Learning to Extract Information for the Semantic Web

Christian Siefkes*

Berlin-Brandenburg Graduate School in Distributed Information Systems†
Database and Information Systems Group, Freie Universität Berlin
Takustr. 9, 14195 Berlin, Germany
christian@siefkes.net

Abstract: The goal of information extraction (IE) is to find desired pieces of information in natural language texts and store them in a form that is suitable for automatic querying and processing; the Semantic Web provides formats and standards for storing and processing such data. Hence IE-based automatic or semi-automatic approaches are a promising approach for building the Semantic Web. My Ph.D. thesis focuses on developing and refining trainable algorithms for this purpose.

1 Introduction

1.1 Information Extraction

Most of the information stored in digital form is hidden in natural language (NL) texts. While *information retrieval* (IR) helps to locate documents which might contain the facts needed, there is no way to answer queries. The goal of *information extraction* (IE) is to find desired pieces of information (*slot fillers* and *templates*) in NL texts and store them in a form that is suitable for automatic querying and processing. Slot fillers contain a single piece of data (e.g., a name or a date); templates contain a list or tuple of slot fillers and/or other templates (e.g., an *address list* contains a list of *address* templates, a *person/job title/company relationship* contains a tuple of three slot fillers). Thus templates can be arranged in a hierarchical structure.

The precursor of information extraction, *text understanding*, tried to create a complete formal representation of the contents of a text, but this aim has been found over-ambitious and impossible to realize. To avoid this trap, IE requires a predefined output representation (template structure) and only searches for facts that fit this representation. All other information contained in the input text is simply ignored, as are aspects of language that resist formalization, e.g., the intentions and moods of the authors.

*Supervisor: Prof. Dr. Heinz F. Schweppe, Database and Information Systems Group, FU Berlin.

†This research is supported by the German Research Society (DFG grant no. GRK 316).

1.2 The Semantic Web

There is another approach to providing text-related data in a form suitable for automatic querying and processing: the *Semantic Web*[1]. While IE aims to extract relevant information from NL texts, the Semantic Web encodes data related to a text (or any URI) in an explicit form, typically in *RDF*[2] format. The explicitly provided data allows more refined IR and automatic querying through ontology languages such as *OWL*[3].

The term *metadata* is sometimes used in a wide sense to refer to any information expressed in RDF or similar formats; in a strict sense it refers to data about documents (e.g., title, author, language of a text). I will use the term *semantic annotations* for the wide sense, reserving my use of *metadata* to the strict sense.

The Semantic Web does not determine how semantic annotations are produced. The most typical approach is to write them manually. While this leads to high-quality results if done well, it is a slow and tedious process. Annotations built this way will always be limited to a small part of the World Wide Web because many authors will not take the additional effort to provide them; they will often be out-of-date because an author forgets to update them when changing a document.

Hence automatic or semi-automatic approaches are an attractive alternative for building the Semantic Web. Such approaches utilize IE techniques to generate machine-readable semantic annotations from NL texts. The resulting data cannot be expected to reach the quality of manually written data; it is certain to contain errors and gaps that can be detected by human judgment only. It is still useful to propose an initial set of semantic annotations as input for a human annotator, thus alleviating the laborious task of writing them manually; and to cover areas not (yet) treated by humans, since imperfect data is (in many cases) better than none. When trainable methods are used to produce the semantic annotations, they can learn from feedback provided by human correctors, thus avoiding to make the same mistakes over and over.

My Ph.D. thesis focuses on developing and refining such trainable algorithms that employ IE techniques for helping to build the Semantic Web.

2 What to Extract: Target Structure and Output Format

The main task of IE consists in filling predefined templates with facts extracted from texts. Typically this is done in two steps: *element filling* to find suitable slot fillers and if necessary normalize them (conversion to the required target datatype); and *template unification* to resolve coreferences and assemble the slot fillers into templates.

In IE, there are no standard formats for specifying target structures and for storing extracted templates. In the Semantic Web context, an obvious choice is to use RDF for

[1] `http://www.w3.org/2001/sw/`
[2] `http://www.w3.org/RDF/`
[3] `http://www.w3.org/TR/owl-guide/`

storing the extracted facts and consequently *RDF Schema*[4] for specifying the target structure.

RDF is built on the concepts of *resources* and *properties*. Everything described by RDF is an `rdfs:Resource`, identified by an URI or by a literal. RDF resources are typed. Except for the container (`rdfs:Container`) and collection (`rdf:List`) classes that allow to group resources, there is no need for build-in types in RDF, since the XML Schema datatypes[5] or any compatible typing schema can be used. The type (class) hierarchy can be extended through the `rdfs:subClassOf` property.

An `rdf:Property` (also called *predicate*) is a relationship between two kinds of resources (*subject* and *object* of the property)—the combination of subject, predicate, and object is called a *triple*. Property relationships are typed: the required type of the object is called the `rdfs:range` of a property; its `rdfs:domain` specifies the type of the subject. Properties themselves are arranged in a type hierarchy: a property *P1* is a `rdfs:subPropertyOf` of a property *P0* if all resources related by *P1* are also related by *P0*.

In addition to the *inheritance hierarchy* (superclass/subclass), there is another kind of hierarchy that is important for entity recognition: the *aggregation hierarchy*, i.e., has-part/is-part-of relations. Aggregation relates a record with its members, e.g., *address* contains *street*, *city*, *ZIP code*. It can also relate a list with its elements, e.g., *address* is part of *address list*. The Dublin Core Metadata Initiative[6] recommends `isPartOf` and `hasPart` properties that could be used to express aggregation.

For IE, resources can be considered candidates for slot fillers; while properties can express very simple templates (expressing a relationship between two slot fillers). More complex templates can be expressed through aggregation or through a conjunction of properties. E.g., the previously mentioned *person/job title/company* relationship is expressed when the object of a *jobTitleOf* property and the subject of a *worksFor* property refer to the same *person* entity.

3 How to Extract: Learning Algorithm and Context Representation

3.1 Linguistic Preprocessing: Making the Text Structure Visible

RDF resources and properties are thus arranged in a class hierarchy. On the other hand, the input text originally might appear to be flat data without visible structure; just a sequence of characters. Yet this is a wrong impression—there is structure in any text. At a low level, text can be considered as a sequence of tokens (words, numbers, punctuation). In natural language texts, tokens are arranged in sentences. Several sentences are grouped in paragraphs, which are grouped in sections (which in turn might be grouped in higher-order sections). By interpreting this structure, a text can be converted to a tree representation. In structured text formats (e.g., HTML, DocBook or other XML/SGML-based formats)

[4] `http://www.w3.org/TR/rdf-schema/`
[5] `http://www.w3.org/TR/xmlschema-2/`
[6] `http://www.dublincore.org/documents/dcmi-terms/`

the higher-level structure (usually down to the paragraph level) is explicitly coded, but the lower-level structure (sentences; sentence elements like verb groups or noun phrases; tokens) must usually be induced. Existing software components can be used or adapted for this purpose.

A tokenizer is used to split the text into tokens. Tokens are the lowest-level leafs in the tree (above the character level). To provide additional context information, tokens are typically tagged, both syntactically (part of speech tagging, morphologic analysis) and semantically (synonym sets and hypernym/hyponym relations provided by a thesaurus). A sentence splitter divides the text into sentences. A parser builds a (more or less complete) linguistic representation of each sentence. Most parsers split a sentence into "chunks" like verb groups, noun phrases and prepositional phrases.

3.2 The Main Tasks: Hierarchical Entity and Relationship Recognition

When an RDF Schema defines the target structure, the main task of IE is to extract suitable RDF triples and store them in an RDF document that conforms to the predefined Schema. A triple is a *relationship* between two typed *entities* (resources), so a preparatory step is to find all entities in a document and identify their types.

3.2.1 Entity Recognition and Normalization

Entity recognition can be handled as a classification task, where each token (typically a word) in a text is classified as the begin of an entity of a certain type, as a continuation of the previously started entity (if any), or as not belonging to any entity. The type hierarchy of RDF Schema (expressed by the `subClassOf` property) is relevant for entity recognition: in an initial step, only the top-most types (those without a `subClassOf` property) of entities must be determined. The classification of entities is refined in further steps, determining for each found entity whether it belongs to one of the direct subclasses of the original type. This process iterates until the type hierarchy is exhausted.

The entity recognizer can utilize the aggregation hierarchy in the same way it uses the type hierarchy, first locating entities whose type is not part of any other type and then iteratively locating the parts of the containing entity. In a first step, a large *address list* might be identified in the text. The list is split in a number of *addresses* in the second step. Finally, *street*, *city*, *ZIP code* and other parts of each address are determined.

The raw form that is used to express an entity in the text will often differ from the ideal format that should be used for entities of this type. E.g., a *date* referred to in a text as *October 15th, 2003* should be stored as *2003-10-15* according to ISO 8601. Aggregation can be very useful for such *normalizations*: It would require complex rules to convert a whole date from any of the forms used in texts to a standard form, but when its three parts (*year=2003; month=October; day=15th*) have also been tagged, the task becomes considerably easier. For entity types that are limited to a finite number of known instances (the 12 months of a year, the up to 31 days of a month), classification can handle the rest.

3.2.2 Coreference Resolution and Relationship Recognition

The goal of *coreference resolution* is to decide whether two entity references of the same type denote the same individual. Coreference resolution can be treated as a binary classification of whether or not two entities are likely to corefer, based on their contents (contained tokens and aggregated entities), respective positions (distance from each other, location within the containing element etc.), and contexts. Coreference can be considered a special kind of relationship: the `owl:sameIndividualAs` relationship defined by OWL.

The existence of other relationships (RDF properties) can be determined in a similar way, by classifying whether a relationship exists between an entity of the domain type and an entity of the range type required by this property.[7] After coreferences have been resolved, it should be sufficient to restrain the search space to entities mentioned near to each other in the text.

RDF properties are arranged in a type hierarchy just as other RDF resources, allowing an iterative refinement of the recognized relationships. So after finding an *isRelativeOf* property between two entities, in the next step the algorithm can decide whether to refine this to a subproperty such as *isSisterOf*.

Some relationships are exclusive, while others allow an entity to take the domain resp. range role of several such relationship. So somebody can be the object (range) of several *isSisterOf* properties, but only of one *isMotherOf* property. In the first case, the classifier can accept several "competing" relationships, while in the second case it should only keep the most likely one. Such cardinality constraints cannot be expressed in RDF Schema, but require a more powerful language such as OWL which allows `owl:cardinality`, `owl:minCardinality`, and `owl:maxCardinality` restrictions on properties.[8]

3.3 The Learning Algorithm

The learning algorithm should fulfill several functional requirements:

1. It should be trainable on a wide range of input texts and linguistic preprocessing components.
2. It should provide a measure of confidence in its results.
3. It should degrade gracefully, delivering as much results as possible when a complete analysis fails.
4. The algorithm should be able to update the extraction model without starting from scratch, thus allowing incremental training.
5. It should respond in a reasonable time—efficiency is not a top priority, but it should not be neglected.

Approaches to information extraction can be classified into three categories: hand-written rule-based algorithms (e.g., *GATE/Annie* [CHGW97]); rule-based machine learning (ML)

[7] A similar approach to *relationship recognition* is used by [MCF+98].

[8] OWL Lite only allows cardinality values of 0 or 1 (or unlimited) which should be enough for most purposes.

algorithms (e.g., *WHISK* [Sod99]); and statistical methods (e.g., *maximum entropy modeling* [Rat98]).

Hand-written rules are obviously not trainable and thus fail the first requirement. Machine-learned rules usually work in a purely binary "all or nothing" fashion, while statistical approaches attach a probability or weight to their decisions, as postulated by the second and third requirement.

Many statistical algorithms are unsuited for incremental training. Among those that remain are the "sparse binary polynomial hashing" of the *CRM114*[9] text classifier [Yer03] and the context modeling technique used by *PPM (Prediction by Partial Match)* [WBMT99] (which is also adaptable to hierarchical contexts [Che01]). I will adapt and extend these or other suited incrementally trainable statistical algorithms for the purposes of my work.

3.4 Hierarchical Context Representation

Training requires an *anchoring* of extracted entities and relationships with their original representation in the input text (augmented text). Then a trainable algorithm can learn to classify them based on their textual contents and contexts.

Typically, the *context window* considered by IE algorithms comprises either the nearest tokens/words (e.g., [Cir01]) or some predefined syntactic elements of the current sentence (e.g., [SFAL95]). Regarding the input text as a hierarchical tree structure results in a more flexible context model: the context of a node contains the nearest nodes around it—its parents, children and (near) siblings. Thus the context window can include non-local information, e.g., a section heading or the head and stub cells of a table.

For this purpose, *dependency trees* (where content can also be attached to internal nodes) are more suited than simple trees (where all content is represented in the leafs). In a dependency tree, a section element has the section title as content and other elements like subsections and paragraphs as child nodes. In XML, dependency trees can be simulated through a cross-referencing mechanism (a node can have a "head-child" attribute that references one of its direct child nodes).

The actual content representation will depend on the used input formats and preprocessing components, as there is no single canonical way of embedding a text in a tree structure. This is no serious hindrance, as the learning algorithm does not postulate any particular representation, but will adjust to the given features. The same context representation must be used for training and evaluation/application, thus a change in preprocessing components will require a full retraining.

[9] `http://crm114.sourceforge.net/`

4 What to Do: Application Scenarios

For evaluation and demonstration purposes, the chosen algorithms and context representations should be tested and utilized in several related application scenarios:

Metadata Extraction: The goal of this task is to extract metadata in the strict sense, i.e., data about text, based on the Dublin Core standard. Extracting metadata from a text requires mainly entity recognition, as the object of the relationship is already known (the text itself).

Filtered Information Retrieval allows to constrain terms in IR queries to selected entity types (e.g., documents in which *Ford* refers to a person, not a company).

Entity Referencing augments web pages with cross-references from contained entities to external background information, e.g., to *CiteSeer*[10] for bibliographic entries, to a map for locations, or to an encyclopedia entry.

Semantic Highlighting and Indexing: Semantic highlighting marks all entities of a certain type in a text or all coreferences to a chosen entity (maybe on mouse-over via Dynamic HTML). Semantic Indexing shows a hyperlinked index of the names of all persons (or any other entity type) mentioned in a document. An index can also list the subjects or objects of a chosen relationship, e.g., all persons working for company X.

Making such demo applications publicly available is not only useful as a showcase, but especially for capturing user feedback that can be employed for further training or for evaluation.

5 Related Work

Approaches that employ similar methods have been referenced throughout the text. Approaches that have related goals comprise the ML-generated CiteSeer citation index [LGB99], the *Web⇒KB*[11] project (where rules are learned to extract predefined entities and relations from Web pages [CDF+00]), and the *CORPORUM*$_{OntoExtract}$ system [EBJ01] (that uses grammar-based methods to extract Dublin Core metadata and "light-weight ontologies" from NL texts). [KSS01] use the classification and annotation data provided by Web directories such as *Open Directory* for learning indicator phrases for extracting metadata from texts. All of these approaches utilize rule learning.

I am not aware of any other projects using incrementally trainable statistical methods for Semantic Web–related information extraction. The employment of general hierarchical models for both input and output representation is another novel trait of my approach.

[10] `http://citeseer.nj.nec.com/`
[11] `http://www-2.cs.cmu.edu/~webkb/`

References

[CDF+00] Mark Craven, Dan DiPasquo, Dayne Freitag, Andrew K. McCallum, Tom M. Mitchell, Kamal Nigam, and Seán Slattery. Learning to Construct Knowledge Bases from the World Wide Web. *Artifi cial Intelligence*, 118(1/2):69–113, 2000.

[Che01] James Cheney. Compressing XML with Multiplexed Hierarchical PPM Models. In *Proceedings of IEEE Data Compression Conference*, pages 163–172, 2001.

[CHGW97] Hamish Cunningham, Kevin Humphreys, Robert Gaizauskas, and Yorick Wilks. Software Infrastructure for Natural Language Processing. In *5th Conference on Applied Natural Language Processing*, Washington, 1997.

[Cir01] Fabio Ciravegna. $(LP)^2$, an Adaptive Algorithm for Information Extraction from Web-related Texts. In *Proceedings of the IJCAI-2001 Workshop on Adaptive Text Extraction and Mining*, Seattle, USA, 2001.

[EBJ01] R. H. P. Engels, B. A. Bremdal, and R. Jones. CORPORUM: A Workbench for the Semantic Web. In *Proceedings of the Semantic Web Mining Workshop at ECML/PKDD-2001*, 2001.

[KSS01] Martin Kavalec, Vojtech Svátek, and Petr Strossa. Web Directories as Training Data for Automated Metadata Extraction. In *Proceedings of the Semantic Web Mining Workshop at ECML/PKDD-2001*, 2001.

[LGB99] Steve Lawrence, C. Lee Giles, and Kurt Bollacker. Digital Libraries and Autonomous Citation Indexing. *IEEE Computer*, 32(6):67–71, 1999.

[MCF+98] Scott Miller, Michael Crystal, Heidi Fox, Lance Ramshaw, Richard Schwartz, Rebecca Stone, Ralph Weischedel, and the Annotation Group. Algorithms That Learn to Extract Information—BBN: Description of the SIFT System as Used for MUC. In *Proceedings of the Seventh Message Understanding Conference (MUC-7)*, 1998.

[Rat98] Adwait Ratnaparkhi. *Maximum Entropy Models for Natural Language Ambiguity Resolution*. PhD thesis, University of Pennsylvania, 1998.

[SFAL95] Stephen Soderland, David Fisher, Jonathan Aseltine, and Wendy Lehnert. CRYSTAL: Inducing a Conceptual Dictionary. In Chris Mellish, editor, *Proceedings of the Fourteenth International Joint Conference on Artifi cial Intelligence*, pages 1314–1319, San Francisco, 1995. Morgan Kaufmann.

[Sod99] Stephen Soderland. Learning Information Extraction Rules for Semi-Structured and Free Text. *Machine Learning*, 34(1–3):233–272, 1999.

[WBMT99] Ian H. Witten, Zane Bray, Malika Mahoui, and W. J. Teahan. Text Mining: A New Frontier for Lossless Compression. In James A. Storer and Martin Cohn, editors, *Proceedings Data Compression Conference*, pages 198–207, Los Alamitos, CA, 1999. IEEE Press.

[Yer03] William S. Yerazunis. Sparse Binary Polynomial Hashing and the CRM114 Discriminator. In *Proceedings of the 2003 Spam Conference*, Cambridge, MA, 2003. MIT.