Overview of AutoFeed: An Unsupervised Learning System for Generating Webfeeds

Bora Gazen and **Steven Minton**
Fetch Technologies
2041 Rosecrans Ave.
El Segundo, California, USA
{gazen,sminton}@fetch.com

Abstract

The AutoFeed system automatically extracts data from semi-structured web sites. Previously, researchers have developed two types of supervised learning approaches for extracting web data: methods that create precise, site-specific extraction rules and methods that learn less-precise site-independent extraction rules. In either case, significant training is required. AutoFeed follows a third, more ambitious approach, in which unsupervised learning is used to analyze sites and discover their structure. Our method relies on a set of heterogeneous "experts", each of which is capable of identifying certain types of generic structure. Each expert represents its discoveries as "hints". Based on these hints, our system clusters the pages and identifies semi-structured data that can be extracted. To identify a good clustering, we use a probabilistic model of the hint-generation process. This paper summarizes our formulation of the fully-automatic web-extraction problem, our clustering approach, and our results on a set of experiments.

Introduction

This paper describes a new approach to data extraction from the web (previously published in KCAP '05 (Gazen & Minton 2005)). Our approach uses unsupervised learning to analyze the structure of a site and its pages, with the objective of extracting and structuring the data on the site, so that the data can be transformed into a webfeed (such as an RSS feed). For example, from an e-commerce retail site we would like to be able to automatically create a product catalog webfeed. Other sites of interest would include news sites, classified ads, electronic journals, and so forth. This is an ambitious goal, since the system must discover the site's structure on its own. In contrast, the well-studied "wrapper induction" problem is much easier, since a wrapper induction system is only shown examples of a single page type, and moreover, a human marks up each example page.

The "site extraction problem" is a relatively natural problem, since sites are generally well-structured so that humans can easily navigate through a site. One possible approach would be to identify pages that share the same grammar, and then use the grammar to extract the data. As demonstrated by RoadRunner (Crescenzi, Mecca, & Merialdo 2001), if a human selects a set of similar example pages, grammar induction techniques may be able to automatically learn a grammar for those pages. Unfortunately, this is a chicken and egg problem; without knowing anything in advance about the data on the pages, it is difficult to automatically identify pages that have the same grammar.

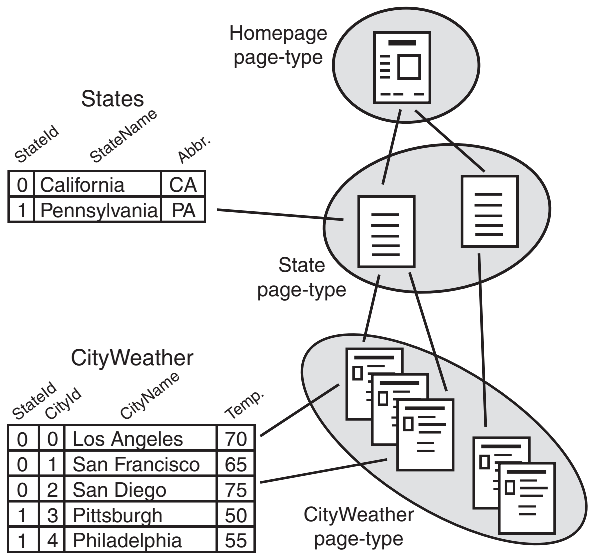
Our solution takes advantage of the fact many different types of structure often exist within a site. This includes the graph structure of the site's links, the URL naming scheme, the content on the pages, the HTML structures within page types, etc. To take advantage of this, we developed a set of "experts" that analyze the links and pages on a site and recognize different types of structure. Based on the structure that is identified, our system clusters the pages and the data within pages, so that it can create a relational structure over the data. A human can then identify which columns in the resulting relational table should be published in a webfeed.

A key algorithmic problem is how to cluster the pages and data, given that the experts are heterogeneous and identify very different types of structure. This is a common problem in AI, and is analogous to the problem of combining syntax, semantics and pragmatics for natural language understanding. To address this, our experts output their observations using a common representation. In this scheme, all experts produce "hints" indicating that two items should be in the same cluster. The clustering process then uses a probabilistic model of the hint-generation process to rate alternative clusterings.

Problem Description

Consider the task of creating a web site for current weather conditions in U.S. cities. We might start by defining a relational database table with the current weather for all the cities, one row per city. Next, we might write a script to generate an HTML page for each row of the table. At this point, we have a set of pages that contain information from a single relational database table and are similarly formatted. We will call such a set of pages a *page type*.

The weather site may involve other page types as well. For instance, to help users navigate to the city-weather pages, we might have pages for each state, with each state page containing links to the city-weather pages. We can do this by creating a new page type for states. To do so, we create a new table that holds state information, such as state



States

StateId	StateName	Abbr.
0	California	CA
1	Pennsylvania	PA

CityWeather

StateId	CityId	CityName	Temp.
0	0	Los Angeles	70
0	1	San Francisco	65
0	2	San Diego	75
1	3	Pittsburgh	50
1	4	Philadelphia	55

Figure 1: Relational Model of a Web Site

name and abbreviation, one row per state. Figure 1 shows our hypothetical weather site, its three page-types, and the underlying relational data, from the homepage down to our city-weather pages.

Now, suppose we are given the exact opposite task: Given a web site, recover the underlying relational database tables. That is the task we are faced with here. Relational learning approaches would seem to offer a possible methodology. However, existing relational learning methods start with data that is already in relational format and attempt to find an underlying model that would generate the data, such as a PRM (Getoor *et al.* 2001). In our case, we do not have a set of relations to start with; instead we need to discover both the relational data and the model by analyzing the HTML pages.

Approach

Our input is the set of HTML pages on a site (including links on each page), which we obtain by spidering the site. The pages are tokenized into individual components (i.e., strings, URLs, numbers, etc.) based on the analysis performed by a set of "experts" which examine the pages for URLs, lists, template structures, etc. As the experts examine the HTML pages and report various types of structure, the start/end character positions of the hypothesized structure are noted. The page is then tokenized so that these start and end positions are the token boundaries, minimizing the number of tokens generated. For example, if an expert generates a hypothesis (or "hint") that refers to a long URL such as "http://news.yahoo.com/news?templ=story&..." and no other expert generates a hint that refers to the substrings of this URL, then we have no reason to break it apart into smaller parts.

To discover the relational data on a site, we first focus on finding the individual columns of the tables. Specifically, we use a clustering approach, where we attempt to place tokens on the HTML pages into clusters, so that eventually each cluster contains the data in a column of one of the underlying tables. Once we achieve this, it is relatively straightforward to identify the rows of the table, producing the complete tables from the clusters.

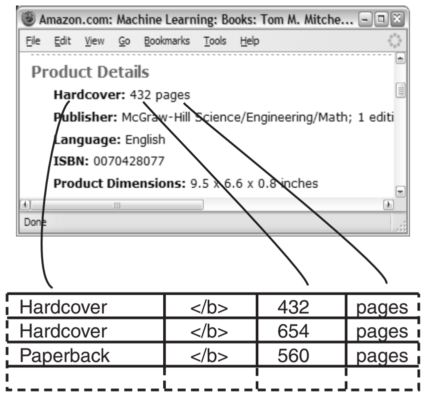
As we process each HTML page, clustering the tokens, it is also convenient to cluster the pages themselves, in order to identify page types. Thus, we use the following representation. A page-cluster contains a set of pages and is also the parent of a set of token-clusters. All of the tokens of the pages in a page-cluster are clustered into the child token-clusters of that page-cluster. For example, suppose we are working on the weather site which contains a home page listing all the states, state pages listing all the cities in that state, and weather condition pages that display the current conditions in a particular city. When we find page-clusters for this site, we hope to find three clusters: one for weather-condition pages, one for the state pages, and one containing just the home page. The token-clusters for the weather-condition page-cluster might include a cluster for city names, another for low temperatures, and another for high temperatures.

One approach to discovering page-clusters is to apply a distance metric based on surface-structure, such as viewing a page as a bag-of-words and measuring the similarity between the document vectors. This type of approach is not very helpful for correctly clustering web-pages from a single web-site, because the true similarity of pages typically can only be discovered after we have some understanding of the deeper structure of the page. For example, determining that a web-page with a short (or empty!) list is similar to one with a long list requires that we discover that the first one has a similar context surrounding the list, even though these pages may not appear similar in words, length, etc.

Rather than using a metric based on surface-structure to directly measure similarity of pages, we rely on multiple experts to generate "hints" that describe local structural similarities between pairs of pages (or between pairs of tokens). The hints are then used to cluster the pages and tokens. Our approach relies on having a heterogeneous set of experts such that for any given site, the discoveries of at least some of the experts will make the solution obvious, or at least nearly so. This is based on our observation that individual experts are successful in finding relevant structure some of the time, but not all of the time.

The hints provide a common language for experts to express their discoveries. A *page-level* hint is a pair of page references indicating that the referred pages should be in the same cluster. For example, if the input contains $page_1$ with URL "weather/current_cond/lax.html" and $page_2$ with URL "weather/current_cond/pit.html", a URL-pattern expert might generate ($page_1$, $page_2$) as a page-level hint. Similarly, a *token-level* hint is a pair of token sequences; the hint indicates the tokens of the two sequences should be in the same token-clusters. A list expert might generate ("New Jersey", "New Mexico") as a token-level hint, among many other similar hints, on examining a page which contains a list of states.

In our system, we use experts that focus on the following structures: URLs, lists, templates, and layout.


Amazon.com: Machine Learning: Books: Tom M. Mitche...

File Edit View Go Bookmarks Tools Help

Product Details

Hardcover: 432 pages

Publisher: McGraw-Hill Science/Engineering/Math; 1 editi

Language: English

ISBN: 0070428077

Product Dimensions: 9.5 x 6.6 x 0.8 inches

Done

Hardcover	</b>	432	pages
Hardcover	</b>	654	pages
Paperback	</b>	560	pages

Figure 2: Extracted Data in Relational Form

After the experts analyze the input pages, there are a great many hints, some of which may be conflicting, which complicates the clustering process. For this reason, we employ a probabilistic approach that gives us a flexible framework for combining multiple hints in a principled way. In particular, we use a probabilistic model that allows us to assign a probability to hints (both token hints and page hints) given a clustering. The model is based on the process of hint generation from clusterings. In the generation process, we assume a cluster is picked at random with uniform probability and then within the chosen cluster, a pair of pages (or tokens) is picked at random, again with uniform probability. The selected pair forms the hint. Combining the probabilities of picking particular items at each step gives the overall probability of generating a particular hint from a given clustering.

The probabilistic model, in turn, allows us to search for clusterings that maximize the probability of observing the set of hints. For this work, we use the basic leader-follower algorithm, which greedily clusters pages and tokens, to find a locally optimum solution.

Once we cluster the pages and tokens, producing the corresponding tables (i.e, the relational view of the data as in Figure 2) is a relatively simple task. For each page cluster there is a set of tables. Each column in each table is given by a token cluster.

Experiments

Evaluating our system is challenging because of the size and scope of the problem we address. Manually evaluating the system is difficult because of the size of the output. So instead of evaluating the full output, we focus on evaluating how well our system does in extracting target data by comparing it against the output of supervised web-page wrappers. Note that such wrappers normally extract only a few fields whereas our system would cluster all the tokens on the pages.

Our evaluation proceeds as follows. For each target field produced by the wrapper, we find the AutoFeed cluster that contains the most target values. Then we calculate the retrieved and relevant count RR, i.e., the number of target values in this cluster. We also report the total number of values in the cluster (Ret), and the total number of target values (Rel). We define precision as RR/Ret and recall as RR/Rel.

Table 1: AutoFeed Results Summary for E-Commerce Sites, Journals, and Job-Listings

	Field	RR	Ret.	Rel.	Precision	Recall
commerce	mfg	81	81	81	100%	100%
	item no.	100	100	103	97%	97%
	model no.	66	68	71	96%	93%
	name	97	100	103	97%	94%
	price	77	88	103	85%	75%
journals	authors	1173	1197	1212	98%	97%
	title	1173	1188	1212	99%	97%
	link	528	528	1212	100%	44%
jobs	position	1391	1391	1453	100%	96%
	req. id	1278	1278	1422	100%	90%
	location	1302	1302	1445	100%	90%

We report experiments in three domains (Table 1). In the first experiment, we compared AutoFeed to seven wrappers that return data from retail sites. These wrappers were originally built for SRI's CALO project. For the second experiment, we spidered all the pages from four electronic journals and for the third, pages for job listings from fifty employers.

The results show that AutoFeed successfully extracts most of the fields almost perfectly from a number of different sites in three different domains. AutoFeed's performance is comparable to that of supervised wrappers, even though it requires no training data.

Related Work

Much of the previous research on web extraction has focused on supervised methods. One approach is to learn field-specific rules that work across a variety of page-types, but good rules require many training examples (Craven *et al.* 1998). Another approach, referred to as "wrapper induction" (Kushmerick 2000; Cohen, Hurst, & Jensen 2002), focuses on learning field- and site-specific rules.

Recently, grammar induction methods have been proposed for unsupervised extraction from web pages (Crescenzi, Mecca, & Merialdo 2001; Arasu & Garcia-Molina 2003). These require the input pages to be of the same page-type, and rely solely on the syntactic structure of the pages. Other work on unsupervised learning has focused on table extraction, where the objective is on detecting the existence of tables and understanding their contents (Embley, Tao, & Liddle 2002; Lerman *et al.* 2004). We believe that the work on grammar-induction and automatic table-analysis is quite relevant to AutoFeed, in that these methods could be incorporated into AutoFeed as experts.

There has also been some closely-related work on clustering web pages (Crescenzi, Merialdo, & Missier 2003; Reis *et al.* 2004; Caverlee, Liu, & Buttler 2004). Generally, these projects have focused on developing a single distance metric for clustering, whereas here we focus on combining multiple experts probabilistically. In our experience, we have found that no single method (that we have developed) has worked sufficiently robustly, and simple methods for combining experts (e.g., using them in sequence) have also proved insufficient.

Finally, in other areas of AI, researchers have combined multiple learners and multiple experts with success. For instance, Slipper (Cohen & Singer 1999) uses a boosting method to generate rulesets from a simple rule-learner. In contrast to Slipper, which combines multiple instances of the same rule-learner, Proverb (Shazeer, Littman, & Keim 1999) uses multiple heterogeneous experts to solve crossword puzzles. Proverb's experts generate answer lists to clues of the crossword puzzle. These lists are then combined within a probabilistic constraint satisfaction framework to find the best answers to fill in the puzzle.

Conclusions

Unsupervised web-site extraction is a new challenge in making the web machine-understandable. In this paper, we have presented our formulation of the problem and our approach based on multiple heterogeneous experts.

We believe our preliminary experiments demonstrate the broad potential of our approach. By using multiple experts, each capable of discovering a basic type of structure, we are able to piece together clues which in turn lead us to the relational data underlying the site.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant DMI-0441563 and by the Defense Advanced Research Projects Agency (DARPA), through the Department of the Interior, NBC, Acquisition Services Division, under Contract No. NBCHD030010.

References

Arasu, A., and Garcia-Molina, H. 2003. Extracting structured data from web pages. In *SIGMOD Conference*, 337–348.

Caverlee, J.; Liu, L.; and Buttler, D. 2004. Probe, cluster, and discover: Focused extraction of qa-pagelets from the deep web. In *ICDE*, 103–115.

Cohen, W. W., and Singer, Y. 1999. A simple, fast, and effictive rule learner. In *AAAI/IAAI*, 335–342.

Cohen, W. W.; Hurst, M.; and Jensen, L. S. 2002. A flexible learning system for wrapping tables and lists in html documents. In *WWW '02: Proc. of the eleventh international conference on World Wide Web*, 232–241. ACM Press.

Craven, M.; DiPasquo, D.; Freitag, D.; McCallum, A. K.; Mitchell, T. M.; Nigam, K.; and Slattery, S. 1998. Learning to extract symbolic knowledge from the World Wide Web. In *Proc. of AAAI-98, 15th Conference of the American Association for Artificial Intelligence*, 509–516. Madison, US: AAAI Press.

Crescenzi, V.; Mecca, G.; and Merialdo, P. 2001. Roadrunner: Towards automatic data extraction from large web sites. In *Proc. of 27th International Conference on Very Large Data Bases*, 109–118.

Crescenzi, V.; Merialdo, P.; and Missier, P. 2003. Fine-grain web site structure discovery. In *WIDM '03: Proc. of the 5th ACM international workshop on Web information and data management*, 15–22. ACM Press.

Embley, D. W.; Tao, C.; and Liddle, S. W. 2002. Automatically extracting ontologically specified data from html tables of unknown structure. In *ER '02: Proc. of the 21st Int. Conf. on Conceptual Modeling*, 322–337. Springer-Verlag.

Gazen, B., and Minton, S. 2005. Autofeed: an unsupervised learning system for generating webfeeds. In *Proc. of the 3rd Int. Conf. on Knowledge Capture (K-CAP), Banff, Alberta, Canada*, 3–10.

Getoor, L.; Friedman, N.; Koller, D.; and Taskar, B. 2001. Learning probabilistic models of relational structure. In *Proc. 18th Int. Conf. on Machine Learning*, 170–177. Morgan Kaufmann, San Francisco, CA.

Kushmerick, N. 2000. Wrapper induction: Efficiency and expressiveness. *Artificial Intelligence* 118(1-2):15–68.

Lerman, K.; Getoor, L.; Minton, S.; and Knoblock, C. 2004. Using the structure of web sites for automatic segmentation of tables. In *SIGMOD '04: Proc. of the 2004 ACM SIGMOD international conference on Management of data*, 119–130. ACM Press.

Reis, D. C.; Golgher, P. B.; Silva, A. S.; and Laender, A. F. 2004. Automatic web news extraction using tree edit distance. In *WWW '04: Proc. of the 13th international conference on World Wide Web*, 502–511. ACM Press.

Shazeer, N. M.; Littman, M. L.; and Keim, G. A. 1999. Solving crossword puzzles as probabilistic constraint satisfaction. In *AAAI/IAAI*, 156–162.