Extracting Data From Web Sources

Giansalvatore Mecca
Università della Basilicata
mecca@unibas.it
http://www.difa.unibas.it/users/gmecca/

EDBT Summer School 2002
August, 26 2002

Outline

ο Subject of this talk
  ⇒ *Information extraction from the Web*

ο Scenario
  ⇒ the Web is probably the largest “knowledge base” ever developed
  ⇒ information conceived to be consumed ("browsed") by humans
  ⇒ the goal is to make this information available to computer applications ("agents")
A general description of a software agent’s tasks
1. receive a goal
2. locate relevant sources on the Web
3. gather information from these sources
4. process this content
5. deliver results to the user

In this talk
⇒ we concentrate exclusively on task 3

Task carried on by “wrappers”
Outline

Part I: Information Extraction from HTML
- wrapper inference techniques
- limitations

Part II: Grammar Inference
- Gold's theorem and identification in the limit
- algorithms from grammar inference
- limitations

Part III: Bridging the Gap
- the RoadRunner Project

Outline

Part I: Information Extraction from HTML

- What's in a wrapper
- Recent research on semi-automatic wrapper generation techniques
  - machine learning approaches
  - wrapper induction
- Limitations of these approaches
  - related to supervised learning
  - wrappers tend to be brittle
  - the maintenance problem
Outline
Part II: Grammar Inference

- Wrappers are essentially grammar parsers
  - wrapper inference is strongly related to grammar inference
- A jump backward: 30 years of grammar inference
  - accomplishments of the grammar inference community
  - why grammar inference techniques are not applicable to information extraction

Outline
Part III: Bridging the Gap

- The Schema Finding Problem
  - a variant of the traditional grammar inference problem for information extraction
- Markup Grammars
  - a class of grammars that can be practically inferred with unsupervised algorithms
- Limitations and opportunities
  - future research directions
Part I
Information Extraction from HTML

Information Extraction Task

- Information extraction task
  - source format: plain text with HTML markup (no semantics)
  - target format: database table or XML file (adding structure, i.e., semantics)
  - extraction step: parse the HTML and return data items in the target format

- Wrapper
  - piece of software for the extraction step
  - intuition: use extraction rules based on HTML markup
Intuition Behind Extraction

**Italian Football Teams**

- Atalanta - Bergamo
- Inter - Milano
- Juventus - Torino
- Milan - Milano
- ... - ...

**Wrapper Procedure:**

Scan document for `<ul>`

While there are more occurrences

- scan until `<b>`
  - extract `teamName` between `<b>` and `<b>`
- scan until `<i>`
  - extract `town` between `<i>` and `<i>`

Output `[teamName, town]`

A Complex Extraction Task

>20-30KB of HTML

>10 attributes with nesting
Why Information Extraction is Relevant?

Wrappers are largely used today
- shopping agents (e.g., Jango, BargainFinder, ShopBot ...)
- integration agents (e.g., molecular biology)
- personal information brokers

In these applications, wrappers are usually written by hand
- e.g., Excite’s shopping agent, Jango, relies on several hundred wrappers [Kushmerick, 2000]

Wrapper Development

There are several tools that can be used to support the wrapper development process
- both research and commercial tools
  - for a list see: www.wifo.uni-mannheim.de/~kuhlns/wrappertools/

Using these tools it is usually possible to rapid prototype a wrapper very quickly
- example: Lixto [Baumgartner et al, VLDB 2001], a GUI-based wrapper generation toolkit

Problem: wrappers tend to be brittle
Problem: Brittlness

- Web sites change quite frequently
- Any change in the HTML layout may disrupt the extraction rules
  - Jango’s wrapper mean time to failure was approximately one month [Kushmerick, 2000]
- Maintaining a wrapper is very costly
  - this has motivated the study of (semi)-automatic wrapper generation techniques

A Note on the Semantic Web

- It is a common opinion that neither XML nor the Semantic Web initiative will represent for now a solution to the information extraction problem
- Reason I
  - the Web is a giant “legacy system”
  - many sites will never move to the new technology
- Reason II
  - many organizations will not be willing to expose their data in XML
Semi-Automatic Wrapper Generation

- Many proposals, from different inspirations
- A rough classification
  - grammar-based (“finite-state”) machine learning approaches
  - relational machine learning approaches
  - ontology based approaches [Embley et al, 98]
- We discuss only finite-state approaches
  - see the references for a complete list
  - we also don’t cover information extraction from free text (a related but quite different problem)

Finite-State Approaches

- Typical learning system
  - a supervisor provides a set of training samples (i.e., labeled HTML pages)
  - a learning algorithm is used to learn the extraction rules
  - a wrapper is generated based on the extraction rules
  - the wrapper is used on the target pages
- This process is called “wrapper induction”
Wrapper Induction

- Seminal work by Kushmerick et al
  - [Kushmerick et al, 1997] [Kushmerick, 2000]
- Six wrapper classes
  - increasing expressibility
  - decreasing efficiency in the learning
- The simplest class: LR (left-right wrappers)
  - targeted at pages containing multiple records
  - one extraction rule per field
  - each extraction rule is a pair of delimiting strings (left and right)

LR-wrappers: Example

```html
<html>
<body>
<h1>Italian Football Teams</h1>
<ul>
<li><b>Atalanta</b> - <i>Bergamo</i></li>
<li><b>Inter</b> - <i>Milano</i></li>
<li><b>Juventus</b> - <i>Torino</i></li>
<li><b>Milan</b> - <i>Milano</i></li>
</ul>
</body>
</html>
```

```
LR-wrapper = { 
[b, ] [i, /i] 
}
delimiters for field teamName
delimiters for field town

Wrapper Procedure:
While there are more occurrences
scan until <b>
extract teamName between <b> and </b>
and </b>
scan until <i>
extract town between <i> and </i>
output [teamName, town]
```
Wrapper Induction

Note the connection with grammars

- the extraction rules define a regular grammar with labeled nonterminals
- wrapper: parser for the grammar + knowledge about the target structure (set of records)

\[ LR\text{-}wrapper = \{ \{<b>,</b>\} \} \]

\( \Sigma^* \)

\((<b>teamName</b> \Sigma^* <i>town</i> )^* \] \( \Sigma^* \) : any string

output: records of the form \((teamName, town)\)

Wrapper Induction

In order to induce the wrapper, the first step is to label the training data

A human inspects some sample pages and manually labels the fields

```html
<html><body>
<h1>Italian Football Teams</h1>
<ul>
<li><b>teamName</b> - <i>town</i></li>
<li><b>teamName</b> - <i>town</i></li>
<li><b>teamName</b> - <i>town</i></li>
<li><b>teamName</b> - <i>town</i></li>
</ul>
</body></html>
```
Wrapper Induction

- Then a learning algorithm is run on the sample pages to find the delimiters
- The algorithm is quite simple
  - it enumerates over potential values for each delimiter selecting the first such that the wrapper works correctly on the training data
  - the learning is efficient because delimiters can be learned independently
  - quadratic time

- Obviously, not all pages can be wrapped using LR-wrappers
  - in [Kushmerick, 2000] it is reported that LR-wrappers were able to handle 53% of the sites in a survey conducted by the author

```html
<html>
  <body>
    <b>Italian Football Teams</b>
    <ul>
      <li><b>Atalanta</b> - <i>Bergamo</i></li>
      <li><b>Inter</b> - <i>Milano</i></li>
      <li><b>Juventus</b> - <i>Torino</i></li>
      <li><b>Milan</b> - <i>Milano</i></li>
    </ul>
  </body>
</html>
Wrapper Induction

To handle these cases, Kushmerick introduces more complex classes of wrappers

Example: HLRT (head-left-right-tail wrappers)

- two additional delimiters, h and t
- h is used to skip potentially confusing text in the head page
- t is used to skip potentially confusing text in the tail of the page

HLRT-wrappers: Example

```
<html><body>
<b>Italian Football Teams</b>
<ul>
<li><b>Atalanta</b> - <i>Bergamo</i>
<br>
</li>
<li><b>Inter</b> - <i>Milano</i>
<br>
</li>
<li><b>Juventus</b> - <i>Torino</i>
<br>
</li>
<li><b>Milan</b> - <i>Milano</i>
<br>
</li>
</ul>
</body></html>
```

HLRT-wrappers

{  [ul], </html>
[<b>, </b>
[<i>, /i>]  }

Wrapper Procedure:
Scan document for <ul>
While (there are more occurrences of <b> before </html>)
scan until <b>
extract teamName between <b> and </b>
scan until <i>
extract town between <i> and </i>
output [teamName, town]
Wrapper Induction

Six classes of wrappers

- LR (left-right)
- HLRT (head-left-right-tail)
- OCLR (open-close-left-right): open and close delimiters identify each record in the document
- HOCLRT (head-close-left-right-tail)
- NLR (nested LR): to handle nested tabular data
- NHLRT (nested HLRT)

Tradeoff between expressibility and complexity of the learning

- the first classes can be learned in polynomial time wrt to the length of the training documents
- in last two classes the learning is unfeasible (exponential in the size of the training documents)

Overall

- the six classes were able to handle 70% of the sites selected by the author in his survey
Extended Wrapper Induction Techniques

- Subsequent works have extended Kushmerick’s work in several respects
  - missing attributes
  - multiple orderings
  - disjunctive delimiters
  - arbitrary nesting
- Stalker [Muslea et al, 1999]
- SoftMealy [Hsu, Dung, 98]
- NoDoSE [Adelberg, 1998]
- Can wrap virtually any form of HTML layout

Limitations of Supervised Techniques

- If the site changes, it is necessary to maintain the wrapper
- Maintenance Problem I: verification
  - it is necessary to detect when the wrapper stops to work properly
  - delimiter-based wrappers do not always allow to detect changes in the site
- Maintenance Problem II: re-induction
  - in general, a new user intervention is necessary in order to learn the new wrapper
Wrapper Verification Example

```
<html><body>
<h3>Italian Football Teams</h3>
<table>
<tr>  <td>Atalanta</td>  <td>Bergamo</td>  </tr>
<tr>  <td>Inter</td>        <td>Milano</td>  </tr>
<tr>  <td>Juventus</td> <td>Torino</td>  </tr>
<tr>  <td>Milan</td>       <td>Milano</td>  </tr>
</table>
<b>Last modified</b>: <i>Aug. 2002</i>
</body></html>
```

The reason for this is that the grammar specified by the delimiters is not very strict, and allows a wide degree of variations.

Wrapper Maintenance

- There has been some work on wrapper maintenance:
  - [Kushmerick, WWWJ 2000]: regression testing to detect wrapper disruption
  - [Lerman, Minton, 2000]: data extracted when the wrapper was working properly can help to automatically relabel the new samples
- However:
  - These results are quite preliminary
Wrapper Maintenance

Ideal Goals

Ideally, we would like to achieve two goals:

- First, the wrapper should give a tighter description of the grammatical structure of the page:
  - this would simplify wrapper verification.
- Second, the wrapper should be learnable in an unsupervised way (fully automatically, without the need for training examples):
  - this would simplify wrapper reinduction.

Wrapper Maintenance

In this way, wrapper induction becomes a typical grammar inference problem:

- given a set of samples, automatically infer the minimal grammar to which they belong.

Grammar inference is a well established research field, with a rich literature:

- nice theoretical framework
- unsupervised algorithms

Why not grammar inference for information extraction?
Part II
Grammar Inference

Grammar Inference

Grammar Inference is a 30-years old subject (early studies in the sixties)
⇒ focus on computational theory, and not on information extraction

Preliminaries:
⇒ let's fix a finite alphabet of symbols \( \Sigma \)
⇒ a language over \( \Sigma \) is any collection of strings over \( \Sigma \)
⇒ a class of languages over \( \Sigma \) is any collection of languages over \( \Sigma \)
Grammar Inference

The problem
- given a collection of sample strings, find the language they belong to

The computational model: identification in the limit [Gold, 1967]
- the inference system is presented with a larger and larger corpus of examples
- it simultaneously makes a sequence of guesses about the language to infer
- if the sequence eventually converges, the inference is correct

Identification in the Limit
Gold’s Theorem

- Any class of languages over $\Sigma$ that contains all finite languages over $\Sigma$ and at least one infinite language cannot be identified in the limit from positive samples only [Gold, 1967]
- This means, for example, that regular grammars cannot be identified in the limit from positive samples only

Gold’s Theorem: Intuition

- Consider a set of samples of the form $a, ab, abb, abbb, abbbb, \ldots$
  - the learner will never be able to definitely tell whether the target language is $a \cup ab \cup abb \cup abbb \cup \ldots$ or $ab^*$
- Hint: the problem is related to disjunction
  - but r.e. without $\cup$ are not identifiable either
- Research directions
  - sub-classes that are identifiable in the limit
  - different computational model (e.g., positive and negative examples or additional information)
Restricted Classes of Regular Expressions

Dana Angluin has characterized the classes of languages identifiable in the limit

A class of languages $L$ is identifiable in the limit iff each language of the class admits a characteristic sample [Angluin, 1980]

Characteristic sample

A finite set $T \subseteq L \in L$ is said a characteristic sample of $L$ in $L$ if $L$ is the smallest language from $L$ containing $T$

Restricted Classes of Regular Expressions

Several subclasses of regular expressions have been proven to identifiable in the limit

- reversible languages [Angluin, 1982]
- terminal-distinguishable languages [Radakrishnan and Nagaraja, 1987]
- testable languages [Garcia and Vidal, 1990]

The definition of these classes is quite involved
f-Distinguishable Languages

- Fernau has recently generalized many previously known classes of identifiable languages under the notion of f-distinguishable languages.

- Class of f-Distinguishable Languages:
  - class of regular languages identified by a distinguishing function [Fernau, 2000].

- Distinguishing function:
  - $f: \Sigma^* \rightarrow F$ (with $F$ finite) such that
  - $f(w) = f(z) \Rightarrow f(wu) = f(zu)$, for all $w, z, u \in \Sigma^*$.

- All these classes of languages are identifiable in the limit (polynomial algorithm).

k-Reversible Languages

- 0-reversible regular language $L$:
  - Consider the automaton $A$ associated with $L$.
  - Consider the reversed automaton $A'$ of $A$ obtained by exchanging initial and final states and reversing each transition edge.
  - $L$ is 0-reversible if both $A$ and $A'$ are deterministic.

- k-reversible regular language $L$:
  - $A$ is deterministic.
  - $A'$ is deterministic with “lookahead” $k$. 
k-Reversible Languages

- Characteristic sample for a k-reversible language (intuition)
  - a set of strings that touch each state of A, and exercise each transition of A
  - the strings need to have minimum length

- Formally, given $A=(Q, \Sigma, \delta, q_0, Q_F)$
  
  $\chi = \{ u(q)v(q) \mid q \in Q \}$
  
  $\cup \{ u(q)av(\delta(q,a)) \mid q \in Q, a \in \Sigma \}$

  where $u(q)$ and $v(q)$ are words of minimal length such that: $\delta^*(q_0, u(q)) = q$ and $\delta^*(q, v(q)) \in Q_F$

k-Reversible Languages: Example

- Professors and their courses
  - name, email (optional), one or more courses

```html
<html><body>
<h1>John Doe</h1> email: <b>doe@dot.edu</b>
<p>Courses:<br> Distributed Systems</p>
</body></html>
```

```html
<html><body>
<h1>John Doe</h1>
<p>Courses:<br> Databases<br> Operating Systems</p>
</body></html>
```
k-Reversible Languages: Example

- This is a 1-reversible language

For each fixed k, there is a polynomial time algorithm that correctly infers a k-reversible language from positive samples.

The algorithm is completely unsupervised
  \( \Rightarrow \) automata theoretic, state-merging techniques

Please note:
  \( \Rightarrow \) the algorithm requires the knowledge of k
  \( \Rightarrow \) to identify the language, the input must contain a characteristic sample (otherwise, the grammar is not correctly inferred)
k-Reversible Languages

For example, the samples below cannot be used to identify the language although they touch each state of A and exercise all transitions.

Why Not Grammar Inference for IE?

Grammar inference techniques might in principle solve the wrapper maintenance problem:
- tight grammars
- unsupervised algorithms

However, virtually none of the recent approaches to information extraction from HTML is based on grammar inference techniques.

There are several reasons for this.
Why Not Grammar Inference for IE?

Reason I: Assumptions on the input to the inference algorithm
- availability of a characteristic sample
The minimal length requirement makes highly unlikely its availability
- a simple probabilistic argument shows that in practice there is a very low probability to find a characteristic sample in a bunch of randomly selected HTML pages

Reason II: The grammar by itself is not a wrapper
- Recall that a wrapper is
  - a grammar
  - plus labeled terminals (attribute names)
  - plus a target structure for the output
- In essence, the wrapper needs to know
  - how to output strings that have been parsed
Why Not Grammar Inference for IE?

Ideally, grammar inference algorithms would be useful if we could find a class of grammars that:
- are natural with respect to HTML pages
- have a more practical notion of characteristic sample
- are such that the target structure can be identified easily by looking at the grammar itself

Part III
Bridging the Gap
The RoadRunner Approach

- **RoadRunner**
  - [Crescenzi, Mecca, Merialdo 2001] a research project on information extraction from Web sites

- **The goal**
  - developing fully automatic techniques for information extraction from Web sites

- **Contributions**
  - it bridges the gap between wrapper induction and traditional grammar inference techniques

---

The RoadRunner Approach

- **The target**
  - large Web sites with HTML pages generated by programs (scripts) based on the content of an underlying data store
  - the data store is usually a relational database

- **The typical page-generation process**
  - data are extracted from the relational tables and possibly joined and nested
  - the resulting dataset is exported in HTML format by attaching tags to values
Example: A Fictional Bookstore

<table>
<thead>
<tr>
<th>Name</th>
<th>Books</th>
<th>Editions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Title</td>
<td>Description</td>
</tr>
<tr>
<td>John Smith</td>
<td>Database Primer</td>
<td>No book ...</td>
</tr>
<tr>
<td>Paul Jones</td>
<td>HTML and Script</td>
<td>JavaScript</td>
</tr>
<tr>
<td></td>
<td></td>
<td>first Edition, Paperback</td>
</tr>
</tbody>
</table>

The Schema Finding Problem

- **Page class**
  - collection of pages generated by the same script from a common dataset
  - these pages typically share a common structure

- **Schema finding problem**
  - given a set of sample HTML pages belonging to the same class, automatically recover the source dataset

- **Solution: Wrapper**
  - structure of the underlying dataset
  - extraction rules
The Schema Finding Problem

- **In essence**
  - we are not making any assumptions on the target structure (which can be arbitrarily nested)
  - we are not relying on user inputs (the algorithm should be unsupervised)

- **This is a grammar inference problem**
  - but we want the data source schema, not only the grammar
  - we want a more natural notion of characteristic sample
The RoadRunner Approach

- Nested Types
  - nested relation types with optional attributes

- Source Dataset
  - instance of a nested type

- Target class of grammars
  - markup-grammars
  - intuition: languages obtained by serializing a dataset with HTML markup

Nested Relations with Optionals

Schema:  

- Tuple
  - Name
  - Optional Email
  - Course

Smith's instance:

- Tuple
  - Frank Smith
  - Optional null
  - Set

Source Dataset

<table>
<thead>
<tr>
<th>Name</th>
<th>Email</th>
<th>Courses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frank Smith</td>
<td>null</td>
<td>Operating Systems Databases</td>
</tr>
<tr>
<td>John Doe</td>
<td><a href="mailto:doe@dot.edu">doe@dot.edu</a></td>
<td>Distributed Systems</td>
</tr>
<tr>
<td>Paul Jones</td>
<td>null</td>
<td>Advanced Database Systems</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Serializing Instance Mark-Up Encodings

Tagged Schema:  

```
<HTML><BODY>
  <H1> Name </H1>  
  <P> Optional </P>  
  <UL> Set </UL>  
    <LI> Email </LI>  
    <LI> Tuple </LI>  
</BODY></HTML>
```

Smith’s Instance Encoding:

```
<HTML><BODY>
  <H1> #PCDATA </H1>  
  <P> ( <B> #PCDATA </B> )? </P>  
  <UL> ( <LI> #PCDATA </LI> )* </UL>  
</BODY></HTML>
```

Mark-up Language:

```
<HTML><BODY>
  <H1> </H1>  
  <P> </P>  
  <B> </B>  
  <UL> </UL>  
  <LI> </LI>  
</BODY></HTML>
```

Prefix Markup-Encodings

- **Mark-up grammar**
  - any regular language generated by a markup-encoding of a nested type

- **Prefix markup grammar (intuition)**
  - markup grammar such that the resulting language is LL(1) in both directions

- **In essence**
  - we have identified a subclass of union-free regular expressions
  - this class has a number of nice properties
Prefix Markup-Encodings

Property I: There is a straightforward mapping from grammars to database types
  - concatenation in the grammar correspond to database tuples
  - each + in the grammar corresponds to a database set
  - each ? in the grammar corresponds to an optional

In fact
  - given a mark-up language, it is possible to recover the underlying type in linear time

Prefix Markup-Encodings

Property II: It is identifiable in the limit
  - note that regular expressions with . and * only are not identifiable in the limit

In fact
  - it is possible to prove that is a subclass of a (slight generalization of) a class of f-distinguishable languages [Fernau, 2000]
  - therefore each language has a characteristic sample
Prefix Markup-Encodings

Property III: It has a very natural notion of characteristic sample
  ⇒ we can drop the minimality requirement

More specifically
  ⇒ all we need is a rich set of instances

Rich set of instances
  ⇒ intuitively: a set of instances that makes “full use” of the underlying type

Rich set of instances
  ⇒ basic richness: each leaf node has at least two distinct values
  ⇒ set richness: each set node has instances of distinct cardinalities
  ⇒ optional richness: for each optional node has at least one null and one not null instance

It is possible to prove that each rich set is a characteristic sample for the corresponding prefix markup-language
A simple probabilistic argument shows that in practice

- the probability of finding a rich set in a random sample is quite high

Consider a type with k set types

- call p the probability that 2 pages have all instances of a set type of the same cardinality
- the probability to find 2 distinct cardinalities in n random pages is \((1-p^{n-1})\)
- the probability that the n samples are set rich is \((1-p^{n-1})^k\) – if k=5, n=10, p=50% then 99%
Contributions of RoadRunner

- We have developed an unsupervised learning algorithm for identifying prefix-markup languages in the limit
  - given a rich-sample of HTML pages, the algorithm correctly identifies the language, i.e., the wrapper
  - the algorithm runs in polynomial time

Note
- with respect to f-distinguishable languages, the algorithm is significantly different due to the different notion of characteristic sample

Prefix-Markup Languages

- This seems to represent a reasonable compromise between two aspects
- Expressibility of the grammar formalism
  - we have been able to wrap many real-world sites like amazon.com, cnn.com, ebay.com etc.
- Effectiveness of the wrapper induction task
  - it is possible to fully automate the step
  - there are reasonable assumptions on the samples to inspect
Open Problems

- The algorithm needs multiple pages
  - as any grammar inference algorithm
  - how to wrap single pages?
- The algorithm is not capable of inferring field names
  - these are anonymous in the target wrapper
  - some heuristics, but preliminary work
- Disjunctions are necessary in some cases
  - this makes the learning much more complex
  - a form of “non-disruptive” disjunction?

References
References

Wrapper Induction - Finite State Techniques

- B. Adelberg. *NoDoSE - a tool for semi-automatically extracting structured and semistructured data from text documents*. In *SIGMOD’98*.

References

Wrapper Induction - Other Approaches

References

Grammar Inference


References

RoadRunner