5 Non-Statistical Approaches

While most modern approaches to information extraction are based on a statistical models, the field has been originally dominated by non-statistical approaches, and any discussion of the work in this area would be incomplete without covering them. Most non-statistical approaches are focused on learning *extraction rules* from the annotated training data, while a few others rely heavily on provided *knowledge bases* such as ontologies or thesauri.

Since the focus of our own work is in the area of statistical approaches, the discussion of non-statistical trainable systems has been limited to representatives of a few typical models. Please see [Sie05b] for other rule-learning and knowledge-based approaches, such as case-based approaches and approaches based on automatic pattern and template creation. That paper also contains additional details on the algorithms presented in the following sections.

5.1 Covering Algorithms

A number of IE systems are based on *covering (separate-and-conquer) algorithms* [Für99], a special type of *inductive learning*. These systems learn rules that extract the tagged attribute values from an annotated training set. After learning rules that cover a part of the training instances, they remove (separate) these instances from the training set and continue to learn rules that cover (conquer) some of the remaining instances, looping until all or most of the training instances are covered. What is regarded as an instance and which features are considered depends on the system.

Crystal: *Crystal* [Sod95, Sod97a] is one of the earliest trainable IE systems. It builds on a chunk parser that identifies syntactic constituents (subject, verb phrase, direct and indirect object, prepositional phrases) and a domain-specific dictionary that specifies semantic classes for all words. Crystal looks for constituents that fit predefined conceptual types (e.g. *diagnosis, symptom*) and subtypes (a diagnosis is either *confirmed, ruled-out, suspected, pre-existing,* or *past*).

The definitions learned to extract subtypes identify a constituent to extract if certain constraints are fulfilled by the surrounding constituents. Constraints may test for word sequences contained in a phrase or for semantic classes of the head noun or a modifier of a phrase. For example, an "absent symptom" is extracted from the direct object if the head of the direct object is of the class *[Sign or Symptom]*, the verb is "denies" in the active voice, and the subject includes the word "patient" and has the head class *[Patient or Disabled Group]*. Negative constraints are not supported.

In a later work [Sod01], the problem of negation is solved by learning different kinds of semantic relations (classes) in a predefined order. Since rules for negative cases are

learned first, specialized negative rules such as *verb group includes "not observed"* are learned (and applied) first, and the examples covered by these rules are removed prior to learning general rules like *verb group includes "observed"* for the positive case.

Crystal learns suitable definitions by generalization, i.e. bottom-up: each training instance is used as a highly constrained initial definition. Crystal tries to unify "similar" definitions by relaxing constraints. Two definitions are unified by finding the most restrictive constraints that cover both.

For template unification (multi-slot extraction), Crystal treats each subset of attribute combinations as a concept to be learned—this can result in data sparseness. Crystal does not extract exact phrases, it only identifies a constituent to extract from. These are the major limitations of the system.

Whisk: The *Whisk* system developed later by the same author [Sod99] is aimed at handling a larger range of texts, from free texts as found in newspapers and books to semi-structured texts (often ungrammatical or in "telegram style") that are common on the World Wide Web or in advertisements.

Whisk is targeted at handling template unification (multi-slot extraction) at the sentence level. The learned rules similar to regular expressions; they can contain verbatim text, character classes (e.g. digit), and wildcards like "*" which lazily skips any characters until the next part of the pattern can match. In addition to the hard-wired character classes, semantic classes of equivalent terms can be defined by the user, e.g. a class *Bdrm* that contains different forms and abbreviations of the term "bedroom."

Rules are derived top-down (starting with the most general rule) by a covering algorithm. For judging the quality of rules, Whisk uses the Laplacian expected error: $Laplacian = \frac{e+1}{n+2}$, where n is the number of extractions made and e is the number of errors among these. In case of a tie, the more general rule is used. The found rules might not be optimal due to the limitations of hill climbing—each specialization is evaluated in isolation, so if two specializations (adding two terms) must be applied together to yield a better rule (according to the Laplacian), they will not be found.

Whisk incorporates *active learning*, so only a small part of the training corpus needs to be tagged in advance. The system proceeds by selecting three kinds of untagged instances for hand-tagging by the user: instances covered by a rule (which will either increase the support of the rule or force further refinement), "near misses" (to check and adapt the boundaries of rules), and a random sample of instances not covered by any rule (to check whether there are still rules to discover).

A disadvantage is that semantic classes must be predefined by the user, they are not learned by the system. Another drawback is the strict ordering constraints of each rule—different rules must be learned for each possible arrangement of attributes.

(LP)^2: *(LP)^2* [Cir01] learns rules to add SGML/XML tags to a text. (LP)^2 is based on *tagging rules* that insert a single (starting or ending) SGML tag into the text. This means that the task of each rule is to recognize the start or the end of a supposed attribute value in the text, not to extract/tag a whole attribute value (or several attribute values) at once, as in most other systems.

The tagging rules are learned from the hand-tagged training corpus. Rules are learned bottom-up, taking an instance as an initial rule whose constraints are subsequently relaxed (e.g. requiring only a lexical class instead of a specific word) or completely dropped. The k best generalizations of each initial rule found by a beam search are stored in a "best rules pool." As $(LP)^2$ is a covering algorithm, the training instances covered by a rule in this pool are removed from the training set.

$(LP)^2$ proceeds in four steps:

1. The *tagging rules* from the "best rules pool" are applied.
2. *Contextual rules* are applied to resulting text. These are tagging rules whose overall reliability was not high enough for the best rules pool but that perform better when restrained to the vicinity of tags inserted in the first step (for example, a rule that inserts an end tag is applied provided that a corresponding start tag occurred some words before).
3. *Correction rules* do not add or delete tags, they only change the position of a tag, moving it some words forward or backward.
4. Finally, invalid markup (unclosed tags etc.) is deleted in a *validation* step.

In the *Amilcare* system, $(LP)^2$ is employed in a "LazyNLP" setting where the amount of utilized linguistic information can be dynamically adjusted [Cir02]. The learner initially induces rules without any linguistic knowledge; then it iterates adding linguistic information, stopping when the effectiveness of the generated rules no longer increases. The adequate amount of linguistic input is learned for each attribute separately, since recognizing a person name might require more NLP input than recognizing a date or time.

$(LP)^2$ is targeted at fragment extraction and does not perform any template unification. In Amilcare a shallow discourse representation module is added for this purpose [Han02, Sec. 5]. Attribute values are unified in templates or subtemplates with the nearest preceding attribute values of a suitable type. E.g. when describing hotels, *address* data and *room types* (single room, double room) will be attached to the last mentioned *hotel*; *price* information might in turn be attached to the last mentioned *room type*.

5.2 Relational Rule Learners

The basic approach of the systems presented in this section is similar to those of the previous section—indeed, they are based on covering algorithms too. The main difference is that the systems presented here explicitly take relations—especially positional relations—between a (potentially unlimited) number of features into account, while those in the previous section are limited to predefined (finite) combinations of features.

Rapier: The *Rapier* [Cal98a, Cal03] system uses syntactic (POS tags) and semantic (WordNet classes) information to induce rules for attribute values. Each rule consists in three parts, a pre-attribute value pattern, a pattern for the actual attribute

value and a post-attribute value pattern. Each pattern contains an ordered list (whose length might be zero for pre/post-attribute value patterns) of constraints that restrict the POS tag, the semantic class, and/or the word itself (disjunctions are allowed). Instances are most specific rules with all their constraints set. The pre- and post-attribute value patterns of instances contain every word from the start / to the end of the document, there is no "context window" of limited length.

New rules are created by randomly selecting two rules and creating the least general generalization for the attribute value pattern. Actually, there are several reasonable generalizations (different values of a constraint can be disjuncted or the constraint can be simply dropped), so each of these generalizations is re-specialized by adding generalized pieces of the pre- and post-attribute value patterns of the original rules. A list of n best candidates is kept until the best generalization is found.

Semantic classes are generalized by finding the nearest common ancestor in the WordNet hypernym hierarchy (dropping the constraint if no common ancestor exists). Instances covered by the found best generalization are subsequently ignored and further rules are learned based on the other instances.

Rapier has also been extended to use *active learning* [Tho99].

SRV: *SRV* [Fre98b] considers any combination of simple features (mapping a token to a value, e.g. *word length: 5, character type: alpha, POS tag: noun*) and relational features (mapping a token to another token, e.g. *next-token*, *subject-verb*). Feature values can be sets, e.g. all synonyms and hypernyms (superordinate concepts) listed by WordNet are combined in a set for each token. Different rule sets are learned for classifying each text fragment as an instance or non-instance of a single attribute value; there is no component for template unification or other postprocessing.

The learning algorithm is similar to the relational rule learner FOIL [Qui95]. SRV learns top-down, greedily adding predicates of some predefined types: the number of tokens in the fragment (*length*), whether a condition is matched by one or several (*some*) or by all (*every*) tokens in the fragment; *position* specifies the position of a token in a *some* predicate, *relpos* constrains the ordering and distance between two tokens. Rules are validated and their accuracy estimated by three-fold cross validation; the three resulting rule sets are merged. The accuracy estimations are available for each prediction.

An advantage of relational learners is their being able to acquire powerful relational rules that cover a larger and more flexible context than most other rule-learning and statistical approaches. The downside is that the large space of possible rules can lead to high training times and there is no guarantee of finding optimal rules (local maxima problem).

5.3 Wrapper Induction

The approach of *wrapper induction* (WI) is mainly targeted at structured and semi-structured documents that have been generated automatically, e.g. Web pages offering products or listing events.

Stalker: *Stalker* [Mus01] is a classical WI algorithm that covers documents that can be described in the so-called *embedded catalog (EC)* formalism. This formalism represents a document as a tree whose leaves contain the relevant data (items of interest for the user). The inner nodes contain lists of k-tuples (e.g. of restaurant descriptions). Each item in a tuple is either a leaf or another (embedded) list. Extraction is based of the EC description of a document and an extraction rule that extracts the contents of each node or tuple from the contents of its parent. List nodes require an additional *list iteration rule* that splits the list into tuples.

Extraction rules are based on groups of successive tokens called *landmarks*. *Start rules* locate the start of an item by find the first matching landmark from the begin of the parent; *end rules* locate the end of the item by finding the last matching landmark before the end of the parent. The text matched by a landmark itself can either be included (SkipUntil condition) or excluded (SkipTo condition) from the item text. Rules can combine several conditions, e.g. SkipTo(Name) SkipTo(<b>) means that the item starts immediately after the first HTML tag <*b*> that follows the word *Name*. They can refer to specific tokens or to wildcards like Number, Punctuation, or HtmlTag.

Disjunctions (either ... or) are allowed to handle formatting variations. Disjunctions are ordered so the first successful match is used. Each node is extracted independently of the other nodes within its parent, so no fixed ordering is required. Rules are learned by a covering algorithm that tries to generate rules until all instances of an item are covered (without false extractions, if possible) and returns a disjunction of the found rules. Rules with fewer false extractions (or more correction extractions, in case of a tie) are preferred when ordering the disjunction.

In a later work [Mus03], several variants of Stalker are combined in a *Co-Testing* approach to support *active learning*.

Boosted Wrapper Induction: Typical WI algorithms such as Stalker are only suited for documents whose structure and layout are regular and consistent. They are inadequate for free text, where information is mainly expressed in natural language. The *BWI (Boosted Wrapper Induction)* system [Fre00a] aims at closing this gap and making WI techniques suitable for free text.

The rules learned by BWI are simple contextual patterns for finding the start and end of the field to extract. A pattern has two parts: a token sequence that immediately precede/follow the field to extract (outside) and a token sequence starting/ending it (inside); these sequences can also contain wildcards. These specialized simple patterns will often reach high precision but low recall because there are many other ways to express a fact, especially in natural language texts. To address this issue, a large number of simple patterns are learned and their results combined. For this purpose BWI applies the technique of *boosting,* i.e. repeatedly applying the learning algorithm to the training data, each time adjusting the weight of training examples to emphasize those examples where the algorithm failed before.

While one of the goals of BWI is to make WI algorithms suitable for free (unstructured) text, BWI still performs significantly worse on free text than on highly or

partially structured text [Kau02]. Most patterns learned from free text merely memorize specific training examples. Also the algorithm is biased towards overfitting to the particularities of the training data—the final rounds of boosting actually lower the reliability of the results. Both precision and recall on free texts can be increased by incorporating the output of a shallow parser into the model, splitting the text into a number of noun, verb, and prepositional phrase segments [Kau02, Sec. 8].

5.4 Hybrid Approaches

The *IE*^{2} system [Aon98] submitted by SRA International to the MUC-7 conference is an early IE system that combines classical hand-written rules with trainable components in an interesting way. The output of a standard named entity recognizer is complemented by a custom component that recognizes domain-specific attribute values (e.g. different kinds of vehicles). Another component recognizes domain-specific types of noun phrases and relations between them (e.g. *employee_of*, *location_of*). Both these components are based on hand-written rules, no learning is involved.

However, the IE^2 system goes further than most other IE approaches in also handling template unification beyond the sentence level. For coreference resolution, different strategies are employed: one strategy uses simple hand-written rules, but another one learns decision trees from a tagged corpus. Optionally these strategies are combined in a hybrid method where the decision tree algorithm works on a subset of possible candidates chosen by the hand-written rules.

IE^2 also handles *event merging*, i.e. deciding whether or not two descriptions refer to the same event and can be merged. Here hand-written rules are combined with external knowledge sources to check the consistency of locations (*Miami* is in *Florida*) and times (can *Wednesday* and *tomorrow* refer to the same day within the current text?).

While in most aspects IE^2 is a typical representative of the classical hand-written rules approach that was dominant in the MUC conferences, its hybrid nature has interesting traits. Template unification and event merging beyond the sentence level are complex challenges that so far have been largely out of reach for learning systems. Combining trainable modules with external knowledge sources and specialized hand-written code could be a viable approach to tackle problems where single-paradigm solutions fail.

5.5 Knowledge-based Approaches

There are only a few extraction algorithms that are mainly based on knowledge sources. This section describes the thesaurus-based *TIMES* system as an exemplary algorithm. For other knowledge-based algorithms, see [Sie05b, Sec. 4].

The *TIMES* system developed by Bagga and Chai [Bag97] requires a number of knowledge sources: the WordNet thesaurus, a general English dictionary, a domain-specific dictionary, and a gazetteer of location names. Texts are preprocessed with an entity recognizer that identifies named and numeric entities and a partial parser.

Training is done by a user through a graphical interface. For each of the head words identified by the parser, the user selects the appropriate sense (concept) if WordNet defines several senses for this word. Then the user builds a *semantic network* to represent the content of each training text. Selected head words from the text are stored as nodes or relations within the network. For example, from the phrase *IBM Corp. seeks jobs candidates in Louisville*, the user might build a relation *seek* between two nodes *IBM Corp.* and *job candidate*.

The text-specific extraction rules created this way are then generalized according to the hypernym/hyponym (super-/subordinate terms) relations defined in WordNet. Generalization replaces a term by its hypernym n steps higher in the WordNet hierarchy. For named entities (NE), the category determined by the NE recognizer is generalized. E.g. *IBM Corp.* is identified as a company—generalizing this concept one step yields business, concern; three steps yields organization. A generalized rule matches any terms that are hyponyms of the generalized term. Increasing the generalization level results in higher recall at the cost of precision, because the generalized rules find instances missed by specialized rules but also produce more false positives.

In later versions of the system, the user only has to mark the target information to extract from a text. The system automatically builds relations between the marked information and generalizes extraction rules to the most suitable level [Cha99]. Since the hypernyms of a word are sense-dependent, the extended version also learns rules for sense disambiguation of head words based on the user-provided word senses.