Domain Independent Knowledge Base Population From Structured and Unstructured Data Sources

Michelle Gregory, Liam McGrath, Eric Bell, Kelly O'Hara, and Kelly Domico

Pacific Northwest National Laboratory
902 Battelle Boulevard
Richland, WA 99354
{michelle, liam.mcgrath, eric.bell, kelly.o'hara, kelly.domico}@pnl.gov

Abstract

In this paper we introduce a system that is designed to automatically populate a knowledge base from both structured and unstructured text given an ontology. Our system is designed as a modular end-to-end system that takes structured or unstructured data as input, extracts information, maps relevant information to an ontology, and finally disambiguates entities in the knowledge base. The novelty of our approach is that it is domain independent and can easily be adapted to new ontologies and domains. Unlike most knowledge base population systems, ours includes entity detection. This feature allows one to employ very complex ontologies that include events and the entities that are involved in the events.

Introduction

Ontologies are widely used in knowledge management, but are seeing resurgence within the NLP community in applications such as information extraction and question answering. More specifically, NLP applications are running in the context of external knowledge sources (i.e., ontologies). A knowledge base is a stored representation of information. The task of Knowledge Base Population (KBP) is identifying instances of your external knowledge source in textual data and storing them in a knowledge base. The incorporation of extracted information into an existing knowledge base is such a prevalent problem that it has had a special track at the Text Analysis Conference since 2008 (McNamee et al., 2010).

In this paper we present a system designed to extract text from both structured and unstructured data sources in order to populate a knowledge base based on an existing ontology. While others have introduced systems that are designed to do the same general task (see Maynard et al., 2010, who have shown that using a combination of rule-based approaches and machine learning approaches works well for knowledge base population tasks), our system is unique in that it is data type agnostic and it has been built in a modular manner that makes easy to generalize to new datasets and domains. In addition, our KBP system allows for very complex ontologies that include event structures in addition to entities and taxonomies.

System Description

Our system is an end-to-end process with modular components that populates a knowledge base with information extracted from structured data sources and unstructured natural language text. The process can be broken down into three phases: Extraction, mapping and entity disambiguation. The system architecture is shown in Figure 1.

Extraction and mapping for structured and unstructured data sources are handled separately. Often a single ontology is used to populate a knowledge base with evidence from both structured and unstructured data types. While mapping structured data to an ontology can be fairly straightforward, when both data types are present we combine them for improved performance. Structured data sources are processed first, as these high-confidence data can be used to inform the processing of unstructured data.

For the structured data sources, we use a rule-based system to map field in the data to types in the target ontology. These high-confidence data can then be added directly to the knowledge base. For unstructured data, we use a natural language processing (NLP) pipeline, described in the next section, to extract named entities and events from the input document and add them to an unmapped triple store. In the mapping phase, the extracted entities and events are then mapped to a domain-specific ontology.

Once all input documents have been processed, we disambiguate entities across the entire knowledge base. This is a cyclical process, where disambiguated entries are added back to the knowledge base, and processed again to ensure that all duplicate entities are found.

We have designed the system such that each phase occurs in independent modular components that can be modified or swapped out entirely. We describe each component in the following section. While many of our components are standard open source tools, we believe the novelty of our system lies in the modular design and functionality the combination provides.

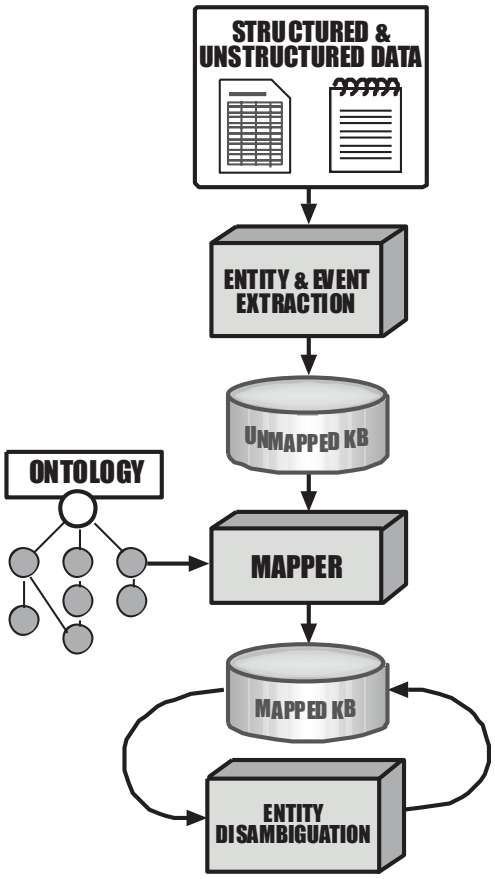

Figure 1 – The system consists of three main components, entity and event extraction, a mapper, and entity disambiguation. Structured data, unstructured data, and an ontology are the inputs to the system.

Processing Structured Data

In addition to unstructured textual data, often structured data is available as a source and it needs to be mapped to the ontology to augment the population of the knowledge base.

In order to map structured data into the knowledge base, a customized rule-based mapping was created that reads in the appropriate fields of the structured data and maps them to the corresponding ontologically valid entries. The resultant triples are then added directly into a mapped knowledge base that can be combined and de-duplicated with the knowledge base obtained from unstructured data sources. Figure 5 diagrams the flow of structured and unstructured data through our system.

However, the strength in integrating structured data comes from the inherent nature of that data. Structured data can be used as high-confidence data that helps populate and augment dictionaries for named entities, events, and entity disambiguation.

Entity and Event Extractor

Unstructured data sources naturally require more complex processing to identify, extract, and map entities and events. Here we describe the NLP pipeline we have created to process free natural language text.

The extractor is designed to take advantage of the capabilities of the Apache UIMA Framework; it consists of a series of components (annotators) that can be added or removed from the pipeline as desired. The full text of documents is internally represented in a specialized object. These objects allow the document to be marked up, by storing start and end indices of annotations as well as metadata provided by the annotator components. The information stored in these objects can be accessed by any component in the pipeline.

The extractor contains four components: two named entity extractors, an event finder and a triple producer. The first three components are independent of each other, and their definitions specify the set of annotation data types used by each component. The triple producer is an aggregator; it defines which

components to use and the order in which to run them, and combines their results. These components are described in more detail in the following sections below. A visual overview the extractor's overview in the overall system is shown in Figure 2.

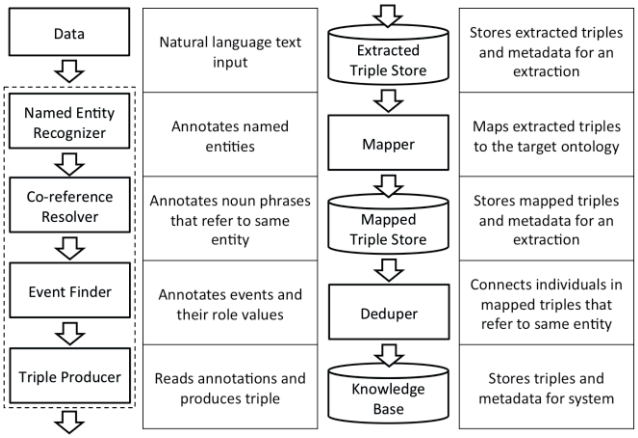

Figure 2 - The NLP Pipeline. Text is annotated with named entities and events. Then, OWL triples are created, mapped to the target ontology, and entities are disambiguated.

Named Entity Extraction

Our system is designed to extract both events and entities from the data. For named entity extraction, we run two independent named entity recognizers – one dictionary-based, and one statistical – and then combine the results.

Dictionary-based Named Entity Recognition. We built our dictionary-based named entity recognizer utilizing functionality from LingPipe[1]. User-created dictionaries are stored as text files. We have general purpose dictionaries, and domain-specific dictionaries can be added and edited easily if desired. For the most part we use exact dictionary matching to identify entities in the input text, but also include some heuristics for combining matches from dictionaries of common first and last names to identify people. Entities identified by this component are annotated with their start and end indices and the label defined in the dictionary. There are no restrictions on the labels that can be used in the dictionaries.

Statistical Named Entity Recognition. Our second NER component is based on the Stanford Named Entity Recognizer. We use the pre-trained model available from Stanford, which identifies entities and labels them as Person, Organization, or Location. These are added as annotations similar to the dictionary-based annotations.

Combining Results. After both recognizers have run, an aggregator combines the results. If an entity is identified by one recognizer but not the other, we extract the entity and entity type it provided. If the recognizers identify overlapping entities, we use the dictionary-based recognizer to determine both the entity type and, if necessary, the entity boundaries (if the entities identified overlap but don't completely align).

Event Extraction

Events provide a representation of complex relationships among entities. For example, one event type that is used regularly is *communicate*. This event requires two entities related via *meeting, calling, reporting, emailing, etc.* Event extraction is a necessary step in cross-domain knowledge base population. The event extraction process can be broken down into two steps: detection and argument identification.

Detection. Event detection involves finding event triggers: the lexical items in the text that best represent the event. The lexical items are usually verbs; for our purposes we only consider verbs. At this initial stage every verb is selected as an event trigger, and if some arguments are identified, it is stored as an event in the unmapped knowledge base. The selection of the events of interest comes later in the mapping step, allowing event extraction to remain domain independent. In this way

Argument Identification. Event identification involves finding arguments for each event trigger, i.e., the constituents that have a semantic role in the event. Resources such as PropBank and FrameNet define sets of argument types for verb or event types. For this stage of the process, we use a simple set of arguments across all event types, which is later mapped to a more complete set of event-specific arguments dependant on the target ontogloy: AGENT, the entity doing the event; PATIENT, the recipient or them of the event; and ARG, a catch-all for other arguments of the event. To identify and type the arguments, we use a simple set of rules over typed dependency parses from the Stanford Dependency Parser. Statistical parsers are not always completely accurate, but we find it to be sufficient for our purposes. See below for evaluation results.

Event detection and argument identification are run in a single UIMA component. As with the named entity extraction, events are annotated with their indices, with metadata containing the arguments, any temporal expressions modifying the verb, and flag if the verb is negated.

[1] http://alias-i.com/lingpipe

Triples

The extracted entities and events are initially stored as subject-predicate-object statements, or "triples", in a temporary Sesame RDF repository. This is the unmapped knowledge base shown in Figure 1. The triple producer is responsible for converting the annotations into triples and adding the triples into the unmapped triple store. All extracted data such as name, type, temporal information, negation and provenance are saved in this repository.

Mapper

Since one of the overall goals for our system is to be domain-independent, our event detection system uses a separate domain mapping component to be as flexible as possible. We use rule-based techniques for event detection in order to provide a simple mechanism for cross-domain application. However, we also provide an optional second-stage classification using supervised techniques to boost precision when labeled data can be created for a domain.

After a knowledge base of extracted entities and events has been created, the Mapper maps this data to the target ontology. This component enables the pipeline to remain domain independent, yet able to support applications using specific domain ontologies.

Entities and Events are mapped to the domain ontology according to a set of rules. A mapping rule consists of:

1. Target Event Type: the event type from the ontology that an event will become if the rule fires, e.g. *Deployment*

2. Triggers: the lexical items that will fire this rule if they occur as the event trigger, e.g., *introduce, upgrade, implement, launch, install, deliver, deploy ...*

3. Argument Mappings: from verb semantic arguments to event roles, e.g. *agent -> organization, patient -> technology*

Figure 3 shows an example event mapping.

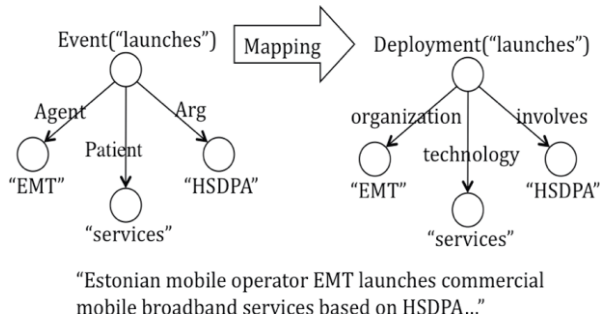

Figure 3 - An example event is mapped to the target ontology. Nodes represent individuals in the knowledge base and are labeled with lexical items from the text and a type. Edges connect events to entities and are labeled with a role type.

Mapping rules are created by hand with the help of some tools for lexical expansion. Using WordNet (Fellbaum,, 1998), each verb can be expanded to include synonyms and hypernyms, which will usually indicate an event of the same type. Note that the system is not dependent on use of handwritten rules, and a component for automatically generating mapping rules could easily be added.

If event-labeled data can be created for the target domain, a second classification step can be used to refine the event mapping. Statistical classification based on Naïve Bayes or k-Nearest Neighbors can be used to improve results. Feature selection is very important in the success of this component. Basic features, part of speech based features, and most importantly syntactic features (dependency relations in the current parse tree, parse tree depth, etc) should be included for the best overall performance (Ahn 2006, Bell 2010)

The mapped triples are then inserted into a hybrid triple store (the mapped knowledge base). This hybrid mapped triple store includes a Sesame RDF repository paired a SQLite relational database. Triples are stored in the RDF repository while provenance information associated with each triple is stored in the SQLite database.

Ontology

We us an OWL ontology to define the target domain. This provides a formal definition of the entities and events of interest to be used by the mapping rules. OWL Classes are used to define entities (e.g. Person, Organization, Location) and DatatypeProperties are

used to define properties on the entities (e.g. name, age, gender). OWL ObjectProperties are used to define relationships between entities (e.g. father, mother, brother). Events - relationships involving more than two entities or additional arguments, such as time and location, are represented using subclasses of an Event class (e.g. Attack, Deployment). ObjectProperties between Events and entities define the Event argument types, and restrictions on the entity types that can fill them (e.g. Attack has a perpetrator, which must be a Person or Organization).

Entity Disambiguation

Once the data has been mapped to the domain ontology, the entity disambiguation component identifies and labels duplicate entries in the knowledge base using simple proper name matching. This step is performed across the entire knowledgebase (i.e. cross-document).

Only entities with entity type Person, Organization and Location are considered to have proper names and, as such, de-duped; events are ignored. Additionally, abbreviations (e.g. *Incorporation* and *Inc.*) and equivalents (e.g. *United Kingdom* and *Britain*) are considered when matching the names. This list of valid abbreviations and equivalents is stored in a reference file that is read in by the disambiguator during initialization.

Duplicate entities are represented in the knowledgebase as entities with a sameAs relationship; this relationship is added for each matching entity to represent bi-directionality. The original entities are kept intact so that any incorrectly labeled entities can later be corrected. We hope to employ more sophisticated features in our entity disambiguation module, such as those described by Drezde et al. (2009).

Evaluation

It is difficult to evaluate our end-to-end system because there is no gold standard for evaluating knowledge base populations in which the ontology includes both entity and event information. For instance, the TAC evaluation datasets are largely based only on entity and location information. We were able, however, to conduct a number of evaluations on individual components as it is certainly the case that weak performance in the individual components will detract from the performance of the overall system. Below, we provide evaluations of various system components.

Event Detection. To evaluate the event extraction portion of our system, the statistical system for event detection was extended in order to run against the 2005 Automatic Content Extraction (ACE) corpus (Walker et al., 2006). The ACE event hierarchy describes an inherent ontology consisting of 8 major event types and 33 subtypes. The ACE corpus used consists of 666 text files.

Multi-way classification from a single event reference to a group of possible class labels was performed. In addition to positive references, negative occurrences were also used because they have been shown to improve classifier performance (Ahn, 2006, Bell et al., 2010). The system runs a single statistical classifier for all event classes, identical to the configuration used in similar evaluations (Bell, 2010). The results of running the statistical system against the ACE corpus are shown in Figure 4. The use of a supervised classifier brings performance to levels that rival that of state-of-the-art tools in the field.

	Rule-based Mapping	Classifier-based Mapping
Precision	0.37	0.74
Recall	0.18	0.53
F1-Measure	0.24	0.62

Figure 4 - Performance of Event Extraction at identifying and classifying events according to the target ontology. achieve close to human agreement, but is not feasible for all domain types.

Event Argument Identification. We evaluate using PropBank by mapping our AGENT, PATIENT and ARG to PropBank types Arg0, Arg1 and all remaining argument types respectively. One caveat in comparing these results to other PropBank evaluated tools is that we do not place the exact bounds restriction on ourselves, because we only identify the head of the argument. (This argument identification could be replaced with a more sophisticated semantic role labeling systems trained on PropBank or FrameNet such as without requiring any downstream code changes, just a different mapping. The

evaluation of argument identification is shown in Figure 5.

We used a modified propbank evaluation for argument detection. It is modified in that we were only concerned with two main argument types, *agent* and *patient*, as generally these were the only two arguments required by the ontology we were using. The rest of the argument types were lumped into a generic Arg category. While the results reported here are very good for argument identification, it should be noted that this modification produces slightly inflated results.

	Precision	Recall	F-measure
Agent	.76	.55	.64
Patient	.86	.39	.54
Arg	.85	.29	.43
Combined	.82	.40	.53

Figure 5 - Evaluation of Argument Identification using a modified PropBank evaluation.

Conclusion

In this paper we have introduced a novel system designed to populate a knowledge base from structured or unstructured text. Our system is designed in a modular format that allows for easy adaptation to new domains and ontologies. We have reported near state of the art results for the components we have built: event extraction, argument identification, and mapping.

We plan on designing a methodology by which we can test the entire system. One method might be to train annotators on a specific ontology and manually populating a knowledge base from unstructured or structured documents. This method would be laborious and expensive. We are also considering the types of evaluations that are more component based, such as testing on existing TAC or ACE datasets.

References

Ahn, D. 2006. *The Stages of Event Extraction*. ARTE'06: Proceedings of the Workshop on Annotating and Reasoning about Time and Events. Morristown, NH.: Association for Computational Linguistics.

Alias-i 2008. LingPipe 4.0.0. http://alias-i.com/lingpipe.

Bell, E., 2010. *Event Detection and Classification by Sense Disambiguation*. Seattle, WA.: University of Washington

Bell, E., McGrath, L., and Gregory, M. 2010. *Verb-Triggered Event Detection and Classification*. NW-NLP Workshop. Redmond, WA.: University of Washington.

Dredze, M., McNamee, P., Rao, D., Gerber, A., and Finin, T., 2010. *Entity Disambiguation for Knowledge Base Population*. Proceedings of Conference on Computational Linguistics (COLING).

Fellbaum, C,, editor. 1998. WordNet: An Electronic Lexical Database. MIT Press, Cambridge, Massachusetts.

Maynard, D., Li, Y. and Peters, W., 2008. *NLP Techniques for Term Extraction and Ontology Population* Buitelaar, P. and Cimiano, P. (eds.), Ontology Learning and Population: Bridging the Gap between Text and Knowledge, pp. 171-199, IOS Press, Amsterdam.

McNamee, P., Dang, H., Simpson, H., Schone, P., and Strassel S., 2010. *An Evaluation of Technologies for Knowledge Base Population* Proceedings of the Seventh Language Resources and Evaluation Conference (LREC).

Stanford University 2009. Stanford Named Entity Recognizer (NER). http://nlp.stanford.edu/software/CRFNER.shtml

Walker, C., Strassel, S., Medero, J., and Maeda, K. 2006. ACE 2005 Multilingual Training Corpus. CD-ROM.