a new custom label or select an existing label. It is not required for the user to label every token in the mention. The user can hit the "DONE WITH LABELING" button to complete the labeling. DVL then derives a generic extraction rule that is consistent with the user labels. For instance, it can derive a rule "⟨CapitalWord⟩⟨CapitalWord⟩,[2]->SUFFIX" that extracts the second token as `SUFFIX` in a two-token substring where both tokens have capital words. To extract more dictionary entries for the concept `SUFFIX`, DVL applies this extraction rule to other mentions in the selected cluster.

Additionally, DVL presents the most uncertain predictions to the user for verification (see Fig 3b). The user only has to uncheck the check-boxes corresponding to incorrect predictions. This feedback is used as labeled data for the extraction rule. In a subsequent generalization step, DVL

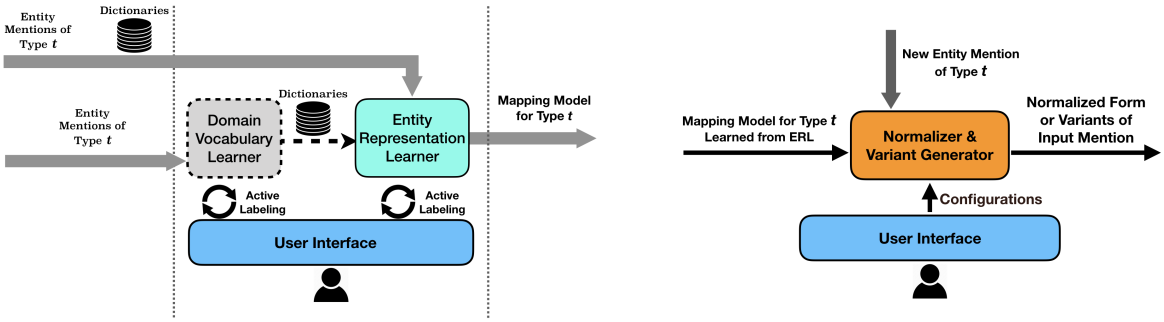


Figure 2: Learning Structured Representations (left); Synthesizing Normalization and Variant Generation Functions (Right)

applies the extraction rule to the entire dataset, identifies more dictionary entries, and seeks user labels on the uncertain predictions. Once the dictionaries are learned, the user can view and edit them before using them for learning structured representations.

Entity Representation Learner

Given a list of mentions of an entity type, ERL learns the structured representations as a function of dictionary-based and regular expression-based matchers. Specifically, it learns how to combine these matchers to constitute the semantic units in a representation such that the semantic units are consistent with the labels provided by the user. Like DVL, it uses an active learning algorithm to iteratively select candidate mentions for labeling. ERL selects a mention such that its structured representation is different from those of previously labeled mentions so different representations can be learned quickly with minimal user effort.

ERL uses a similar interface for labeling as DVL with one key difference. ERL expects the user to label all key semantic units in a mention. However, to make the task easier, it shows the labels for the semantic units that could be inferred from the dictionaries. For instance, given a suffix dictionary, it labels span `INC` as $\langle suffix \rangle$ and expects user to label `VERSARTIS` (e.g., as `corename`). Once it obtains the labels, ERL selects a mapping strategy from the many possible sequences of matchers that are consistent with the user labels. It generates a program for the strategy and updates its model of programs. For instance, from the multiple mapping strategies for `VERSARTIS INC`:

- ⟨corename:AlphaChar⟩⟨suffix:suffix-dict⟩
- ⟨corename:AlphaNum⟩⟨suffix:suffix-dict⟩

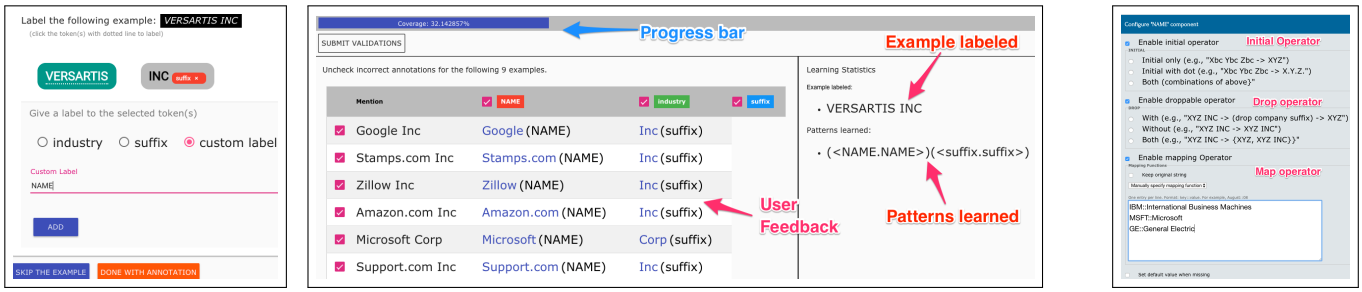
ERL follows a conservative approach and selects the first mapping strategy as the regular expression `AlphaChar` is more specific than `AlphaNum`. Since the programs learned are generic, ERL allows the user control the quality of the model learned by providing additional feedback on a few intermediate, uncertain predictions (see Fig 3b). The user only has to mark these as correct or incorrect. This feedback is used internally to reason how multiple mapping strategies can be used for a mention. This iterative process continues until all the mentions can be mapped to the learned representations or the user is satisfied with the quality of the model learned.

IV. Synthesizing Normalization and Variant Generation Functions

LUSTRE allows users to configure their normalization and variant generation functions over the structured representations learned from the mentions. Specifically, it provides a set of transformation operators that the user can configure to manipulate the semantic units of an unseen mention. These transformed semantic units are then used to compose meaningful variations of entities. LUSTRE provides a configuration interface, as shown in Figures 3d and 3c, to configure four types of transformation operators:

- Initial: Retain first character of each token in the token sequence of a semantic unit. The user can also specify whether a dot should be added to each initial character.
- Order: Define a specific order for the semantic units
- Drop: Ignore token sequence of a semantic unit
- Map: Replace one or more token sequences of a semantic unit with a user-provided string

It allows user to define multiple configurations for different tasks. Clicking the + symbol brings up a configuration panel with the learned semantic units. Optionally, the user can add custom units for special characters (such as "/ for separating month, day, year in date mentions). By reordering (drag-and-drop) and deleting the semantic units, the user can generate different configuration patterns. The user can further configure each unit in the pattern by clicking the edit button (Fig 3c shows the configuration options for $\langle name \rangle$ unit). The user can configure the normalization and variant generation functions independently, i.e., toggle "EXHAUSTIVE MODE" for normalization or variant generation. The configuration for variant generation allows all the transformation operators so that all possible variations with transformed semantic units can be generated (see Fig 3c). However, only some dynamically determined set of operators is used for normalization as all mentions have to be normalized to a unique representation. Fig 3d shows sample configurations for company name mentions. At runtime, these configured functions are applied to the structured representation of a mention. Consider the variant



(a) Labeling interface (b) User feedback (c) Configuration dialog

(d) Configuration interface (e) Variant generation tester

Figure 3: Screenshots of LUSTRE

generation configuration (the lower panel) shown in Fig 3d, wherein (1) $\langle industry \rangle$, $\langle suffix \rangle$, and $\langle location \rangle$ units can all be dropped, (2) if the span mapped to $\langle suffix \rangle$ is "Corp" then it can be transformed to "Corporation", (3) if the span mapped to $\langle location \rangle$ is "DE" then it can be transformed to "Germany", (4) abbreviate the span mapped to $\langle name \rangle$ (with and without dot). LUSTRE also allows the user to test run (see Fig 3e) and adjust the configuration interactively before exporting the corresponding programs for their applications.

V. Demonstration Flow

We will use a list of company names to show how LUSTRE helps a user learn programs for structured representations from a few labeled mentions, and configure normalization and variant generation functions (a demo video is available at https://youtu.be/hhM3BkvTZbE). The user would do the following:

1. *Executing Domain Vocabulary Learner.*
   a) Upload a data file containing company mentions.
   b) Using the interface, label a tokenized mention.
   c) Submit the labels. Go through the intermediate predictions and identify mistakes.
   d) Repeat steps a) and b) until all mentions are processed or the dictionaries meet user's needs. Use the dictionaries in learning structured representations.
2. *Executing Entity Representation Learner.*

   Learn structured representations of company mentions with ERL. The user interface of ERL is identical to that of DVL with two differences: (1) mentions shown in ERL may have been partially labeled, (2) the user needs to identify all the semantic units.
3. *Configuring Normalization & Variant Generation*
   a) Specify a normalization configuration and a variant generation configuration. For normalization, transform the company names into the canonical form "`NAME INDUSTRY SUFFIX (LOCATION)`". For variant generation, use pattern "`NAME INDUSTRY SUFFIX LOCATION`". Show how different transformations can be configured for each semantic unit with only a few clicks and simple text input.
   b) Test provided configurations on multiple company mentions having different structures (see Fig 3e).

The demo participants can also test LUSTRE with person names and dates or their own list of entity mentions.

References

[1] M. Dredze, P. McNamee, D. Rao, A. Gerber, and T. Finin, "Entity disambiguation for knowledge base population," COLING 2010, pp. 277–285.

[2] Z. Zheng, F. Li, M. Huang, and X. Zhu, "Learning to link entities with knowledge base," in *NAACL HLT*, pp. 483–491, 2010.

[3] J. Lehmann, S. Monahan, L. Nezda, A. Jung, and Y. Shi, "Lcc approaches to knowledge base population at tac 2010.," in *TAC*, 2010.

[4] X. Liu, Y. Li, H. Wu, M. Zhou, F. Wei, and Y. Lu, "Entity linking for tweets.," in *ACL (1)*, pp. 1304–1311, 2013.

[5] A. Arasu and R. Kaushik, "A grammar-based entity representation framework for data cleaning," SIGMOD '09, pp. 233–244.

[6] K. Qian, L. Popa, and P. Sen, "Active learning for large-scale entity resolution," CIKM '17, pp. 1379–1388, 2017.