Information Extraction from HTML: Application of a General Machine Learning Approach*

Dayne Freitag
Department of Computer Science
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
dayne@cs.cmu.edu

Abstract

Because the World Wide Web consists primarily of text, information extraction is central to any effort that would use the Web as a resource for knowledge discovery. We show how information extraction can be cast as a standard machine learning problem, and argue for the suitability of relational learning in solving it. The implementation of a general-purpose relational learner for information extraction, SRV, is described. In contrast with earlier learning systems for information extraction, SRV makes no assumptions about document structure and the kinds of information available for use in learning extraction patterns. Instead, structural and other information is supplied as input in the form of an extensible token-oriented feature set. We demonstrate the effectiveness of this approach by adapting SRV for use in learning extraction rules for a domain consisting of university course and research project pages sampled from the Web. Making SRV Web-ready only involves adding several simple HTML-specific features to its basic feature set.

The World Wide Web, with its explosive growth and ever-broadening reach, is swiftly becoming the default knowledge resource for many areas of endeavor. Unfortunately, although any one of over 200,000,000 Web pages is readily accessible to an Internet-connected workstation, the information *content* of these pages is, without human interpretation, largely inaccessible.

Systems have been developed which can make sense of highly regular Web pages, such as those generated automatically from internal databases in response to user queries (Doorenbos, Etzioni, & Weld 1997) (Kushmerick 1997). A surprising number of Web sites have pages amenable to the techniques used by these systems. Still, most Web pages do not exhibit the regularity required by they require.

There is a larger class of pages, however, which are regular in a more abstract sense. Many Web pages come from collections in which each page describes a single entity or event (e.g., home pages in a CS department; each describes its owner). The purpose of such a page is often to convey essential facts about the entity it describes. It is often reasonable to approach such a page with a set of standard questions, and to expect that the answers to these questions will be available as succinct text fragments in the page. A home page, for example, frequently lists the owner's name, affiliations, email address, etc.

The problem of identifying the text fragments that answer standard questions defined for a document collection is called *information extraction* (IE) (Def 1995). Our interest in IE concerns the development of machine learning methods to solve it. We regard IE as a kind of text classification, which has strong affinities with the well-investigated problem of document classification, but also presents unique challenges. We share this focus with a number of other recent systems (Soderland 1996) (Califf & Mooney 1997), including a system designed to learn how to extract from HTML (Soderland 1997).

In this paper we describe SRV, a top-down relational algorithm for information extraction. Central to the design of SRV is its reliance on a set of *token-oriented* features, which are easy to implement and add to the system. Since domain-specific information is contained within this features, which are separate from the core algorithm, SRV is better poised than similar systems for targeting to new domains. We have used it to perform extraction from electronic seminar announcements, medical abstracts, and newswire articles on corporate acquisitions. The experiments reported here show that targeting the system to HTML involves nothing more than the addition of HTML-specific features to its basic feature set.

Learning for Information Extraction

Consider a collection of Web pages describing university computer science courses. Given a page, a likely task for an information extraction system is to find the title of the course the page describes. We call the title a *field* and any literal title taken from an actual page, such as "Introduction to Artificial Intelligence," an *instantiation* or *instance* of the title field. Note that the typical information extraction problem involves multiple fields, some of which may have multiple instantiations in a given file. For example, a course page might

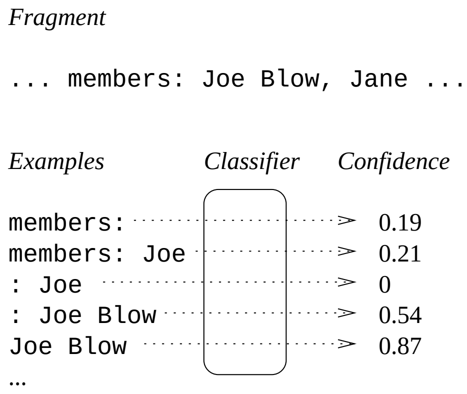

Figure 1: Information extraction as text classification. Above, a hypothetical fragment of text from a document describing a research project. Below, some of the corresponding *examples*. Each is assigned a number by a classifier designed to recognize, say, project members. This illustration assumes that any fragment containing two or three tokens (terms or units of punctuation) is an example, but the actual range in example lengths depends on the task.

include, in addition to the title, the official course number and the names of the instructors. In traditional IE terms, this collection of tasks is a *template*, each field a *slot*, and each instantiation a *slot fill*.

The general problem of information extraction involves multiple sub-tasks, such as syntactic and semantic pre-processing, slot filling, and anaphora resolution (Cardie 1997). The research reported here addresses only the slot filling problem, and we use the term *information extraction* to refer to this problem. In all results we report, therefore, our system is attempting to solve the following task: *Find the best unbroken fragment (or fragments) of text to fill a given slot in the answer template.* Note that this task is attempted for each field in isolation. A solution to this task, simple as it may seem, could serve as the basis for a large number of useful applications, involving both Web and non-Web documents. Moreover, our focus allows us to study more carefully the behavior of the machine learning approaches we develop.

Extraction as Text Classification

Information extraction is a kind of text classification. Figure 1 shows how the problem of finding instances of a "project member" field can be re-cast as a classification problem. Every candidate instance in a document is presented to a classifier, which is asked to accept or reject them as project members, or more generally, as in the figure, to assign a score to each, the size of which is its confidence that a fragment is a project member.

In contrast with the document classification problem, where the objects to classify (documents) have clear boundaries, the IE problem is, in part, to *identify* the boundaries of field instances, which always occur in the context of a larger unit of text. One way to implement a learned classifier for Figure 1 would be to treat each fragment as a mini-document and use a bag-of-words technique, as in document classification. There is reason to believe, however, that this would not yield good results. The terms in a fragment by themselves do not typically determine whether it is a field instance; its relation to surrounding context is usually of great importance.

Relational Learning

Relational learning (RL), otherwise known as inductive logic programming, comprises a set of algorithms suited to domains with rich relational structure. RL shares with traditional machine learning (ML) the notion of a universe consisting of class-labeled instances and the goal of learning to classify unlabeled instances. However, in contrast with traditional ML, in which instances are represented as fixed-length attribute-value vectors, the instances in a relational universe are embedded in a domain theory. Instance attributes are not isolated, but are related to each other logically. In a typical covering algorithm (e.g., CN2 (Clark & Niblett 1989)), predicates based on attribute values are greedily added to a rule under construction. At each step the number of positive examples of some class is heuristically maximized while the number of negative examples is minimized. Relational learners are rule learners with on-the-fly feature derivation. In addition to simple attribute-value tests, a relational learner can also logically derive new attributes from existing ones, as, for example, in FOIL (Quinlan 1990).

SRV

Our learner must induce rules to identify text fragments that are instances of some field. When presented with an instance of the field, these rules must say "yes"; when given any other term sequence drawn from the document collection, they must say "no." The set of positive examples for learning, therefore, is simply the set of field instances. Because the set of all possible text fragments is intractably large, however, we make the assumption that field instances will be no smaller (in number of terms) than the smallest, and no larger than the largest seen during training. Any non-field-instance fragment from the training document collection which matches these criteria is considered a negative example of the field and counted during induction.

Features In a traditional covering algorithm, the learner is provided with a set of features, defined over examples, which it can use to construct predicates. Unfortunately, multi-term text fragments are difficult to describe in terms of simple features. In contrast, individual terms (or tokens), lend themselves much more readily to feature design. Given a token drawn from a document, a number of obvious feature types suggest themselves, such as length (e.g., single_character_word),

word	punctuationp	sentence_punctuation_p
capitalized_p	all_upper_case	all_lower_case
numericp	singletonp	hybrid_alpha_num_p
doubletonp	tripletonp	quadrupletonp
longp	**prev_token**	**next_token**

Table 1: SRV"core" token features. The two features in bold face are relational features.

character type (e.g., numeric), orthography (e.g., capitalized), part of speech (e.g., verb), and even lexical meaning (e.g., geographical_place). Of course, a token is also related to other tokens by a number of different kinds of structure, and this structure suggests other, relational feature types, such as adjacency (e.g., next_token) and linguistic syntax (e.g., subject_verb).

SRV differs from existing learning systems for IE by requiring and learning over an explicitly provided set of such features. These features come in two basic varieties: *simple* and *relational*. A simple feature is a function mapping a token to some discrete value. A relational feature, on the other hand, maps a token to another token. Figure 1 shows some of the features we used in these experiments. We call these the "core" features, because they embody no domain-specific assumptions.

Search SRV proceeds as does FOIL, starting with the entire set of examples—all negative examples and any positive examples not covered by already induced rules—and adds predicates greedily, attempting thereby to "cover" as many positive, and as few negative examples as possible. An individual predicate belongs to one of a few predefined types:

- length(Relop N): The number of tokens in a fragment is less than, greater than, or equal to some integer. Relop is one of $<$, $>$, or $=$. For example, length($= 3$) accepts only fragments containing three tokens.
- some(Var Path Feat Value): This is a feature-value test for some token in the sequence. An example is some(?A [ ] capitalizedp true), which means "the fragment contains some token that is capitalized." One argument to this predicate is a variable. For a rule to match a text fragment, each distinct variable in it must bind to a distinct token in the fragment. How the Path argument is used is described below.
- every(Feat Value): Every token in a fragment passes some feature-value test. For example, every(numericp false) means "every token in the fragment is non-numeric."
- position(Var From Relop N): This constrains the position of a token bound by a some-predicate in the current rule. The variable From takes one of two values, fromfirst or fromlast. These values control whether the position is specified relative to the beginning or end of the sequence. For example, position(?A fromfirst < 2) means "the token bound to ?A is either first or second in the fragment."
- relpos(Var1 Var2 Relop N): This constrains the ordering and distance between two tokens bound by distinct variables in the current rule. For example, relpos(?A ?B $= 1$) means "the token bound to ?A immediately precedes the token bound to ?B."

At every step in rule construction, all documents in the training set are scanned and every text fragment of appropriate size counted. Every legal predicate is assessed in terms of the number of positive and negative examples it covers.[1] That predicate is chosen which maximizes the gain metric used in FOIL.

Relational paths Relational features are used only in the Path argument to the some predicate. This argument can be empty, in which case the some predicate is asserting a feature-value test for a token actually occurring within a field, or it can be a list of relational features. In the latter case, it is positing both a relationship about a field token with some other nearby token, as well as a feature value for the other token. For example, the assertion:

some(?A [prev_token prev_token] capitalized true)

amounts to the English statement, "The fragment contains some token preceded by a capitalized token two tokens back." There is no limit to the number of relational features the learner can string together in this way. Thus, it is possible in principle for the learner to exploit relations between tokens quite distant from each other. In practice, SRV starts each rule by considering only paths of length zero or one. When it posits a some-predicate containing a path, it adds to its set of candidate paths all paths obtained by appending any relational feature to the path used in the predicate. In this way, it builds its notion of field context outward with each rule.

Validation In the experiments reported here, each rule in a learned rule set is validated on a hold-out set. A randomly selected portion (one-third, in this case) of the training data is set aside for validation prior to training. After training on the remaining data, the number of matches and correct predictions over the validation set is stored with each rule. In order to get as much out of the training data as possible, this procedure—training and validation—is repeated three times, once for each of three partitions of the training data. The resulting validated rule sets are concatenated into a single rule set, which is used for prediction. Figure 2 summarizes all the steps involved in training SRV.

During testing, Bayesian m-estimates are used to assess a rule's accuracy from the two validation numbers (Cestnik 1990). All rules matching a given fragment are used to assign a confidence score to SRV's extraction of the fragment. If C is the set of accuracies for matching

[1]For example, a *position*-predicate is not legal unless a *some*-predicate is already part of the rule. See the following discussion for restrictions on how the relational component of the *some*-predicate is used.

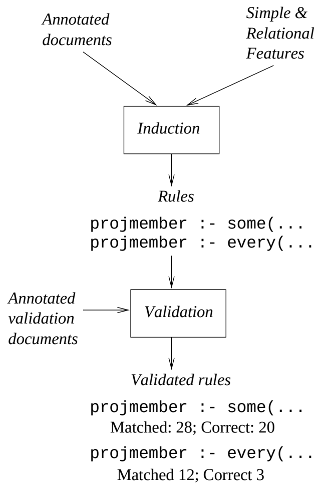

Figure 2: Input/output of the SRV algorithm.

rules, then the combined confidence is $1 - \prod_{c \in C}(1 - c)$. We found in practice that this yields better results than, say, taking the score of the single matching rule of highest confidence.

SRV and FOIL It is important to distinguish between SRV and FOIL, the general-purpose learning algorithm on which it is based. FOIL takes as input a set of Horn clauses, which define both the set of training examples, as well as the structure of the search space. SRV, in contrast, takes a document collection tagged for some field and a set of *features* (see Figure 2). Although it might be possible to get some of the functionality of SRV by encoding a field extraction problem in first-order logic (an ungainly encoding at best), it is doubtful that FOIL would perform as well, and it certainly would perform less efficiently. In addition to heuristics SRV shares with FOIL, it encompasses many additional heuristics which render its search through typically huge negative example sets tractable. Examples are handled implicitly, on a token-by-token basis; because a token is generally shared by many examples (see Figure 1), this permits a much more rapid accounting than if examples were explicitly given. And SRV's exploration of relational structure is restricted in a way that makes sense for the IE problem; in contrast with traditional ILP systems, for example, it cannot infer recursive rules.

Experiments

To test our system, we sampled a set of pages from the Web and tagged them for relevant fields. This section describes the data set and the approach we took to adapting SRV for HTML.

Adapting SRV for HTML

Making SRV able to exploit HTML structure only involved the addition of a number of HTML-specific features to its default set. Table 2 shows the features we added to the core set for these experiments. The *in* features return true for any token occurring within the scope of the corresponding tag. The *after* features return true only for the single token following the corresponding tag. The feature in_emphatic is a disjunction of in_i, in_em, in_b, and in_strong. In addition, several relational features were added which capture relations between tokens occurring together in HTML tables.

in_title	in_a	in_h
in_h1	in_h2	in_h3
in_list	in_tt	in_table
in_b	in_i	in_font
in_center	in_strong	in_em
in_emphatic	after_br	after_hr
after_p	after_li	after_td
after_th	after_td_or_th	
table_next_col	**table_prev_col**	
table_next_row	**table_prev_row**	
table_row_header	**table_col_header**	

Table 2: HTML features added to the core feature set. Features in bold face are relational.

Data

For these experiments we sampled and labeled course and research project pages from four university computer science departments: Cornell, University of Texas, University of Washington, and University of Wisconsin. Course pages were labeled for three fields: course title, course number, and course instructor. Course instructor was defined to include teaching assistants. Project pages were labeled for two fields: project title and project member. After collection and tagging, we counted 105 pages in our course collection and 96 in our research project collection.

Procedure

To gauge the performance of SRV, we partitioned each of the two data sets into training and testing sets several times and averaged the results. We tried SRV using the core features shown in Table 1, and again with this same set augmented with the HTML features shown in Table 2. We also performed separate experiments for each of two ways of partitioning: *random* partitions, in which a collection (course or project) was randomly divided into training and testing sets of equal size; and *leave-one-university-out* (LOUO) partitioning, in which pages from one university were set aside for testing and pages from the other three university were used for training. Using random partitioning, we split each data set 5 times into training and testing sets of equal size. The numbers reported for these experiments represent average performance over these 5 sessions. Using LOUO partitioning, the numbers reported represent average performance over a 4-fold experiment, in which each single experiment involves reserving the pages from one university for testing.

		Full		≈ 80%		≈ 20%	
		Acc	*Cov*	*Acc*	*Cov*	*Acc*	*Cov*
Course Title							
ran	html	45.7	94.4	53.8	80.4	100.0	19.6
ran	core	33.0	91.1	37.2	79.9	51.1	20.1
unv	html	48.5	96.6	55.7	79.5	83.3	20.5
unv	core	30.8	93.2	32.9	79.5	31.6	20.5
Course Number							
ran	html	84.0	100.0	87.2	80.1	91.7	19.9
ran	core	84.1	98.3	86.9	80.1	89.6	19.9
unv	html	79.2	95.7	81.1	73.4	87.0	22.3
unv	core	52.1	95.7	65.8	75.5	77.3	20.2
Project Title							
ran	html	26.3	99.5	31.0	79.9	50.0	20.1
ran	core	7.7	99.0	8.2	79.4	9.8	20.6
unv	html	34.0	97.5	39.2	80.0	81.2	20.0
unv	core	9.9	93.8	11.3	80.0	6.3	20.0

Table 3: SRV performance on "one-per-document" fields. The label *ran* stands for random partitioning; *unv* stands for leave-one-university-out partitioning. Rows labeled *core* show SRV's performance using the feature set shown in Table 1; those labeled *html* used the same feature set augmented with the features shown in Table 2.

Results

We distinguish between two kinds of IE task, depending on whether a field is likely to have only one or multiple instantiations in a document. For example, a research project page refers to a single project, so can have only a single project title, even though the project title may occur (and be tagged) multiple times in the page. On the other hand, a project typically has multiple members.

The former case, "one-per-document" (OPD), is obviously a much simpler IE task than the latter. Although a system, heuristic or learned, may recognize multiple project titles in a page, we can simply take the single most confident prediction and have done with it. In the case of project member, a "many-per-document" (MPD) field, a system will ideally extract every occurrence. The answer of the system in this case to the extraction task must be to return every prediction it makes for a document.

Table 3 shows the performance of SRV on the three OPD fields: course title, course number, and project title. Here, the unit of performance is a single file. Accuracy (*Acc*) is the number of files for which a learner correctly identifies a field instance, divided by the number of files for which it made a prediction. Coverage (*Cov*) is the number of files containing instances for which any prediction was made.

It is common in machine learning and information retrieval to try to exploit the spread in the confidence of learner predictions by trading off coverage for increased accuracy. Perhaps by sacrificing all predictions except

Field		Full		≈ 20%	
		Prec	*Rec*	*Prec*	*Rec*
Course Instr.	ran html	21.6	55.9	63.6	20.1
	ran core	20.6	53.2	47.7	20.1
	unv html	13.9	47.2	37.7	19.9
	unv core	16.6	47.2	55.0	20.4
Project Member	ran html	30.0	41.0	66.4	20.0
	ran core	26.2	35.2	46.0	20.0
	unv html	29.1	42.8	64.1	20.1
	unv core	26.7	35.9	52.6	20.1

Table 4: SRV performance on the "many-per-document" fields.

those with confidence greater than x, for instance, we can realize accuracy much higher than baseline. The standard way of measuring the feasibility of such a trade-off is with an accuracy-coverage (or precision-recall) graph. To construct such a graph, we sort all learner predictions by confidence, then, for various n from 1 to 100, plot the accuracy of the n% most confident predictions. In lieu of such graphs, we have included two additional columns in Table 3, one for accuracy at approximately 80% coverage and one for approximately 20% coverage. [2]

Table 4 shows SRV performance on the two MPD fields: course instructor and project member. The unit of performance in this table is the individual prediction and field instance. Because the accuracy and coverage statistics bear a strong relationship to the standard IR notions of *precision* and *recall*,[3] and in order to emphasize the different way of measuring performance, we use different labels for the columns. *Prec* is the number of correctly recognized field instances divided by the total number of predictions. *Rec* is the number of correctly recognized field instances divided by the total number of field instances. Because this is a fundamentally harder task than in Table 3, in no case does SRV achieve 80% coverage; thus, only a 20% column is presented. By the same token, the accuracy-coverage trade-off is perhaps more crucial here, because it shows the effect of discarding all the low-confidence predictions that are naturally filtered out in the OPD setting.

Finally, Table 5 shows the performance of two strawman approaches to the task. The `Rote` learner simply memorizes field instances it sees in the training set, making a prediction on any test sequences that match an entry in its learned dictionary and returning a confidence that is the probability, based on training statistics, that a sequence is indeed a field instance. We take this learner to be the simplest possible machine learning approach to the problem. We have found in other

[2] Actual coverages are listed, because it is often impossible to choose a confidence cut-off that yields exactly the desired coverage.

[3] Field instances correspond to "relevant" documents, incorrect predictions to irrelevant ones.

Field	*Part.*	Rote *Acc*	Rote *Cov*	Guess *Acc*
Course Title	*rand*	44.3	49.5	1.3
	univ	36.2	47.7	1.4
Course Number	*rand*	44.4	28.8	3.1
	univ	0.0	0.0	2.9
Project Title	*rand*	18.8	21.6	8.3
	univ	28.6	7.5	8.5
		Prec	*Rec*	*Prec + Rec*
Course Instr.	*rand*	69.4	11.5	0.9
	univ	0.0	0.0	0.9
Project Member	*rand*	70.3	10.7	7.1
	univ	44.1	2.0	7.6

Table 5: Performance of two simple baseline strategies.

```
coursenumber :-
  length(= 2),
  every(in_title false),
  some(?A [previous_token] in_title true),
  some(?A [] after_p false),
  some(?B [] tripleton true)


<title> Course Information CS213 </title>
<h1> CS 213 C++ Programming </h1>
```

Figure 3: A learned rule for course number with some sample matching text. This rule matched 11 examples in the validation set with no false positives.

contexts that it performs surprisingly well in a small number of "naturally occurring" IE problems.

The Guess column shows the expected performance of a random guesser, given unrealistically optimistic assumptions. The learner is "told" how many field instances occur in a test file and what their lengths are. For each instance it is allowed to make one guess of the appropriate length. Because it always makes exactly as many predictions as there are test field instances, its precision and recall are equal on MPD fields.

Discussion

Not surprisingly, SRV performs better than the baseline approaches in all cases. This is especially apparent for the OPD fields. Note that, although Rote may appear to have comparable accuracies in some cases, these accuracy figures show only performance over "covered" files. In cases where SRV and Rote accuracies appear comparable, Rote coverage is generally much lower.

As expected, the addition of HTML features generally yields considerable improvement, especially at the high precision end of the curve. HTML information appears to be particularly important for recognizing project titles. The single exception to this trend is the course instructor problem in the LOUO setting. It appears that formatting conventions for a course page are relatively specific to the department from which it comes, as page templates are passed around among instructors and re-used from semester to semester. SRV takes advantage of the resulting regularities, and its performance suffers when some of the rules it learns do not generalize across university boundaries.

SRV performs best by far on the course number field, even in the LOUO case, where (as the results for Rote indicate) memorization yields no benefit. Figure 3 shows one HTML-aware rule responsible for this good performance. The *core* results show that much of its performance on this field is attributable to the strong orthographic regularities these numbers exhibit.

As noted above, Rote is sometimes a viable approach. This appears to be true for the course title field, for which Rote achieves reasonable performance at surprisingly high coverage levels. This is obviously an effect of the generic character of course titles. The title "Introduction to Artificial Intelligence" is quite likely to be a course title, wherever it is encountered, and is probably used at many universities without variation. Rote's high coverage for this field allows us to measure its accuracy at the approximate 20% level: 80.6% accuracy at 16.4% coverage, in the random-split experiments, and 57.9% at 20.5% coverage, in the LOUO experiments. Armed with HTML-specific features, SRV achieves much better accuracy at this coverage level for both partitioning methods.

A comparison between Table 3 and Table 4 makes the difficulty of the MPD extraction problem evident. Of course, the lower performance in Table 4 is also due to the fact that names of people are being extracted—probably a more difficult task for an automated system than fields made up of common English terms—and that formatting conventions for course instructors and project members vary more than for, say, course titles. Note, however, that if we can be satisfied with only finding 20% of the names of instructors and project members (in the case of random partitioning), we can expect about two-thirds of our predictions to be correct.

A comparison between the two partitioning methods shows, not surprisingly, that random partitioning makes for an easier learning task than LOUO partitioning. This is especially apparent for Rote on the person-name fields; faculty members tend to teach multiple courses, and researchers at a university become involved in multiple projects. But Web formatting conventions also tend to be shared within a department, so we might hope that SRV could benefit from intra-department regularities. Surprisingly, this is not uniformly evident. In the case of project title, SRV actually does worse in all three columns. This effect is probably in part due to differences in training set size between the two partitioning regimes (half of the data in random vs. three-fourths of the data, on average, in LOUO).

Related Work

Soderland originally showed the viability of a covering (rule learning) approach to the slot filling problem (Soderland 1996). More recently, Califf and Mooney have demonstrated a similar system with relational extensions (Califf & Mooney 1997). In both of these systems, rules are patterns which stipulate what must occur in and around field instances, and both systems generalize by starting with maximally specific patterns and gradually dropping constraints. Generalization is halted when a rule begins to accept too many negative examples.

This "bottom-up" search represents an efficient and useful approach, but it must rely on heuristics to control how constraints are dropped. Typically, there are many ways in which a rule can be generalized so that it does not cover negative examples. In contrast, top-down search is entirely controlled by the distribution of positive and negative examples in the data. SRV does this in part with the aid of a set of features, which are separate from the core algorithm. In the two bottom-up systems discussed here, the features are implicit and entangled with the search heuristics.

Soderland describes modifications which must be made to CRYSTAL, his learning system for IE, in order to use it for HTML (Soderland 1997). CRYSTAL's assumption that text will be presented in sentence-sized chunks must be satisfied heuristically. How this segmentation is performed depends in part on the domain and requires manual engineering. In contrast, because SRV searches at the token level, it requires no modifications to be retargeted. *Exploiting* HTML structure only involves the addition of several new HTML-specific features to its basic feature set. These feature are only additional information, and SRV does not require them in order to work with HTML.

It is common to report the performance of an IE system in terms of two summary numbers, precision and recall, and the systems described above adhere to this convention. In SRV, we have added a mechanism whereby these numbers can be varied to achieve the balance most advantages for the particular application.

Conclusion

Proceeding from general considerations about the nature of the IE problem, we have implemented SRV, a relational learner for this task. Adapting SRV for HTML requires no heuristic modifications to the basic algorithm; instead, HTML structure is captured by the addition of simple, token-oriented features. There is clear evidence that, armed with such features, SRV achieves interesting and effective generalization on a variety of tasks in two HTML domains.

Among the contributions made by this work are:

- **Increased modularity and flexibility.** Domain-specific information is separate from the underlying learning mechanism. This permits, among other things, the rapid adaptation of the system for use with HTML.
- **Top-down induction.** SRV demonstrates the feasibility of conducting learning for IE in the direction from general to specific.
- **Accuracy-coverage trade-off.** In contrast with other work in this area, the learning framework includes a mechanism for associating confidence scores with predictions. This allows the system to trade coverage for increased accuracy.

With the introduction of SRV, and other IE systems like it, we can begin to address the larger problem of designing artificially intelligent agents for mining the World Wide Web.

Acknowledgments Thanks to the other members of the WebKB project at CMU, whose data collection made this work possible. This research was supported in part by the DARPA HPKB program under contract F30602-97-1-0215.

References

Califf, M. E., and Mooney, R. J. 1997. Relational learning of pattern-match rules for information extraction. In *Working Papers of ACL-97 Workshop on Natural Language Learning.*

Cardie, C. 1997. Empirical methods in information extraction. *AI Magazine* 18(4):65–79.

Cestnik, B. 1990. Estimating probabilities: A crucial task in machine learning. In *Proceedings of the Ninth European Conference on Artificial Intelligence.*

Clark, P., and Niblett, T. 1989. The CN2 induction algorithm. *Machine Learning* 3(4):261–263.

Defense Advanced Research Projects Agency. 1995. *Proceedings of the Sixth Message Understanding Conference (MUC-6)*, Morgan Kaufmann Publisher, Inc.

Doorenbos, R.; Etzioni, O.; and Weld, D. S. 1997. A scalable comparison-shopping agent for the world-wide web. In *Proceedings of the First International Conference on Autonomous Agents.*

Freitag, D. 1998. Toward general-purpose learning for information extraction. In *Proceedings of COLING-ACL '98.* In submission.

Kushmerick, N. 1997. *Wrapper Induction for Information Extraction.* Ph.D. Dissertation, University of Washington. Tech Report UW-CSE-97-11-04.

Quinlan, J. R. 1990. Learning logical definitions from relations. *Machine Learning* 5(3):239–266.

Soderland, S. 1996. *Learning Text Analysis Rules for Domain-specific Natural Language Processing.* Ph.D. Dissertation, University of Massachusetts. CS Tech. Report 96-087.

Soderland, S. 1997. Learning to extract text-based information from the world wide web. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining.*