Incremental Information Extraction Using Tree-based Context Representations

Christian Siefkes

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

Abstract. The purpose of *information extraction* (IE) is to find desired pieces of information in natural language texts and store them in a form that is suitable for automatic processing. Providing annotated training data to adapt a trainable IE system to a new domain requires a considerable amount of work. To address this, we explore *incremental learning*. Here training documents are annotated sequentially by a user and immediately incorporated into the extraction model. Thus the system can support the user by proposing extractions based on the current extraction model, reducing the workload of the user over time.
We introduce an approach to modeling IE as a token classification task that allows incremental training. To provide sufficient information to the token classifiers, we use rich, tree-based context representations of each token as feature vectors. These representations make use of the heuristically deduced document structure in addition to linguistic and semantic information. We consider the resulting feature vectors as ordered and combine proximate features into more expressive joint features, called "Orthogonal Sparse Bigrams" (OSB). Our results indicate that this setup makes it possible to employ IE in an incremental fashion without a serious performance penalty.

1 Introduction

The purpose of *information extraction* (IE) is to find desired pieces of information in natural language texts and store them in a form that is suitable for automatic querying and processing. IE requires a predefined output representation (*target structure*) and only searches for facts that fit this representation. Simple target structures define just a number of *slots*. Each slot is filled with a string extracted from a text, e.g. a name or a date (*slot filler*).

To adapt an IE system to a new domain, it is necessary to either manually rewrite the rules used in the system (in case of *static* rule-based systems) or to provide annotated training data (in case of *trainable* systems). Manual rewriting of rules is a time-consuming and intricate task that must be done by experts

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

which are usually hard to get. Providing annotated training data is less costly but still requires a considerable amount of work.

To address this, some approaches use *active learning* [5,14] where the system actively selects texts to be annotated by a user from a pool of unannotated training data. Thus adaptation to a new domain still requires a large amount of raw (unannotated) training data (which are usually cheap), but only a reduced amount of annotated (and thus expensive) training data which are chosen to be especially valuable for building the extraction model, e.g. those texts or fragments whose annotation is least certain.

An alternative setup is *incremental learning*. Here training documents are annotated sequentially by a user and immediately incorporated into the extraction model. Except for the very first document(s), the system can support the user by proposing slot fillers. Thus the work to be done by the user is reduced over time, from largely manual annotation of slot fillers to mere supervision and correction of the system's suggestions.

While incremental learning generally requires more annotated documents than active learning to reach the same level of accuracy (since the system cannot select the most informative samples), the work required by the user for annotating each document is reduced. Also the user keeps control about which documents are processed. Moreover an incremental setup fits better in situations where information is to be extracted from a stream of incoming documents ("text stream management"), for example email messages or newspaper articles.

In an incremental setup, the workflow for processing a document comprises the following steps:

1. Extract text fragments to fill the slots defined by the given target structure.
2. Show the predicted information to a user; ask the user to review the information and to correct any errors and omissions.
3. Adapt the extraction model based on the user's feedback.

In this paper we introduce an approach to IE that allows incremental training. In the following section we explain our approach to modeling IE as a classification task and the classification algorithm we use. We then describe the preprocessing steps of our system and the rich tree-based context representations we generate as input for the classifier. After reporting experimental results, we conclude by discussing related approaches and future work.

2 Classification-based Extraction

2.1 Fragment Extraction

The extraction of slot fillers can be handled as a token classification task, where each token (typically a word) in a text is classified as the begin of a slot filler

of a certain type (B-*type*), as a continuation of the previously started slot filler, if any (I-*type*), or as not belonging to any slot filler (O).[1]

Thus there are $2n+1$ classes for n slot types. Not all of them are allowed for all tokens—the B-*type* classes and the O class are always allowed, but there is at most one I-*type* class allowed, and only if the preceding token has been classified to be part of a slot filler of the same *type* (B-*type* or I-*type*).

2.2 The Winnow Classification Algorithm

Most refined classification algorithms are unsuitable for incremental training. One exception is the *Winnow* algorithm [11].

We use a variant of Winnow introduced in [16] which is suitable for both binary (two-class) and multi-class (three or more classes) classification. It keeps an n-dimensional weight vector $w^c = (w_1^c, w_2^c, \ldots, w_n^c)$ for each class c, where w_i^c is the weight of the ith feature. The algorithm returns 1 for a class iff the summed weights of all active features (called the score Ω^c) surpass a predefined threshold θ:

$$\Omega^c = \sum_{j=1}^{n_a} w_j^c > \theta.$$

Otherwise ($\Omega^c \leq \theta$) the algorithm returns 0. $n_a \leq n$ is the number of active (present) features in the instance to classify.

The goal of the algorithm is to learn a linear separator over the feature space that returns 1 for the true class of each instance and 0 for all other classes on this instance. The initial weight of each feature is 1.0. The weights of a class are updated whenever the value returned for this class is wrong. If 0 is returned instead of 1, the weights of all active features are increased by multiplying them with a *promotion factor* α, $\alpha > 1$: $w_j^c \leftarrow \alpha \times w_j^c$. If 1 is returned instead of 0, the active weights are multiplied with a *demotion factor* β, $0 < \beta < 1$: $w_j^c \leftarrow \beta \times w_j^c$.

The used threshold is not fixed, but set to the number n_a of features that are active in a given instance: $\theta = n_a$. Thus initial scores are equal to θ since the initial weight of each feature is 1.0.

We use a *thick threshold* for training Winnow (cf. [3,16]). Instances are trained even if the classification was correct if the determined score was near the threshold. Two additional thresholds θ^+ and θ^- with $\theta^- < \theta < \theta^+$ are defined and each instance whose score falls in the range $[\theta^-, \theta^+]$ is considered a mistake. In this way, a large margin classifier will be trained that is more robust when classifying borderline instances.

We use the parameter values recommended in [16], setting the promotion factor $\alpha = 1.23$, the demotion factor $\beta = 0.83$, and the threshold thickness to 5%.[2]

[1] This is the so-called **IOB2** combination strategy. There are other combination strategies for combining single-token classification decisions into slot fillers that can comprise several tokens (cf. Sec. 5); but in preliminary experiments we found this one to perform best.

[2] In either direction, i.e. $\theta^- = 0.95\,\theta$, $\theta^+ = 1.05\,\theta$.

The scores Ω^c generated by Winnow are converted into confidence estimates using the sigmoid function $\sigma^c(\theta, \Omega^c) = 1/(1 + \frac{\theta}{\Omega^c})$. The resulting σ^c values are normalized by dividing them by $\sum \sigma^c$ so they sum up to 1.

While our Winnow variant supports multi-class classification, initial experiments indicated that is advantageous to use multiple binary classifiers in a "one-against-the-rest" setup. We train a separate classifier for each B-*type* and I-*type* class, considering the context representations of all tokens of a class as positive instances for the corresponding classifier and all other token contexts as negative instances. If several classifiers predict their positive class, the most confident classifiers wins.

2.3 Orthogonal Sparse Bigrams (OSB)

Winnow is a linear separator in the Perceptron sense, but by providing a feature space that itself allows conjunction and disjunction, complex non-linear features may be recognized by the composite feature-extractor + Winnow system.

For this purpose, the OSB (orthogonal sparse bigrams) technique introduced in [16] has proved valuable. OSB slides a window of length N over the original feature list. For each window position, joint output features are generated by combining the right-most (newest) input feature with each of the other input features in the window, memorizing the order of the two input features and the distance between them.

Each of these joint features can be mapped to one of the numbers from 1 to $2^N - 1$ with two bits "1" in their binary representations ($2^n + 1$, for $n = 1$ to $N - 1$) where original features at "1" positions are visible while original features at "0" positions are hidden and marked as skipped. Thus $N - 1$ combinations with exactly two input features are produced for each window position. We use OSB with a window length of $N = 5$ as recommended in [16].

With a sequence of five input features, $i1, \ldots, i5$, OSB produces four output features:

		$i4$	$i5$
	$i3$	$<skip>$	$i5$
$i2$	$<skip>$	$<skip>$	$i5$
$i1$	$<skip>$	$<skip>$	$i5$

3 Preprocessing and Context Representation

3.1 Preprocessing

Regarded naively, an input text feed to an IE system might appear to be flat data without visible structure; just a sequence of characters. 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).

In structured text formats the higher-level structure (usually down to the paragraph level) is explicitly coded, but the lower-level structure (sentences; sentence constituents such as verb groups or noun phrases; tokens) must usually be induced.

The native format of our IE system is XML-based; any well-formed XML document is accepted as input. Documents in other formats must be converted to an XML dialect before they can be processed. Currently, converters from SGML-based (legacy) HTML to XHTML (*JTidy* [9]) and from plain text to XHTML (*txt2html* [20]) are integrated into the system—the latter uses heuristics to deduce the text structure from ASCII markup, recognizing section headers, lists and tables, emphasized text etc. Other document formats can be processed by integrating a suitable converter into the system or by converting them to XML or HTML prior to processing.

In a second step, the text is augmented with explicit linguistic information. We use the well-known *TreeTagger* [19] to:

- Divide a text into sentences;[3]
- Split sentences into "chunks" such as verb groups, noun phrases and prepositional phrases;[4]
- Tokenize the input into a sequence of *parts-of-speech* (words, numbers and punctuation) and determine their syntactic categories and normalized base forms.[5]

The output of the tagger is converted to the XML markup mentioned in the footnotes and merged with the explicit markup of the source document. The merging algorithm is described in [15]. After preprocessing, a text is represented as a DOM (Document Object Model) tree. The structure of the DOM tree for a simple HTML document (containing a section heading and several paragraphs) is shown in Fig. 1.

3.2 Tree-based Context Representation

Typically, the *context window* considered by IE algorithms comprises either the nearest tokens/words (e.g. [2]) or some predefined syntactic elements of the current sentence (e.g. [17]). The hierarchical tree structure obtained by preprocessing yields a more flexible context model: the context of a node contains the nearest nodes around it. The context we consider for each token includes features about:

- The token itself and the POS (part-of-speech) element it is in.
- Up to four preceding and four following siblings[6] of the POS element (neighboring parts-of-speech, but only those within the same sentence chunk).

[3] `sent` element

[4] `const` element with a `type` attribute that identifies the chunk/constituent type

[5] `pos` element with `type` and `normal` attributes

[6] We use the terms *preceding sibling*, *following sibling*, *parent*, and *ancestor* as defined by XPath [21].

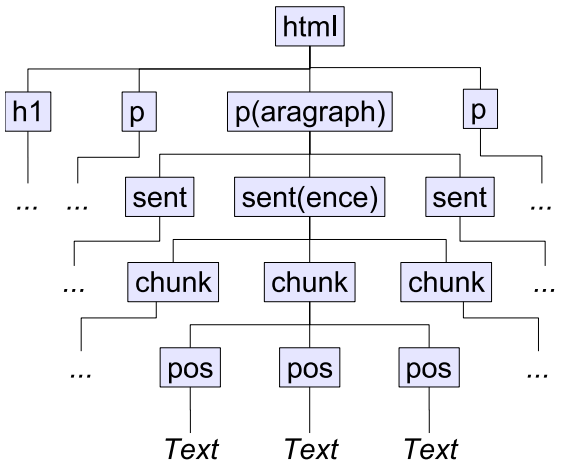

Fig. 1. Partial DOM Tree of a Simple HTML Document with Linguistic Annotations

- Up to four ancestors of the element (typically the embedding chunk, sentence, paragraph or related unit, etc.)
- Preceding and following siblings of each ancestor—the number of included siblings is decremented for each higher level of ancestors (three for the direct parent, i.e. three preceding and three following chunks; two for the "grandparent", i.e. sentence; etc.)

In addition to this DOM tree–based context, we add information on the last four slot fillers found in the current document, similar to the *lastTarget* variable used in [12].

In the DOM tree creating during preprocessing, all leaf nodes are POS elements. Each POS element contains a text fragment for which we include several features:

- The text fragment, both in original capitalization and converted to lowercase;
- Prefixes and suffixes from length 1 to 4, converted to lower-case;[7]
- The length of the fragment;[8]
- The type of the fragment (one of *lowercase, capitalized, all-caps, digits, punctuation, mixed* etc.)

Additionally, the semantic class(es) the fragment belongs to are listed, if any. For this purpose a configurable list of dictionaries and gazetteers are checked. Currently we use the following semantic sources:

[7] Pre-/suffixes that would contain the whole fragment are omitted.
[8] Both the exact value and the rounded square root as a less sparse representation.

– An English dictionary;[9]
– Name lists from US census;[10]
– Address suffix identifiers from US Postal Service;[11]
– A list of titles from Wikipedia.[12]

All other elements are inner nodes which contain child elements, they do not directly contain text. For *chunk* elements, we include the normalized form of the last POS that is not part of a sub-chunk as head word. For elements containing chunks (such as *sentence*), the head words of the left-most and the right-most chunk are included. For other elements, no features are generated, except the name of the element and any attributes stored in the DOM tree[13] which are included for all elements. For the represented POS element and its ancestors, we also store the position of the element within its parent.

Th result is a fairly high number of features representing the context of each token. The features are arranged in an ordered list to allow recombination via OSB (Sec. 2.3); the resulting feature vector is provided as input to the classifier (cf. Sec. 2.2).

4 Experimental Results

We have tested our approach on the *CMU Seminar Announcements*[14] corpus, a standard corpus that is used very often to evaluate IE systems. The corpus contains 485 seminar announcements (plain text files) collected from university newsgroups; the task is to extract up to four fragment slots from each document (if present): *speaker*, *location*, *start time (stime)* and *end time (etime)* of the talk.

We randomly shuffled the order of documents in the corpus and used the first 50% of documents for training and the other 50% for evaluation, averaging results over five random shuffles.[15] As usual for this corpus, we used the "one answer per slot" approach for evaluation (cf. [10]): at most one instance of each slot is to be extracted from each document; if there are several annotated fragments in a document, it is sufficient to find one. If our system finds multiple extraction candidates, it selects the most probably one. Only exact matches are accepted—partial matches are counted as errors.

Extraction results are evaluated in the usual way by counting *true positives tp* (correct slot fillers), *false positives fp* (spurious slot fillers), *false negatives fn*

[9] `http://packages.debian.org/testing/text/wamerican`
[10] `http://www.census.gov/genealogy/names/`
[11] `http://www.usps.com/ncsc/lookups/abbreviations.html`
[12] `http://en.wikipedia.org/wiki/Title`
[13] Type of parts-of-speech and chunks, normalized form of parts-of-speech, etc.
[14] `http://www-2.cs.cmu.edu/~dayne/SeminarAnnouncements/__Source__.html`
[15] The most typical evaluation setup for this corpus; some other systems average over ten shuffles.

Table 1. F1 Results Compared to Other Approaches

Approach	**TIE**		**BWI**	**ELIE**		**HMM (LP)^2**	**MaxEnt**	**MBL**	**SNoW-IE**
	Inc.	**Iter.**		**L1**	**L2**				
Reference			[7]	[6]		[8]	[2]	[1]	[22]	[13]
etime	96.7	97.5	93.9	87.0	96.4	59.5	95.5	94.2	96	96.3
location	79.3	80.6	76.7	84.8	86.5	83.9	75.0	82.6	87	75.2
speaker	80.9	85.2	67.7	84.9	88.5	71.1	77.6	72.6	71	73.8
stime	99.2	99.3	99.6	96.6	98.5	99.1	99.0	99.6	95	99.6
Average	88.3	89.9	83.9	88.8	92.1	81.7	86.0	86.9	86.6	85.3

Table 2. Results With Incremental Feedback

F1	**Evaluation Set**	**All Files**
etime	97.8	94.2
location	80.2	73.2
speaker	83.9	77.0
stime	99.2	98.0
Average	89.5	84.8

(missing slot fillers) and calculating *precision* $P = \frac{tp}{tp+fp}$ and *recall* $R = \frac{tp}{tp+fn}$. *F1 measure* is the harmonic mean of precision and recall:

$$F1 = \frac{2 \times P \times R}{P + R}.$$

To combine the results from all slot types into a single measure, we report the *weighted average* as used in [1] where each slot type is weighted by the total number of expected slot fillers in the corpus (485 *start times*, 464 *locations*, 409 *speakers*, 228 *end times*). All reported performance figures are *F1* percentages.

Table 1 compares our system (called TIE, "Trainable Information Extractor") with other approaches evaluated in the same way.[16] When trained incrementally (first column), our system is better than all other approaches, except one (the ELIE system described in [6]).[17] ELIE uses Support Vector Machines in a two-level approach, while our system so far is limited to a single level. When resigning incrementality and iteratively training our system until accuracy of the token classifiers on the training set stops increasing (second column), our system

[16] One other approach, BIEN [12], is not directly comparable, since it uses an 80/20 split instead of 50/50. When run with an 80/20 split, the overall result of our system (in incremental mode) is 89.5%; BIEN reaches 88.9%.

The reported results are all from *trainable* systems (mainly statistical ones, while some—BWI, (LP)^2—use rule-learning). In the past, domain-specific rule-based systems haven often been able to outperform trainable approaches. However, for this corpus we are not aware of comparable or superior results reached by static, handcrafted systems.

[17] Testing the statistical significance of performance differences is not possible since it would require detailed test results of the other systems which are not available.

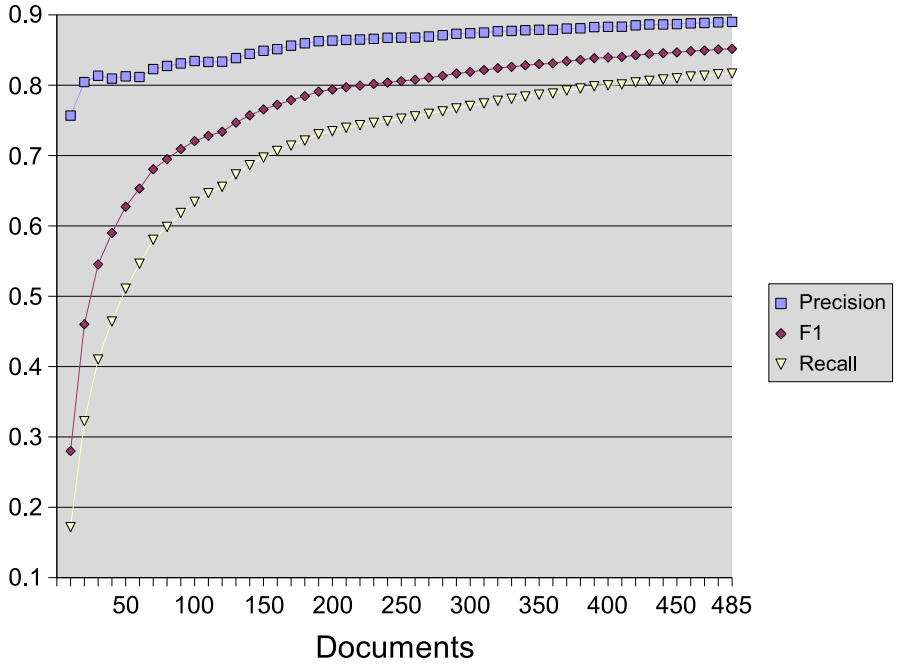

Fig. 2. Incremental Feedback: Learning Curve (precision, recall, and F1 on all documents processed so far)

outperforms their first level by more than 1%. Further improvement should be possible by adding a second level similar to theirs.

In the usual setup, 50% of all documents are used for training and the rest for evaluation (50/50 split). In an incremental approach, it is possible to adapt the extraction model even during the evaluation phase, by allowing the classifier to train the correct slot fillers from each document after evaluating its own proposals for this document.[18] With this feedback added, the F1 measure on the evaluation set increases to 89.5% (Table 2, left column).

With this feedback mechanism it is not strictly necessary to start with a training-only phase; the system can be used to propose slot fillers to be evaluated from the very start, using the whole corpus as evaluation set (0/100 split). Tested in this way, our system still reaches almost 85% F1 over *all documents* (right column). This means the system can be beneficial to use very soon, without requiring a tedious manual annotation phase to provide initial training data.

[18] This corresponds to the workflow from the Introduction where the system proposes slot fillers which are reviewed and corrected by a human supervisor. After the supervisor has corrected a document, the system updates its extraction model prior to processing the next document. In this way the quality of the extraction proposals will continually improve.

Table 3. Ablation Study

F1	Default	No Semantic	No HTML	No Linguistic	No OSB
etime	96.7	97.2	97.0	89.2	95.5
location	79.3	78.0	76.8	68.0	69.3
speaker	80.9	77.0	72.8	53.6	64.9
stime	99.2	99.4	99.4	99.1	98.7
Average	88.3	87.0	85.6	76.8	80.9

Fig. 2 shows the learning curve in this setup. As can be seen, precision is high from the very start—more than 75% after the first 10 documents, more than 80% after 20. Initial recall is far lower, but it exceeds 50% after processing 50 documents and 70% after 160 documents.

Table 3 shows the relative importance of different sources of information. Semantic information is less important than for other systems—without it, F1 drops by only 1.3%, while $(LP)^2$ reports a performance drop by 23% (from 86% to 63.1%); for BIEN it is 11% (from 88.9% to 77.8%). This indicates that our approach makes efficient use of syntactic and linguistic features to compensate for missing explicit semantic data.

More relevant is the heuristic preprocessing step to recognize document structure in the plain text input (*txt2html*, cf. Sec. 3.1). Linguistic annotation (*TreeTagger*) contributes most to the results, not surprisingly. We also find that the OSB feature combination technique (cf. Sec. 2.3) is indeed useful—without it, F1 degrades by 7.4%.

5 Related Work

There are several other approaches modeling IE as a classification task: [1] uses Maximum Entropy modeling with four classes for each slot type (*X*-begin, *X*-continue, *X*-end, *X*-unique). [6] uses two SVM classifiers for each slot type, one for detecting the begin and the other for detecting the end of slot fillers. [22] uses Memory-based Learning (MBL) with the *IOB1* strategy[19].

While there are multiple approaches to statistical IE, most of them use methods that are unsuitable for incremental training. One other approach, SNoW-IE [13], employs the Winnow algorithm, but since it uses several parameters that are determined from the training data it cannot be trained incrementally.[20] We are not aware of any approach that supports incremental training.

[19] Which differs from *IOB2* (Sec. 2.1) in using B-*type* only when necessary to avoid ambiguity; otherwise I-*type* is used even at the beginning of slot fillers.

[20] SNoW-IE realizes a two-step approach. Among a small number of possible candidate fragments identified in a *filtering* stage, the (presumably) correct text fragment is determined and extracted in a *classifying* stage. The complete training data is inspected to determine minimum scores necessary to pass the filtering stage as well as specific conditions fulfilled by all or most positive instances (such as the maximum length of slot fillers).

The employment of a (DOM) tree-based representation of the input documents and the use of heuristics to recognize document structure (*txt2html* conversion) appear to be other novel traits of our approach.

6 Conclusion and Future Work

We have introduced an approach to modeling information extraction as a token classification task that allows incremental updating of the extraction model. To provide sufficient information to the token classifiers, we use rich, tree-based context representations of each token as feature vectors. These representations make use of the heuristically deduced document structure in addition to linguistic and semantic information. We consider the resulting feature vectors as ordered and combine proximate features into more expressive joint features, using the OSB combination technique. Our results indicate that this setup makes it possible to employ IE in an incremental fashion without a serious performance penalty.

There are several directions for future work. We plan to try our system for other tasks and to explore variations of the used context representations in more detail. To augment our current single-level setup, we will add a correction mode that reconsiders misclassified tokens near extraction candidates. We are also experimenting with a sentence filtering step to reduce the number of tokens to be presented to the token classifiers, similar to the approach proposed in [4].

Currently our system is limited to very simple target structures—it handles only *slot filling* (extraction of text fragments). We plan to add support for more complex target structures by extending the system to handle *relationship recognition* (e.g. a **works-for** relation between a *person* and a *company*) and *template unification* (deciding which slot fillers give details about the same complex object, e.g. a seminar). In the used CMU task this isn't necessary because each document contains only one seminar announcement, but in real-life applications there will often be multiple relevant objects per document.

The IE system presented in this paper is available as free software [18].

References

1. H. L. Chieu and H. T. Ng. A maximum entropy approach to information extraction from semi-structured and free text. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI 2002)*, pages 786–791, 2002.
2. F. Ciravegna. $(LP)^2$, an adaptive algorithm for information extraction from Web-related texts. In *Proceedings of the IJCAI-2001 Workshop on Adaptive Text Extraction and Mining*, Seattle, USA, 2001.
3. I. Dagan, Y. Karov, and D. Roth. Mistake-driven learning in text categorization. In C. Cardie and R. Weischedel, editors, *Proceedings of EMNLP-97, 2nd Conference on Empirical Methods in Natural Language Processing*, pages 55–63, Providence, US, 1997. Association for Computational Linguistics.

4. A. De Sitter and W. Daelemans. Information extraction via double classification. In *Proceedings of the International Workshop on Adaptive Text Extraction and Mining (ATEM-2003)*, 2003.
5. A. Finn and N. Kushmerick. Active learning selection strategies for information extraction. In *Proceedings of the International Workshop on Adaptive Text Extraction and Mining*, 2003.
6. A. Finn and N. Kushmerick. Information extraction by convergent boundary classification. In *AAAI-2004 Workshop on Adaptive Text Extraction and Mining*, San Jose, USA, 2004.
7. D. Freitag and N. Kushmerick. Boosted wrapper induction. In *AAAI/IAAI*, pages 577–583, 2000.
8. D. Freitag and A. K. McCallum. Information extraction with HMMs and shrinkage. In *Proceedings of the AAAI-99 Workshop on Machine Learning for Information Extraction*, 1999.
9. JTidy. `http://jtidy.sourceforge.net/`.
10. A. Lavelli, M. Califf, F. Ciravegna, D. Freitag, C. Giuliano, N. Kushmerick, and L. Romano. A critical survey of the methodology for IE evaluation. In *Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC 2004)*, 2004.
11. N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.
12. L. Peshkin and A. Pfeffer. Bayesian information extraction network. In *IJCAI*, 2003.
13. D. Roth and W.-t. Yih. Relational learning via propositional algorithms: An information extraction case study. In *IJCAI*, 2001.
14. T. Scheffer, S. Wrobel, B. Popov, D. Ognianov, C. Decomain, and S. Hoche. Learning hidden Markov models for information extraction actively from partially labeled text. *Künstliche Intelligenz*, (2), 2002.
15. C. Siefkes. A shallow algorithm for correcting nesting errors and other well-formedness violations in XML-like input. In *Extreme Markup Languages (EML) 2004*, 2004.
16. C. Siefkes, F. Assis, S. Chhabra, and W. S. Yerazunis. Combining Winnow and orthogonal sparse bigrams for incremental spam filtering. In J.-F. Boulicaut, F. Esposito, F. Giannotti, and D. Pedreschi, editors, *Proceedings of the 8th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD 2004)*, volume 3202 of *Lecture Notes in Artificial Intelligence*, pages 410–421. Springer, 2004.
17. S. Soderland, D. Fisher, J. Aseltine, and W. Lehnert. CRYSTAL: Inducing a conceptual dictionary. In *IJCAI*, 1995.
18. Trainable Information Extractor. `http://www.inf.fu-berlin.de/inst/ag-db/software/tie/`.
19. TreeTagger. `http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/`.
20. txt2html. `http://txt2html.sourceforge.net/`.
21. *XML Path Language (XPath) 2.0*, 2004. W3C Working Draft, 29 October 2004.
22. J. Zavrel and W. Daelemans. Feature-rich memory-based classification for shallow NLP and information extraction. In J. Franke, G. Nakhaeizadeh, and I. Renz, editors, *Text Mining, Theoretical Aspects and Applications*, pages 33–54. Springer Physica, 2003.