User-centric Web Information Extraction

Shawn Tan* and Kan Min-Yen†
Department of Computer Science,
School of Computing,
National University of Singapore

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 evaluation results show that the framework may not perform well under all circumstances, but is comparable to related work in robust wrapper induction.

1. INTRODUCTION

Web Information Extraction (Web IE) is extracting data from the web from an unstructured or semi-structured into a structured form. Many web IE systems have been developed, mainly for corporations, since they are commonly used to extract huge amounts of data from the web. Also, process of getting them up and running takes quite a bit of time as it involves many hours of training and labelling work.

However, for the users who brows the web for leisure do not have access to such resources. 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.

Our primary goal of user-centric web IE is to offer an 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: 1. No need to know how to program and 2. The system must work seamlessly even at the even after a layout change.

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

1. Provide a visual interface for labelling that gives the user immediate visual feedback as to the items that will be extracted.

2. Provide a robust frame wark for extraction of the selected information using machine learning. Users would have their wrappers work seamlessly after layout changes.

*Student
†Supervisor

2. RELATED WORK

2.1. Visual Feedback for User Annotation

Lixto was one of the first systems that has tried to make the selection of items a more visual process (Baumgartner, Flesca, & Gottlob, 2001). Systems like *Lixto* usually provide users a point-and-click interface in which they can specify items that they want extracted. Other examples from commercial products include Mozenda and Needlebase. MIT's solvent , which is part of a mash-up creation system, also provides users with a similar visual feedback system. These systems are useful because it allows the user to make any required corrections immediately, but, the abovementioned systems, with the exception of Needlebase, requires users to have customised software in order to perform this labelling. In addition, all 3 systems required users to have some experience with programming.

In contrast, our proposed system, grab*smart*, we aim to allow users to have a sense of what will be extracted, with the added advantage of keeping the installation simple.

2.1.1. XPath Generation & Robust Wrappers

As the medium for transmitting the selected items back and forth between the bookmarklet and they browser, we have chosen XPath due to its simplicity, and the ubiquity of XPath as a query language for many HTML parsers.

However, the generation of XPaths is still the responsibility of the user, and may introduce human error. A method for extracting XPaths is described in (Anton, 2005). 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. The method creates a robust XPath, by taking note of how the page of the site changes over time.

There were also other systems not based on XPath that dealt with extracting data independent of the HTML DOM structure. One method of approaching this was to make use of the visual features of the page. PARCELS (Lee, Kan, & Lai, 2004) looked at classifying page blocks, which were chunks of pages that were made up of 1 or more HTML tag. Gatterbauer also approached the problem of extracting data from tables on pages by analysing its 2D output (Gatterbauer, Bohunsky, Herzog, Krüpl, & Pollak, 2007).

The issue with these methods are that the XPath generated is not immediate, and the user is unable to know what will be extracted using the XPath until it is extracted.

In contrast, grab*smart* will generate XPaths as well, but will be able to do this fast, so that the user can be updated via visual changes to the interface. In order to still be robust, our system will employ machine learning to learn a model for extraction, and use this once the generated XPath fails to work.

For the rest of the paper, we will define users of the system as users who do not have access to resources required by current IE systems. Also, we will define robustness as the wrapper's resistance to changes in HTML structre from the time that the wrapper was derived.

3. METHOD & IMPLEMENTATION

3.1. Path Traversal Alignment (PTA) & The Selection Interface

A tag's traversal path from the `<BODY>` tag is an important aspect which describes tags which are visually similar. This also implies that the elements tend to be items that the user would like to extract (e.g. search result entries).

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.

We make the following modification 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 aparameter. This is due to the slight differences in which the rejection path is calculated.

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.

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.

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 $\lambda(P_i, Q_j) = Q_j - (P_i \cap Q_j)$.

One of the limitations of this method 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. Web Application & Machine Learning Component

Once the user has made use of the bookmarklet to select a suitable extraction region, the corresponding XPath for his selection region would be sent back to the web application written in Ruby on Rails. In order to be able to perform any kind of extraction, the user would have to navigate to our web application. He should then be presented with his dashboard, on which he would be able too see a list of all **extractors** he has created. From here, he can select one of the extractors to view in detail.

The detailed view of the extractor shows several items:

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.

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.

Here, the user can also choose to do an extraction now, or set a daily time for the extraction to be done. When the system performs an extraction, it also creates a model for future use if the XPath it is relying on fails.

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) and Header (Discrete).

In order to search for more instances for learning, we crawl the site for more similar pages. However, the elements returned from applying the XPath on these new crawled pages are not extracted, but merely used for learning.

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 set number of instances, which is now set at 500.

The entire process of crawling more pages for more learning instances is transparent to the user. Also, when the layout does change, the switch to the learned model is also seamless, and the user will be able to find the extracted data just in the same way if the XPath has extracted it.

4. EVALUATION

4.1. 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.

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

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.

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.

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.

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.

4.2. Machine Learning evaluation

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.

4.2.1. 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. Huge spikes in these values tend to indicate a layout change. We checked these, and noted those in which the old XPaths failed to work.

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.

4.2.2. Analysis

The results of the experiment are in Figure **??**. 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 caused 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.

We also compare our results to those in Dalvi, Bohannon, & Sha (2009) by estimating our robustness performance if we performed the same experiment. We intend to compare our performance by estimating how well our classification method will do, given the same method of evaluation.

4.2.3. 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 number of 'hits' each model got when applied from

We have set $\alpha = 90\%$ as a threshold for our models. That is, if a model has an F_1 score $> \alpha$, the model is considered a 'hit'. 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. There are four models in total.

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 recall.

At the very least, the data collected suggests that our approach achieves comparable results with the state-of-the-art.

5. CONCLUSION

While our 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 also other 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.

Our system, grab*smart*, 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. REFERENCES

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

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

[3] 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.

[4] 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.

[5] 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.

[6] 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.