

Exposé—Dissertation Plans

Learning for Information Extraction Based on Tree-like Models

Christian Siefkes

20th March 2003

(Initial Version: 10th January 2003)

Contents

1	Introduction and Overview	2
1.1	Information Extraction	2
1.2	Text Mining and Other Related Areas	2
1.3	IE as Text Transformation Using Tree Models	3
1.4	Context in Trees	4
2	Application Flow: Building and Transforming the Tree	5
2.1	Building the Tree	5
2.2	Learning to Transform to the Output Structure	6
3	The Learning Algorithm	6
4	Aims and Requirements	7
	References	8

1 Introduction and Overview

1.1 Information Extraction

The goal of information extraction (IE) is to transfer “items of interest” expressed in a natural language text in a more formal representation, e.g. a database. What these “items of interests” are is not always clear. I will call them *facts* with the following definition:¹

A *fact class* is a pair of a key (name) and a typed value slot. The type of the value slot is either a simple data type² (atomic type/fact) or a list of facts of the same class or a tuple of facts of different classes (compound type/fact). A *fact* is an instantiation of a fact class whose value slot is filled with a suitable value.³

Thus facts are arranged in a hierarchical structure. The structure of the output representation, i.e. the classes of the expected facts, must be predefined. Facts can be represented by elements in XML (or more exactly *Minimal XML*⁴ as they cannot have any attributes or mixed content) or by entities in an E/R model.

1.2 Text Mining and Other Related Areas

Hearst [1999] defines the goal of data mining and also of text (data) mining (TM) as “to discover or derive new information from data, finding patterns across datasets, and/or separating signal from noise”, while the goal of information retrieval (IR) is “to help users find documents that satisfy their information needs”.

IE takes a middle position between IR (locating relevant texts) and Text Mining (generating new knowledge by finding unknown patterns). It does not try to generate new knowledge, but only to represent items of interest (facts) explicitly represented in a text in a more formal structure (cf. table 1).

Information Retrieval	Information Extraction	Text Mining
finding documents or text segments	filling predefined structures	discovering and filling unknown structures

Table 1: Applications of IR, IE, and Text Mining

¹ Adapted from the EC formalism used by *Stalker* [Muslea et al. 2001, sec. 2].

² According to some standard like XML Schema or Java, e.g. a string, number, or date.

³ This is only a tentative model that is probably too simple—additionally, there should be an cross-referencing mechanism between facts to avoid duplication.

⁴ <http://www.docuverse.com/smldev/minxml.jsp>

IE can be used as a first step in text mining, by extracting facts from the unstructured text to a database or other structured representation. In a second step, usual data mining techniques can be applied to the resulting database structure to discover interesting relationships in the data. This approach is utilized by Nahm and Mooney [2000].

The structure filled by IE can also be utilized for more flexible IR (using a structured query language like SQL). Thus IE might also be useful as a preparatory step for information retrieval.

The extracted information can also be used in databases and ontologies for further processing (while IE extracts explicit facts, ontologies might extract additional implicit knowledge).

1.3 IE as Text Transformation Using Tree Models

As defined above, facts are arranged in a hierarchical way. Each fact can be regarded as a tree whose leafs are atomic facts; the expected result of IE is a forest of such trees.

On the other hand, the input text originally appears to be flat data without visible structure⁵; it is 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) 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.

Any approach to IE utilizes at least some of this structure. So IE can be understood as multi-level text transformation: initially it transforms the input text into a structured representation (e.g. a tree model) to make implicit structures visible; then it converts (parts of) the resulting representation to the required target representation.

IE thus transforms the more-or-less unstructured input data to the output format as required by the fact classes to be filled. In intermediate steps, additional transformations are applied to add more structure to the text (sentence splitting,

⁵ Assuming we have plain text or disregarding formatting or markup information for the moment.

parsing etc.), which can be utilized for the final transformation where most of the original data is discarded.

In this regard, IE can be compared to the transformation standard of XML, XSLT⁶ (XSL Transformations—XSL is the *Extensible Stylesheet Language*). XSL also employs a tree model for element access. Navigation and reference to other elements in the tree is possible through XPath⁷, the XML Path Language.

However, IE goes in the other direction than “normal” XSL transformations. XSLT is usually utilized to transform data-centered (highly structured) documents either to surface-centered representations (less structure or fewer dependencies between document structure and underlying data) for presentation purposes or to other data-centered documents for processing by other application. IE converts surface-centered texts to data-centered representations.

1.4 Context in Trees

Regarding IE as a tree transformation leads to a re-definition of the concept of *context*. Typically, the context window considered by IE algorithms comprises either the nearest tokens/words (e.g. *LP*)² [Ciravegna 2001] or some predefined syntactic elements of the current sentence (e.g. *Crystal* [Soderland et al. 1995]).

Utilizing tree structures 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. the 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 leaf). 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).

⁶ <http://www.w3.org/TR/xslt20/>

⁷ <http://www.w3.org/TR/xpath20/>

2 Application Flow: Building and Transforming the Tree

2.1 Building the Tree

The goal of the preparatory first steps of IE is to make the structure contained in a text visible and usable. For this purpose, a tree model is built from the (more or less) flat text document. Additionally, the nodes of the tree can be enhanced with additional data to provide input that can be utilized in later transformation steps (these data are typically called “feature structures”; they can be stored in attribute/value pairs of the XML elements).

For building the tree, existing software components should be used or adapted—there are solutions available for any of the steps described below.

The Token and Entity Level

A *tokenizer* is used to split the text into *tokens* (words, numbers, punctuation). Tokens are the lowest-level leaves in the tree (above the character level).

For further processing, tokens will typically be *tagged*:

Syntactic tagging analyzes the part of speech (POS) role of each token (whether it is a noun, a verb etc.). For languages with rich inflection like German this should include a morphologic analysis.⁸

Semantic tagging establishes semantic similarities between word tokens, by arranging them in synonym sets and hypernym/hyponym relations (super-ordinated/sub-ordinated concepts) as specified by a general dictionary and/or a domain-specific thesaurus. Alternatively or additionally, the semantic similarity of word tokens can be determined by statistical approaches like *LSA* (Latent Semantic Analysis) [Deerwester et al. 1990].

A somewhat higher level of leafs in the tree are *named entities* (names of persons, organizations/companies, locations) and other *generic entities*, e.g. quantitative values (monetary amounts, percentages, dates, times). It is useful to handle them early because it makes things easier for sentence splitting, as punctuation characters often occur in entities.

The Sentence Level

A *sentence splitter* divides the text into sentences.

⁸ Case, gender, voice etc. of words can be stored in under-specified features structures; stemming identifies the base form of a word.

3 The Learning Algorithm

A *parser* builds a (more or less complete) linguistic tree representation of each sentence. Most parsers split a sentence into “chunks” like verb groups, noun phrases and prepositional phrases, which in turn can be parsed further down to the token or entity level.

The Document Level

At the highest level, a document can be split into sections of different order, which are split into paragraphs and into other structures like lists and tables. Structured documents formats explicitly contain these higher-order structures. They are typically ignored in traditional IE but can be induced by *wrapper induction* systems like Stalker [Muslea et al. 2001].

2.2 Learning to Transform to the Output Structure

The main task of IE consists in filling the predefined fact classes with facts extracted from the texts (“template filling”). Typically this occurs in two steps:

- Element filling: find suitable slot fillers (atomic facts) and normalize them (conversion to the required target data type) if necessary.
- Unification: merge the atomic facts into compound facts, handling co-reference resolution as required.

This will be the central focus of my work. My approach is to regard these tasks as tree transformations. This requires an *anchoring* of facts in the target structure with nodes in the tree representation of the original text. Then a learning algorithm can detect suitable transformations from the input text tree to the target tree, utilizing the context model described above (sec. 1.4).

3 The Learning Algorithm

Approaches to information extraction can be classified into three categories:

- Hand-written rule-based algorithms (e.g. *GATE/Annie* [Cunningham et al. 1997])
- Rule-based machine learning algorithms (e.g. (LP)², *Crystal*, or *WHISK* [Soderland 1999])
- Stochastic methods (e.g. *maximum entropy modeling* [Berger et al. 1996; Ratnaparkhi 1998])

Hand-writing the transformation rules will not be considered, because this would require a complete rewrite for porting the solution to another application domain or language (or just for exchanging one of the components that were used to build the tree).

So the other options remain: learning the transformations, one way or the other. In the ML domain, *inductive learning algorithms* as used by Crystal are especially interesting. Crystal learns by unification of similar rules through generalization (constraint relaxation), thus making the high search space tractable [Soderland et al. 1995, sec. 3]. In the statistical field, entropy-based methods like maximum entropy or the context modeling technique used by *PPM (Prediction by Partial Match)* [Witten et al. 1999] tend to be quite successful. A third option might be data mining methods for discovering association rules like the *Apriori* algorithm [Agrawal and Srikant 1994].

Active learning techniques (as used by WHISK [Soderland 1999, sec. 1.4]) are a useful refinement: these techniques select examples at the boundaries of the rule set for hand-tagging, thus reducing the training cost. Another way of minimizing the human effort consists in employing the current model to propose a solution for correction by the user. The corrected sample is then used for further training of the model. For this purpose (among others), the learning algorithm should be suitable for *incremental training*, e.g. refining the extraction model without needing a re-run on the complete training set. This might be hard or impossible for many statistical methods where an implicit model is built, and quite a challenge for rule-based algorithms. It is also hard to see how purely statistical approaches can handle slot normalization.

4 Aims and Requirements

In my dissertation, I plan to refine the proposed tree models for input (text) and output (facts classes) and to choose a representation of the context of nodes in trees that is convenient for learning.

I will develop a learning algorithm suitable for tree transformations and—in cooperation with the members of the FEx project—integrate it in a system for information extraction. The system should include components for building the tree and a user interface for interactive training.

I will employ this approach for finding facts in natural language texts and combining them to fill the desired fact hierarchy. I will evaluate the resulting system, refining it on the basis of the evaluation results.

References

The learning algorithm should fulfill these functional requirements:

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

There are some additional implementation requirements:

- The architecture and development API of the system should be clean and well-documented.
- The software should be portable to different systems (this should not be hard to realize, as Java is the chosen implementation language).

References

- Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, pages 487–499. Morgan Kaufmann, 1994. URL <http://citeseer.nj.nec.com/agrawal94fast.html>.
- Adam L. Berger, Stephen A. Della Pietra, and Vincent J. Della Pietra. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71, 1996. URL <http://www.cs.cmu.edu/~aberger/ps/compling.ps>.
- Fabio Ciravegna. (LP)², an adaptive algorithm for information extraction from Web-related texts, 2001. URL <http://www.smi.ucd.ie/ATEM2001/proceedings/ciravegna-atem2001.pdf>.
- 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. URL <http://citeseer.nj.nec.com/cunningham97software.html>.
- Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990. URL <http://citeseer.nj.nec.com/deerwester90indexing.html>.

- Marti A. Hearst. Untangling text data mining. In *Proceedings of ACL'99: the 37th Annual Meeting of the Association for Computational Linguistics*. University of Maryland, 1999. URL <http://www.sims.berkeley.edu/~hearst/papers/acl99/acl99-tdm.html>.
- Ion Muslea, Steven Minton, and Craig A. Knoblock. Hierarchical wrapper induction for semistructured information sources. *Autonomous Agents and Multi-Agent Systems*, 4(1/2):93–114, 2001. URL <http://citeseer.nj.nec.com/muslea01hierarchical.html>.
- Un Yong Nahm and Raymond J. Mooney. Using information extraction to aid the discovery of prediction rules from text. In *Proceedings of the Sixth International Conference on Knowledge Discovery and Data Mining (KDD-2000) Workshop on Text Mining*, Boston, MA, 2000. URL http://www-2.cs.cmu.edu/~dunja/KDDpapers/Nahm_TM_IE.ps.
- Adwait Ratnaparkhi. *Maximum Entropy Models for Natural Language Ambiguity Resolution*. PhD thesis, University of Pennsylvania, 1998. URL <ftp://ftp.cis.upenn.edu/pub/ircs/tr/98-15/98-15.ps.gz>.
- Stephen Soderland. Learning information extraction rules for semi-structured and free text. *Machine Learning*, 34(1–3):233–272, 1999. URL <http://citeseer.nj.nec.com/soderland99learning.html>.
- 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 Artificial Intelligence*, pages 1314–1319, San Francisco, 1995. Morgan Kaufmann. URL <http://citeseer.nj.nec.com/soderland95crystal.html>.
- 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. URL <http://www.cs.waikato.ac.nz/~nzdl/publications/1999/IHW-ZB-MM-WJT-Text-Mining.p>