Boosted Wrapper Induction

Dayne Freitag
Just Research
Pittsburgh, PA, USA
dayne@cs.cmu.edu

Nicholas Kushmerick
Department of Computer Science
University College Dublin, Ireland
nick@ucd.ie

Abstract

Recent work in machine learning for *information extraction* has focused on two distinct sub-problems: the conventional problem of filling template slots from natural language text, and the problem of *wrapper induction*, learning simple extraction procedures ("wrappers") for highly structured text such as Web pages produced by CGI scripts. For suitably regular domains, existing wrapper induction algorithms can efficiently learn wrappers that are simple and highly accurate, but the regularity bias of these algorithms makes them unsuitable for most conventional information extraction tasks. *Boosting* is a technique for improving the performance of a simple machine learning algorithm by repeatedly applying it to the training set with different example weightings. We describe an algorithm that learns simple, low-coverage wrapper-like extraction patterns, which we then apply to conventional information extraction problems using boosting. The result is BWI, a trainable information extraction system with a strong precision bias and F1 performance better than state-of-the-art techniques in many domains.

Introduction

Information extraction (IE) is the problem of converting text such as newswire articles or Web pages into structured data objects suitable for automatic processing. An example domain, first investigated in the Message Understanding Conference (MUC) (Def 1995), is a collection of newspaper articles describing terrorist incidents in Latin America. Given an article, the goal might be to extract the name of the perpetrator and victim, and the instrument and location of the attack. Research with this and similar domains demonstrated the applicability of machine learning to IE (Soderland 1996; Kim and Moldovan 1995; Huffman 1996).

The increasing importance of the Internet has brought attention to all kinds of automatic document processing, including IE. And it has given rise to problem domains in which the kind of linguistically intensive approaches explored in MUC are difficult or unnecessary. Many documents from this realm, including email, Usenet posts, and Web pages, rely on extra-linguistic structures, such as HTML tags, document formatting, and ungrammatical stereotypic language, to convey essential information. Much recent work in IE, therefore, has focused on learning approaches that do not require linguistic information, but that can exploit other kinds of regularities. To this end, several distinct rule-learning algorithms (Soderland 1999; Califf 1998; Freitag 1998) and multi-strategy approaches (Freitag 2000) have been shown to be effective. Recently, statistical approaches using hidden Markov models have achieved high performance levels (Leek 1997; Bikel *et al.* 1997; Freitag and McCallum 1999).

At the same time, work on information integration (Wiederhold 1996; Levy *et al.* 1998) has led to a need for specialized *wrapper* procedures for extracting structured information from database-like Web pages. Recent research (Kushmerick *et al.* 1997; Kushmerick 2000; Hsu and Dung 1998; Muslea *et al.* 2000) has shown that wrappers can be automatically learned for many kinds of highly regular documents, such as Web pages generated by CGI scripts. These *wrapper induction* techniques learn simple but highly accurate contextual patterns, such as "to retrieve a URL, extract the text between `<A href="` and `">`". Wrapper induction is harder for pages with complicated content or less rigidly-structured formatting, but recent algorithms (Hsu and Dung 1998; Muslea *et al.* 2000) can discover small sets of such patterns that are highly effective at handling such complications in many domains.

In this paper, we demonstrate that wrapper induction techniques can be used to perform extraction in traditional (natural text) domains. We describe BWI, a trainable IE system that performs information extraction in both traditional (natural text) and wrapper (machine-generated or rigidly-structured text) domains. BWI learns extraction rules composed only of simple contextual patterns. Extraction is triggered by specific token sequences preceding and following the start or end of a target field. For example, BWI might learn the pattern ⟨`[< a href = "]`,`[http]`⟩ for finding the beginning of a URL, and the pattern ⟨`[. html]`,`[" >]`⟩ for finding the end, which would extract "`http://xyz.com/index.html`" from "`···<a href="http://xyz.com/index.html">···`".

Of course, the documents used for traditional IE tasks do not exhibit the kind of regular structure assumed in wrapper induction. Consider the task of extracting the speaker's name from a seminar announcement. A significant fraction of documents in the seminar announcement corpus we use for experiments have the speaker's name prefixed by "`Who:`". Similarly, many speaker names begin with honorifics such as "`Dr.`". These observations suggest that simple contextual patterns such as ⟨`[who :]`, `[dr .]`⟩ could be used to identify the start of the speaker's name with high precision.

However, while such a pattern may have high precision, it will generally have low recall: this pattern strongly indicates the beginning of a speaker's name, but it occurs in only a fraction of documents. To apply wrapper induction techniques to IE from natural text, we must generate many such simple patterns, and then combine their predictions in some reasonable way. Previous work on learning rules for information extraction assume that the final extractor will be some combination, typically a disjunction, of individual rules, each one covering only a fraction of the training phrases. However, whereas previous techniques learn individual rules that cover as many of the training examples as possible, BWI learns rules that individually have limited power and coverage, but that can be learned efficiently and are easy to understand.

Boosting is a procedure for improving the performance of a "weak" machine learning algorithm by repeatedly applying it to the training set, at each step modifying training example weights to emphasize examples on which the weak learner has done poorly in previous steps (Schapire and Singer 1998). The ability of boosting to improve upon the performance of the underlying weak learner has been verified in a wide range of empirical studies in recent years.

In this paper, we demonstrate that boosting can be used effectively for information extraction. Our BWI algorithm uses boosting to generate and combine the predictions from numerous extraction patterns. The result is a trainable information extraction system that, in our experiments, performs better than previous rule learning approaches, and is competitive with a state-of-the-art statistical technique.

In the remainder of this paper, we (1) formally describe the extraction patterns BWI learns; (2) describe the BWI algorithm; and (3) empirically evaluate BWI on eight document collections.

Problem Statement

We begin by explaining how we treat IE as a classification problem, and then describe the classifiers (*wrappers*) that BWI learns.

Information Extraction as Classification. We treat *documents* as sequences of *tokens*, and the IE task is to identify one or more distinguished token subsequences called *fields*. Specifically, IE involves identifying the *boundaries* that indicate the beginning and end of each field; below, the symbols i and j refer to boundaries. We cast this as a classification problem, where instances correspond to boundaries—the space between any two adjacent tokens—and the goal is to approximate two target extraction functions, X_{begin} and X_{end}:

$$X_{\text{begin}}(i) = \begin{cases} 1 & \text{if } i \text{ begins a field} \\ 0 & \text{otherwise} \end{cases}$$

Similarly, X_{end} is 1 for field-ending boundaries and 0 otherwise.

To learn such a function X (either X_{begin} or X_{end}), a learning algorithm is given a training set $\{\langle i, X(i)\rangle\}$ and must output a function that approximates X.

Wrappers. A *pattern* is a sequence of tokens (e.g., `[who :]` or `[dr .]`). A pattern *matches* a token sequence in a document if the tokens are identical. (Below we enrich this notion of matching when we describe BWI's use of wildcards.)

A *boundary detector* $d = \langle p, s\rangle$ is a pair of patterns: a *prefix* pattern p and a *suffix* pattern s. A boundary detector (hereafter, simply "*detector*") $d = \langle p, s\rangle$ *matches* a boundary i if p matches the tokens before i and s matches the tokens after i. We treat a detector d as a function from a boundary to $\{0, 1\}$: $d(i) = 1$ if d matches i, and 0 otherwise. Finally, associated with every detector d is a numeric *confidence* value C_d.

A *wrapper* $W = \langle F, A, H\rangle$ consists of two sets $F = \{F_1, F_2, \ldots, F_T\}$ and $A = \{A_1, A_2, \ldots, A_T\}$ of detectors, and a function $H : [-\infty, +\infty] \to [0, 1]$. The intent is that F (the "fore" detectors) identifies field-starting boundaries, A (the "aft" detectors) identifies the field end boundaries, and $H(k)$ reflects the probability that a field has length k.

To perform extraction using wrapper W, every boundary i in a document is first given a "fore" score $F(i) = \sum_k C_{F_k} F_k(i)$ and an "aft" score $A(i) = \sum_k C_{A_k} A_k(i)$. W then classifies text fragment $\langle i, j\rangle$ as follows:

$$W(i,j) = \begin{cases} 1 & \text{if } F(i)A(j)H(j-i) > \tau \\ 0 & \text{otherwise} \end{cases},$$

where τ is a numeric threshold.

The rationale is that W compares τ to an estimate of the probability of correct classification. The value of $H(\cdot)$ is proportional to a maximum-likelihood estimate that a fragment is a particular length, given that it is a target fragment. If we assume that $F(\cdot)$ and $A(\cdot)$ are also proportional to the conditional probability of finding a beginning or ending boundary, respectively, then the product of the three values is proportional to a naive Bayesian estimate with uniform priors.

By varying τ one can force a tradeoff between precision and recall. In our experiments we use the full-recall setting $\tau = 0$, because (as our experiments demonstrate) BWI is generally biased toward precision.

The BWI algorithm

Learning a wrapper W involves determining the "fore" and "aft" detectors F and A, and the function H, given the example sets S and E. In this section, we describe BWI, our algorithm that solves problems of this form.

Fig. 1 lists BWI. The function H reflects the prior probability of various field lengths. BWI estimates these probabilities by constructing a frequency histogram $H(k)$ recording the number of fields of length k were encountered in the training set. To learn F and A, BWI boosts LearnDetector, an algorithm for learning a single detector.

procedure BWI(example sets S and E)
    $F \leftarrow$ AdaBoost(LearnDetector, S)
    $A \leftarrow$ AdaBoost(LearnDetector, E)
    $H \leftarrow$ field length histogram from S and E
    return wrapper $W = \langle F, A, H \rangle$

Figure 1: The BWI algorithm.

procedure LearnDetector(example set Y)
    prefix pattern $p \leftarrow$ [ ]
    suffix pattern $s \leftarrow$ [ ]
    loop
        prefix pattern $p' \leftarrow$ BestPreExt($\langle p, s \rangle, Y$)
        suffix pattern $s' \leftarrow$ BestSufExt($\langle p, s \rangle, Y$)
        if score($\langle p', s \rangle$) $>$ score($\langle p, s' \rangle$)
            if score($\langle p', s \rangle$) $>$ score($\langle p, s \rangle$)
                $p \leftarrow$ the last $|p| + 1$ tokens of p'
            else return detector $\langle p, s \rangle$
        else
            if score($\langle p, s' \rangle$) $>$ score($\langle p, s \rangle$)
                $s \leftarrow$ the first $|s| + 1$ tokens of s'
            else return detector $\langle p, s \rangle$

Figure 2: The LearnDetector weak learner.

Boosting. Freund and Shapire's generalized AdaBoost algorithm maintains a distribution $D_t(i)$ over the training examples. Initially, D_0 is uniform. On iteration t, the weak learner is invoked, resulting in hypothesis d_t (what we have called a detector), a weight C_{d_t} is assigned to d_t, and then the distribution is updated as follows:

$$D_{t+1}(i) = D_t(i) \exp(-C_{d_t} d_t(i)(2X(i) - 1))/N_t$$

where $X(i) \in \{1, 0\}$ is the label of i, and N_t is a normalization factor. AdaBoost simply repeats this learn-update cycle T times, and then returns a list of the learned weak hypotheses with their weights.

The LearnDetector Weak Learner. Fig. 2 shows the weak learning algorithm LearnDetector. LearnDetector generates a single detector; BWI invokes LearnDetector (indirectly through AdaBoost) T times to learn the "fore" detectors F, and then T more times to learn the "aft" detectors A. LearnDetector iteratively builds out from the empty detector $\langle$ [ ], [ ] $\rangle$. At each step, LearnDetector invokes the functions BestPreExt and BestSufExt, which search for the best extension of length L (the *lookahead* parameter) or less to the prefix and suffix (respectively) of the current detector. These extensions are exhaustively enumerated.

The current detector and the best extensions are then compared to maximize a scoring function. If an extension is found which results in a detector that scores better than the current one, then the first token of the extension (the rightmost of a prefix extension, the leftmost of a suffix extension) is added to the appropriate part of the current detector, and the process repeats. The procedure returns when no extension yields a better score than the current detector.

The function score(d) computes the score of detector $d = \langle p, s \rangle$. Cohen and Singer 1999 describe SLIPPER, a boosting algorithm which infers a single rule at each step. BWI's scoring method is identical to that used by SLIPPER. For a detector d, let M_d^+ be the correctly classified training boundaries (i.e., the boundaries in the training set Y matched by d and labeled 1), and M_d^- be the incorrectly classified examples. Cohen and Singer showed that training error is minimized by using:

$$\text{score}(d) = \sqrt{W_d^+} - \sqrt{W_d^-},$$

and assigning detector d the confidence value:

$$C_d = \frac{1}{2} \ln \left(\frac{W_d^+ + \epsilon}{W_d^- + \epsilon} \right),$$

where $W_d^+ = \sum_{i \in M_d^+} D_t(i)$, is the total weight of the correctly classified boundaries, $D_t(i)$ is the weight of boundary i during boosting iteration t, W_d^- is a similar sum over the set M_d^-, and ϵ is a small smoothing parameter.

Wildcards. As described so far, BWI learns only wrappers that match exact token sequences. We have extended BWI to handle *wildcards*. A wildcard is a special token that matches one of a set of tokens. BWI uses the following wildcards:

- `<Alph>` matches any token that contains only alphabetic characters
- `<ANum>`, contains only alphanumeric characters
- `<Cap>`, begins with an upper-case letter
- `<LC>`, begins with a lower-case letter
- `<SChar>`, any one-character token
- `<Num>`, containing only digits
- `<Punc>`, a punctuation token
- `<*>`, any token

Extending BWI to handle wildcards is straightforward: we simply modify BestPreExt and BestSufExt to enumerate all sequences of tokens and wildcards, rather than just tokens.

Experimental Results

We evaluated BWI on 16 information extraction tasks defined over eight distinct document collections:

- *SA*: A collection of 486 seminar announcements. Fields: *speaker* (speaker's name); *location* (seminar location); *stime* (starting time); and *etime* (ending time).
- *Acq*: A collection of 600 Reuters articles detailing coporate acquisitions. Fields: *acq* (name of purchased company); and *dlramt* (purchase price).
- *Jobs*: A collection of 298 Usenet job announcements. Fields: *id* (message identifier); *company* (company name); *title* (job title).
- *CS*: A collection of Web pages listing the faculty of 30 computer science departments. Field: *name* (faculty member names).

domain	top-scoring boundary detectors	example
SA-stime	$F_1 = \langle$[time :], [<Num>]$\rangle$	··· Time: $\boxed{2:00}$ - 3:30 PM ⇓ ···
	$A_1 = \langle$[], [- <Num> : <*> <Alph> ⇓]$\rangle$	
CS-name	$F_1 = \langle$[<LC> <*> <*> <Punc> <*> <*> <ANum> " <Punc>], [<FName>]$\rangle$	··· cgi-bin/facinfo?awb"> $\boxed{\text{Alan⇓Biermann}}$ </A><···
	$A_1 = \langle$[⇓ <LName>], [< <Punc> a <Punc> <Punc>]$\rangle$	

Figure 3: Examples of boundary detectors learned by BWI for the *SA* and *CS* domains. "⇓" is the return character. The wildcards <FName> and <LName> stand for first name and last name; see details below.

- *Zagats*: A collection of 91 Web pages containing restaurant reviews. Field: *addr* (addresses of restaurants).
- *LATimes*: A collection of 20 Web pages containing restaurant descriptions. Field: *cc* (credit cards accepted by restaurants).
- *IAF*: A collection of 10 pages from an Web email search engine. Fields: *altname* (person's alternate name); and *org* (host organization).
- *QS*: A collection of 10 pages from an Web stock quote service. Fields: *date* (quote date); and *vol* (trading volume).

We chose these collections because they have been widely used in previous research. The first three domains are typical for the traditional IE techniques, while the last five are typical for the wrapper induction techniques. Fig. 3 gives examples of wrappers learned by BWI for two of these tasks.

In our experiments, we adopt the standard *cross validation* methodology: the document collection is partitioned several times into a training set and a testing set. We learn a wrapper using the training set, and then measure its performance using the testing set.

Given a test document, BWI extracts zero or more fields. In order for an extraction to be counted as correct, the exact boundaries of a target fragment must be identified. For the "traditional" domains (*SA*, *Acq* and *Jobs*), we exploit the assumption that each document contains at most one field, and discard all but the highest-confidence prediction. In the "wrapper" domains (*CS*, *Zagats*, *LATimes*, *IAF* and *QS*), documents may contain multiple fields, so we keep all predictions with confidence greater than $\tau = 0$.

We evaluate performance in terms of three metrics: *precision*, the number of correctly extracted fields divided by the total number of extractions; *recall*, the number of correct extractions divided by the total number of fields actually present in the documents, and *F1*, the harmonic mean of precision and recall. We report all values as percentages.

Our experiments were designed to answer four questions:

1. What effect does the number of rounds of boosting T have on performance?
2. What effect does the look-ahead parameter L have on performance?
3. How important are wildcards?
4. How does BWI compare with other learning algorithms on the same tasks?

To address Questions 1–3, we devised a set of experiments using the four *SA* tasks, and we discuss the other domains when the results are significantly different. To answer Question 4, we compare BWI with four alternative learning algorithms.

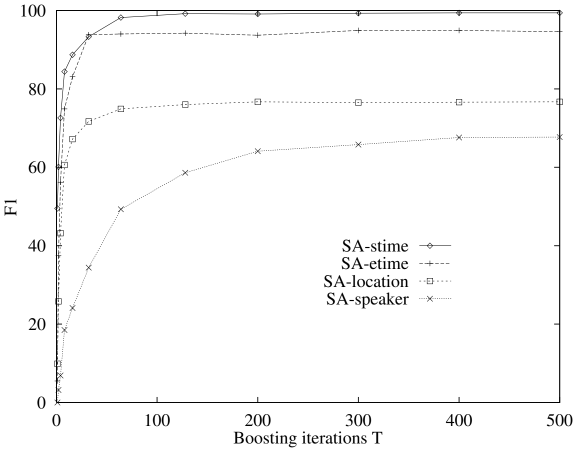

Figure 4: F1 performance on the *SA* tasks as a function of the number of rounds of boosting T.

Question 1: Boosting. To measure sensitivity to the number of rounds of boosting, we fixed look-ahead at $L = 3$, gave BWI the default wildcard set, and varied the number of rounds of boosting from $T = 1$ to 500. As Fig. 4 suggests, the number of rounds required by BWI to reach peak performance depends on the difficulty of the task. On easy tasks, such as *SA-stime*, BWI quickly achieves its peak performance. On more difficult tasks, such as *SA-speaker*, as many as 500 rounds may be required.

On the "wrapper" tasks, many fewer rounds of boosting are required. For example, BWI stops improving after $T = 5$ on *IAF-altname*, and a single round of boosting yields perfect performance on *QS-Date*. These sources are formatted in a highly regular fashion, so just just a handful of boundary detectors are needed.

Question 2: Look-Ahead. Performance improves with increasing look-ahead L, as indicated in Fig. 5. For the *CS* tasks, performance improvements are marginal beyond a look-ahead of 3. This is fortunate, because training time increases exponentially with look-ahead. With look-ahead set to $L = 3$, BWI took 3,183 sec. to complete 100 rounds of boosting using the default feature set for the speaker task on a 300 MHz. Pentium II; with $L = 4$, it took 13,958 sec.

L	*speaker*	*location*	*stime*	*etime*
1	7.0	40.1	27.7	7.4
2	51.9	76.6	95.0	84.9
3	67.7	76.7	99.4	94.6
4	69.3	75.5	99.6	93.9

Figure 5: F1 performance on the *SA* tasks as a function of look-ahead L.

wildcards	*speaker*	*location*	*stime*	*etime*
none	15.1	69.2	95.7	83.4
just <*>	49.4	73.5	99.3	95.0
default	67.7	76.7	99.4	94.6
lexical	73.5	—	—	—

Figure 6: F1 performance on the *SA* tasks as a function of various wildcard sets.

However, much deeper lookahead is required in some domains. For example, in the *IAF-altname* task, $L = 3$ results in $F1 = 0$, but $L = 8$ yields $F1 = 58.8$. The explanation is that *IAF-altname* requires a 30-token "fore" boundary detector prefix, but shorter prefixes have a very low score, so BWI can find the correct prefix only with very deep lookahead.

Question 3: Wildcards. Wildcards are important to achieve good performance on traditional IE problems. Fig. 6 presents F1 performance on the *SA* task with various wildcard sets. In each case, we performed 500 rounds of boosting with look-ahead set to 3. The "*none*" row lists performance in the absence of wildcards; "*just* <*>" lists performance with only the <*> wildcard; "*default*" lists performance using the full set of eight wildcards listed above.

Finally, we evaluated BWI after adding task-specific lexical resources in the form of three additional wildcards: <FName> matches tokens in a list of common first names released by the U.S. Census Bureau; <LName> matches tokens in a similar list of common last names; and <NEW> matches tokens not found in /usr/dict/words on Unix systems ("NEW" stands for "not an English word"). These lexical resources increase F1 from 67.7 to 73.5 on the *SA-speaker* task. It is common in traditional IE to use task-specific lexicons as part of the extraction process. Our results show how such lexicons can be integrated with learning in a way that leads to improved performance.

Question 4: Other Algorithms. Figs. 7–8 compares BWI on the sixteen extraction tasks with four other state-of-the-art learners: two rule learners (SRV (Freitag 1998) and Rapier (Califf 1998)), an algorithm based on hidden Markov models[1], and the Stalker wrapper induction algorithm (Muslea *et al.* 2000).

For the four "traditional" domains, we used $T = 500$ boosting iterations and set the lookahead to $L = 3$, and used the default wildcards. For the five "wrapper" domains we use $T = 50$ and two different settings for L and the wildcards. For *CS-name*, *Zagats-addr*, *LATimes-cc* and *QS-date*, we used $L = 3$ and the lexical wildcards. For *IAF-altname*, *IAF-org* and *QS-vol*, we used $L = 8$ because (as described above) the tasks require very long boundary detectors. Since BWI is exponential in L, the only feasible way to run the BWI with $L = 8$ was to use just the <*> wildcard.

We include precision and recall scores in order to illustrate an interesting aspect of BWI's behavior—its precision bias. The extractors produced by BWI tend to achieve higher precision than the other learners, particularly the HMM, while still managing good recall.

Conclusions

The automatic processing of machine-readable text is becoming increasingly important. Techniques for information extraction—the task of populating a pre-defined database with fragments from free text documents— are central to a broad range of text-management applications.

We have described BWI, a novel approach to building a trainable information extraction system. Like wrapper induction techniques, BWI learns relatively simple contextual patterns identifying the beginning and end of relevant text fields. BWI repeatedly invokes an algorithm for learning such boundaries. By using the AdaBoost algorithm, BWI repeatedly reweights the training examples so that subsequent patterns handle training examples missed by previous rules.

The result is an extraction algorithm with a bias to high precision (because the learned contextual patterns are highly accurate) but with reasonable recall in many domains (due to the fact that dozens or hundreds—but not millions—of such patterns suffice for broad coverage). We have evaluated BWI on a broad range of IE tasks, from traditional free text to machine-generated HTML, and find that BWI is competitive with state-of-the-art algorithms in most domains, and superior in many.

Acknowledgements. This research was funded in part by grant N00014-00-1-0021 from the US Office of Naval Research, and grant ST/1999/071 from Enterprise Ireland.

References

D. Bikel, S. Miller, R. Schwartz, and R. Weischedel. Nymble: a high-performance learning name-finder. In *Proc. ANLP-97*, pages 194–201, 1997.

M.-E. Califf. *Relational Learning Techniques for Natural Language Information Extraction*. PhD thesis, University of Texas at Austin, 1998.

W. Cohen and Y. Singer. A simple, fast, and effective rule learner. In *Proc. Sixteenth National Conference on Artificial Intelligence*, 1999.

[1] The HMM in question has four fully connected "target" states, a prefix and suffix, each of length four, and uses shrinkage to mitigate data sparsity. This model has shown state-of-the-art performance on a range of IE tasks; see Freitag and McCallum (1999) for details

	SA-speaker			*SA-location*			*SA-stime*			*SA-etime*					
	Prec	*Rec*	*F1*	*Prec*	*Rec*	*F1*	*Prec*	*Rec*	*F1*	*Prec*	*Rec*	*F1*			
HMM	77.9	75.2	76.6	83.0	74.6	78.6	98.5	98.5	98.5	45.7	97.0	62.1			
Rapier	80.9	39.4	53.0	91.0	60.5	72.7	93.9	92.9	93.4	95.8	96.6	96.2			
SRV	54.4	58.4	56.3	74.5	70.1	72.3	98.6	98.4	98.5	67.3	92.6	77.9			
BWI	79.1	59.2	67.7	85.4	69.6	76.7	99.6	99.6	99.6	94.4	94.9	93.9			
	Jobs-id			*Jobs-company*			*Jobs-title*			*Acq-acq*			*Acq-dlramt*		
HMM	—	—	—	38.6	72.3	50.4	53.2	63.0	57.7	32.8	29.2	30.9	49.3	63.5	55.5
Rapier	98.0	97.0	97.5	76.0	64.8	70.0	67.0	29.0	40.5	57.3	19.2	28.8	63.3	28.5	39.3
SRV	—	—	—	—	—	—	—	—	—	40.7	39.4	40.1	48.1	67.0	56.0
BWI	100	100	100	88.4	70.1	78.2	59.6	43.2	50.1	55.5	24.6	34.1	63.4	42.6	50.9
	LATimes-cc			*Zagats-addr*			*IAF-altname*			*IAF-org*					
HMM	98.5	100	99.3	97.7	99.5	98.6	1.7	90.0	3.4	16.8	89.7	28.4			
Stalker	100	—	—	100	—	—	100	—	—	48.0	—	—			
BWI	99.6	100	99.8	100	93.7	96.7	90.9	43.5	58.8	77.5	45.9	57.7			
	CS-name			*QS-date*			*QS-vol*								
HMM	41.3	65.0	50.5	36.3	100	53.3	18.4	96.2	30.9						
Stalker	—	—	—	0	—	—	0	—	—						
BWI	77.1	31.4	44.6	100	100	100	100	61.9	76.5						

Figure 7: BWI compared with four competing algorithms on sixteen tasks.

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

D. Freitag and A. McCallum. Information extraction using HMMs and shrinkage. In *Proc. AAAI-99 Workshop on Machine Learning for Information Extraction*, 1999. AAAI Technical Report WS-99-11.

D. Freitag. Information extraction from HTML: Application of a general machine learning approach. In *Proc. Fifteenth National Conference on Artificial Intelligence*, 1998.

D. Freitag. Machine learning for information extraction in informal domains. *Machine Learning*, 39(2/3), 2000.

C. Hsu and M. Dung. Generating finite-state transducers for semistructured data extraction from the web. *J. Information Systems*, 23(8), 1998.

S. Huffman. Learning information extraction patterns from examples. In *Connectionist, Statistical, and Symbolic Approaches to Learning for Natural Language Processing*, volume 1040 of *Lecture Notes in Artificial Intelligence*, pages 246–260. Springer-Verlag, Berlin, 1996.

J.-T. Kim and D. Moldovan. Acquisition of linguistic patterns for knowledge-based information extraction. *IEEE Trans. on Knowledge and Data Engineering*, 7(5):713–724, 1995.

N. Kushmerick, D. Weld, and R. Doorenbos. Wrapper Induction for Information Extraction. In *Proc. 15th Int. Conf. Artificial Intelligence*, pages 729–35, 1997.

N. Kushmerick. Wrapper induction: Efficiency and expressiveness. *Artificial Intelligence*, 2000. In press.

T. Leek. Information extraction using hidden Markov models. Master's thesis, UC San Diego, 1997.

A. Levy, C. Knoblock, S. Minton, and W. Cohen. Trends and controversies: Information integration. *IEEE Intelligent Systems*, 13(5), 1998.

I. Muslea, S. Minton, and C. Knoblock. Hierachical wrapper induction for semistructured information sources. *J. Autonomous Agents and Multi-Agent Systems*, 2000. In press.

R. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. In *Proc. Eleventh Annual Conference on Computational Learning Theory*, 1998.

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

S. Soderland. Learning information extraction rules for semi-structured and free text. *Machine Learning*, 34(1/3):233–272, 1999.

G. Wiederhold. *Intelligent Information Integration*. Kluwer, 1996.

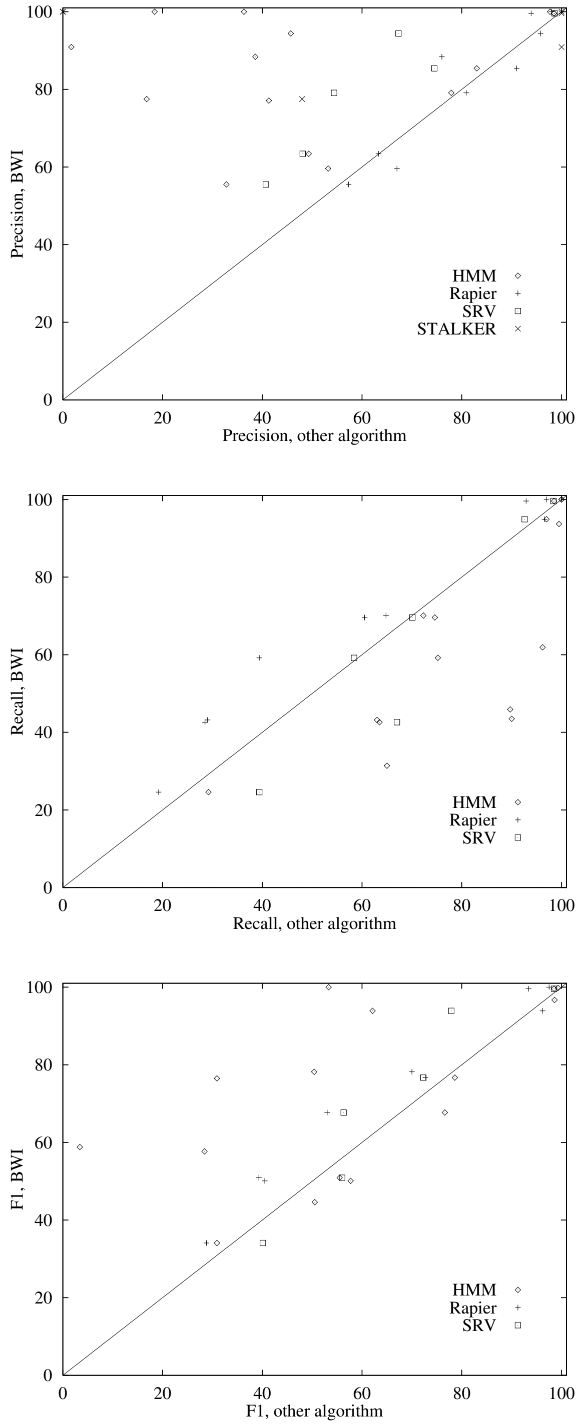

Figure 8: Graphical summaries of the data in Fig. 7. Each point represents a comparison between BWI and one other algorithm; points above the straight lines indicate domains in which BWI performs better.