Undergraduate Research Opportunity Program
(UROP) Project Report

User-centric Web Information Extraction

By

Shawn Tan

Department of Computer Science

School of Computing

National University of Singapore

2010/11

Undergraduate Research Opportunity Program
(UROP) Project Report

User-centric Web Information Extraction

By

Shawn Tan

Department of Computer Science

School of Computing

National University of Singapore

2010/11

Project No: U096883L
Advisor: A/P Kan Min-Yen
Deliverables:
    Report: 1 Volume
    Source Code: 1 DVD

Abstract

In this report, I present an algorithm providing an iterative method for users to generate a wrapper visually from a given web page, and also a machine learning solution for extraction of data after a site layout change. These were implemented as a system, and then evaluated both quantitatively and qualitatively.

Subject Descriptors:
H3.5 Web-based Services
H2.8 Data Mining
I5.2 Pattern Analysis

Keywords:
bookmarklet, XPath, alignment, information extraction

Implementation Software and Hardware:
Java, Ruby 1.9.2, Javascript

Acknowledgement

I would like to extend my gratitude to my supervisor, A/P Kan Min-Yen, for his patience when explaining certain concepts to me, and guidance throughout the project. I am grateful for this opportunity to work on this UROP project under his care. I would also like to thank the entire WING group, for their valuable advice, especially Jesse Prabawa, for answering so many of my questions.

List of Figures

List of Tables

Table of Contents

Chapter 1

Introduction

Information extraction deals with extracting data from input documents into a structured form. In contrast, information retrieval is concerned with retrieving relevant documents from a collection of documents. Traditionally, the task of information extraction is usually performed on unstructured free text and as such, makes use of certain Natural Language Processing techniques. Web IE however, is concerned with documents extracted from the web. Many of these documents today are in HTML, which are semi-structured, and increasingly, generated by server scripts and template-based. Thus, web information extraction is slightly different from information extraction from free text. In IE, procedures that extract certain information from structured or semi-structured documents like HTML are known as *wrappers*.

1.1 Motivation

While many systems for web information extraction have been developed over the years, many of these are used in corporate settings, and generally extract huge amounts of data from the web into a form more suited for search and retrieval. While this may be within the reach of corporations, a user who browses the web for leisure does not have access to such resources. Another drawback of these systems is that the process of getting them up and running usually involve many hours of labelling and training work. As a result, these systems are inaccessible to users who do not have the time to do this.

	RSS Feeds	Screen Scrapers
Availability	Lies with content provider	Anything that is displayed can be scraped
Content Extracted	Lies with content provider	Lies with user
Affected by Layout	Content provider provides content, no layout involved	Breaks on layout change
Format	Still requires parsing of RSS XML in order to maipulate data.	Can extract into any format
Technical Knowledge required	User has only to know how to use RSS.	Requires programming experience

Table 1.1: Pros & Cons of using RSS or wrappers (screen-scrapers)

One might argue that this is where RSS feeds come in, to provide users updates to constantly changing data on their pages. However, the data provided on RSS is usually controlled by the site in question, and may not provide the data that the user actually needs. One other alternative to tackle this problem would be to write "screen scrapers", as this would provide complete control over the data extracted. This approach faces another set of problems, the most glaring being that most users are not familiar with programming, let alone the many other technical issues faced when doing screen scraping.

As such, we see a gap here which needs to be filled by bringing web IE closer to the average user by making creation of wrappers more user friendly, and at the same time, creating wrappers that are robust, and resistant to layout changes. In essence, a "super scraper" for user-centric web information extraction.

1.2 Goals & Challenges

In this report, we present a system that attempts to solve some of these problems by doing the following:

1. Provide an intuitive interface for labelling that is platform agnostic, and takes advantage of the many advancements in Javascript. This will provide the user with an immediate visual feedback as to the items that he/she will be extracting, and at the same time reduce the amount of labelling that needs to be done.

2. Create a more robust framework for extraction of the selected information using machine

learning. The classifier will have less focus on the HTML structural information of the tags in order to be resistant to any layout changes made to the page.

1.3 System Overview

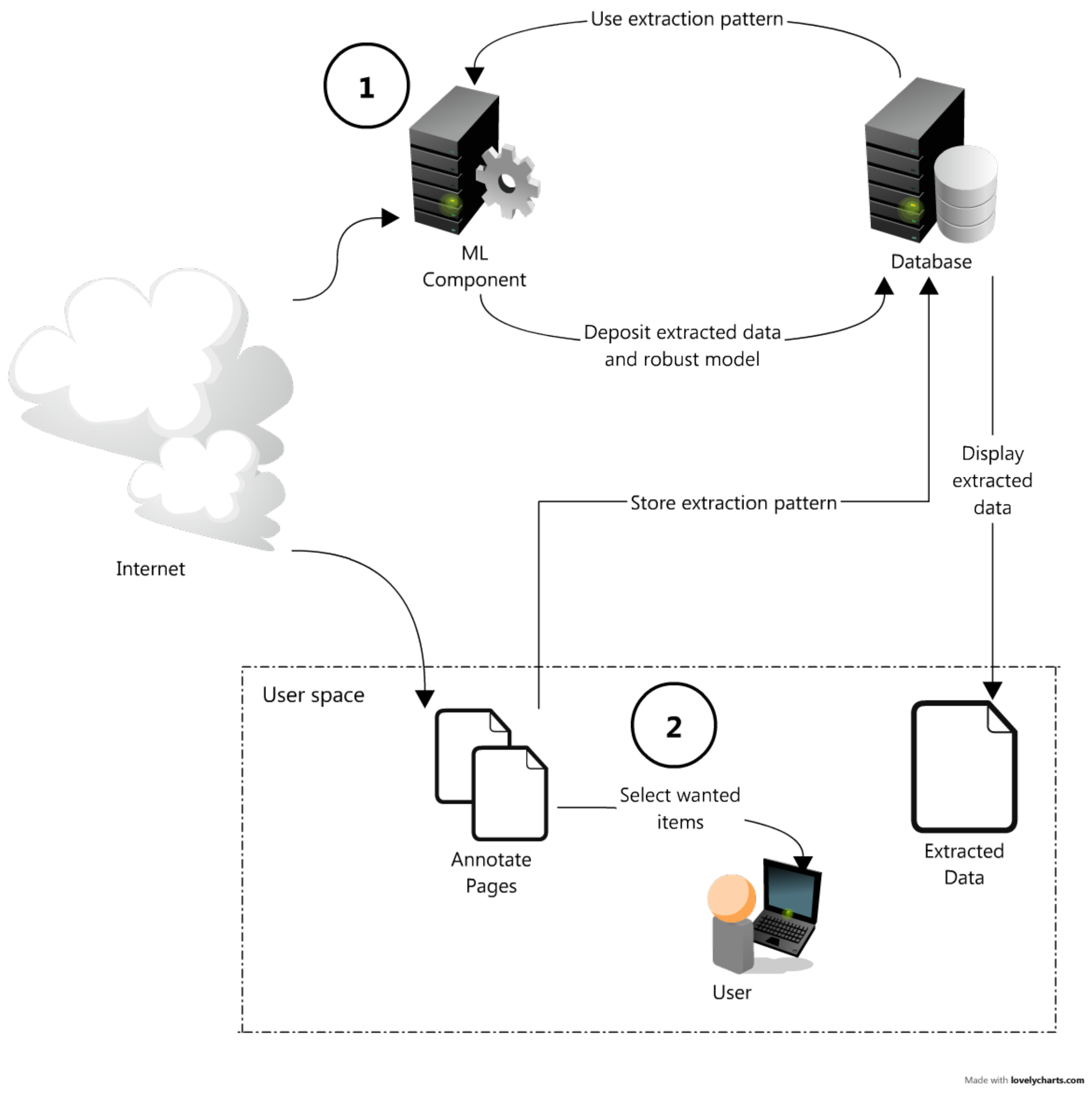


Figure 1.1: Overview of the system

Figure 1.1 shows the workflow of the system presented in this report. The section labelled 1 will be described in detail in Chapter **??**, and the section labelled 2, which consists of the bookmarklet and user interface, will be described in Chapter 4.1. Both chapters will elaborate on related work, methodology, the sub-system's individual evaluation and analysis of the results.

Chapter 2

Related Work

Initially, web IE involved manually written wrappers that catered to specific needs and information sources (Chawathe, Garcia-Molina, Hammer, & K, 1994; Perkowitz & Etzioni, 1995) . While this provided accuracy in the information extracted, it required users of the such IE systems to have some background in programming. Later, many wrapper induction tools came about after Kushmerick's paper (Kushmerick, Weld, & Doorenbos, 1997), using several different approaches to the problem (e.g. machine learning) (Freitag, 1998; Soderland, 1999). Subsequently, from 2001, systems that required no user intervention were developed (Chang & Lui, 2001; Crescenzi, Mecca, & Merialdo, 2002; Arasu & Garcia-Molina, 2003). This suggests a trend toward information extraction systems with less human interaction.

However, IE systems cannot completely be free of user intervention. "Supervised approaches, although require annotation from users, extend well to non-template page extraction if proper features are selected for extraction rules" (Chang, Kayed, Girgis, & Shaalan, 2006). Since user annotation is crucial to web IE, it is important to improve current user interfaces for labelling. Attempts such as *Lixto*, which has some focus on the user interface aspect of labelling, have tried to make this process more visual (Baumgartner, Flesca, & Gottlob, 2001). Other ways to do this would be to reduce the amount of sample data required by the system, or to provide a way to give the user immediate feedback as to what the result of the learnt classifier would be. Another problem faced when trying to automate web IE is having to regenerate or re-implement the wrapper every time there is a change in the site layout. This is an important problem given the ease in which templating systems allow pages to be “themed” and redesigned. Wrappers need to be robust enough to handle such frequent changes, and be less reliant on the underlying HTML structure of the page.

One method of approaching this was to make use of the visual features of the page. The PARCELS system is not strictly an information extraction system, but rather, is concerned with dividing up the page and classifying the resulting blocks correctly. As input, the user is required to input several example pages with labelled blocks as examples. The system then extracts features from 2 different aspects of the blocks: Its *lexical* features and its *stylistic* features. Using both of these the classifier uses a technique called co-training, machine classified blocks from either view is fed into the training data of the other view(Lee, Kan, & Lai, 2004). The system was then improved to include usage of inter and intra similarities between pages as features(Lau, 2005). Gatterbauer also approached the problem of extracting data from web tables by looking directly at the rendered 2D output, extracting tables (or grids) within them, and then analysing these in order to extract relevant information (Gatterbauer, Bohunsky, Herzog, Krüpl, & Pollak, 2007).

Dalvi explored the approach of developing a tree-edit model of HTML, modelling changes to a page as a stochastic process, using it to improve wrapper construction for robustness(Dalvi, Bohannon, & Sha, 2009). The method generates a list of candidate XPaths for a certain selected region within the page. It used archived data of a given website to calculate its *compute transformation probability.* Using this, the most robust XPath candidate was selected from the list. This process, however, required access to archives of older versions of the site.

After studying the drawbacks and advantages of some of the previous work in the area, I hope to make 2 main contributions with my system:

1. Provide an intuitive interface for labelling that is platform agnostic, and takes advantage of the many advancements in Javascript. This will provide the user with an immediate visual feedback as to the items that he/she will be extracting, and at the same time reduce the amount of labelling that needs to be done.

2. Create a more robust framework for extraction of the selected information using machine

learning. The classifier will have less focus on the HTML structural information of the tags in order to be resistant to any layout changes made to the page.

For the rest of the paper, I will define users of the system as users who do not have access to resources required by current information extraction system. This includes not only the hardware and processing power required, but also the time needed for the labelling work done. Also, I will define robustness as the wrapper's resistance to changes in HTML structure from the time that the wrapper was derived.

2.1 Visual Selection of Data

One of the first systems to introduce a visual method for extraction of data was *Lixto* (Baumgartner et al., 2001). In order to accomplish this, the system has an in built browser that allows users to select items, giving them visual feedback in the form of highlighted elements. There are several commercial systems that also do this, like Mozenda and Needlebase. MIT's Solvent is part of a mash-up creation system that also provides users with a similar visual feedback system. Enabling the users to see what the system will extract helps give a sense of transparency, and in this way, allows the user to make any required corrections immediately.

However, the abovementioned systems, with the exception of Needlebase, require users to perform the changes within a customised browser inside the system. In addition, Mozenda, Needlebase and Solvent require users to have had experience with programming, as part of their extraction process requires the user to define the "flow" of the extraction.

2.1.1 XPath Generation

XPath is a powerful query language commonly used for selecting particular elements in XML documents. Recently, many scrapers are using XPath as a method of retrieving the relevant element within an HTML page. However, the process of coming up with the appropriate XPath lies with the programmer of the scraper. This requires prerequsite knowledge of DOM trees and XPath, and creating the XPath manually also introduces human error. In our system, we automate the generation of XPaths for extraction from the user-selected pages.

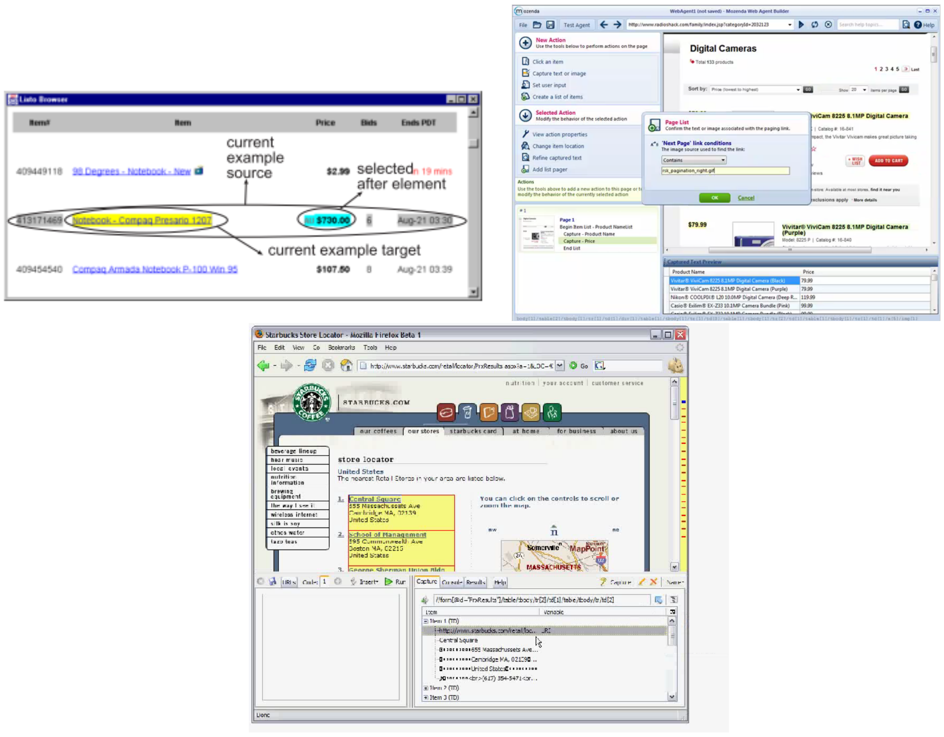


Figure 2.1: The *Lixto*, Mozenda and Solvent interfaces.

A method for extracting XPaths is described in (Anton, 2005). The approach models the DOM Tree as a traversal graph, and uses this to try to create an XPath that gets the wanted elements for all pages it is given. Later, (Dalvi et al., 2009) explored the approach of developing a tree-edit model of HTML, modelling changes to a page as a stochastic process, using it to improve wrapper construction for robustness. The method generates a list of candidate XPaths for a certain selected region within the page. It used archived data of a given website to calculate its *compute transformation probability*. Using this, the most robust XPath candidate was selected from the list. This process, however, required access to archives of older versions of the site. Such methods of XPath generation do are not immediate, and, for the most part, does not involve a lot of input from the user, but the drawback of that would be that the user is unaware of what will be extracted until he/she sees the extracted results.

In order to be able to extract data even after a change in layout, another method of extraction without the use of XPath is required. Our approach for this system uses a machine learned model. The rest of the chapter elaborates on the details of the implementation.

Looking at any web information extraction system using machine learning classification techniques, the workflow of a user can be generalised into the following steps:

1. The user finds a page or site that he/she would like information to be mined from.
2. Samples of the page presented to the user for labelling. Positive examples are selected and these are fed into the machine learner to create a model.
3. Using this model, data is extracted from similar pages, this data is then put into storage (e.g. database)
4. Some systems repeat steps 2 and 3 iteratively until an accurate model is achieved.
5. The user can then access the data storage for the mined information.

Chapter 3

Method

In the following sections, we will elaborate on the contributions of this report, namely the traversal path alignment algorithm, the framework for learning models for classification of DOM nodes, and the features used.

3.1 Path Traversal Alignment

An important aspect which describes tags which are visually similar is its traversal path from the `<BODY>` tag. Parents of selected tags with similar tag names or that have similar attributes are likely to be tags that look visually similar on the page. This also implies that the elements are probably items that that the user would also be looking for.

We propose using a modification of the Needleman-Wunsch global alignment algorithm for this purpose. Each traversal path could be seen as a sequence of symbols that need to be aligned based on their common attributes. The more attribute-values they have in common, the more likely they are ancestors of elements of the same type. However, when reconstructing the aligned sequence, one needs to be aware that the symbols are not entirely equivalent, and that only the common attributes should be retained.

Given the input of 2 arrays P and Q of elements, the scoring function is given as $\textsc{Score}(P_i, Q_j) = |P_i \cap Q_j|$, where P_i and Q_j are elements (sets of attributes) along P and Q. After the alignment, a new generalised traversal path is created, P', such that $P'_k = P_i \cap Q_j$, where P_i and Q_j are

Algorithm 1 PTAlign(P, Q, λ)

```
1: for $i \leftarrow 1$ to $|P| + 1$ do
2:     for $j \leftarrow 1$ to $|Q| + 1$ do
3:         $M_{i,j} \leftarrow \max(M_{i-1,j}, M_{i-1,j-1}, M_{i,j-1}) + \textsc{Score}(P_i, Q_j)$
4:     end for
5: end for
6: $R \leftarrow []$
7: while $i > 1$ and $j > 1$ do
8:     if $M_{i,j} - \textsc{Score}(P_i, Q_j) = M_{i-1,j-1}$ then
9:         $R \leftarrow \lambda(P_i, Q_j) + R$
10:        $i, j \leftarrow i - 1, i - j$
11:    else if $M_{i,j} - \textsc{Score}(P_i, Q_j) = M_{i-1,j}$ then
12:        $R \leftarrow * + R$
13:        $i, j \leftarrow i - 1, j$
14:    else if $M_{i,j} - \textsc{Score}(P_i, Q_j) = M_{i,j-1}$ then
15:        $R \leftarrow * + R$
16:        $i, j \leftarrow i, j - 1$
17:    end if
18: end while
19: return $R$
```

Figure 3.1: The *Traversal Path Alignment* algorithm

elements in the original traversal paths that were aligned. Figure 3.1 gives the pseudocode for the *Path Traversal Alignment* algorithm (PTA).

When selecting additional elements for generalising the XPath, the λ function would be defined as $\lambda(P_i, Q_j) = P_i \cap Q_j$. This would reconstuct a generalised traversal path with common elements and their common attributes. Gaps in the path are represented as "*" in the output. During the serialisation of the XPath, groups of these are converted to `//` to represent any descendant.

This process is a bottom-up approach: Users start with a single element, and as more items are selected, the items captured by the XPath generated grows. For each element along the selected item's path, the similarities between the tag's attributes are kept, and the differences removed. This results in an extraction rule with fewer and fewer constraints as more elements are selected.

Using alignment of the path elements, XPath rules can be generalised for elements at different levels of the DOM tree. Figure 3.3 is an example of the array generated when 2 `<a>` tags are

P	`div`	`div`	`div`	`ul`	`li`	`h3`	`a`
id	mainContent						
index	4	1	1	3	1	1	1
class		content	newsl, pagedNewsList	newslTop			

Q	`div`	`div`	`div`	`ul`	`li`	`h3`	`a`
id	mainContent						
index	4	1	1	3	2	1	1
class		content	newsl, pagedNewsList	newslTop			

After PTA:

Result	`div`	`div`	`div`	`ul`	`li`	`h3`	`a`
id	mainContent						
index	4	1	1	3		1	1
class		content	newsl, pagedNewsList	newslTop			

Figure 3.2: Example of fewer constraints after each iteration

compared, one at a depth of 5 from the `<body>` tag, and another at depth 3. The `<body>` tag is omitted in order to reduce the running time of the algorithm, as it is common in the paths for all visible elements on a given page. The algorithm is then able to generalise an XPath: `//div[@id='main']/div//a`. This XPath will include both selected elements, as well as other anchor tags that the user may be interested in.

It is important to note that serialising the generalised model to its XPath form would result in loss of information. As such the model in its unserialised form is kept, and used again when another item on the page is selected. Since this process has an associative property, the order in which items are selected on the page does not matter, and the user can then select 3 items or more in order to specify the items he wants.

3.1.1 Element rejection

Once the user has defined a region for extraction, he/she may find that the region includes items are irrelevant. Our objective was to make this process part of the selection process as well. In order to achieve this, we used the same alignment algorithm, with *P* as the already

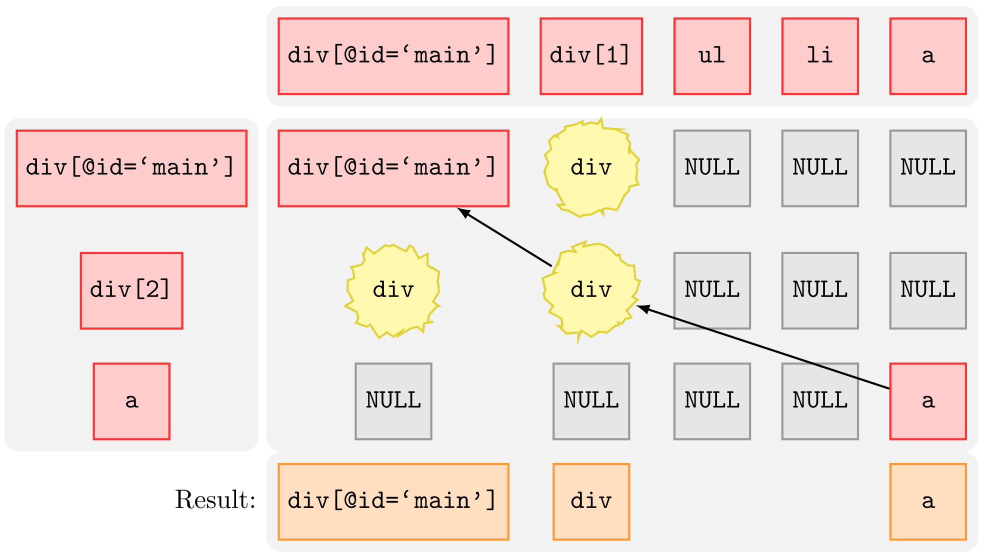


Figure 3.3: Table generated by the LCAS algorithm

generalised path, and Q as the new traversal path of the element that the user wishes to reject, with λ defined as $\lambda(P_i, Q_j) = Q_j - (P_i \cap Q_j)$.

3.1.2 Limitations

There are still several limitations to this method. When $Q_j = P_i \cap Q_j$ for all i, j in the aligned path, that rejection path cannot be used. This usually implies that the traversal paths of the wanted elements and the rejected elements are indistinguishable using the considered attributes. In such a case, the user is not allowed to select elements that cause such an event. Once the first rejection element has been generalised with the above method, additional rejection elements can then be generalised using the first algorithm.

Also, the mentioned method of generalising the XPath is a form of a multiple sequence alignment, but since it is approached using a sequence of pairwise alignments, the optimal alignment may not be achieved. However, the selection scopes of the generated XPaths are generally usable for scraping.

3.2 Robust Extraction

The above method enables us to generate a fixed rule (XPath) to extract data from the given pages. However, XPath is heavily reliant on the structure of the DOM page in its present form: Any element extracted is defined by its ancestor nodes and their attributes. Once this is changed, which is likely due to a layout change, the same XPath may no longer be applicable to the current page.

In order to be able to extract data even after a change in layout, another method of extraction without the use of XPath is required. Our approach for this system uses a machine learned model. The rest of the chapter elaborates on the details of the implementation.

In our framework, we use the provided XPaths together with the URLs of the pages to be extracted as annotated documents. Each node in the DOM is treated as a separate instance, and labelled based on the XPath it was extracted by. The XPaths are assumed to be mutually exclusive. We have chosen to use decision trees because the generated models are small, and can be easily interpreted. The generation of these models are also relatively faster than other classification models. Since CPU time and memory space are constrained resources, decision trees were suitable for our needs.

3.2.1 Features used

To keep the model more robust and resistant to layout changes on the original site, content features are extracted in order to use less of the HTML structure for extraction. In this system, a J48 decision tree classifier is used. Positive examples of HTML elements are those returned when the XPath is applied to the given page. The following features are extracted:

- Word occurrence count, or tokens (Numeric)
- Tag name (Discrete)
- Previous sibling, next sibling and parent tag name (Discrete)
- Number of words/tokens (Numeric)

- Ending with character (:,-,.) (Discrete)
- Header (Discrete)

For subsequent extractions, the server will first use the available XPath for extraction from the site. If the XPath rule fails to find elements for extraction, the learnt classification model is used. Figure 3.4 shows a decision tree when the process is run on a Google search result with the result entries selected for extraction.

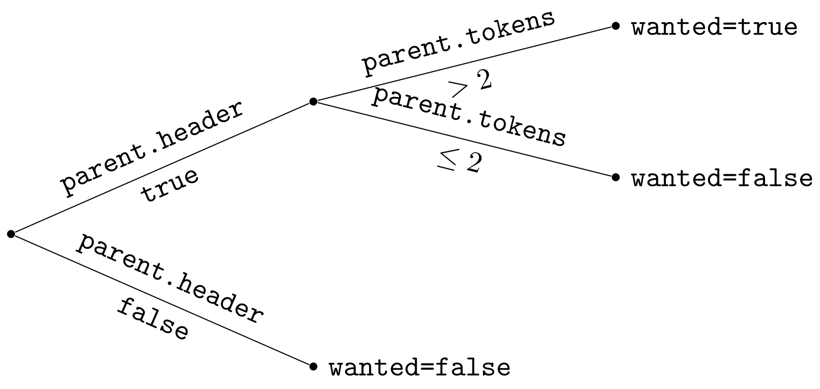


Figure 3.4: Decision tree generated when search entries on Google are selected.

As we can see, the generated decision tree seems pretty reasonable given a Google search result. The `parent.` prefix is the equivalent to the statement "if the parent of the element". We can see that the generated tree is looking for elements with parents that are headers and contain 2 or more words.

3.2.2 Crawling for more instances

We found that many annotations capture very few elements per page when compared to the total number of elements for that page. By attempting to balance the number of elements that belong to labels with those that were not captured by any XPath, we hoped to improve the performance of the classifier. However, this still yielded relatively poor results. Eventually, we crawled the site for other pages which had elements captured by the same set of XPaths. This gave us more instances to train the classifier on, and gave better results.

The crawler starts with a given set of pages, and crawls links with URLs similar to the given pages. The similarity score is given by the edit distance between the link URL and the URL of the wanted pages. A heap is used to store the seen links, and the most similar URL selected at every round. The crawler stops when it has collected a fixed number of instances, which is now set at 500.

3.2.3 Limitations

In Weka's implementation of their machine learning algorithms, the serialised models created tend to store all attributes and their values. Due to this, some of the models are really large, with a lot of redundant data. This takes up much I/O time and storage space on the server.

Also, from our tests, the learnt model does better on heading-type results, while other fields tend to have a low precision score, we will discuss the evaluation results in detail in Chapter 4.

Chapter 4

Implementation

This chapter details the implementation of the methods mentioned before into a working system. The system consists of 3 main components, the architecture is depicted in Figure 4.1. The components shown are the following:

1. **Selection Interface** This is implemented as a bookmarklet which the user can simply drag into the toolbar of the browser.

2. **Web Application** This is the frontend of the system, allowing the user to create, update and delete their extractors, and also to be able to view their extracted data.

3. **Machine Learning Component** This is the component that creates a classification model based on the generated XPath and the pages selected to be extracted from.

4.1 Selection Interface

The selection interface described in this section aims to provide visual feedback to the user when building a suitable wrapper for the page chosen by the user. The interface is implemented in the form of a bookmarklet which the user simply has to drag and drop into his/her browser toolbar. This bookmarklet can then be activated when the user reaches a page which he/she wants to have something extracted. Clicking on items will select them in green, and subsequent clicking will expand the scope of the extraction, with the items to be extracted highlighted in yellow.

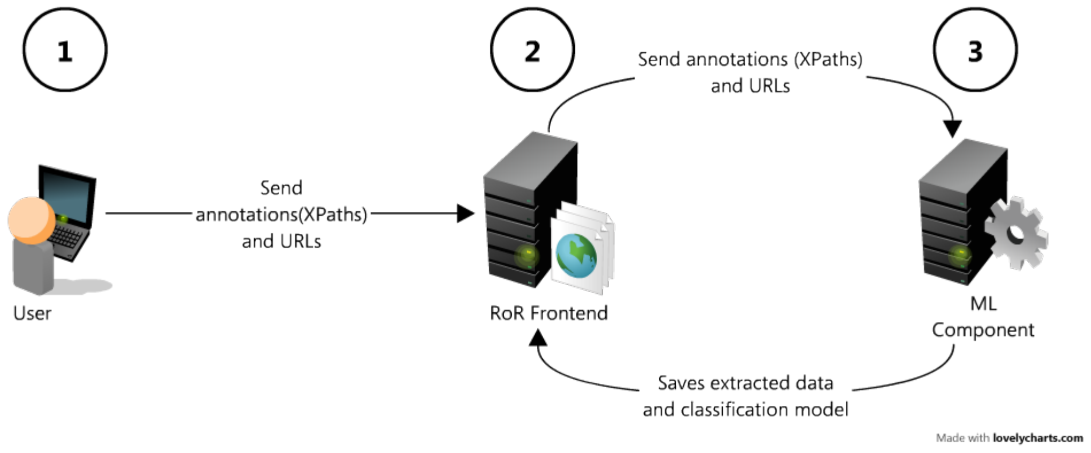
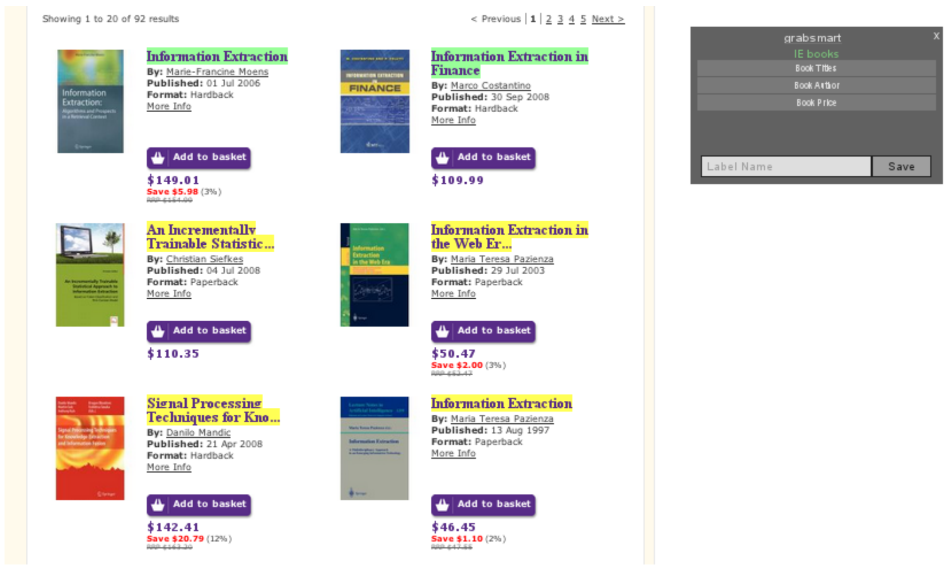

1
2
Send annotations (XPaths) and URLs
3
Send annotations(XPaths) and URLs
User
RoR Frontend
ML Component
Saves extracted data and classification model
Made with lovelycharts.com

Figure 4.1: Overall system architecture

Figure 4.2 shows a screenshot of a search result in bookdepository.co.uk with the bookmarklet activated.



Showing 1 to 20 of 92 results

< Previous | **1** | 2 3 4 5 Next >

Information Extraction
By: Marie-Francine Moens
Published: 01 Jul 2006
Format: Hardback
More Info
Add to basket
$149.01
Save $5.98 (3%)
~~RRP $154.99~~

Information Extraction in Finance
By: Marco Costantino
Published: 30 Sep 2008
Format: Hardback
More Info
Add to basket
$109.99

An Incrementally Trainable Statistic...
By: Christian Siefkes
Published: 04 Jul 2008
Format: Paperback
More Info
Add to basket
$110.35

Information Extraction in the Web Er...
By: Maria Teresa Pazienza
Published: 29 Jul 2003
Format: Paperback
More Info
Add to basket
$50.47
Save $2.00 (3%)
~~RRP $52.47~~

Signal Processing Techniques for Kno...
By: Danilo Mandic
Published: 21 Apr 2008
Format: Hardback
More Info
Add to basket
$142.41
Save $20.79 (12%)
~~RRP $163.20~~

Information Extraction
By: Maria Teresa Pazienza
Published: 13 Aug 1997
Format: Paperback
More Info
Add to basket
$46.45
Save $1.10 (2%)
~~RRP $47.55~~

grabsmart X
IE books
Book Titles
Book Author
Book Price
Label Name
Save

Figure 4.2: An example of the selection interface in action.

For greater automation, the bookmarklet interface attempts to reduce the amount of labelling work the user has to do by trying to predict what the user wants to extract from the page. More specifically, as the user selects the individual HTML elements on the page using the interface, the bookmarklet attempts to generalise an XPath that captures the selected elements, and also elements on the page with similar characteristics, like class names and position within the parent tag.

The process is an iterative one. When the user clicks on a new element on the page, the XPath is recalculated, and then used to highlight the captured items on the page. This way, the user understands the changes he/she has made to the extraction scope as he/she clicks on additional items. The user is then able to perform this task for any number of labels the user thinks is appropriate for the extractor he/she is creating.

4.1.1 Engineering challenges

One of the aims of the interface was to ensure that the item would not require the hassle of installation. This included not only standalone, specialised browsers for selection of items, but also extended to plugins – since a different plugin would have to be written for every browser. We made the decision to go with a bookmarklet because it was one of the few ways we could still run our code on top of another website. In general, a bookmarklet achieves this by injecting a `<SCRIPT>` tag into the host page, with the `src` attribute pointed to the main code hosted on the application's server. Since this meant downloading the code every time the bookmarklet is activated, one of our challenges was also to keep the code small.

The bookmarklet sends the created unserialised generalised traversal paths and XPaths back this by way of AJAX callbacks to the RoR frontend. One of the technical difficulties in implementing this was the fact that the standard XMLHttpRequest method of AJAX callbacks were not usable due to the cross-site security put into place by the Javascript API. The workaround was to use hidden form elements within the interface to send POST requests back to the server. Data was fetched by inserting script tags and callbacks to allow the JSON objects to be passed back into the small Javascript application.

4.2 Machine Learning Component

Once the XPath is sent from the bookmarklet back to the server-side application and inserted into the database, the user then has the option to test his/her created extractor. Once the user selects this option the machine-learning component visits the new page inserted, and crawls within the domain for similar pages. In this context, similar pages refer to HTML documents that return results, of any number, when a query is made using any of the previously labelled XPaths.

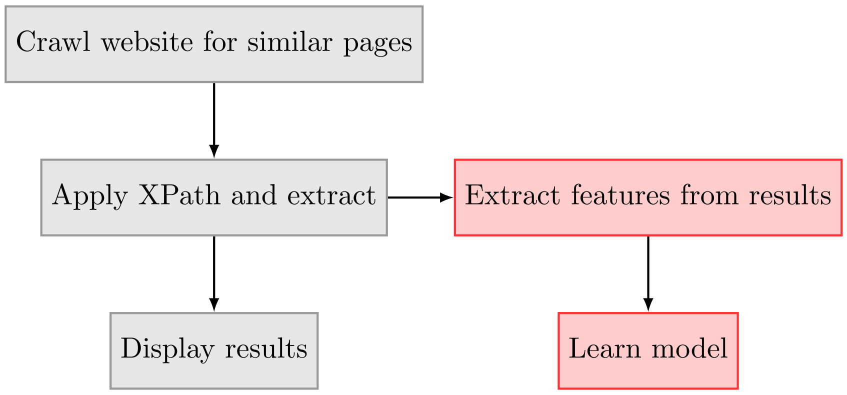

Crawl website for similar pages
Apply XPath and extract
Extract features from results
Display results
Learn model

Figure 4.3: Workflow of the machine learning component

The extracted data is inserted back into the database, and then made available to the user. At the same time, features are extracted from this data in order to create a model for extraction for future use.

Chapter 5

Evaluation

For the evaluation of this part of the system, a group of 30 students were paid to do a simple 30 minute evaluation of the system. A simple tutorial was given to each of the students, and once completed, a simple extraction task from the bookdepository.co.uk website was to be carried out. Lastly, the students were asked to fill out a questionnaire on the following aspects of the system: Ease of installation of the bookmarklet, the ease of item selection, and the ease with which they could view the extracted items.

Chapter 6

Conclusion

The current approach does not solve the issue of unordered elements that are extracted from the pages. While some methods have been devised, none of them have been implemented and tested at this time.

More importantly, in the following semester, more fine tuning of the machine learning component needs to be done. Also, formal evaluation of the methods described has to be carried out.

6.1 Contributions

6.2 Future Work

References

Anton, T. (2005). XPath-Wrapper Induction by generalizing tree traversal patterns. *Lernen, Wissensentdeckung und Adaptivitt (LWA)*, (3), 2005.

Arasu, A., & Garcia-Molina, H. (2003). Extracting structured data from Web pages. *Proceedings of the 2003 ACM SIGMOD international conference on on Management of data - SIGMOD '03*, , 2003, 337.

Baumgartner, R., Flesca, S., & Gottlob, G. (2001). Visual Web Information Extraction with Lixto. *Proceedings of the 27th VLDB Conference*, , 2001.

Chang, C.-H., Kayed, M., Girgis, M. R., & Shaalan, K. F. (2006). A Survey of Web Information Extraction Systems. *IEEE Transactions on Knowledge and Data Engineering*, *18*(10), October, 2006, 1411–1428.

Chang, C.-h., & Lui, S.-C. (2001). IEPAD : Information Extraction Based on Pattern Discovery. *Proceedings of the 10th international conference on World Wide Web*, , 2001, 681–688.

Chawathe, S., Garcia-Molina, H., Hammer, J., & K (1994). The TSIMMIS project: Integration of heterogeneous information sources. *Proceedings of IPSJ*, , 1994, 1–12.

Crescenzi, V., Mecca, G., & Merialdo, P. (2002). RoadRunner : Towards Automatic Data Extraction from Large Web Sites. *Proceedings of the 27th VLDB Conference*, , 2002.

Dalvi, N., Bohannon, P., & Sha, F. (2009). Robust web extraction: an approach based on a probabilistic tree-edit model. *Proceedings of the 35th SIGMOD international conference on Management of data* (pp. 335–348), 2009: ACM.

Freitag, D. (1998). Information Extraction from HTML : Application of a General Machine Learning Approach Extraction as Text Classi cation. *Search*, , 1998.

Gatterbauer, W., Bohunsky, P., Herzog, M., Krüpl, B., & Pollak, B. (2007). Towards domain-independent information extraction from web tables (pp. 71–80), Banff, Alberta, Canada, 2007: ACM.

Kushmerick, N., Weld, D., & Doorenbos, R. (1997). Wrapper induction for information extraction.

Lau, A. M. (2005). Advancing PARCELS : PARser for Content Extraction and Logical Structure Using Inter- and Intra- Similarity Features Advancing PARCELS : PARser for Content Extraction and Logical Structure Using Inter- and Intra- Similarity Features. *2004/2005 Honours Year Project Report, National University of Singapore*, , 2005.

Lee, C. H., Kan, M.-Y., & Lai, S. (2004). Stylistic and lexical co-training for web block classification. *Proceedings of the 6th annual ACM international workshop on Web information and data management - WIDM '04*, , 2004, 136.

Perkowitz, M., & Etzioni, O. (1995). Category translation: Learning to understand information on the internet. *International Joint Conference on Artificial Intelligence*, Vol. 14 (pp. 930–938), 1995: Citeseer.

Soderland, S. (1999). Learning Information Extraction Rules for Semi-structured and Free Text. *World Wide Web Internet And Web Information Systems*, *44*, 1999, 1–44.

Batch	Label	Future Batch	TP	FP	FN	TN		Recall%	Precision%
0	AUTHOR	1	84	38	50	2761		62.6866	68.8525
0	AUTHOR	2	30	28	120	6222		20	51.7241
0	AUTHOR	3	64	37	86	5857		42.6667	63.3663
0	AUTHOR	4	3	2	147	6376		2	60
0	AUTHOR	5	3	9	147	9249		2	25
1	AUTHOR	2	0	0	150	6280		0	#DIV/0!
1	AUTHOR	3	0	28	150	5930		0	0
1	AUTHOR	4	0	75	150	6306		0	0
1	AUTHOR	5	1	0	149	9261		0.66667	100
2	AUTHOR	3	0	13	150	5945		0	0
2	AUTHOR	4	0	13	150	6368		0	0
2	AUTHOR	5	0	111	150	9148		0	0
3	AUTHOR	4	150	132	0	6099		100	53.1915
3	AUTHOR	5	0	56	135	8276		0	0
4	AUTHOR	5	0	180	135	8152		0	0
0	DIGG COUNTS	1	135	84	0	2664		100	61.6438
0	DIGG COUNTS	2	147	181	3	5952		98	44.8171
0	DIGG COUNTS	3	104	195	46	5659		69.3333	34.7826
0	DIGG COUNTS	4	101	126	49	6154		67.3333	44.4934
0	DIGG COUNTS	5	71	291	79	8900		47.3333	19.6133
1	DIGG COUNTS	2	0	33	150	6247		0	0
1	DIGG COUNTS	3	0	22	150	5936		0	0
1	DIGG COUNTS	4	0	6	150	6375		0	0
1	DIGG COUNTS	5	4	9	146	9246		2.66667	30.7692
2	DIGG COUNTS	3	150	382	0	5426		100	28.1955
2	DIGG COUNTS	4	150	391	0	5840		100	27.7264
2	DIGG COUNTS	5	80	830	70	8347		53.3333	8.79121
3	DIGG COUNTS	4	150	150	0	6081		100	50
3	DIGG COUNTS	5	0	170	150	9087		0	0
4	DIGG COUNTS	5	17	153	118	8162		12.5926	10
0	SUMMARY	1	110	74	25	2699		81.4815	59.7826
0	SUMMARY	2	150	142	0	5988		100	51.3699
0	SUMMARY	3	97	117	21	5744		82.2034	45.3271
0	SUMMARY	4	76	108	34	6197		69.0909	41.3043
0	SUMMARY	5	60	73	75	8204		44.4444	45.1128
1	SUMMARY	2	22	30	128	6228		14.6667	42.3077
1	SUMMARY	3	50	2	68	5906		42.3729	96.1538
1	SUMMARY	4	56	53	54	6272		50.9091	51.3761
1	SUMMARY	5	81	34	69	9144		54	70.4348
2	SUMMARY	3	84	109	34	5765		71.1864	43.5233
2	SUMMARY	4	78	104	32	6199		70.9091	42.8571
2	SUMMARY	5	118	213	32	8926		78.6667	35.6495
3	SUMMARY	4	72	213	38	6096		65.4545	25.2632
3	SUMMARY	5	135	249	0	7948		100	35.1562
4	SUMMARY	5	150	548	0	8559		100	21.49
0	TITLE	1	135	501	0	2247		100	21.2264
0	TITLE	2	150	1866	0	4264		100	7.44048
0	TITLE	3	150	1635	0	4173		100	8.40336
0	TITLE	4	150	1619	0	4612		100	8.47937
0	TITLE	5	150	3016	0	6095		100	4.73784
1	TITLE	2	63	212	87	6005		42	22.9091
1	TITLE	3	95	658	55	5205		63.3333	12.6162
1	TITLE	4	96	509	54	5776		64	15.8678
1	TITLE	5	96	356	54	8810		64	21.2389
2	TITLE	3	150	306	0	5502		100	32.8947
2	TITLE	4	150	388	0	5843		100	27.881
2	TITLE	5	150	517	0	8592		100	22.4888
3	TITLE	4	150	171	0	6060		100	46.729
3	TITLE	5	150	456	0	8651		100	24.7525
4	TITLE	5	150	435	0	8672		100	25.641

Table 1: SVM evaluation results

Batch	Label	Future Batch	TP	FP	FN	TN		Recall%	Precision%
0	AUTHOR	1	134	390	0	2359		100	25.5725
1	AUTHOR	2	10	340	140	5930		6.66667	2.85714
2	AUTHOR	3	0	2	150	5956		0	0
3	AUTHOR	4	150	932	0	5299		100	13.8632
4	AUTHOR	5	0	51	150	8831		0	0
0	DIGG COUNTS	1	135	521	0	2227		100	20.5793
1	DIGG COUNTS	2	0	0	150	6280		0	#DIV/0!
2	DIGG COUNTS	3	150	500	0	5308		100	23.0769
3	DIGG COUNTS	4	150	435	0	5796		100	25.641
4	DIGG COUNTS	5	150	2221	0	6511		100	6.32644
0	SUMMARY	1	135	162	0	2586		100	45.4545
1	SUMMARY	2	150	1480	0	4650		100	9.20245
2	SUMMARY	3	1	5	117	5952		0.84746	16.6667
3	SUMMARY	4	0	298	110	6083		0	0
4	SUMMARY	5	150	458	0	8276		100	24.6711
0	TITLE	1	135	576	0	2172		100	18.9873
1	TITLE	2	150	4485	0	1645		100	3.23625
2	TITLE	3	150	25	0	5783		100	85.7143
3	TITLE	4	150	33	0	6198		100	81.9672
4	TITLE	5	150	31	0	8701		100	82.8729

Table 2: Tree evaluation results