UROP Progress Report: User-centric Webpage Information Extraction

Shawn Tan
U096883L

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*.

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.

The system presented in this report attempts to solve these issues by providing an intuitive interface for labelling that cuts down the amount of annotation required from the user, and also provides a robust information extraction framework by using machine learning to create more robust extraction models.

In the next section, I will discuss previous work done in information extraction and highlight some of the more recent work closely related to ours. In Section 3, a detailed description of the system developed will be given. Section 4 will cover some of the evaluation that has been done with the methods, and Section 5 will conclude the report with future steps that are required.

2 Related Work

Initially, web IE involved manually written wrappers that catered to specific needs and information sources [**?**, **?**] . 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 [**?**], using several different approaches to the problem (e.g. machine learning) [**?**, **?**]. Subsequently, from

2001, systems that required no user intervention were developed [**?**, **?**, **?**]. 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" [**?**]. 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 [**?**]. 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[**?**]. The system was then improved to include usage of inter and intra similarities between pages as features[**?**]. 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 [**?**].

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[**?**]. 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. Method In this section, I will present a user-centric system for IE that attempts to tackle the issues of robustness when extracting web content. The system comes in the form of a web application, and uses a bookmarklet interface for annotation from the user.

Looking at a 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.

The goal is to simplify this process for users of the system. In order to make it accessible, it is in the form of a web application, with a labelling interface available a bookmarklet. When the user arrives at an interesting site, he or she would then click on the available bookmarklet to begin labelling positive examples to be extracted. The bookmarklet will reduce the amount of labelling done by predicting the wanted items from the page, and this is described in Section 4. The result will then be sent back to the web application to be processed. The system will then take care of updating the feeds by extracting data from the site at regular intervals. A classification model, which is described in Section 5, is used to ensure that the extraction process is resistant to changes in the structure of the HTML page. This aims to reduce the time taken to reimplement a wrapper in case when the page is changed.

3 System overview

The system comprises of 3 main components.

On the client-side, the **bookmarklet interface** allows the user to label the elements on the page he is interested in. This interface makes use of an algorithm to reduce the number of positive examples the user has to select. In this way, the labelling of documents in order to learn a classification model becomes a less tedious process.

Once the user is done with the labelling process, an XPath string is sent back to the **application server**. This component of the system provides the user interface, and at the same time has a web API to enable communication between the bookmarklet interface and the application.

The XPaths and URLs recieved will be stored on a local database, and accessed at regular intervals by the **processor**. This component will then crawl the site of interest, looking for pages similar to the one of interest to the user. From these, the machine learner will create a model of what the user has selected, which is then stored for later use. At the same time, the extacted information will then be stored on the database, and presented the next time the user accesses the web application.

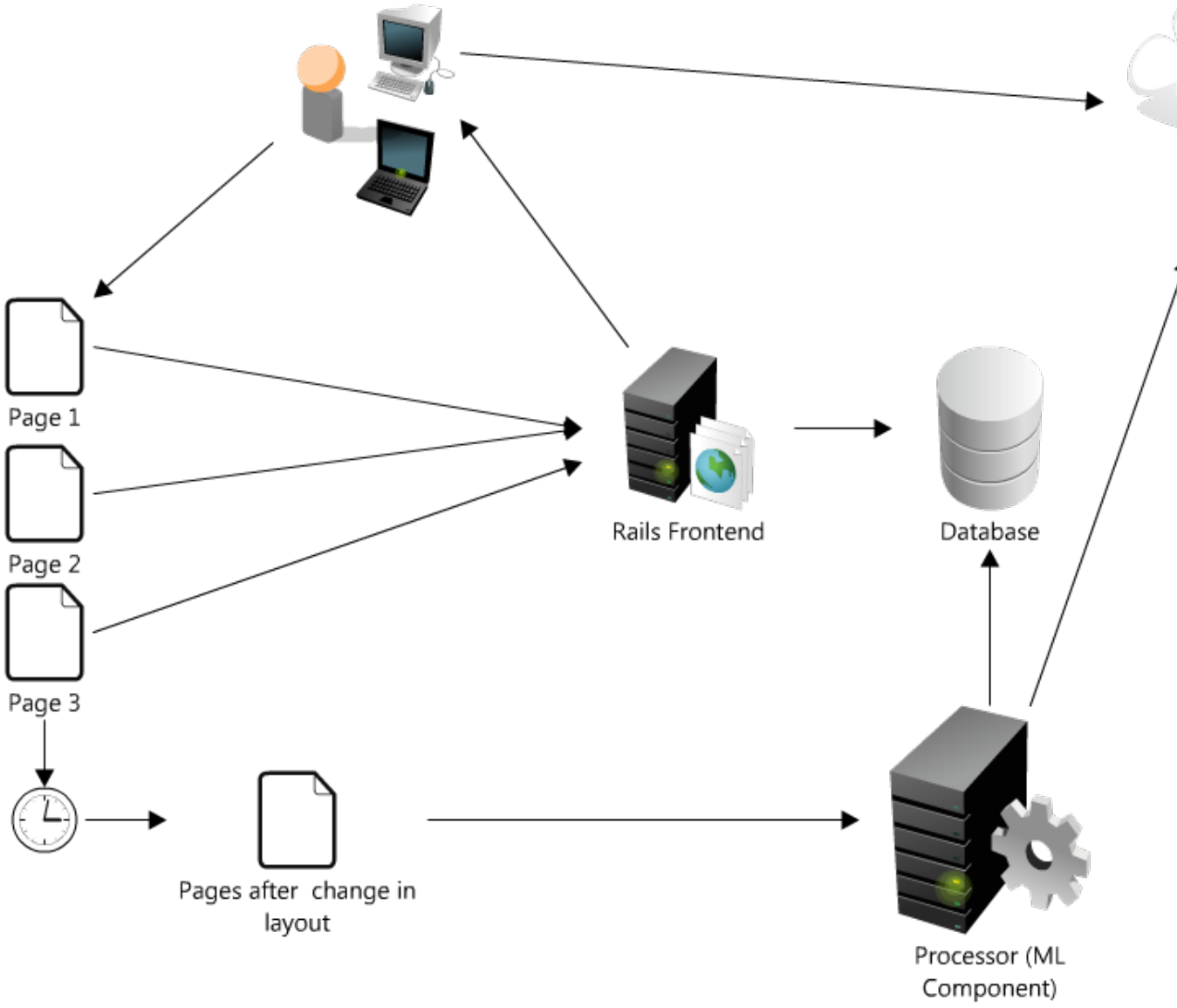

Figure 1: Overview of the system

4 Bookmarklet Interface

For greater automation, the bookmarklet interface attempts to reduce the amount of labeling 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.

Algorithm 1 $\textsc{lcas}(P_1, P_2)$

```
1: M ← new array[LENGTH(P_1) + 1][LENGTH(P_2) + 1]
2: for i ← 1 to LENGTH(P_1) + 1 do
3:   for j ← 1 to LENGTH(P_2) + 1 do
4:     E_1 ← P_1[i], E_2 ← P_2[j], E_p ← {}, M[i, j] ← E_p
5:     for a ∈ A do
6:       if a(E_1) = a(E_2) then
7:         E_p ← E_p + {a(E_1)}
8:         SCORE(E_p) ← SCORE(E_p) + 1
9:       end if
10:    end for
11:    if SCORE(E_p) > 0 then
12:      SCORE(E_p) ← SCORE(M[i − 1][j − 1])
13:    else
14:      SCORE(E_p) ← MAX(SCORE(M[i − 1][j]), SCORE(M[i][j − 1]))
15:    end if
16:    M[i, j] ← E_p
17:  end for
18: end for
```

Figure 2: The *longest common ancestor algorithm*

An important aspect which describes tags which are visually similar is its "ancestry". 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.

Given arrays of elements representing paths to 2 elements P_1 and P_2, and each element E as a set of attributes a in the form $a : E \to V$, where V is the set of attribute values, the algorithm finds the longest common sequence (LCS) of ancestors for the selected elements. Figure 2 gives the pseudocode for the *longest common ancestor algorithm* (LCAS). Modifications were made to LCS in order to find common elements within the 2 element's paths.

This process is a bottom-up approach, since 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 similarties 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.

P_1	div	div	div	ul	li	h3	a
id	mainContent						
index	4	1	1	3	1	1	1
class		content	newsl, pagedNewsList	newslTop			
P_2	div	div	div	ul	li	h3	a
id	mainContent						
index	4	1	1	3	2	1	1
class		content	newsl, pagedNewsList	newslTop			
			After LCAS:				
Result	div	div	div	ul	li	h3	a
id	mainContent						
index	4	1	1	3		1	1
class		content	newsl, pagedNewsList	newslTop			

Figure 3: 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 4 is an example of a diagram 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.

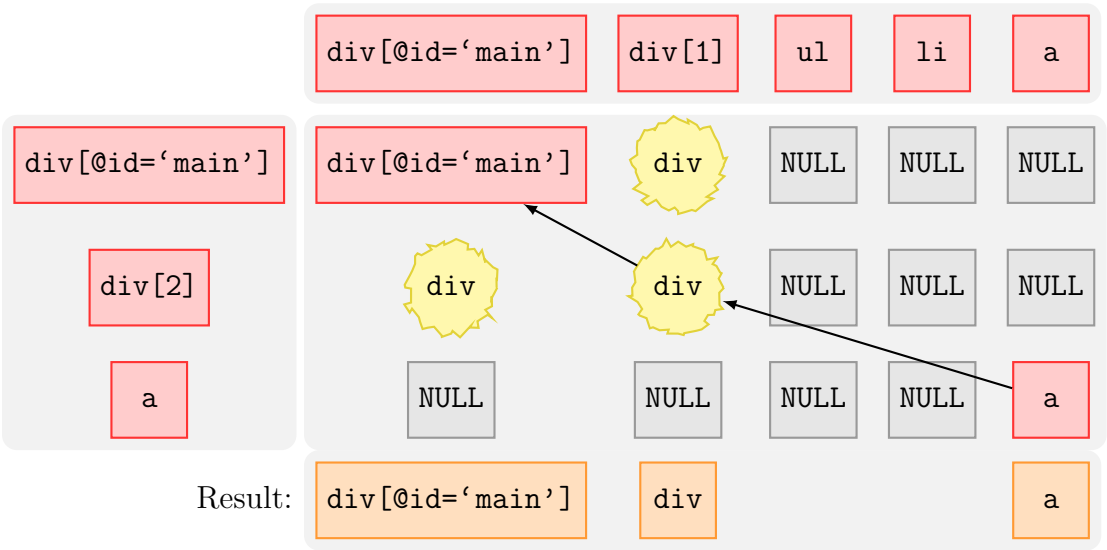

Figure 4: Table generated by the LCAS algorithm

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.

5 Learning the model

Once the XPath is sent from the bookmarklet back to the server-side application and inserted into the database, 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.

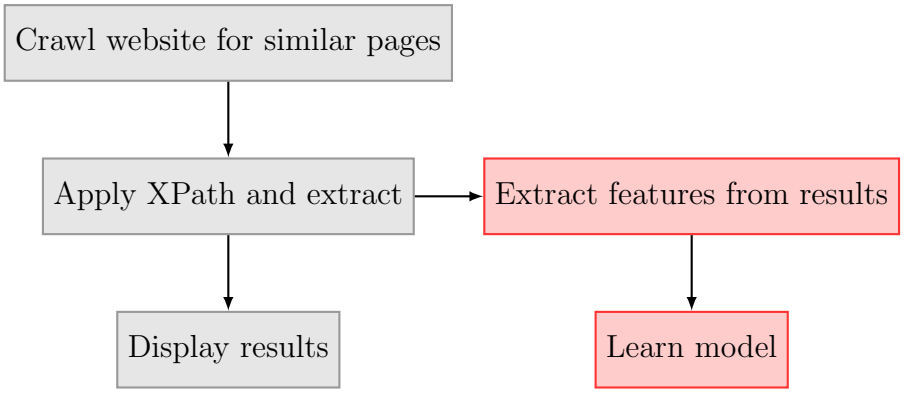

Figure 5: Workflow of the machine learning component

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. In order 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 (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 6 shows a decision tree when the process is run on a Google search result with the result entries selected for extraction.

There are still some issues with the methods describe above. The items extracted from the page are still at the HTML element level. Extraction of smaller units of data which require some Natural Language Processing have not been implemented. Also, the data extracted may not be displayed in the same order as it was displayed on the original site.

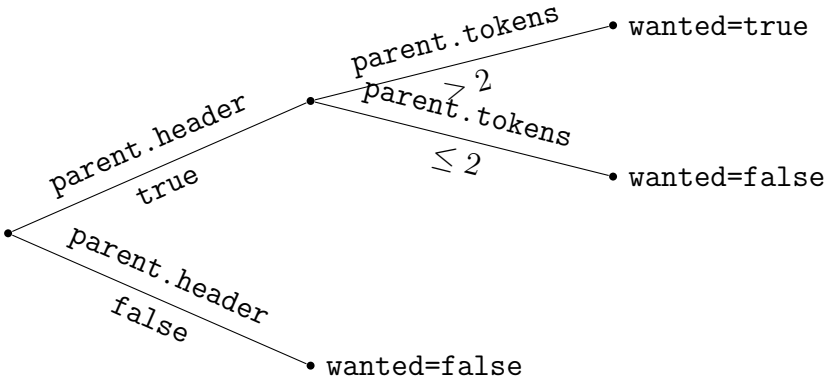

Figure 6: Decision tree generated when search entries on Google are selected.

6 Evaluation

At the writing of this report, no formal evaluation of the methods described have been carried out. However, some preliminary tests have been done for several sites, and the results of the methods described in Sections 4 and 5 look promising.

Figure 7: User's view when selecting examples

Some tests done using the bookmarklet interface show that the algorithm implemented is quite effective in predicting elements wanted by users. This is especially so when the page in question is highly structured, with many similar repeating elements. This may be due to the strict formatting done when users define a template for their Content Management System, and as a result, class names and attributes for similar elements are consistently reused for headlines, entries etc. Figure 7 shows the view of the user when selecting items from a Google search result. The framework used for robust extraction from websites also work well for some sites. On Google search results, the decision tree generated is able to capture most of the entries returned. However, there are some results which do not conform to the constraints specified by the decision tree, and are not selected. This suggests some additional fine tuning must be done to the features

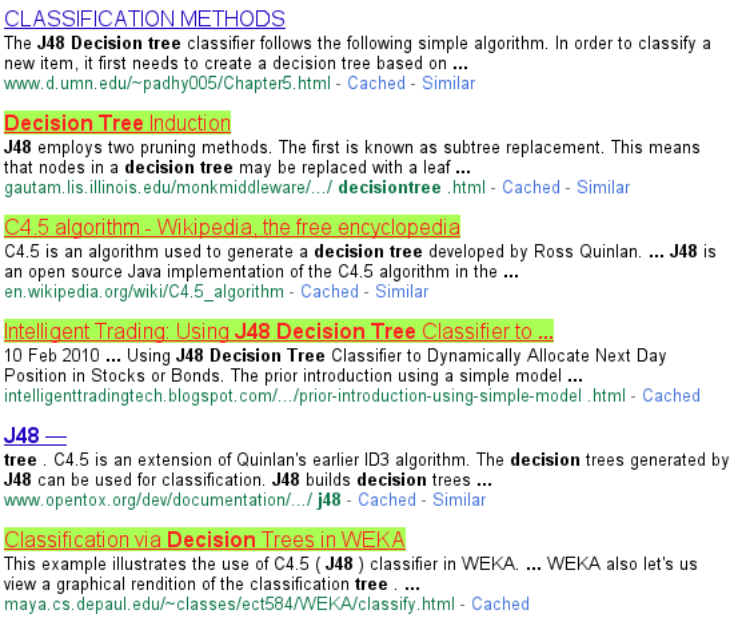

Figure 8: User's view when selecting examples

extracted. Figure 8 shows 2 items not picked up by the rule.

7 Future Work & 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.