To appear in IJCAI-99 Workshop on Text Mining: Foundations, Techniques and Applications

Finite-State Transducers for Semi-Structured Text Mining

Chun-Nan Hsu and Chien-Chi Chang
Institute of Information Science, Academia Sinica
Nankang 115, Taipei City, TAIWAN
{chunnan,top}@iis.sinica.edu.tw
Ph: +886-2-27883799x1801 Fax: +886-2-27824814
http://chunnan.iis.sinica.edu.tw/

Abstract

Text mining for semi-structured documents requires information extractors. Programming extractors by hand is difficult to catch up with the amount and the variation of the documents placed on the World-Wide Web everyday. This paper presents our recent result on applying machine learning techniques to automatize the generation of the extractors. Our goal is to develop a domain and language independent approach that automatically learns an extractor from training examples of extraction. In particular, this paper discusses the use of finite-state transducers (FST) as the representation formalism of the extractors. Previously, we have shown that only a small number of examples is enough for learning perfect FST-based extractors for documents from a variety of real Web sites. In this paper, we introduce two classes of FST-based extractors: *single-pass* and *multi-pass*, and informally analyze their relative advantages and disadvantages. In terms of their sample complexity, we hypothesized that single-pass extractors may have an edge for *tabular documents* while multi-pass extractors may require fewer examples for *tagged-list documents*. We verified this hypothesis empirically and found that by slightly changing the problem setting, a single-pass learner can outperform a multi-pass learner for tagged-list documents. A prototype system based on this work will be presented in the demo session of this workshop.

1 Introduction

An approach to text mining from semi-structured documents on the World-Wide Web is to convert the documents into a collection of structured data and then apply data mining techniques to analyze the data. This conversion involves information extraction from semi-structured documents. For example, consider a fragment of `Caltech CS faculty` Web page[1] shown in Figure 1(a). This page is a typical semi-structured document on the Web. This fragment shows the information about four faculty members. The information of each faculty consists of a sequence of attribute values. In this case, the attributes are URL U, name N, academic title A and administrative title M. An information extractor for this Web page is supposed to take its HTML source as input (see Figure 1(b)), extract the attribute values, and return four faculty data tuples $\langle U, N, A, M\rangle$. The expected correct output is shown in Figure 1(c).

Previous work in information extraction (see e.g., [DARPA, 1995]) are usually aimed at domain-specific free text and require a substantial linguistic knowledge-base. Meanwhile, a widely used approach is to program extractors by hand in some language specialized in string processing such as `Perl`. Since semi-structured documents are rendered in a regular fashion, potentially, machine learning can exploit the regularity of their *appearance* to generate a perfect extractor. To catch up with the amount and the variation of the documents placed on the World-Wide Web everyday, researchers have developed many machine learning approaches to accelerate extractor generation [Kushmerick, 1997; Hsu & Dung, 1998; Muslea, Minton, & Knoblock, 1999]. This paper presents our recent result in this direction.

Although semi-structured documents are structured regularly, frequently encountered irregularities, such as missing attribute values, may impede the learning [Hsu & Dung, 1998]. A critical issue is selecting an appropriate representation for the extractor such that it is not only sufficiently expressive to describe the extraction steps to handle the irregularities in the documents, but also sufficiently simple to be learnable. This issue is known as the selection of *inductive bias* in machine learning (see e.g., [Haussler, 1988]). This paper discusses *finite-state transducers* (FST) as the inductive bias for learning extractors of semi-structured documents.

Since learning FST with unknown graph structure can be shown to be intractable [Biermann & Feldman, 1972; Angluin, 1987], we will restrict the space of possible graph structures or assume that a structure is given.

[1] `http://www.cs.caltech.edu/csstuff/faculty.html`

Faculty and Research Staff

- Mani Chandy
  Professor of Computer Science and *Executive Officer for Computer Science*
- Yaser Abu-Mostafa
  Professor of Electrical Engineering and Computer Science
- Jim Arvo
  Associate Professor of Computer Science
- John Tanner
  Visiting Associate of Computer Science

(a) A fragment of Caltech CS faculty Web page

```
<H1>Faculty and Research Staff</H1>
<DT> <A HREF="http://www.cs.caltech.edu/people/mani.html"><B>Mani Chandy</B></A>
<DD> <I>Professor of Computer Science</I> and
  <I>Executive Officer for Computer Science</I>
<DT> <A HREF="http://electra.caltech.edu/EE/Faculty/Abu-Mostafa.html"><B>Yaser Abu-Mostafa</B></A>
<DD> <I>Professor of Electrical Engineering and Computer Science</I>
<DT> <A HREF="http://www.cs.caltech.edu/~arvo/home.html"><B>Jim Arvo</B></A>
<DD> <I>Associate Professor of Computer Science</I>
<DT> <B>John Tanner</B>
<DD> <I>Visiting Associate of Computer Science</I>
```

(b) The HTML source code

```
U = ``http://www.cs.caltech.edu/people/mani.html''
N = ``Mani Chandy''
A = ``Professor of Computer Science''
M = ``Executive Officer for Computer Science''

U = ``http://electra.caltech.edu/EE/Faculty/Abu-Mostafa.html''
N = ``Yaser Abu-Mostafa''
A = ``Professor of Electrical Engineering and Computer Science''

U = ``http://www.cs.caltech.edu/~arvo/home.html''
N = ``Jim Arvo''
A = ``Associate Professor of Computer Science''

N = ``John Tanner''
A = ``Visiting Associate of Computer Science''
```

(c) Expected results of extraction

Figure 1: Extracting faculty data from Caltech CS Web page

Previously, we proposed a special type of FST called **SoftMealy** and showed that a small amount of training examples is sufficient for learning a perfect SoftMealy extractor for the documents from a variety of real Web sites [Hsu, 1998; Hsu & Dung, 1998].

In this paper, we compare two general classes of SoftMealy extractors: **single-pass** and **multi-pass**, and present an informal comparison of their relative advantages and disadvantages. Our initial results lead to a dichotomy of document layout styles. This dichotomy divides semi-structured documents into two general layout styles: **tabular** and **tagged-list**. We hypothesized that single-pass extractors may have an edge in tabular documents while multi-pass extractors may require fewer examples in tagged-list documents. We implemented the learning algorithms for the two classes of the extractors and verified our hypothesis empirically. Moreover, by slightly changing the problem setting, we found that a single-pass extractor learner can also handle tagged-list documents well. We also compared our result with previous work in the literature and the result is also consistent with our hypothesis.

The remainder of this paper describes the categorization of finite-state transducer-based extractors, the analysis, and the experimentation.

2 SoftMealy Machines

The finite-state transducer (FST), also known as the *"Mealy Machine"* [Hopcroft & Ullman, 1979], is a variation of the finite-state automaton. Since learning finite-state automata with unknown graph structure is known to be intractable [Biermann & Feldman, 1972; Angluin, 1987], learning FST can be at least as difficult. Therefore, we need to restrict the space of possible graph structures or assume that a structure is given. Previously, we proposed a new type of finite-state transducers called a *SoftMealy machine* [Hsu & Dung, 1998] specialized in semi-structured document information extraction. SoftMealy machines embody a minimal set of the restrictions that we believe to be necessary to make the learning problem practical yet preserve the flexibility to assure sufficient expressiveness for a variety of semi-structured documents. This section presents version 1.1 of SoftMealy machines.

SoftMealy assumes that an input document will be tokenized in some granularity before it was fed into a finite-state transducer. We define a *separator* as the invisible borderline between two adjacent tokens. A SoftMealy machine takes **a sequence of the separators** rather than the raw document string as input. Each time a new separator is read, the FST checks the patterns of its *context* to determine the state transition. That is, the FST will check the tokens at the left of the separator as well as those at its right. The purpose is to allow

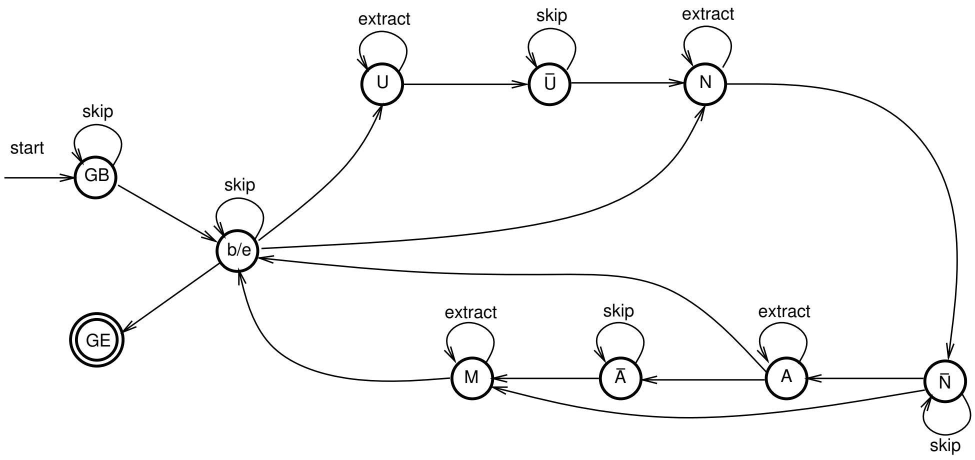


Figure 2: Single-pass SoftMealy extractor

for flexibility so that transitions in the FST can be determined by examining the tokens that have been read as well as look-ahead tokens. In a nutshell, the input scanning of a SoftMealy machine can be thought of as moving a flashlight beam through a token string [Hsu & Dung, 1998].

States in a SoftMealy machine are designed to designate when to extract a token and when to skip, depending on whether it is in a state corresponding to an attribute or a dummy attribute. A *dummy attribute* is a string that we intend to skip between two attributes. We use $\bar{k}$ to denote the dummy attribute after attribute k. For example, in the first tuple in Figure 1(b), $\bar{N}$ =“`</B></A>\newline<DD> <I>`” is a dummy attribute between N =`‘‘Mani Chandy’’` and the next attribute A =`‘‘Professor of Computer Science’’`.

State transitions in a SoftMealy machine are determined by matching the context of the input separator with the *contextual rule* associated with each edge. Contextual rules can be of any forms that describe the pattern of a context; that is, the right and left sides of a separator. A SoftMealy machine is formally defined as follows:

Definition 1 (SoftMealy Machines) *A SoftMealy machine is a 5-tuple* $(\Sigma_1, \Sigma_2, Q, R, E)$ *such that:*

- Σ_1 *is the set of* **separators** *as the input alphabet.*
- Σ_2 *is the set of characters as the output alphabet.*
- Q *is a finite set of states containing:*
  - *State GB and GE (global begin and global end), the initial state and the final state;*
  - *State b/e, representing both the beginning and the end of an extraction loop;*
  - *State k for each attribute k to be extracted;*
  - *states for* dummy attribute*, the tokens that we intend to skip.*
- R *is the set of contextual rules. R contains a special contextual rule, denoted as* $'?'$*, which stands for a rule covering any input separator that does not match on any other outgoing edge.*
- $E \subseteq Q \times R \times \Sigma_2^* \times Q$ *is the set of edges. A state transition from i to j is legal if there exists an edge* $\langle i, r, o, j \rangle \in E$ *such that the next input separator satisfies r. For all reflexive edges* $(i = j)$, $r = '?'$. *The output o is defined as follows:*
  - *“start-to-extract”: when $i \neq j$ and j corresponds to an attribute to be extracted, then the output o is the concatenation of “j =” and the token right next to the position of the current input separator.*
  - *“extract”: when $i = j$ and j represents an attribute to be extracted, then o is the token right next to the position of the current input separator.*
  - *“skip”: Otherwise, o is ε.*

3 SoftMealy Extractors

There are two general classes of extractors based on SoftMealy machines:

- **Single-pass** A single-pass SoftMealy extractor is an extractor based on a single SoftMealy machine, which scans an input document and extracts all attributes and data tuples in a single pass.
- **Multi-pass** A multi-pass SoftMealy extractor, in contrast, scans an input document for several passes. In each pass, the extractor extracts just one type of objects, such as a string corresponding to a tuple, or a string corresponding to an attribute. A multi-pass extractor is based on a collection of SoftMealy machines.

Example 1 (Single-Pass Extractor) Figure 2 shows the diagram of a single-pass SoftMealy extractor for the

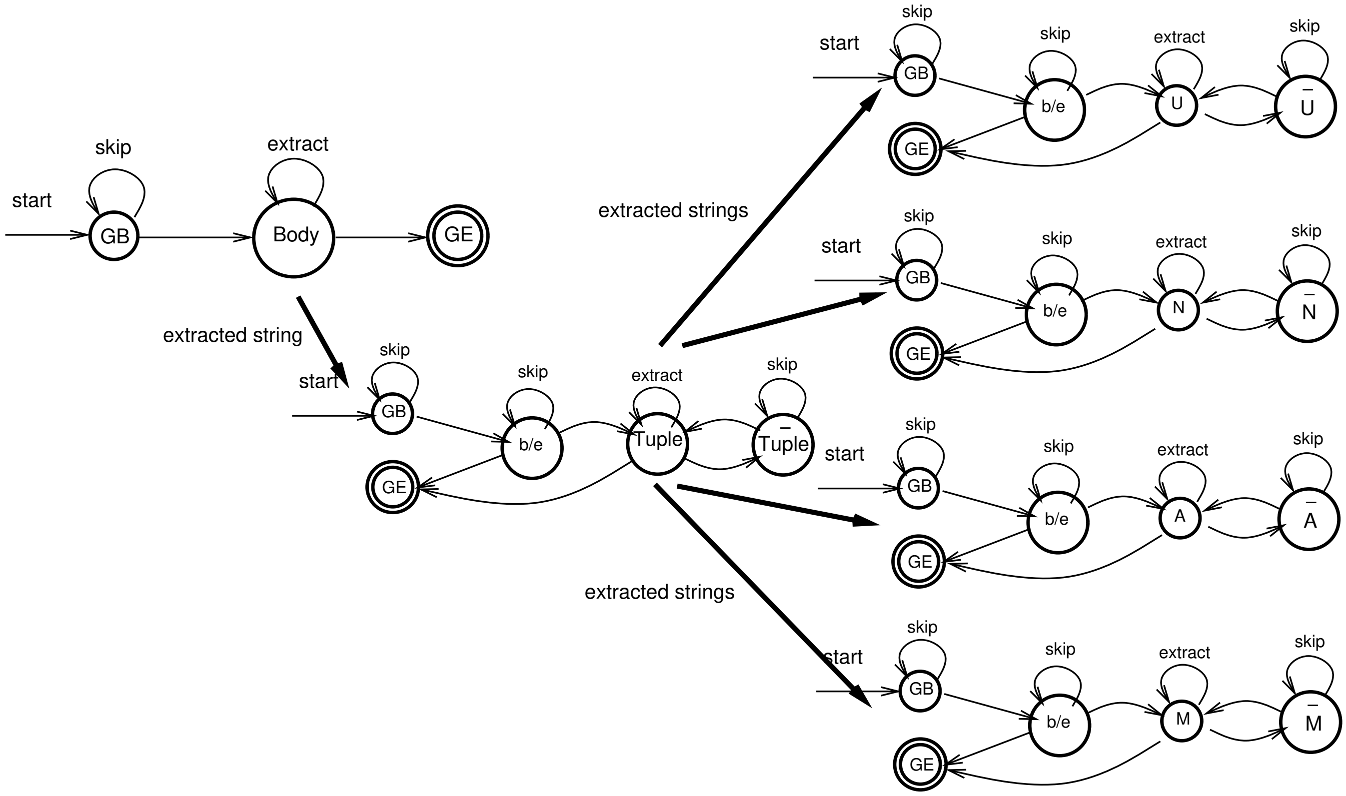


Figure 3: Multi-pass SoftMealy extractor

Web page in Figure 1. This FST has ten states, including GB, GE, b/e, four states corresponding to attributes U,N,A,M, and the three states for dummy attributes $\bar{U}$,$\bar{N}$,$\bar{A}$.

To extract faculty data from Figure 1 (b), the extractor starts in GB and remains there until it recognizes the beginning of the first data tuple. That is, the extractor will skip the string starting from `‘‘<H1>Faculty...’’` to `‘‘HREF="’’` before it enters b/e. Then, the extractor will extract attributes U,N,A,and M, skip the dummy attributes in between, and return to b/e to complete an extraction cycle for the first data tuple.

For the second data tuple, the extractor will extract only U, N, and A, since M is missing. The extractor will then choose the edge from A to b/e to complete the extraction. This process continues until the extractor recognizes that there is no more data tuple and enters GE to complete the extraction of the entire Web page. □

Example 2 (Multi-Pass Extractor) Figure 3 illustrates an instance of multi-pass extractors for the same Web page. This extractor consists of six SoftMealy machines. The extraction starts at the leftmost FST, which extracts the part of the input document (referred to as the "body") that contains the target data tuples. The extracted body is the input of another FST which repeatedly extracts data tuples. Next, a set of four FSTs is invoked to extract each of the four attributes. These FSTs take each extracted data tuple as the input, repeatedly search for the target attributes, and extract them until their control reach the end of the input string. □

In terms of the expressiveness, both classes of extractors can handle irregular attribute permutations frequently encountered in semi-structured documents. Irregular attribute permutations occur in the following situations:

- *Missing attributes:* For example, a faculty may not have an administrative title.
- *Multiple attribute values:* For example, a faculty may have two or more administrative titles.
- *Variant attribute ordering:* For example, the academic title that appears before the administrative title in most data tuples may appear after the administrative title in others.

In `Caltech CS Faculty` Web page, there are a total of four distinct attribute permutations of the four attributes:

$$(U, N, A, M)\ (U, N, A)\ (N, A)\ (U, N, M).$$

For a single-pass extractor to handle these irregularities, each possible attribute permutation in the target documents must be represented as a cycle of inter-state transitions that starts and finishes in b/e. This is because

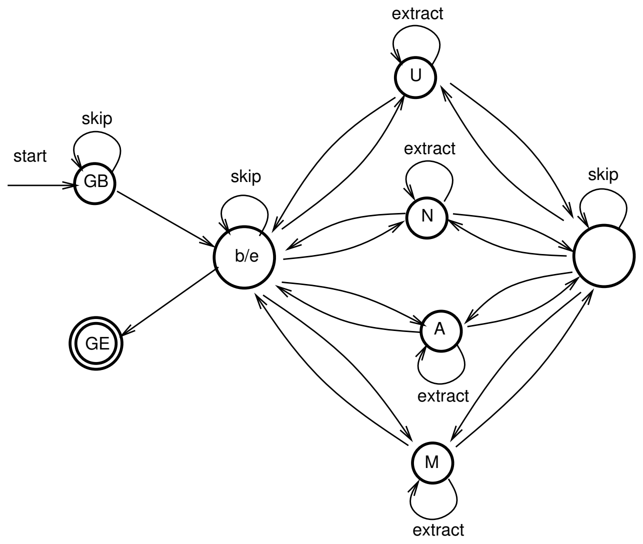


Figure 4: Universal single-pass SoftMealy extractor

a single-pass extractor must determine exactly which attribute is the next to be extracted after extracting an attribute string. The extractor illustrated in Figure 2 is a correct single-pass extractor for the target page, because it covers all of the four distinct attribute permutations. In fact, the graph structure of this extractor is *economical* with respect to the number of edges because there is an edge between a pair of states only if their corresponding attributes are adjacent in the target document.

In contrast, a multi-pass extractor allows any attribute permutations in an input document. A single-pass extractor can achieve the same flexibility using a single dummy attribute state connected bi-directionally with all attribute states. We call this type of single-pass extractors as **universal single-pass** extractors. Figure 4 illustrates such an extractor for `Caltech CS Faculty` Web page. Compared with the single-pass extractor in Figure 2, the graph structure of this extractor is more complex.

In terms of the efficiency of parsing, a multi-pass extractor needs to scan a document several times and may not be adequate to the applications where on-line extraction is required. In contrast, a single-pass extractor can read in an on-line document as a stream and return partially extracted data tuples even if the communication network is stalled during the extraction. This is almost impossible for a multi-pass extractor.

4 Learning SoftMealy Extractors

Now we consider the problem of learning about the graph structure and the contextual rules of a SoftMealy extractor given training examples of extraction. The training examples consist of the intervals of the offsets of the attribute strings in one or more sample documents with the same layout. More precisely, for each sample document, these offsets include:

- The interval of the region, referred to as the *"body"*, that contains the data tuples to be extracted.
- The intervals of one or more *example data tuples*. Each interval for an example data tuple consist of the offsets of the beginning and the end of the string corresponding to a data tuple.
- The intervals of all attributes presenting in each example data tuple.

Example 3 (Training Examples) Suppose we select the first faculty item in `Caltech CS Faculty` Web page as an example data tuple. The interval of this example is $[51, 211]$. Offset 51 is between `` ` ` <DT> <A HREF="' ' `` and `` ` `http://' ' `` in line 2, and offset 211 is between `` ` `Science' ' `` and `` ` `</I>' ' `` in line 4. The interval of faculty name N in this example is $[99, 109]$, the offsets of the beginning and the end of the string `` ` `Mani Chandy' ' ``, respectively. □

The learnable components of a SoftMealy extractor include its graph structure and the contextual rules associated with its edges. A learner can select either single-pass or multi-pass as its inductive bias of the graph structure. If the selected bias is a multi-pass or universal single-pass extractor, the learner can fully determine the graph structure solely based on the number of the attributes in the target documents.

To learn a single-pass extractor with an economical graph structure, a conservative approach is to create edges when it is necessary:

- create an edge from b/e to i for each attribute i appears at the beginning of an example data tuple;
- create an edge from state i to $\bar{i}$ and an edge from $\bar{i}$ to j for each pair of attributes i and j that are adjacent in any example data tuples;
- create an edge from i to b/e for each attribute i appears at the end of an example data tuple.

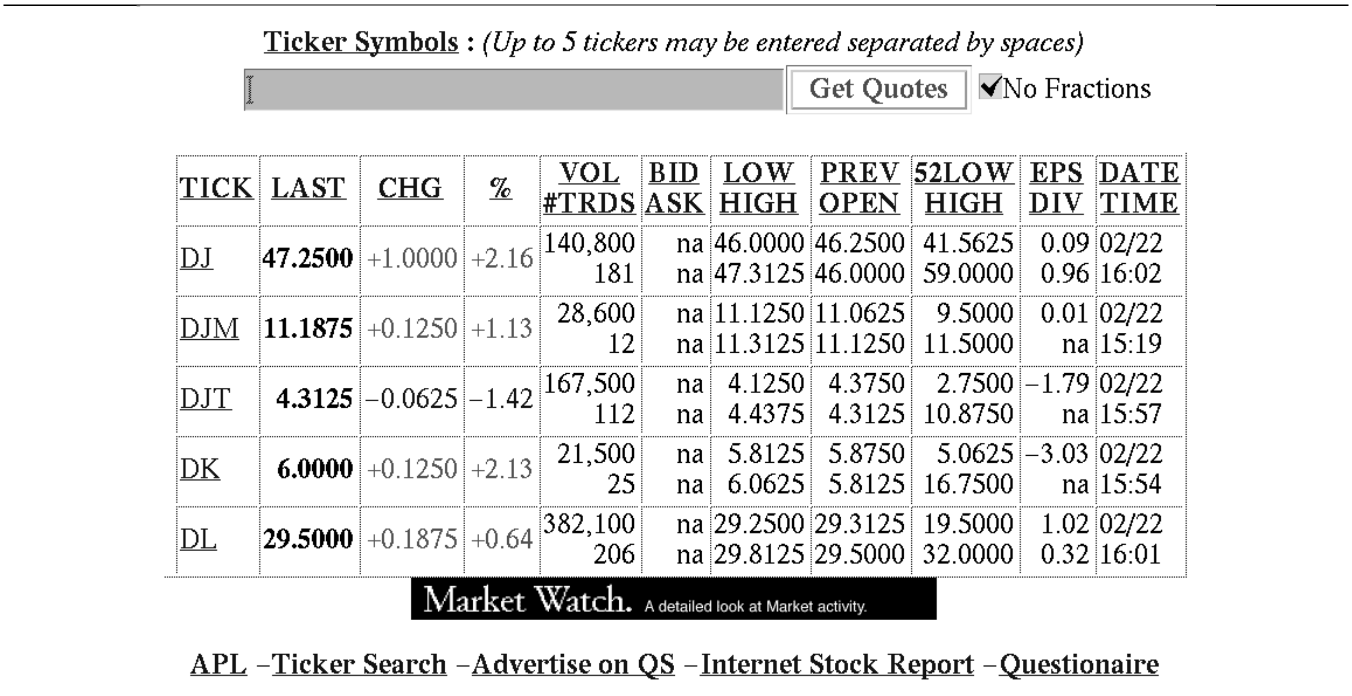
The resulting edges will form cycles that cover all possible attribute permutations appearing in the example data tuples.

Example 4 (Learning Graph Structure) Suppose the faculty items in Figure 1 are selected as example data tuples where we have the following attribute permutations:

$$(U, N, A, M)\ (U, N, A)\ (N, A).$$

To cover these attribute permutations, the learner will create the edges between the following pairs of states:

$\langle b/e, U\rangle$,
$\langle b/e, N\rangle$,
$\langle U, \bar{U}\rangle$, $\langle \bar{U}, N\rangle$,
$\langle N, \bar{N}\rangle$, $\langle \bar{N}, A\rangle$
$\langle A, \bar{A}\rangle$, $\langle \bar{A}, M\rangle$,
$\langle A, b/e\rangle$,
$\langle M, b/e\rangle$. □


Ticker Symbols : *(Up to 5 tickers may be entered separated by spaces)*

Get Quotes ✔No Fractions

TICK	LAST	CHG	%	VOL #TRDS	BID ASK	LOW HIGH	PREV OPEN	52LOW HIGH	EPS DIV	DATE TIME
DJ	**47.2500**	+1.0000	+2.16	140,800 181	na na	46.0000 47.3125	46.2500 46.0000	41.5625 59.0000	0.09 0.96	02/22 16:02
DJM	**11.1875**	+0.1250	+1.13	28,600 12	na na	11.1250 11.3125	11.0625 11.1250	9.5000 11.5000	0.01 na	02/22 15:19
DJT	**4.3125**	−0.0625	−1.42	167,500 112	na na	4.1250 4.4375	4.3750 4.3125	2.7500 10.8750	−1.79 na	02/22 15:57
DK	**6.0000**	+0.1250	+2.13	21,500 25	na na	5.8125 6.0625	5.8750 5.8125	5.0625 16.7500	−3.03 na	02/22 15:54
DL	**29.5000**	+0.1875	+0.64	382,100 206	na na	29.2500 29.8125	29.3125 29.5000	19.5000 32.0000	1.02 0.32	02/22 16:01

Market Watch. A detailed look at Market activity.

APL –**Ticker Search** –**Advertise on QS** –**Internet Stock Report** –**Questionaire**

Figure 5: Example of a tabular page: Quote Server

Once the graph structure is determined, the component remains to be learned is the contextual rule associated with each inter-state edge. Recall that a SoftMealy machine reads a sequence of separators as input. Each offset in the training example actually corresponds to an example of a class of separators between two particular attributes, and the extractor must learn to recognize these separators using the contextual rules. We first examine how a SoftMealy machine determines its state transition to precisely specify the problem of contextual rule learning.

Suppose a state i in an extractor has a set of outgoing edges of the form $\langle i, r_n, o_n, j_n\rangle$ and a reflexive edge $\langle i, '?', o_0, i\rangle$. Note that the contextual rule of a reflexive edge is always $'?'$ which will be satisfied when no other contextual rule of the outgoing edges is satisfied.

When the control is in state i and reading a new separator, the extractor will match the context of this separator with every contextual rule associated with its edge to determine the next state. r_n is a correct contextual rule if all of the following conditions hold:

- r_n is satisfied when the control is reading a separator between (dummy) attributes i and j_n;
- For all $m \neq n$, r_n can **not** be satisfied when the control is reading a separator between (dummy) attributes i and j_m;
- r_n can **not** be satisfied when the control is reading a separator within the string corresponding to (dummy) attribute i, where the control should stay in state i.

These conditions also guarantee that a SoftMealy machine is deterministic.

Now we are ready to cast our problem as a rule induction problem that can be solved by a set-covering algorithm [Haussler, 1988]: A correctly learned contextual rule r_n must cover all positive examples and exclude all negative examples, where the *positive examples* of r_n are the offsets between attributes i and j_n, and the *negative examples* are the offsets between i and j_m, $m \neq n$, as well as the offsets within the string corresponding to i.

Note that this specification is independent of the graph structure. To directly apply a rule induction algorithm for this problem, however, we still need to convert the context strings as feature vectors with fixed length. [Hsu & Dung, 1998] describes this conversion in detail.

Example 5 (Learning Contextual Rules) Suppose the learner is trying to learn a single-pass extractor as shown in Figure 2. Also suppose that the learner has the offsets of the four faculty member data in Figure 1 (b) as the example data tuples.

Consider the problem of learning the contextual rule for the edge from b/e to U. There are three positive examples. The first one is at offset 51, in line 2, the second one is in line 5 and the third is in line 8. All are between ‘‘ `<DT> <A HREF="`’’ and ‘‘`http://`’’.

An instance of the negative examples is the separator between ‘‘ `<DT> <B>`’’ and ‘‘`John Tanner</B>`’’ because it marks the transition between b/e and N. All separators between the end of the previous data tuple and the beginning of attribute U such as the separa-

1.	Name:	**'Lithium' J Smith**
	E–Mail:	**aulmer@u.washington.edu**
	Last Update:	**08/01/95**
	Organization:	**University of Washington**
2.	Name:	**'Sir Brand' Gregrobin Smith**
	Alt. Name:	**Smith Gregrobin**
	E–Mail:	**sirbrand@u.washington.edu**
	Organization:	**university of washington**
	Last Update:	**06/21/96**
	Organization:	**University of Washington**
3.	Name:	**(raig Smith**
	E–Mail:	**chs@maxwell.cs.uoregon.edu**
	Last Update:	**08/01/94**
	Organization:	**University of Oregon**
4.	Name:	**– Richard Smith**
	Alt. Name:	**Richard**
	E–Mail:	**GBORDERS@SFASU.EDU**
	Last Update:	**11/12/95**
	Organization:	**Stephen F. Austin State University**
5.	Name:	**– David S Smith**
	Alt. Name:	**David S**
	E–Mail:	**dssmith@INDIANA.EDU**
	Last Update:	**11/16/95**
	Service Provider:	**Indiana University**

Figure 6: Example of a tagged-list page: Internet Address Finder. Notice the multiple occurrences of ``Organization'' in 2. and a typo in 3.

tor between `` <DT>'' and `` <A HREF=...'' are also instances of the negative examples, because they correspond to the reflexive edge of b/e.

The learned contextual rule for the edge between b/e and U may look like

TRANSFER FROM b/e **TO** U **IF**
 LEFT CONTEXT = ``HREF="'' **AND**
 RIGHT CONTEXT = ``http://''.

In this case, we use raw strings to encode a contextual rule. Contextual rules may also contain abstract token classes (or "wildcards"), like ⟨capitalized string⟩'', "*.html", etc. to cover a wide range of strings. □

5 Learnability Analysis

Our learnability analysis focuses on the sample complexity, that is, how many example data tuples are required to learn a perfect extractor. Generally speaking, the sample complexity depends on the complexity of the graph structure and the irregularities of the target documents.

Table 1 compares the complexity of the graph structures of a variety of SoftMealy extractors, assuming that the target documents have k attributes. The entries for single-pass extractors provide the figures in the simplest case where the extractor contains a single inter-state cycle. Such an extractor works when the target documents have only one attribute permutation. In the worst case, all attributes may be adjacent to one another and the complexity of the graph structure of a single-pass extractor is as large as a universal one.

The total number of edges in an extractor is a good indicator of the sample complexity because a large number of edges translates to a high probability to learn an incorrect rule. The number of outgoing edges of each state is also a good indicator because increasing this number amounts to increase the number of categories in a classification problem. Accordingly, learning a universal single-pass extractor may become increasingly difficult as the number of attributes grows, while the total number of edges for multi-pass extractors also grows with a large constant factor. Therefore, learning a single-pass extractor with the conservative approach, which creates an edge only when there exists an example, become appealing

However, in the worst case, the conservative approach requires at least an instance of each attribute permutation to learn a correct graph structure, while multi-pass extractors can deal with documents with an arbitrary number of attribute permutations inherently. [Hsu, 1998] derives a PAC upper bound of the minimal number of example data tuples required to select at least an instance for each attribute permutation with a high probability.

As a result, the tie breaker of single-pass and multi-pass extractors is a dichotomy of document layout styles based on their tendency of containing variant attribute permutations:

- **Tabular** Tabular documents are characterized by the use of ordering as a hint for readers to recognize the boundaries of data tuples and individual attributes. Typical cases of tabular documents are those arranging data in a table of rows and columns. An example of tabular documents is an output page from the Web site `Quote Server`[2]. This Web site returns a page of stock quotes in respond to a user request. Figure 5 shows an example page of this Web site.
- **Tagged-list** Tagged-list documents, in contrast, uses tags, such as an explicit ``Name:'' string, to point out the meaning of the following string. An example is the `Internet Address Finder` Web site[3]. Figure 6 shows an example page returned from this Web site.

Intuitively, learning a single-pass extractor for tabular documents may be easier because single-pass extractors can take advantage of the hint of attribute ordering. Consider learning an extractor for `Quote Server`, since most of the attributes are of the numeric type, the only hint to extract them is their relative positions in the

[2] http://www.scapl.com/cgi-bin/qs

[3] http://www.ifa.net

Extractor Type	Total Edges	Avg. outgoing edges per state	Max. outgoing edges per state	Example
Single-Pass	$> 4k+3$	$> \frac{4k+3}{2k+2}$	> 2	Figure 2
Multi-Pass	$10k+14$	$\frac{10k+14}{5k+8}$	3	Figure 3
Single-pass Universal	$5k+5$	$\frac{5k+5}{k+4}$	$k+1$	Figure 4
Single-pass Alternative	16	2.285	3	Figure 7

Table 1: Graph complexity of SoftMealy-based extractors, given k attributes

page. Since in this document collection, most attributes are of the numeric data type and a multi-pass extractor extracts one attribute in each pass of scanning, the only way to recognize a particular attribute is by counting how many columns it has read. This will become very difficult to generalize if there are occasional missing attributes.

On the other hand, multi-pass extractors may require fewer examples for tagged-list documents because the explicit tags allow such a document to contain a large number of attribute permutations without confusing its readers. Multi-pass extractors can use the tags as the hints to recognize boundaries of attributes in each pass of scanning, while single-pass extractors need to encode all the different attribute permutations and thus require more examples.

6 Alternative for Tagged-List Documents

We can significantly simplify a single-pass extractor for tagged-list documents by slightly changing the problem setting of information extraction. In a common problem specification, an information extractor is supposed to extract only the attribute values and skip all the other strings. In this alternative approach, we take advantage of the tags in tagged-list documents and reformulate the problem so that the goal is to extract sequences of *tag-and-value* pairs rather than just attribute values. A straightforward post-processing step can then convert tag-and-value pairs into desired data tuples of attribute values.

Example 6 (Alternative Information Extraction) The information extraction problem for the Web page shown in Figure 6 is usually defined as to extract data tuples of the six attributes and the correct output for the first data tuple is like:

```
Name = "'Lithium' J Smith"
E-Mail = "aulmer@u.washington.edu"
Last-Update = "08/01/95"
Organization = "University of Washington"
```

In the alternative problem specification, the correct output for the same data tuple is like:

```
Tag = "Name",
Value = "'Lithium' J Smith"
Tag = "E-Mail",
Value = "aulmer@u.washington.edu"
Tag = "Last Update",
Value = "08/01/95"
Tag = "Organization",
Value = "University of Washington"
```

□

The diagram of the single-pass extractor in this new problem setting is illustrated in Figure 7, and the last row of Table 1 gives its graph complexity factors. This alternative extractor has two critical advantages. First, its graph structure is fixed and applicable for **any** tagged-list documents. Second, its graph structure is simple, with small average and maximal outgoing edges per state. Therefore, learning this extractor can be considerably simpler than its counterparts for tagged-list documents, especially when the number of attributes is large or when the target documents contains many irregularities. Another advantage of the alternative extractor is that since it is single-pass, it can parse a document more efficiently than its multi-pass counterpart.

7 Empirical Result

This section reports our experiment to verify our hypotheses:

1. Single-pass extractors are more appropriate for tabular documents while multi-pass extractors are more appropriate for tagged-list.
2. For tagged-list documents, the alternative single-pass extractors that extract tag-and-value pairs can be significantly simpler to be learned than those extracting only attribute values.

We chose `Quote Server` and `Internet Address Finder` as the experimental document collections partly because they are exemplars of tabular documents and tagged-list documents, respectively, and partly because

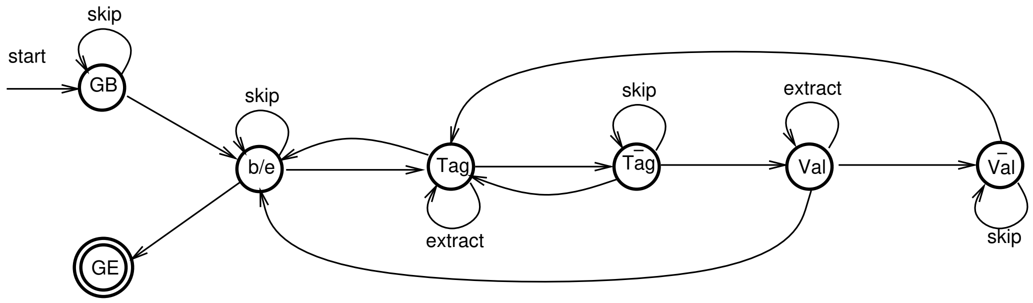


Figure 7: Alternative single-pass SoftMealy extractor for tagged-list documents

as reported in the literature [Kushmerick, Weld, & Doorenbos, 1997; Kushmerick, 1997; Muslea, Minton, & Knoblock, 1999], the extractors of these Web sites are particularly difficult to learn due to their frequent typos and irregularities. Table 2 summarizes the characteristics of these document collections.

The experiment is conducted as follows. For each type of extractors and each document collections, we run the test using one to ten data tuples as the training examples. Each run generates a learning curve of ten data points for each number of training data tuples. Each data point corresponds to the average of 300 hold-out set test accuracies. The hold-out set contains ten randomly selected documents for each document collection. The accuracy is the ratio of the number of correctly extracted tuples and the number of all tuples in the hold-out set. This quantity is equivalent to "recall" in information retrieval. An extracted data tuple is considered correct if and only if all attribute values are correctly extracted.

In each 300-trial test, the training data tuples are selected from different numbers of training pages. Suppose five training data tuples are to be selected. They can be selected from one to five different sample pages. Data tuples concentrating in a small number of pages may reveal quite different patterns from those spreading in a large number of pages. To minimize the impact of how training data tuples distribute in sample pages, the number of sample pages will be uniformly distributed for the 300 trials. That is, in the 300 trials of the test run with five training data tuples, in 60 trials the five training data tuples are selected from one randomly selected page (if such a page exists). In another 60 trials the tuples are selected from two pages, another 60 are from three, four, and five pages, respectively, to form a total of 300 trials. The same procedure applies to the test runs with one to ten training data tuples.

Figure 8 and 9 display the resulting learning curves. The result is consistent with our hypothesis. Multi-pass extractors perform moderately well for both collections, and single-pass extractors perform quite well for tabular collection `Quote Server` but poorly for tagged-list collection `Internet Address Finder`. Meanwhile, the alternative approach – by extracting tag-and-value pairs – significantly outperforms the other two extractor biases for `Internet Address Finder`. The result suggests

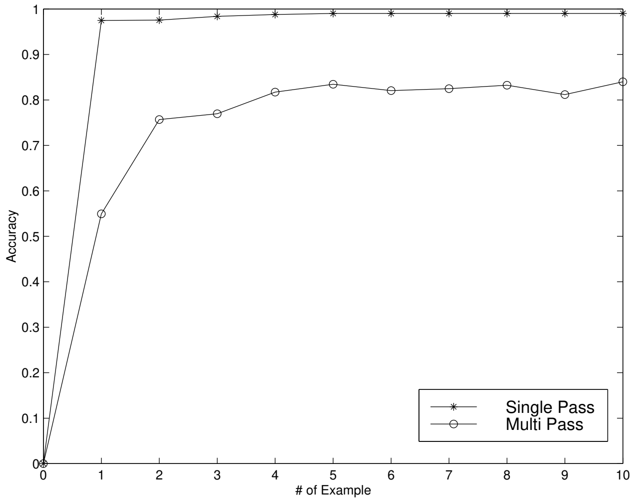


Figure 8: Learning curves for Quote Server

single-pass extractors as the bias for tabular documents and alternative single-pass for tagged-list documents.

8 Related Work in Wrapper Induction

[Kushmerick, Weld, & Doorenbos, 1997] coins the term "wrapper induction" and describes a machine learning approach called *"WIEN"*. In their work, a wrapper is essentially an extractor for semi-structured documents on the Web. Like in our work, their "wrappers" are induced from training examples of extraction. [Kushmerick, 1997] defines a family of wrapper classes that can be considered as a special case of the single-pass SoftMealy extractor that contains a single cycle starting and finishing in b/e. Figure 10 illustrates such an extractor for `Caltech CS Faculty` Web page. [Kushmerick, 1997] establishes a PAC bound on the sample complexity for WIEN wrappers. However, his representation is too restrictive to handle semi-structured documents with variant attribute permutations.

More recently, Muslea et al. proposed an approach called *"STALKER"* [Muslea, Minton, & Knoblock, 1999] that can learn wrappers more expressive than Kushmerick's work. Their wrappers are based on a set of "disjunctive landmark automata" organized as a hierarchy.

Source	Layout Style	# of Attributes	Sample Pages	Data tuples in Sample Pages
Quote Server	tabular	18	200	743
Internet Address Finder	tagged-list	6	20	180
Okra	tagged-list	6	250	2736
Big-Book	tabular	4	244	4099

Table 2: Document collections used in the empirical study

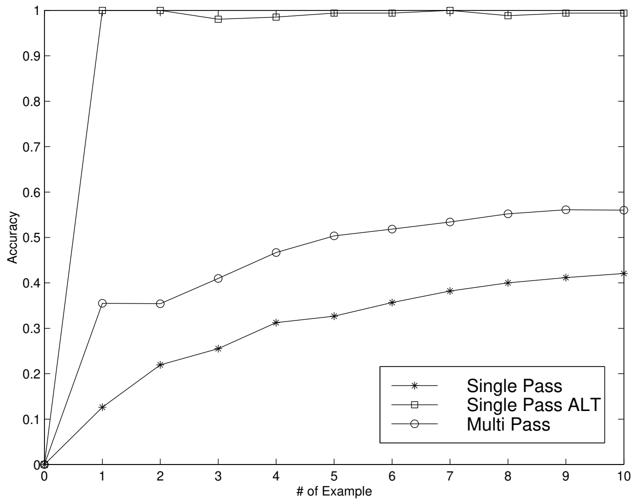


Figure 9: Learning curves for Internet Address Finder

To extract data, their extractor scans the document several times using the automata at the same level in the hierarchy. Each automaton is specialized in extracting one type of the objects. Their representation is similar to multi-pass SoftMealy extractors. In fact, their work inspired our study on the multi-pass extractors.

Both STALKER and WIEN have been tested on `Quote Server` and `Internet Address Finder` as well as the other two document collections: `Okra` and `Big-Book`. Extractors for `Okra` and `Big-Book` are relatively easy to learn because their layouts are regularly structured. Table 2 summarizes the characteristics of these document collections[4]. We now informally compare our results with the results reported in [Muslea, Minton, & Knoblock, 1999] and [Kushmerick, 1997].

- **Quote Server:**
  - STALKER: Given ten example data tuples, STALKER achieves a median of 79% accuracy in 500 trials, from a suite of ten sample pages. This tabular collection seemed particularly difficult for STALKER. [Muslea, Minton, & Knoblock, 1999] reports that one of the authors has tried to build a **handcrafted** extractor based on their representation and only to achieve 88% accuracy.
  - WIEN: This collection is beyond their learner's capability.
  - SoftMealy: As shown in Figure 8, our multi-pass learner achieves an average of 85% accuracy in 300 trials given ten example data tuples. The result appears to match the result achieved by the handcrafted extractor in STALKER, which is also based on a multi-pass representation. Figure 8 also shows that our single-pass extractor can achieve an average of 97% accuracy given **one** example data tuple. However, since the suite they used is too small, we collected more pages to form a total of 200 sample pages. The results may not be comparable in the rigorous sense.
- **Internet Address Finder:**
  - STALKER: Given ten example data tuples, STALKER achieves median accuracies ranging between 85% and 100% in 500 trials, with a suite of ten sample pages. However, according to their Web page[5], their result was obtained by omitting multiple occurrences of attribute `Organization` (see Figure 6).
  - WIEN: This collection is beyond their learner's capability.
  - SoftMealy: Figure 9 shows that given ten example data tuples in 300 trials, our multi-pass learner achieves an average of 57% accuracy, the single-pass learner achieves an average of 41% accuracy and **99%** with the alternative approach. Note that, in all trials, our learner and extractors consider multiple occurrences of `Organization`.
- **Okra:**
  - STALKER achieves 97% median accuracy given one example.
  - WIEN achieves 100% accuracy using an average of 3.5 example **pages**, which amounts to approximately 46 example data tuples, in 30 trials.

[4] Sample pages for experimentation can be obtained from `http://www.isi.edu/~muslea/RISE/`

[5] `http://www.isi.edu/~muslea/RISE/wIAF/Model.html`

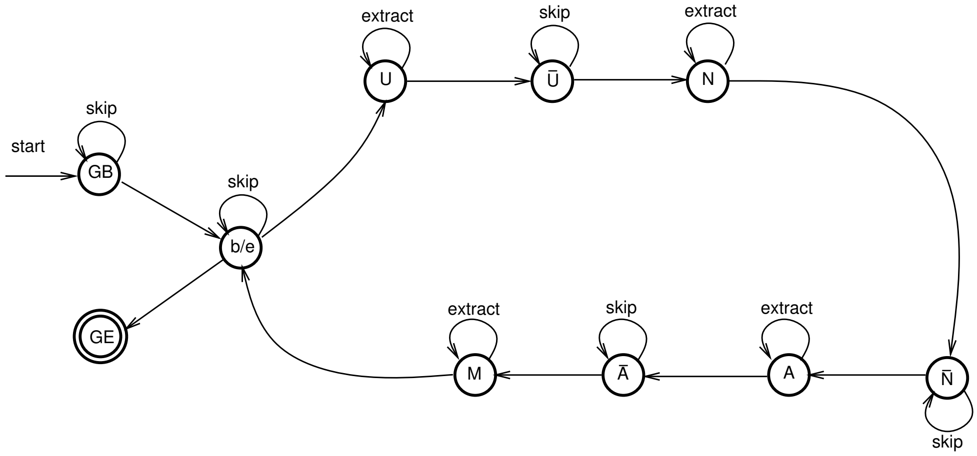


Figure 10: WIEN family-equivalent SoftMealy Extractor

  - SoftMealy: Our single-pass learner achieves 100% accuracy given **one** example in all 30 trials.

- **Big-book:**
  - STALKER achieves a median of 97% accuracy given eight examples.
  - WIEN learns perfect extractors using an average of 15 example pages, which amounts to approximately 274 data tuples, in 30 trials.
  - SoftMealy: Our single-pass learner achieves an average of 97% accuracy given four examples and an average of 100% accuracy given six examples, both in 30 trials.

9 Conclusion and Future Work

We have presented SoftMealy machines as the representation formalism for learning extractors. We have described single-pass and multi-pass extractors based on SoftMealy machines and briefly explained how to learn these extractors. Our analysis shows that both classes of the extractors are sufficiently expressive to handle documents with irregular attribute permutations, but a single-pass extractor can parse a document more efficiently. In terms of their learnability, we hypothesized that a single-pass extractor is a better bias for tabular documents while a multi-pass extractor is more suitable for tagged-list documents. Our hypothesis is consistent with the experimental result and the comparison with the previous work also provides an evidence. Another important contribution of this work is our alternative approach for tagged-list documents. By extracting tag-and-value pairs instead of only attribute values, a single-pass extractor can significantly outperform a multi-pass extractor for the tagged-list documents in our experiment.

Many factors of the sample complexity deserve further investigation. One of the factors is the representation of contextual rules. The design of abstract token classes may also play an important role. Another factor is the distribution of the training data tuples in the sample pages. Our impression is that training data tuples from a small number of pages tend to yield worse performance than the same number of tuples distributing in a large number of pages. This is the case especially for documents that require a large number of examples. We plan to verify this assumption more rigorously soon.

In conclusion, we have presented a machine learning approach to generating extractors for tabular and tagged-list documents and demonstrated that a small number of training examples is sufficient to learn perfect extractors. We therefore propose this approach as a practical tool for text mining.

Acknowledgements

We wish to thank Chia-Che Ma and Ya-Ting Yang for their help in preparing the experimental result data.

References

[Angluin, 1987] Angluin, D. 1987. Learning regular sets from queries and counterexamples. *Information and Computation* 75:87–106.

[Biermann & Feldman, 1972] Biermann, A., and Feldman, J. 1972. On the synthesis of finite state machines from samples of their behavior. *IEEE Transactions on Computers* C-21:592–597.

[DARPA, 1995] DARPA. 1995. *Proceedings of the sixth message understanding conference (MUC-6).* Defense Advanced Research Projects Agency, Morgan Kaufmann Publisher Inc.

[Haussler, 1988] Haussler, D. 1988. Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. *Artificial Intelligence* 36:177–221.

[Hopcroft & Ullman, 1979] Hopcroft, J. E., and Ullman, J. D. 1979. *Introduction to Automata Theory, Languages, and Computation.* Reading, Massachusetts: Addison-Wesley.

[Hsu & Dung, 1998] Hsu, C.-N., and Dung, M.-T. 1998. Generating finite-state transducers for semistructured data extraction from the web. *Information Systems* 23(8). Special Issue on Semistructured Data.

[Hsu, 1998] Hsu, C.-N. 1998. Initial results on wrapping semistructured web pages with finite-state transducers and contextual rules. In *Proceedings of AAAI-98 Workshop on AI and Information Integration, Technical Report WS-98-01.* Menlo Park, CA: AAAI Press.

[Kushmerick, Weld, & Doorenbos, 1997] Kushmerick, N.; Weld, D.; and Doorenbos, R. 1997. Wrapper induction for information extraction. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97).*

[Kushmerick, 1997] Kushmerick, N. 1997. *Wrapper Induction for Information Extraction.* Ph.D. Dissertation, Department of Computer Science and Engineering, University of Washington, Seatle, WA.

[Muslea, Minton, & Knoblock, 1999] Muslea, I.; Minton, S.; and Knoblock, C. 1999. A hierarchical approach to wrapper induction. In *Proceedings of the Third International Conference on Autonomous Agents (Agents'99), Seattle, WA.*