Undergraduate Research Opportunity Program
(UROP) Project Report

grab*smart*: A User-centric Web Information Extraction System

By

Shawn Tan

Department of Computer Science

School of Computing

National University of Singapore

2010/11

Undergraduate Research Opportunity Program
(UROP) Project Report

grab*smart*: A User-centric Web Information Extraction System

By

Shawn Tan

Department of Computer Science

School of Computing

National University of Singapore

2010/11

Project No: U07932
Advisor: A/P Kan Min-Yen
Deliverables:
Report: 1 Volume

Abstract

We present grab*smart*, a user-centric web information extraction system that addresses two challenges of today's web information extraction systems. Firstly, users without any programming language should be able to easily and simply specify information they want to extracted. Secondly, the system should be robust enough to work seamlessly after changes in website layouts. To address these challenges grab*smart*: 1. provides users with an interactive visual interface for information selection, and 2. uses a robust machine learning framework that extracts information even after layout changes.

Our interface for graphical selection of data for extraction has been found useful by the surveyed students and our estimated evaluation of our robust wrapper shows that our system is comparable with the state-of-the-art, performing at 91.07% according to their measure of robustness, while their performance is around 80%.

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

2.1 The *Lixto*, Mozenda and Solvent interfaces 5

3.1 The *traversal path alignment* algorithm 10
3.2 Example of fewer constraints after each iteration 11
3.3 Table generated by the LCAS algorithm 12
3.4 Decision tree generated when search entries on Google are selected. 15

4.1 Overall system architecture 17
4.2 An example of the selection interface in action. 18
4.3 Dashboard screenshot 21
4.4 Extractor view 21
4.5 Workflow of the machine learning component 22

5.1 Graph of Tree Edit Distance between adjacent pages 27

List of Tables

1.1 Pros & Cons of using RSS or wrappers (screen-scrapers) 2

5.1 $D^{(t)}$ intervals and their corresponding layout changes 29
5.2 Learnt model evaluation results for digg.com 30

Table of Contents

Title **i**

Abstract **ii**

Acknowledgement **iii**

List of Figures **iv**

List of Tables **v**

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Goals & Challenges . . . 2

2 Related Work **4**
- 2.1 Visual feedback for User annotation . . . 4
- 2.2 Generation of XPath for wrappers . . . 6
- 2.3 Robust wrappers . . . 7
- 2.4 Conclusion . . . 7

3 Method **9**
- 3.1 Path Traversal Alignment (PTA) . . . 9
  - 3.1.1 Element rejection . . . 12
  - 3.1.2 Limitations . . . 13
- 3.2 Robust Extraction . . . 13
  - 3.2.1 Features used . . . 14
  - 3.2.2 Crawling for more instances . . . 15
  - 3.2.3 Limitations . . . 16

4 The grab*smart* System **17**
- 4.1 Selection Interface . . . 18
  - 4.1.1 Engineering challenges . . . 19
- 4.2 Web Application . . . 20
  - 4.2.1 Extractor list . . . 20
  - 4.2.2 Viewing the extractor . . . 20
- 4.3 Machine Learning Component . . . 22

5 Evaluation **24**
5.1 Human evaluation of Selection Interface 24
5.1.1 Evaluator demographics 25
5.1.2 Analysis 25
5.2 Machine Learning evaluation 26
5.2.1 Methodology 27
5.2.2 Analysis 28

6 Conclusion **31**
6.1 Contributions 32
6.2 Future Work 32
6.2.1 User Interface 32
6.2.2 Machine learning component 33
6.2.3 User-centric Web IE 33

References **34**

A Evaluation **A-1**
A.1 Bookstore scenario A-1
A.2 Survey & Feedback results A-2
A.2.1 Preliminary questions A-2
A.2.2 System evaluation questions A-2

Chapter 1

Introduction

1.1 Motivation

Web Information Extraction (Web IE) is defined as extracting data from the web, which are commonly unstructured or semi-structured into a structured form. Since many web documents today are increasingly template-based, generated by server scripts, web IE is slightly different from IE from free text. This allows for more accurate methods for IE, and systems have been developed to extract the differing information from the repetitive raw data. In IE, procedures that extract selected information from structured or semi-structured documents like HTML are known as *wrappers*.

While many systems for web IE have been developed over the years, many of these are used in corporate settings, and are commonly used to extract huge amounts of data from the web into a form to aid search and retrieval. This is within the reach of corporations, but users who browse the web for leisure do not have access to such resources. Another drawback of these systems is that the process of getting them up and running usually involves 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.

One might argue that this is where RSS feeds come in, to provide users updates to constantly changing data on their favourite web pages. However, the data provided through RSS is usually controlled by the corresponding sites, and may not provide the exact information that the user

	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)

actually wants. An alternative for users is for them to write their own "screen scrapers", which are simply wrappers. This would provide complete control over the data extracted from the page of their liking, but, however, most users are unfamiliar with programming, let alone the many technical issues involved in screen scraping.

Considering various aspects of RSS feeds and screen scrapers (see Table 1.1), we see an opportunity to make web IE more accessible for users by making it more user friendly,without the need for any programming knowledge. More than that, we also want the created wrappers to be robust and resistant to website layout changes with minimal user intervention. In essence, a "super scraper" for user-centric web information extraction.

1.2 Goals & Challenges

Our primary goal of user-centric web IE is to offer a simple and easy way for users to be able to extract data from the web.

We have identified two challenges for this goal that we will address in this thesis. Firstly, the system must allow users to specify the information they want extracted without requiring them to know how to program. Secondly, the system must work seamlessly even at the presence of a web site layout change.

To address these challenges, we propose grab*smart*, a system with the following features:

1. Provide an intuitive and visual interface for labelling that is platform agnostic. grab*smart*'s user interface for specifying information on a web page to extract provides the users immediate visual feedback as to items that will be extracted. This makes the labelling process

more interactive, and at the same time reduces the amount of labelling that needs to be done.

2. Provide a robust framework for extraction of the selected information using machine learning. Users would have their wrappers still work after layout changes, with minimal user intervention.

The remaining parts of the thesis is organised as follows. In Chapter 2 we will look at some of the other work that has been done in the areas of visual annotation of training data, XPath generation and creation of robust wrappers. In Chapter 3, we will describe the various algorithms and methodologies involved in the creation of the system. In Chapter 4, we will look at the implementation of the system, while finally in Chapter 5 we will see the 2 main components of the system being evaluated on 2 fronts, a human evaluation for the selection interface, and an evaluation of the machine learning component.

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 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.

2.1 Visual feedback for User annotation

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.

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 from the system, and in this way, allows the user to make any required corrections immediately. Examples of these systems in action are seen in Figure 2.1.

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.

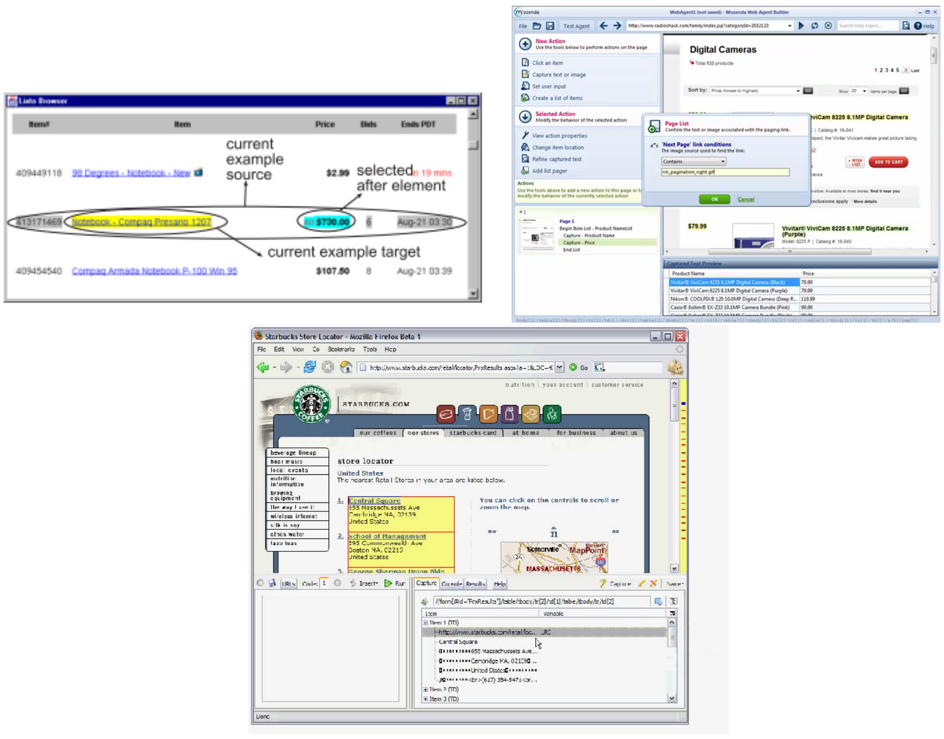


Figure 2.1: The *Lixto*, Mozenda and Solvent interfaces

In contrast,our proposed system, grab*smart*, we aim to provide a system which allows users to have a sense of what will be extracted, with the added advantage of keeping the installation process simple by introducing this to the user in the form of a bookmarklet, so that all that is needed is a drag-and-drop action.

We have identified XPath as a suitable format to query the selected pages for extraction, due to its widespread use[1], so by being able to generate XPath queries, we are also able to provide a method for writers of their own scrapers to do so more easily.

2.2 Generation of XPath for wrappers

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. This is because its representation of the selected items is somewhat similar to the way we access directories in our file systems, and also the ubiquity of XPath as a query language for many HTML parsers makes XPath viable as a tool to retrieve the relevant elements within a 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.

Methods are available to automate the creation of these XPath queries. 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, Bohannon, & Sha (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 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 contrast, our system trades the robustness of the XPath query with the speed of genera-

[1]Implementations of Javascript in most browsers have a native API for querying using XPath

tion. It will also be able to create, and display elements on the page that will be extracted as the user is selecting items.

2.3 Robust wrappers

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 CMSs allow pages to be "themed" and redesigned. Wrappers need to be robust enough to handle such changes, and be less reliant on the underlying structure of the page. We have already covered one method of robust wrapper generation by using probabilistic tree edit models in the previous section.

One other method of approaching this was to make use of the visual features of the page. The PARCELS (Lee, Kan, & Lai, 2004) 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. 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).

2.4 Conclusion

To address the shortcomings of existing work in the area, grab*smart* makes two main contributions:

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 report, we 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, we will define robustness as the wrapper's resistance to changes in HTML structure from the time that the wrapper was derived.

Chapter 3

Method

In the following sections, we will elaborate on the different techniques used in this report. In the first section we will describe traversal path alignment algorithm, which we propose to overcome the issues of current methods of XPath generation by providing a way to create viable XPaths based on elements that users have selected or rejected, giving them a visual understanding of what will be extracted as they complete the process.

Next, we address the approach we take in trying to create a robust wrapper for the site. We propose a framework for learning models for classification of HTML tags or DOM nodes by learning from extracted instances using the XPath.

3.1 Path Traversal Alignment (PTA)

A tag's traversal path from the `<BODY>` tag is an important aspect which describes tags which are visually similar. 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 (Needleman & Wunsch, 1970) 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.

Hence, we make the following modifications to the Needleman-Wunsch algorithm:

1. The scoring function used is based on the number of attributes in common between the two elements.

2. The method for reconstructing the aligned path is given as a parameter. We will describe in the following sections why this is needed.

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

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 elements in the original traversal paths that were aligned. Figure 3.1 gives the pseudocode for our proposed *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. Figure 3.2 shows an example of generalisation between 2 equal length traversal paths. The attribute values highlighted in yellow are the ones being removed.

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

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 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.

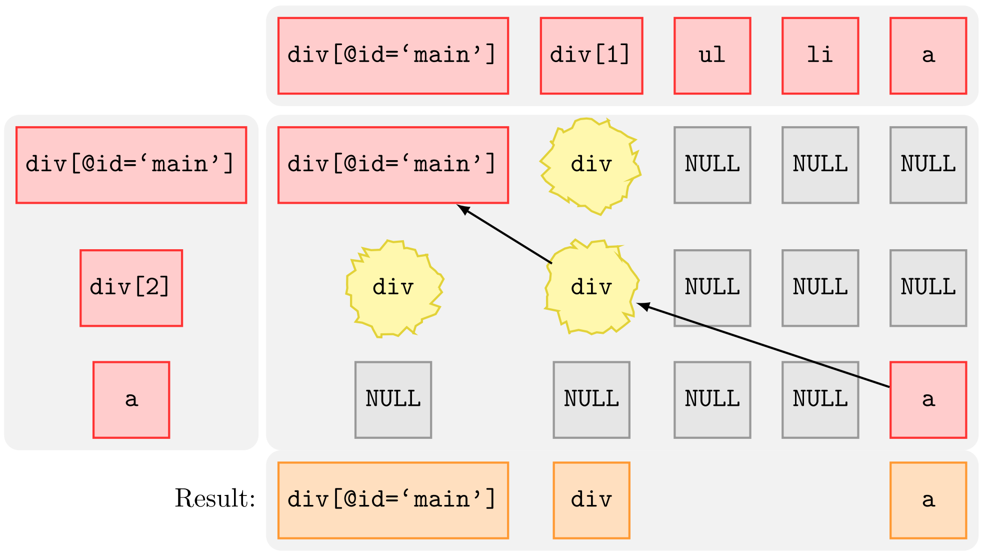


Figure 3.3: Table generated by the LCAS algorithm

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 that are irrelevant. Our objective was to make this process part of the selection process as well. To achieve this, we used the same alignment algorithm, with P as the already 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 some 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. To address this limitation, the user is not allowed to select elements that cause such an event. Once the first rejection element has been generalised with the PTA algorithm, 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 our method is approached using a sequence of pairwise alignments, the optimal alignment may not be achieved. However, in our evaluation the selection scopes of the generated XPaths are found by users to be usable for scraping.

One final limitation that is that the XPath generated in this manner will not work after a layout change that affects the layout of the DOM tree. In the following section, we will describe how we overcome this problem. Overcoming this limitation is a major contribution of this thesis and will be discussed in greater detail in the next section.

3.2 Robust Extraction

The PTA algorithm 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.

To extract data even after a change in layout, a 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.

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.

Our system follows the same workflow, except for step 4. We will elaborate further how the machine learning component fits into the entire picture in 4.

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.

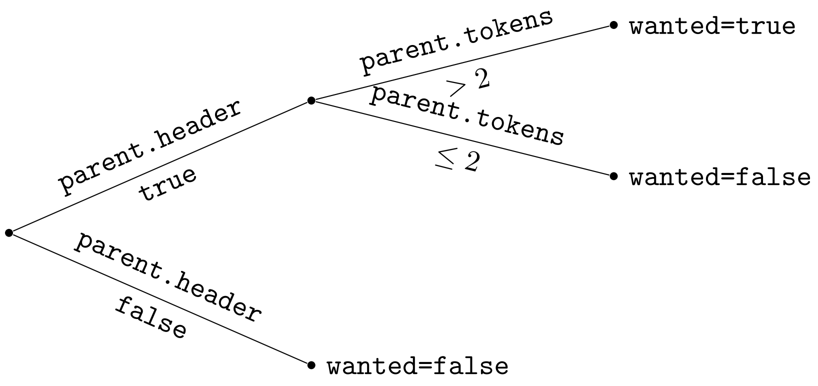


Figure 3.4: Decision tree generated when search entries on Google are selected.

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. Note that this is also done automatically without user intervention.

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. However, in comparison with the implementations of the other classification methods in the Weka library, the decision tree classifier does perform better than the other classifiers in terms of runtime and model size. A decision tree classifier optimised specifically for the system was not implemented due to time constraints.

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

The grab*smart* System

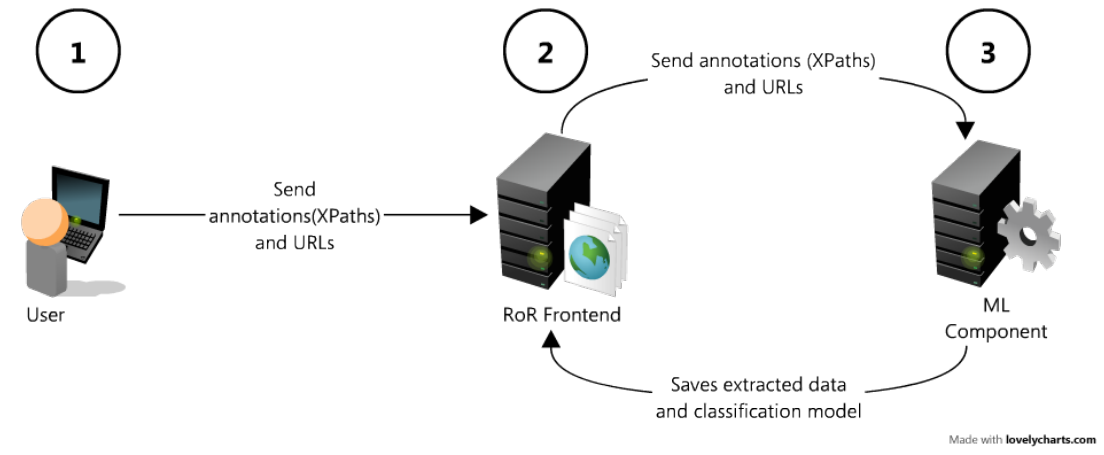

1
2
Send annotations (XPaths) and URLs
3
Send annotations(XPaths) and URLs
User
RoR Frontend
ML Component
Saves extracted data and classification model
Made with lovelycharts.com

Figure 4.1: Overall system architecture

This chapter details the implementation of the methods mentioned before into a working system. The system consists of three 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.

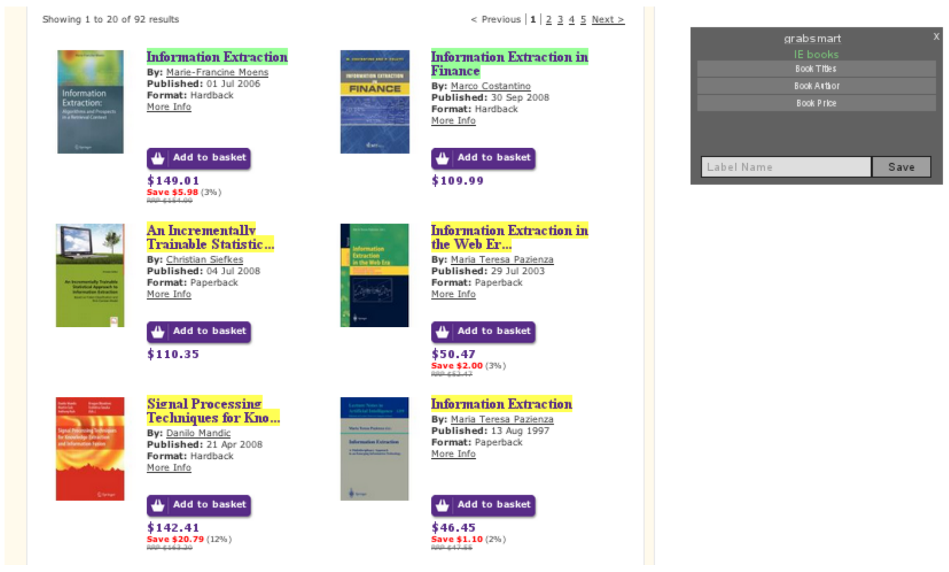
3. **Machine Learning Component** This is the component that creates a classification

model based on the generated XPath and the pages selected to be extracted from.

We intend to introduce this system with the following scenario: John wants to be informed of the latest books about information extraction on bookdelivery.co.uk. In the subsequent sections we will describe John's workflow, and how the methods described in Chapter 3 fit into the overall system.

4.1 Selection Interface

The selection interface described in this section aims to provide visual feedback to John when building a suitable wrapper for the page chosen by John. The interface is implemented in the form of a bookmarklet which John simply has to drag and drop into his browser toolbar. This bookmarklet can then be activated when the he arrives at 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. 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%)
[illegible]

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%)
[illegible]

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%)
[illegible]

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

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 predicting 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 generalises an XPath that captures the selected elements, and other elements on the page with similar characteristics, like class names and position within the parent tag.

When the user clicks on a new element on the page, the XPath is recalculated using the PTA algorithm, and then used to highlight the captured items on the page. Since the generalisation of the XPath is an iterative process, the user is able to perform actions step by step, and see how their actions affect the result. 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.

In this case, John navigates to the search results for "Information Extraction" on bookdepository.co.uk, and activates the bookmarklet. He then selects the first two book titles and the selection interface automatically highlights all the other titles on the page in yellow. This means that the items will be extracted once the extractor is activated. He then saves this label, then does the same for the prices.

4.1.1 Engineering challenges

One of the aims of the interface was to ensure that the software would not require the hassle of installation. This included not only standalone, specialised browsers for selection of items, but also extended to plugins. We made the decision to go with a bookmarklet because it was a commonly used method for a platform-agnostic app that runs on existing web pages. 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. In order to force the bookmarklet to always be the latest version retrieved from the server, we made it perform the request with different parameters each time (random number). 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 via AJAX callbacks to the Ruby on Rails application 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. Our workaround is to use hidden form elements within the interface to send POST requests back to the server. Data is fetched by inserting script tags and callbacks to allow the JSON objects to be passed back into the small Javascript application.

4.2 Web Application

In order to extract data from bookdepository.co.uk, John would have to create a new **extractor**. This can be done through the bookmarklet when he is at the page he wishes to capture. Once he starts selecting items to capture at a given page, he would have created a new **page** that the extractor is required to extract. Once he is done with a capture region, this would be saved under the extractor as a new **label**.

The web application we have developed aims to provide the user a way to modify and manipulate the concepts we have mentioned.

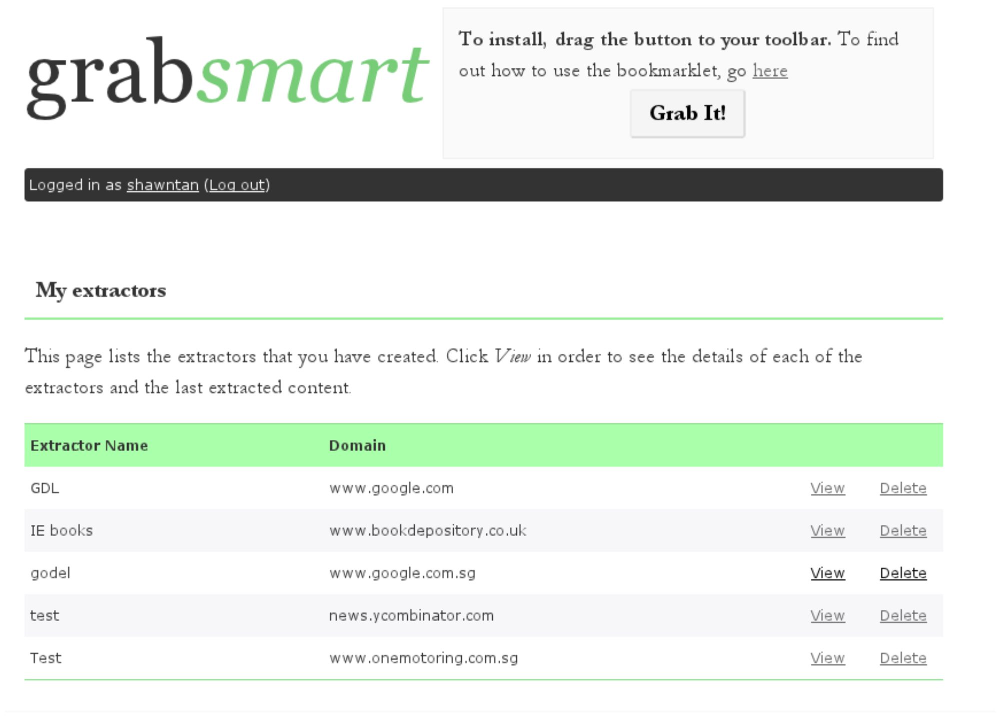
4.2.1 Extractor list

Figure 4.3 shows a screenshot of the John's dashboard. This is the first page that John sees after he has logged in. The page lists the various extractors that he has created, and he is able to delete them or view the extractors in greater detail by clicking “View”.

4.2.2 Viewing the extractor

The detailed view of the extractor shows several items, which can be seen in Figure 4.4:

1. List of the labels, together with their corresponding XPath. We have chosen to supply the generated XPath in the page for the benefit of users who are familiar with writing screen-scrapers.


grabsmart

To install, drag the button to your toolbar. To find out how to use the bookmarklet, go here

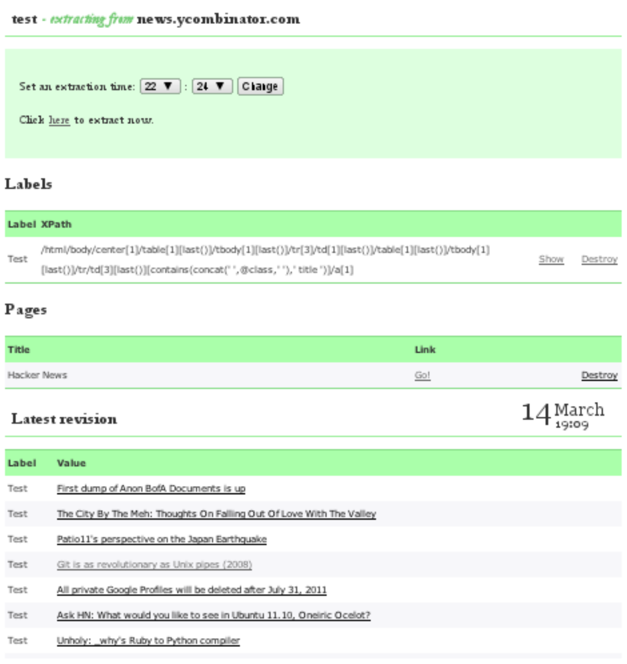
Logged in as shawntan (Log out)

My extractors

This page lists the extractors that you have created. Click *View* in order to see the details of each of the extractors and the last extracted content.

Extractor Name	Domain		
GDL	www.google.com	View	Delete
IE books	www.bookdepository.co.uk	View	Delete
godel	www.google.com.sg	View	Delete
test	news.ycombinator.com	View	Delete
Test	www.onemotoring.com.sg	View	Delete

Figure 4.3: Dashboard screenshot


test - *extracting from* news.ycombinator.com

Set an extraction time: 22 : 24 Change

Click here to extract now.

Labels

Label	XPath		
Test	/html/body/center[1]/table[1][last()]/tbody[1][last()]/tr[3]/td[1][last()]/table[1][last()]/tbody[1][last()]/tr/td[3][last()][contains(concat(' ',@class,' '),' title ')]/a[1]	Show	Destroy

Pages

Title	Link	
Hacker News	Go!	Destroy

Latest revision

14 March 19:09

Label	Value
Test	First dump of Anon BofA Documents is up
Test	The City By The Meh: Thoughts On Falling Out Of Love With The Valley
Test	Patio11's perspective on the Japan Earthquake
Test	Git is as revolutionary as Unix pipes (2008)
Test	All private Google Profiles will be deleted after July 31, 2011
Test	Ask HN: What would you like to see in Ubuntu 11.10, Oneiric Ocelot?
Test	Unholy: _why's Ruby to Python compiler

Figure 4.4: Extractor view

2. The list of pages that the labels will be applied on.

3. The latest list of data extracted from the data, if it exists.

John is, again, able to delete any labels or pages that he no longer wishes to extract using the extractor. Also, the page has a linked RSS feed that the user can subscribe to.

It is at this page that the John is able to do a test run of the extractor. Once satisfactory, he can then set a daily time for the data to be extracted.

Once John has arrived at this point, his extractor is now in place to provide him updates via RSS daily.

4.3 Machine Learning Component

After John clicks on "Extract now" or when the extractor is scheduled to be executed, the machine learning component will retrieve the relevant pages and extract the data using the extractor's XPath. Using this set of data, it will then proceed to retrieve more similar pages with the method described in the previous chapter.

This process is meant to be transparent to the user, so the user should not have to make any changes to his configured extractor. The extractor should then work even though a layout change has occurred.

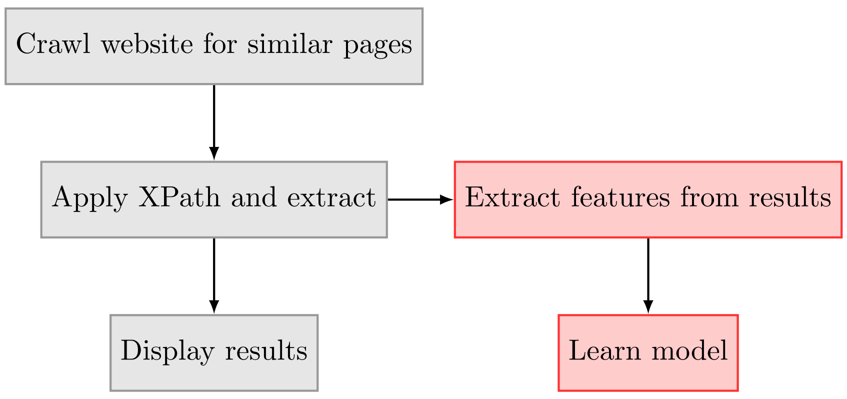

Crawl website for similar pages
Apply XPath and extract
Extract features from results
Display results
Learn model

Figure 4.5: 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. For John, the process takes place completely in the background. When the layout of bookdepository.co.uk changes, the same data is still expected to be extracted. This is in line with what we expect from a user-centric web IE system.

Now that we have seen how the various methods described in Chapter 3 fit into the rest of the system, we will look at how well the system performs in Chapter 5.

Chapter 5

Evaluation

We have divided the evaluation of our system into 2 important sections. The selection interface deals with a more qualitative topic, since its performance is dependent on how easy users find selecting items for extraction. For this component of the system, we conducted a human evaluation on the qualitative performance of the system. In the second section, we will describe the evaluation of the machine learning technique we have used, and how well this performs when attempting to extract data from a webpage after a layout change.

In summary, these are the questions we hope to answer with our evaluation:

1. What is the current awareness level of Web Information Extraction?

2. How easy is it to do various web IE tasks using our system?

3. How well does the robust wrapper perform on modified web page layouts?

5.1 Human evaluation of Selection Interface

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. The full content of the entire survey can be found in the appendix, along with the survey results.

The students were from various tertiary institutions in Singapore, and were given a small sum of money for their time.

5.1.1 Evaluator demographics

We conducted a survey with these 30 students before they interacted with our system, and asked them about their prior experiences with Web IE.

60% of the students have never heard of Web IE. In order to keep up to date with topics they are interested in, a little over 60% manually visit the sites of such topics, while 22% make use of RSS feeds. When asked if they knew what a screen-scraper or a wrapper was, only 1 out of the 30 have heard of them, and 96% have never used any form of Web IE tool before.

This suggests that while companies have the capabilities and resources to harness the power of data mining and Web IE, many users, even those as tech savvy as university students, are either unaware of such technology, or such tools are not as accessible to users.

When asked how they would go about getting specific data from a set of pages into a tabular form for some research or personal interest, many of them said they would approach the problem manually, copying and pasting the required data into a spreadsheet or a text file.

5.1.2 Analysis

The participants were asked to rank the ease of use for the installation of the bookmarklet on a scale of 1 to 5, with 1 being "Really easy" and 5 being "Really difficult". On the average, the score was 1.63 The installation procedure was meant to be simple and easy: Dragging the button from the page to the toolbar. However, some of the participants had an issue with this since some browsers, for example Chrome, do not show their toolbars by default.

Installation was hassle-free. No need to restart browser (like in the case of plugins)

I think that should be the most simple way to install a bookmarklet. I think you should consider the case when the bookmark toolbar is hidden, and if it's possible, *show a tutorial video for beginner.*

On the intuitiveness of the selection procedure, the average score was 2.27 Some of the participants found the interface for modifying existing labels confusing, while some had issues when trying to select elements "behind" its child elements.

> *When the bookmarket is initially started, it does not intuitively guide the user as to how they are to go about choosing the elements they want from the page. Editing existing labels can be confusing.*

76.67% of the participants found the visual feedback for the selection procedure useful in understanding what the extractor would be extracting.

Viewing the extracted data seemed to be a problem for many of the participants. The average score given for the ease of viewing extracted data was at 2.73 This may be due to the way in which the page is updated after the data has been extracted and inserted into the database.

> *Nothing seems to have changed when I click "extract" a couple of times. Maybe include a "estimated time left" bar or progress bar? At least something to indicate whether the extracting is completed or still in progress or something went wrong and nothing was registered.*

In general, we find that the selection interface has been successful in aiding the process of annotation by giving visual feedback. However, there were still glaring issues in the interface used for viewing the extracted data. We intend to correct these cosmetic problems in the next version of the system.

5.2 Machine Learning evaluation

We want to be able to extract the same content from pages which have had a layout change. So our evaluation method involves attempting to extract data after a layout change by training a classifier for the previous.

For this evaluation, we have selected the archived pages of Digg from Web Archives. This is due to the fact that Digg has a front page with structured entries, very much like a search entry. Also, Digg has gone through several layout changes over the years, making it an excellent candidate to perform this experiment on.

5.2.1 Methodology

Since our system only uses the learnt models when the layout of the page changes (which is detected when the XPath does not retrieve any items), we have restricted the evaluation of the learnt models between layout changes.

We use old pages from web.archive.org to simulate pages that have not undergone a layout change. To define a layout change, we use the tree-edit distance (Zhang & Shasha, 1989) between two pages as a simple metric. In order to find layout changes within the list of pages we retrieved from Web Archive, we took the tree-edit distance of the DOM tree between every pair of consecutive pages. The graph of these values is shown in Figure 5.1.

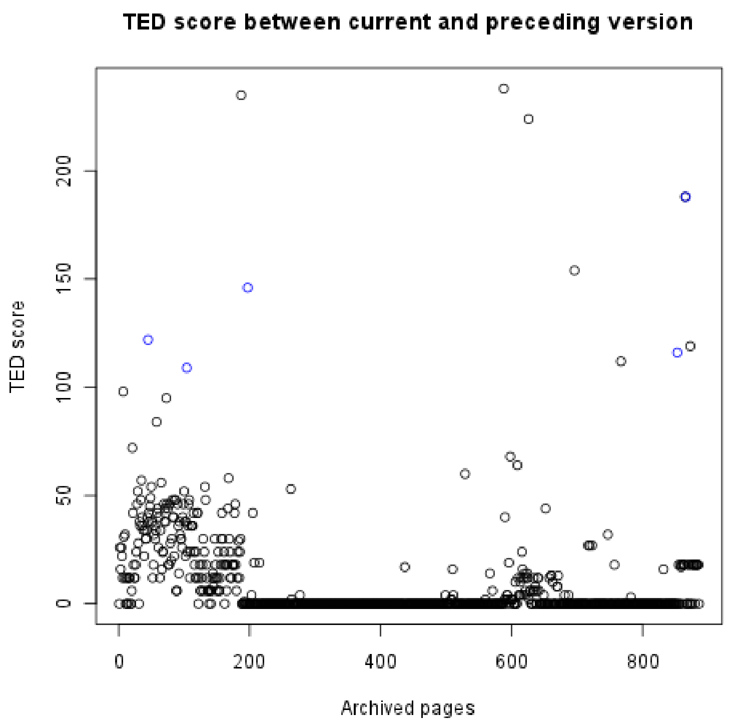


Figure 5.1: Graph of Tree Edit Distance between adjacent pages

The spikes visible in the graph were signals of a high tree-edit distance between the pairs of pages, and signals some sort of layout change. What we are looking for, however, are changes in which the XPaths fail to work. For each of the labels we are hoping to extract, we check if the application of the XPath yields any resulting DOM nodes. If they do not, the page was considered a layout change, and the pages prior the change were randomly sampled for 20 instances for learning. On the graph, these nodes are shown in blue.

We created 5 sets of 20 pages from the pages retrieved from web then trained the classifier on each of these. The test results were then compared to the data extracted if an XPath was created manually for each of these sets.

5.2.2 Analysis

Before we present our results, we compare our results to the performance of the robust wrappers in (Dalvi2009) by estimating our robustness performance if we performed the same experiment.

Their work aims to produce robust XPath wrappers resilient to changes in layout by measuring the probability of different DOM edits in past pages from the Web Archive[1]

We differ from their evaluation methods in the choosing of test corpus. In their testing set, they have used 3 pages for learning, while testing their wrapper on 15 pages, each 2 months apart, without taking into account if the pages have a change in layout.

We intend to compare our performance by estimating how well our classification method will do, given the same method of evaluation.

Estimation method

Due to the popularity of Digg compared to IMDB, the page has been indexed more by Web Archives. In our data set, we have 895 pages downloaded from IMDB. From this, we extracted 15 snapshots with 2 month intervals. We then calculated the accuracy of the extractor in the following manner:

It is unclear from the paper what XPath is defined as working, so we assume that the

[1]`http://web.archive.org`

Algorithm 2 Calculating performance of a wrapper over an interval

```
1: if $D^{(t)} \to D^{(t+1)}$ has layout change then
2:     $M \leftarrow$ models trained using batch corresponding to $D^{(t)}$
3:     $w \leftarrow$ models with $F_1$ score above $\alpha$ in $M$
4:     return $\frac{w}{|M|}$
5: else
6:     return 100% accuracy (this is the way we have defined the different datasets.)
7: end if
```

Interval	Change state	Robustness
$D^{(1)} \to D^{(2)}$	Change	75%
$D^{(2)} \to D^{(3)}$	No Change	100%
$D^{(3)} \to D^{(4)}$	Change	0%
$D^{(4)} \to D^{(5)}$	No Change	100%
$D^{(5)} \to D^{(6)}$	No Change	100%
$D^{(6)} \to D^{(7)}$	No Change	100%
$D^{(7)} \to D^{(8)}$	No Change	100%
$D^{(8)} \to D^{(9)}$	No Change	100%
$D^{(9)} \to D^{(10)}$	No Change	100%
$D^{(10)} \to D^{(11)}$	No Change	100%
$D^{(11)} \to D^{(12)}$	No Change	100%
$D^{(12)} \to D^{(13)}$	No Change	100%
$D^{(13)} \to D^{(14)}$	No Change	100%
$D^{(14)} \to D^{(15)}$	No Change	100%

Table 5.1: $D^{(t)}$ intervals and their corresponding layout changes

XPath fails when it returns no corresponding element. In our learned model, our models do not generally work completely (retrieving a 100% of all items), nor do they fail completely. Therefore, in order to estimate a result, we use a threshold α on the F_1 measure of the model. If $F_1 > \alpha$, then we consider the model to be 'working'.

We have set $\alpha = 90\%$ as a threshold for our models. Since for our dataset the number of extracted items are the same (15) for each page, we have simply calculated the percentage of the models that would work.

Using the numbers in Table 5.1 , we arrive at an overall precision of 91.07%. This result is significantly better than the robust XPath wrappers generated using their method. However, it should be noted that this result is estimated, and the dataset used in this experiment may be drastically different from that of the IMDB archive. Also, this method is slightly more forgiving since the model is still considered "working" even though it does not have a 100% precision and

Batch	Label	TP	FP	FN	TN		Precision	Recall	F_1	Accuracy
0	TITLE	282	0	3	5803		100.00%	98.95%	99.47%	99.95%
0	AUTHOR	276	17	8	5792		94.20%	97.18%	95.67%	99.59%
0	SUMMARY	284	18	1	5783		94.04%	99.65%	96.76%	99.69%
0	DIGG COUNTS	0	0	285	6085		N/A	0.00%	N/A	95.53%
1	TITLE	149	150	1	9557		49.83%	99.33%	66.37%	98.47%
1	AUTHOR	0	150	150	9706		0.00%	0.00%	N/A	97.00%
1	SUMMARY	0	295	150	9561		0.00%	0.00%	N/A	95.55%
1	DIGG COUNTS	0	180	150	9676		0.00%	0.00%	N/A	96.70%
2	TITLE	299	309	1	11319		49.18%	99.67%	65.86%	97.40%
2	AUTHOR	0	1	300	11926		0.00%	0.00%	N/A	97.54%
2	SUMMARY	0	0	247	11927		N/A	0.00%	N/A	97.97%
2	DIGG COUNTS	0	0	300	11927		N/A	0.00%	N/A	97.55%
3	TITLE	195	131	0	12146		59.82%	100.00%	74.86%	98.95%
3	AUTHOR	288	289	12	11895		49.91%	96.00%	65.68%	97.59%
3	SUMMARY	0	6	229	12466		0.00%	0.00%	N/A	98.15%
3	DIGG COUNTS	300	67	0	12105		81.74%	100.00%	89.96%	99.46%
4	TITLE	0	300	300	17815		0.00%	0.00%	N/A	96.74%
4	AUTHOR	248	300	52	17567		45.26%	82.67%	58.49%	98.06%
4	SUMMARY	0	0	300	18115		N/A	0.00%	N/A	98.37%
4	DIGG COUNTS	300	1643	0	16172		15.44%	100.00%	26.75%	90.93%

Table 5.2: Learnt model evaluation results for digg.com

recall.

We have also performed a stricter form of evaluation using our data set that uses pages grouped by layout changes. This means the model is used only after a drastic layout change that causes the XPath to no longer work.

The results of our method of evaluation is shown in Table 5.2. Aside from the first layout change, the other layout changes sometimes do not extract anything at all. This is probably due to the fact that Digg underwent a significant layout change that rearranged the locations of the individual items within each entry, rendering the model learnt useless. Also, some of the labels, like the article poster's username, are harder to learn than others, since not many defining features could be extracted. Of the results retrieved, only TITLE, AUTHOR and SUMMARY for Batch 0. The learner seems to have a problem dealing with elements with short text content. Also, the poor performance is partly by the classifier selecting elements that are either the direct child or parent of the target node. This is due to the fact that the content in the tags are almost the same, since the wrongly selected tag may either be an immediate ancestor or descendant of the wanted node. At the very least however, the data collected suggests that our approach achieves comparable results with the state-of-the-art.

Chapter 6

Conclusion

As of December 2010, there are 255 million websites on the internet, of which 21.4 million were created in 2010[1]. Our high level goal: to allow users to easily and simply extract any information from any of these websites. The sheer size of the web makes this task seem almost insurmountable.

While this primary goal has many challenges, in this report, we have chosen to address two challenges to pave the way towards a more user-centric web IE system. To provide users with a more accessible user interface, grab*smart* utilises a simple but effective alignment algorithm we call PTA through an interactive and visual experience for its users to specify the information they want extracted To create a more robust wrapper that well seamlessly work even after website layout redesign.

Overall, our user evaluation shows that the visual selection interface is doing well, while the machine learnt classifier performs comparably to another work in robust web IE. These are promising results and more can be done to improve upon them.

There are, however, limitations with the current implementation, such as the poor performance of the learnt model when it comes to non-header elements, or elements containing single worded content. Also, the system captures information at the HTML tag level, but more useful information may lie within the text content inside the tags.

[1]`http://mashable.com/2011/01/25/internet-size-infographic`

6.1 Contributions

grab*smart* it has the following advantages when compared to other Web IE solutions:

1. It provides an immediate visual feedback when the user does the annotation process.

2. It is: Easy to install, relatively simple to use.

3. It provides an alternative to the rigidness of the generated XPath with the decision tree wrapper alternative.

4. It employs our proposed method for automating the detection layout changes given a chain of archived pages using the Zhang-Shasha tree edit distance.

6.2 Future Work

With the *grabsmart* system only tackling a few of the problems in User-centric Web IE, it leaves much room for future work to be done:

6.2.1 User Interface

The bookmarklet, should consider more features of the selected elements. For example, the current implementation only takes into account the attribute values of the elements. Since the XPath syntax also allows for regular expression predicates, it is concievable for a method to create a regex pattern common to all selected items. This would give a more "content-based" XPath that may be more selective.

The web application, being less of the focus in this project, could have more functionality built-in. Since the extracted data is already present on the database, one could create ways in which this data could be manipulated and displayed. Currently, only the data from the latest extraction is available. The data from the previous extractions could be used for comparisons with the current extracted data, examples of such data include Linux distribution download rankings or book prices.

6.2.2 Machine learning component

For this project, we have made use of a library from the Apache project known as HtmlUnit. This library provides us with a "headless browser" from which to access the annotated pages, but it has several drawbacks. Unlike a true browser, it is unable to parse the CSS associated with the page, as well as the Javascript. This deprives us of many usefull visual features that would be obvious to the user. An alternative to this would be the WebKit library, since it is a widely used rendering engine. A headless instance of WebKit could function like a browser, allowing us to query the the page for visual features that would be useful for classifying HTML elements.

6.2.3 User-centric Web IE

The next step forward for the system would be to do focussed crawling through the site for a "chain" of pages that the user might want. Such a system is a viable research problem that could be looked into.

Since the focus of many scraping systems are lists of structured data, many of these are paginated, and one of the limitations of our system is that the extractor does not go beyond the pages the user has annotated. Being able to infer a navigation pattern that goes through this paginated data, and then extracting this automatically would be useful for users who do not know how to write a scraper.

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.

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.

Needleman, S., & Wunsch, C. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, *48*(3), 1970, 443–453.

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.

Zhang, K., & Shasha, D. (1989). Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Comput.*, *18*(6), 1989, 1245–1262.

Appendix A

Evaluation

A.1 Bookstore scenario

Now that you have familiarised yourself with the basic usage of the system, we can move on to our first scenario.

Book Depository is an online bookstore that provides free shipping to many countries round the world. This gives it a slight edge over the popular online bookstore Amazon. It's site is a standard e-commerce site, with standard capabilities such as browsing and searching, and sorting search results.

Imagine you are interested in up and coming mathematics books, but, unfortunately, Book Depository does not provide an active feed of such data as it is keyed into its database.

Use the system to create an extractor that provides a feed that would supply the needed information by extracting the relevant data from the e-bookstore. The feed must provide the data in order of latest to oldest.

A.2 Survey & Feedback results

A.2.1 Preliminary questions

How do you use the Internet to keep up to date with topics you are interested in?

Option		Freq	%
I visit the related sites of such topics regularly.		20	66.67%
I use RSS feeds and a feed reader to keep up-to-date.		6	20.00%
The sites I frequent usually have a ”Subscribe to newsletter” function.		0	0.00%
I don’t use the web for this purpose.		1	3.33%
Other		3	10.00%

Have you heard of web information extraction?

Option		Freq	%
Yes		10	33.33%
No		20	66.67%

If you know what a screen-scraping or a wrapper for web information extraction is, have you ever written one?

Option		Freq	%
Yes		2	6.67%
No		28	93.33%

Have you used any of the systems below or other web information extraction systems in the past?

Option		Freq	%
Mozenda		0	0.00%
Needlebase		0	0.00%
MIT’s Solvent		0	0.00%
I have never used such systems before.		29	96.67%
Other		1	3.33%

A.2.2 System evaluation questions

On a scale of 1 to 5, how difficult was the installation procedure?

Option		Freq	%
1-Easy		20	66.67%
2		3	10.00%
3		5	16.67%
4		2	6.67%
5-Really Hard		0	0.00%

Is this the first time you have used a bookmarklet?

Option		Freq	%
Yes		18	10.00%
No		12	40.00%

On a scale of 1 to 5, how difficult was selecting items from the page?

Option		Freq	%
1-No problems		9	30.00%
2		8	26.67%
3		9	30.00%
4		4	13.33%
5-Couldn’t do it at all		0	0.00%

Was the visual feedback helpful in determining the data that will be extracted by the system?

Option		Freq	%
Yes		23	76.67%
No		4	13.33%
Other		3	10.00%

On a scale of 1 to 5, how intuitive was the site in terms of viewing the data that you've selected and want to extract?

Option		Freq	%
1-Easy to get to the data		3	10.00%
2		11	36.67%
3		8	26.67%
4		7	23.33%
5-I didn't know I could view the data		1	3.33%

Was the data extracted presented in a useful format?

Option		Freq	%
Yes		24	80.00%
No		6	20.00%