Automatic Web-Scale Information Extraction

Philip Bohannon
Yahoo! Research,
Santa Clara, CA.
plb@yahoo-inc.com

Nilesh Dalvi
Yahoo! Research,
Santa Clara, CA.
ndalvi@yahoo-inc.com

Yuval Filmus *
Dept. of Computer Science,
University of Toronto, Canada.
yuval.filmus@utoronto.ca

Nori Jacoby
Yahoo!
Tel Aviv, Israel.
nori@yahoo-inc.com

Sathiya Keerthi
Yahoo! Research,
Santa Clara, CA.
selvarak@yahoo-inc.com

Alok Kirpal
Yahoo! Research,
Santa Clara, CA.
kirpal@yahoo-inc.com

ABSTRACT

In this demonstration, we showcase the technologies that we are building at Yahoo! for Web-scale Information Extraction. Given any new Website, containing semi-structured information about a pre-specified set of schemas, we show how to populate objects in the corresponding schema by automatically extracting information from the Website.

Categories and Subject Descriptors

H.0 [**Information Systems**]: General; H.3.3 [**Information Systems**]: Information Search and Retrieval

General Terms

Design, Experimentation, Performance

Keywords

Information Extraction, Web-Scale, Domain-centric

1. INTRODUCTION

One of the grand research challenges in the field of Information Extraction (IE) is to develop effective techniques for web-scale information extraction. In this demo, we showcase the technologies that we are building at Yahoo! towards this problem.

Traditional IE techniques considered in the database community tend to be *source-centric*, i.e., they can only be deployed to extract from a specific website or data source. However, a range of *web-centric* techniques have emerged recently [2, 4, 8, 9, 10, 16] that seek to look at extraction holistically on the entire web. All of these efforts have taken a *domain-independent* approach. E.g. WebTables [4, 8] extracts all simple tables and lists from the Web and store them as relational data. However, domain-independence makes it difficult to attach semantics to the extracted data. Furthermore, the amount of information that can be extracted is quite limited, since most of the information is outside of simple tables and lists. Web-scale extraction, at the level of precision and recall high enough to power real applications, has remained an open challenge.

In contrast, we have advocated [5] a *domain-centric* approach to the problem, where we want to extract all the entities and their attributes from the entire web restricting to a specific domain. For instance, one might be interested in constructing a database of all restaurants, along with their contact information, hours of operation and set of reviews, by extracting information from all the websites on the web that contain information about restaurants. Similarly, one might be interested in constructing databases of all books and their reviews, artists, albums and their discography, or product listings from merchants and so on. An example of a domain-centric extraction system is the DBLife project [7], which focuses on extracting information about Database researchers. However, DBLife was built on substantial manual effort, ranging from specifying a fixed set of sources to providing the extraction rules : a process that we seek to automate at web-scale.

The difference between a *web-centric* and a *domain-centric* approach is that, in the latter, we allow supervision at domain-level, with the objective of populating a specific schema by extracting from the entire web. Thus, if we are interested in constructing a database of restaurants from the web, we can specify the set of attributes that we are interested in, e.g. "name", "address" and "reviews", supply sample dictionaries or regular expressions or language models for attributes, specify domain knowledge like "businesses typically have a single phone number but multiple reviews", and so on. We believe that solving the domain-centric extraction can provide a promising stepping stone towards cracking the grand challenge of a general web-scale IE.

Nonetheless, even with domain-level supervision, Web-scale extraction proves to be an incredibly hard problem. There are several aspects to a complete solution for this end-to-end challenge, which include discovering and identifying websites that contain information of interest, analyzing the websites to identify the correct subset or cluster of pages to

*Work done while the author was at Yahoo! Research, Tel Aviv.

SIGMOD '12 May 20–24, 2012, Scottsdale, Arizona, USA.

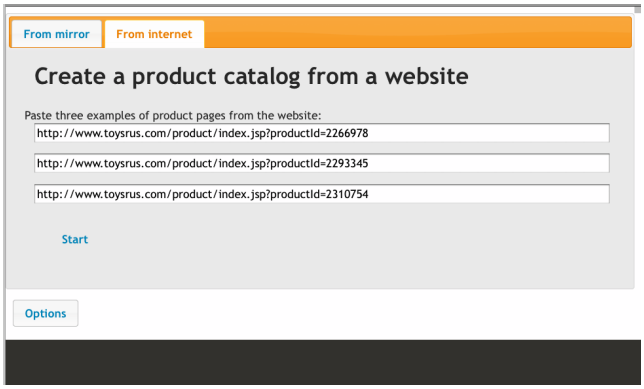

Figure 1: Start Page For A New Website

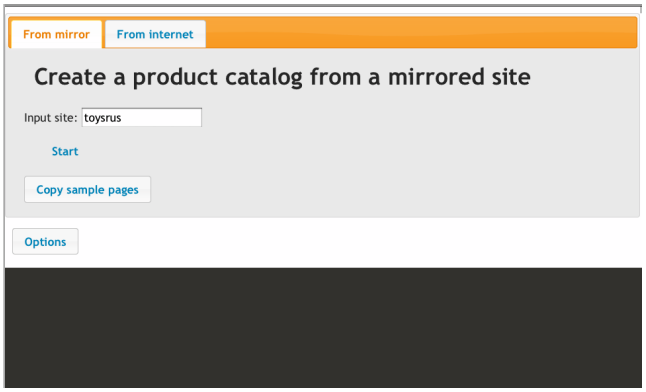

Figure 2: Start Page For A Preprocessed Website

extract from, automatically learning extraction rules, deduplication and linking, all at the scale and diversity of the web. In our recent works [6, 3], we have been developing techniques to address some of these research problems. By building upon these techniques, we will demonstrate the potential of this approach using our system prototype.

2. TECHNOLOGY

In this prototype, we assume a prespecified schema from a domain of interest. The objective of the system is to be able to take any new Website containing information from the domain, and automatically populate the schema by extracting from the entire website. Note that the system only requires human input at the domain level : there is no website-level supervision. We describe the main components below :

2.1 Clusterer

A website typically contains several "*types*" of pages. For example, the website `yelp.com` has restaurant details pages, users pages, list pages, events pages, and so on. The clustering module analyzes the website and organizes the pages of the site into clusters corresponding to these types. This is an important step for two reasons. First, it lets us identify the target set of pages to extract from the site. Second, we can exploit the structural similarity of pages from the same cluster to learn a single, high-quality extraction rule for each cluster.

This component is based on our recent work on efficient structural clustering of websites [3]. The technique relies on URLs, in conjunction with very simple content features, which makes them extremely fast and work easily on websites with millions of pages. Our main observation is that simple pairwise similarity measures between URLs are not meaningful. E.g. consider the following urls:

u_1 : `site.com/CA/SanFrancisco/eats/id1.html`
u_2 : `site.com/WA/Seattle/eats/id2.html`
u_3 : `site.com/WA/Seattle/todo/id3.html`
u_4 : `site.com/WA/Portland/eats/id4.html`

Suppose the site has two types of pages : *eats* pages containing restaurants in each city, and *todo* pages containing activities in each city. We will ideally like to cluster the site into these two clusters. In terms of string similarity, u_2 is much closer to u_3, an url from a different cluster, than the url u_1 from the same cluster. Thus, we need to look at the set of urls holistically, and cannot rely on string similarities for clustering. Our technique [3] is based on the principles of information theory, and constructs a set of patterns that offer the *simplest explanation* for the observed set of URLs/content.

2.2 Annotators

Once we have a set of clusters, we want to automatically annotate a small sample of pages with the attributes of interest in order to learn rules for extraction. This is the only component of the system that is domain-specific : given a new domain, we only need to write annotators for each of the attributes that we are interested in.

We use several different kinds of annotators. A dictionary-based annotator looks for the presence of items from a precompiled dictionary. For instance, we can create a dictionary of restaurant names based on what we have extracted so far, and given a new website, we can annotate restaurant names that overlap with our dictionary. A pattern-based annotator is effective for attributes like phone numbers, dates and prices. A language-model based annotator can be used for detecting reviews, product descriptions, etc. In our annotators, we also use HTML cues like the page title, headers and meta-tags to identify the main content of the page, as well as identify attributes. For this demo, we have written manual rules for pattern-based annotators, and have used a small corpus of reviews to train a language-model based annotator for reviews.

As we will explain in the next section, we only require *weak guarantees* from our annotators. They can have a low recall, since we only need a small sample of pages with annotations, and they can be noisy, since we are able to boost their accuracy quite effectively by looking at structural redundancies within websites. We find that with weak guarantees, it is easy to write annotators for a wide range of attributes across domains. In our experiments [6], we presented concrete results on how the weak guarantees of annotators translate into the overall performance of the extraction system. For instance, annotators with precision of 0.7 and a recall of 0.15 were sufficient for high quality extractions.

2.3 Extractors

The core component of our system is a learning algorithm [6] that learns high-quality, clean extraction rules from

a set of noisy annotations on structurally similar pages. Such a rule is also called a *wrapper* in the literature. While the problem of inducing wrappers from labeled examples has been extensively studied [13, 12, 11, 15, 14, 1], all the traditional wrapper induction algorithms assume a clean set of input annotations, and even one incorrect example breaks the algorithms completely. The ability of our system to learn from a noisy set of examples makes it very powerful, and allows us to use automatic annotators for scaling up extraction.

The main idea behind our algorithm [6] is to *generate-and-test*: given a set of labels that are noisy, we use an efficient enumeration algorithm to construct all possible distinct wrappers generated by subsets of the labels. Each such wrapper is ranked according to its quality, which depends on its likelihood of generating the (noisy) labels and its likelihood of being a good wrapper based on its structural coherency. The intuition is that if the label noise is not too excessive, then one of the generated wrappers will be trained on sufficiently many noise-free labels and our careful definition of quality will cause it to be ranked high.

3. DEMO EXPERIENCE

We will feature a *live* demo where audience can specify any website of their choice, and we will show the run of our system on the website. We will host the demo for two domains : *products* and *restaurants*. For products, we have written annotators for name, price, image and reviews, while for restaurants, we have annotators for name, address, phone, website and reviews.

A user can specify any website that contains information about entities from either of the two domains. There are two ways users can perform this step. First, they can select from a set of websites for which we have already run our entire pipeline and cached the results of each stage. We have chosen these websites to illustrate the interesting aspects of our system, and how it performs in a diverse range of webpages. Alternately, users can specify any new website. Users can enter a small set of URLs from the same website on which they will like to see the extractions. The sample interfaces for the two ways are shown in Figure 1 and Figure 2.

After a user specifies a website, the system crawls several pages from the website, and analyses them to form a set of clusters. The clusters correspond to the different types of pages present in the website. Figure 3 shows a snapshot of the site analysis. It shows the number of pages crawled and used in the analysis, the number of clusters found, URL patterns for clusters, sample pages from each cluster, etc. For example, two sample clusters it discovers from `toysrus.com` are

```
toysrus.com/products/*
rewardsus.toysrus.com/*
```

Next, the system uses its annotators to find annotations in the pages. Based on these annotations, it learns an extraction rule for each attribute in the schema at the cluster level. Figure 5 shows a sample snapshot of extraction rules that the system learned for `toysrus.com`. For instance, it infers that the XPath

```
//meta[@property="og:title"]/@content
```

can be used for extracting the product names for a specific

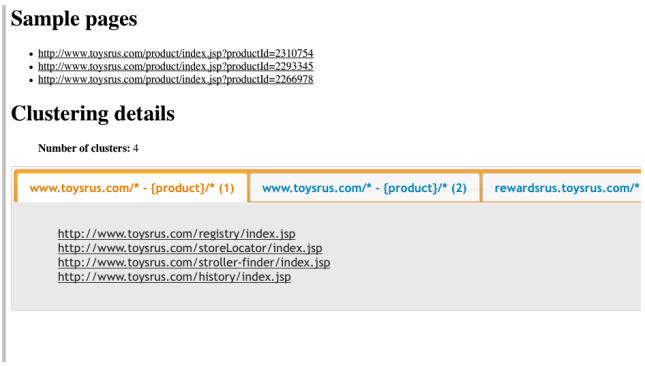

Figure 3: Clusters

Figure 4: Extracted Records

Figure 5: Extraction Rules

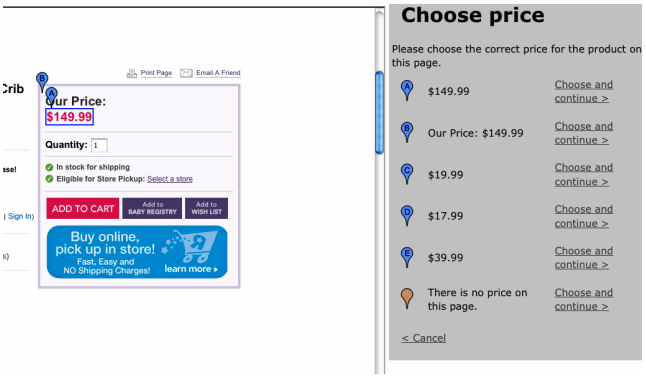

Figure 6: Fixing Extraction Errors

cluster in `toysrus.com`. Figure 4 shows a snapshot of the set of records that we extracted from a specific *toysrus.com* cluster. It shows the *name*, *price* and *image* extracted from each page. Any given website, in general, will contain a subset of attributes from our schema, and we show all the attributes that we were able to find and extract.

Finally, if the system is not able to extract a specific attribute, or extracts it incorrectly, the interface also provides a single-click functionality to fix the errors. Figure 6 shows a snapshot of this interface. Thus, the interface can also be used in a supervised setting, where one can extract from a website instantly with just a few (often zero) clicks.

4. REFERENCES

[1] Tobias Anton. Xpath-wrapper induction by generating tree traversal patterns. In *LWA*, pages 126–133, 2005.

[2] Michele Banko, Michael J. Cafarella, Stephen Soderland, Matthew Broadhead, and Oren Etzioni. Open information extraction from the web. In *International Joint Conference on Artificial Intelligence*, pages 2670–2676, 2007.

[3] Lorenzo Blanco, Nilesh N. Dalvi, and Ashwin Machanavajjhala. Highly efficient algorithms for structural clustering of large websites. In *WWW*, pages 437–446, 2011.

[4] Michael J. Cafarella, Alon Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. Webtables: exploring the power of tables on the web. *VLDB*, 1(1):538–549, 2008.

[5] Nilesh Dalvi, Ravi Kumar, Bo Pang, Raghu Ramakrishnan, Andrew Tomkins, Philip Bohannon, Sathiya Keerthi, and Srujana Merugu. A web of concepts (keynote). In *PODS*. Providence, Rhode Island, USA, June 2009.

[6] Nilesh N. Dalvi, Ravi Kumar, and Mohamed A. Soliman. Automatic wrappers for large scale web extraction. *PVLDB*, 4(4):219–230, 2011.

[7] Pedro DeRose, Warren Shen, Fei Chen, AnHai Doan, and Raghu Ramakrishnan. Building structured web community portals: A top-down, compositional, and incremental approach. In *VLDB*, pages 399–410, 2007.

[8] Hazem Elmeleegy, Jayant Madhavan, and Alon Y. Halevy. Harvesting relational tables from lists on the web. *PVLDB*, 2(1):1078–1089, 2009.

[9] Oren Etzioni, Michael Cafarella, Doug Downey, Stanley Kok, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S. Weld, and Alexander Yates. Web-scale information extraction in Knowitall: (preliminary results). In *WWW*, pages 100–110, 2004.

[10] Rahul Gupta and Sunita Sarawagi. Answering table augmentation queries from unstructured lists on the web. In *VLDB*, 2009.

[11] Wei Han, David Buttler, and Calton Pu. Wrapping web data into XML. *SIGMOD Record*, 30(3):33–38, 2001.

[12] Chun-Nan Hsu and Ming-Tzung Dung. Generating finite-state transducers for semi-structured data extraction from the web. *Information Systems*, 23(8):521–538, 1998.

[13] Nickolas Kushmerick, Daniel S. Weld, and Robert B. Doorenbos. Wrapper induction for information extraction. In *IJCAI*, pages 729–737, 1997.

[14] Jussi Myllymaki and Jared Jackson. Robust web data extraction with XML path expressions. Technical Report RJ 10245, IBM, 2002.

[15] Arnaud Sahuguet and Fabien Azavant. Building light-weight wrappers for legacy web data-sources using w4f. In *VLDB*, pages 738–741, 1999.

[16] Pierre Senellart, Avin Mittal, Daniel Muschick, Rémi Gilleron, and Marc Tommasi. Automatic wrapper induction from hidden-web sources with domain knowledge. In *WIDM*, pages 9–16, 2008.