Learning and Discovering Structure in Web Pages

William W. Cohen
Center for Automated Learning & Discovery
Carnegie Mellon University
Pittsburgh, PA 15213
wcohen@cs.cmu.edu

Abstract

Because much of the information on the web is presented in some sort of regular, repeated format, "understanding" web pages often requires recognizing and using structure, where structure is typically defined by hyperlinks between pages and HTML formatting commands within a page. We survey some of the ways in which structure within a web page can be used to help machines understand pages. Specifically, we review past research on techniques that automatically learn and discover web-page structure. These techniques are important for wrapper-learning, an important and active research area, and can be beneficial for tasks as diverse as classification of entities mentioned on the web, collaborative filtering for music, web page classification, and entity extraction from web pages.

1 Introduction

In spite of recent progress on the semantic web and interchange formats like XML, most of the information available today on the world wide web is targeted at people, not machines. Because of this, the problem of automatically aggregating information from the web [3, 4, 11, 18, 21, 23, 24] is technically difficult: simply put, in order to make web information machine-readable, it is usually necessary to design a program that "understands" web pages the same way a human would—at least within some narrow domain.

The task of "understanding" web pages is broadly similar to classical problems in natural language understanding. One important difference is that most of the information on the web is not presented in smoothly flowing grammatical prose; instead, web information is typically presented in some sort of tabular, structured format. Hence understanding web pages often requires recognizing and using "structure"—a structure typically being defined by hyperlinks between pages, and HTML formatting commands (e.g., tables and lists) within a page. In this paper, we will survey some of the ways in which structure can be used to help machines understand web pages. We will focus on techniques that exploit HTML formatting structure within a page, rather than link structure between pages.

Broadly speaking, there are two ways in which such structure can be used. It may be used *directly* to accomplish some task; typically, to determine how to extract data from a page. As an example, consider the imaginary web page shown in Figure 1. (The markers near the words "Customers", "Mr. Peabody", etc will be explained below.) To populate the relation *titleOf(personName, jobTitle)* from this page, one might extract

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

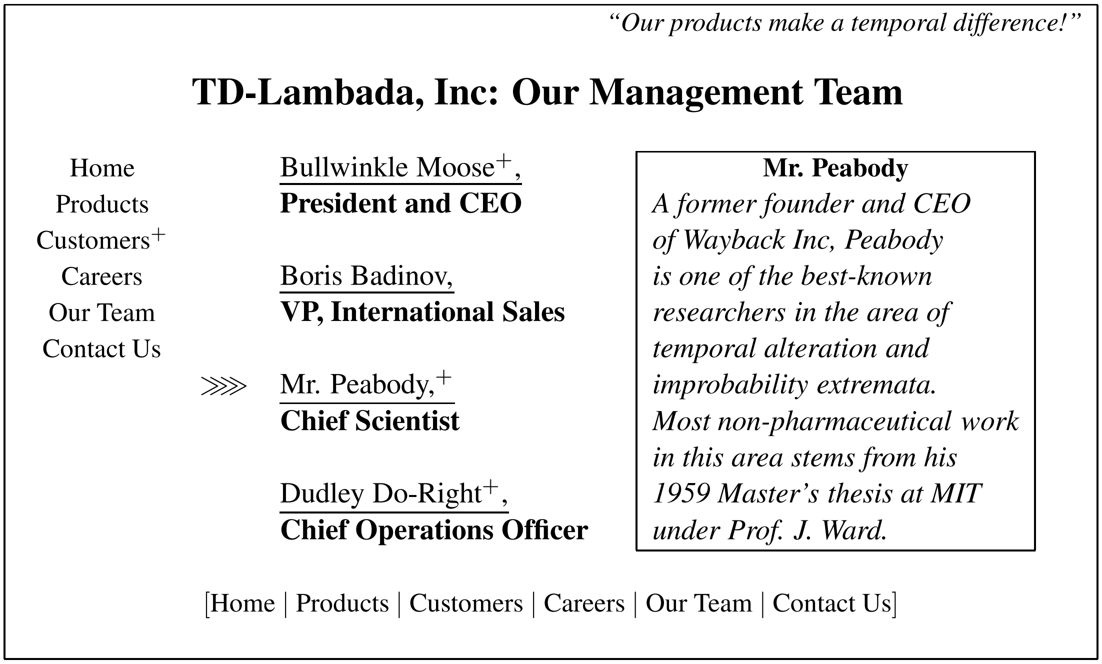

Figure 1: An imaginary web page containing non-trivial regular structure

names and titles from alternate non-blank positions in the second column; thus extraction would be based on the formatting used on the page.

Structure might also be used as one input among many that collectively guide a more complex process. For instance, consider using a statistical machine learning algorithm to identify job-title phrases on web pages. Such an algorithm might exploit content features (like "x contains the word 'CFO' "), formatting features (like "x is rendered in boldface") or structural features (like "x appears in the same table column as a known job-title word"). We will call this sort of use of structure *indirect*, since structure is only one of several factors that affects the learning algorithm.

In this paper, we will focus on techniques that involve learning structure from examples, or automatically discovering possible structure using heuristics. We will also discuss how learned or discovered structure can be used—both directly and indirectly.

2 Learning structure for direct use

A program that derives a database relation from a specific website is called a *wrapper*, and *wrapper learning* is the problem of learning website wrappers from examples. For instance, a wrapper for the website of Figure 1 might produce the relation *titleOf(personName, jobTitle)* described above.

As an alternative to explicit programming, such a wrapper might be learned from a handful of user-provided examples. As a very simple example, the four job titles on this page might be learned from the two examples "President and CEO" and "VP, International Sales". (Notice that these strings are *positive* examples, *i.e.*, instances of strings that *should* be extracted; most learning systems also require *negative* examples, *i.e.*, strings that *should not* be extracted. A common convention in wrapper-learning is for a user to provide *all* positive examples in some prefix of the document being labeled. A set of negative examples then can be derived by using a sort of closed world assumption.)

Wrapper learning has been an active area from the mid-nineties [20] through the present (*e.g.*, [2, 15, 25]).

Typically a wrapper is defined in terms of the format used on a particular web site: in Figure 1, for instance, a wrapper might extract as job titles "all boldfaced strings in the second column". Hence wrapper-learning is a prototypical example of learning a structure for direct use. By way of an introduction to this active area of work, we will discuss below a few successful wrapper-learning systems.

Kushmeric's seminal WIEN system [19, 20] was based on a handful of carefully crafted wrapper languages with very restricted expressive power. For example, one such language was HLRT. An HLRT wrapper for a one-field relation is defined by four strings: a string h that ends the "header" section of a web page, two strings ℓ and r that precede and follow each data field, and a string t that begins a "tail" section of a web page. (The "head" and "tail" sections of a page contain no data fields.) For instance, a plausible HLRT wrapper for job-title strings in an HTML version of the page of Figure 1 might be $h =$"Contact Us⟨/A⟩", $\ell =$"⟨B⟩", r="⟨/B⟩", and t="⟨/B⟩⟨/LI⟩⟨/UL⟩". By limiting a wrapper-learner to explore this restricted set of possible structures, Kushmeric was able to use a learning algorithm that was elegant and well-grounded formally (for instance, it was guaranteed to converge after a polynomial number of examples). The disadvantage of this approach was that some wrappers could not be learned at all by WIEN.

Later wrapper-learning systems such as STALKER [26] and BWI [14] used more expressive languages for wrappers: for instance, the start of a field might be identified by the disjunction of a set of relatively complex patterns, rather than a single fixed string ℓ. Surprisingly, these broader-coverage systems did *not* require more examples to learn simple structures, such as those considered by WIEN; instead the learning algorithms used were designed to propose simpler structures first, and fall back on more complex structures only if necessary. The disadvantage of these systems (relative to WIEN) is greater complexity, and weaker performance guarantees.

In our own previous work, we developed a wrapper-learning system called WL^2 [16, 9]. Like WIEN, WL^2 exploits the fact that many wrappers are based on a few simple structures, and that it is often possible to design restricted languages that capture these common cases. Like STALKER and other systems, WL^2 can find complex wrappers when simple ones are inadequate. Unlike previous wrapper-learners, WL^2 is modular, and designed to be easily extended to accomodate new classes of structures (perhaps structures common in a particular domain). The learning system in WL^2 is based on an ordered set of *builders*. Each builder B is associated with a certain restricted language $\mathcal{L}_B$. However, the builder for $\mathcal{L}_B$ is not a learning algorithm for $\mathcal{L}_B$. Instead, to facilitate implementation of new "builders", a separate master learning algorithm handles most of the real work of learning, and the builder B need support only a small number of operations on $\mathcal{L}_B$—the principle one being to propose a single "least general" structure given a set of positive examples. Builders can also be constructed by composing other builders in certain ways. For instance, two builders for languages L_{B_1} and L_{B_2} can be combined to obtain builders for the language $(L_{B_1} \circ L_{B_2})$, or the language $(L_{B_1} \wedge L_{B_2})$.

WL^2 included builders that detected structures of various kinds. Some structures were defined in terms of string-matches, like HLRT; some were defined in terms of the HTML parse tree for a page; and some were defined in terms of the geometrical properties of the rendered page. The master learning algorithm can find extraction rules based on any of these sorts of structures; or, it can combine structures produced by different builders to form new, more complex extraction rules (e.g., "extract all table cells vertically below a cell containing the words "Job Title" that will be rendered in bold with a font size of 2"). Experiments conducted with a large number of practical extraction problems showed that WL^2 could learn to extract many data fields with three positive examples or less.

3 Discovering structure for direct use

In wrapper learning, a user gives explicit information about each structure. Even if wrapper-learning is very efficient, a system that requires information from many web sites will still be expensive to train. A natural question to ask is: can structure be recognized (and used) without training data?

In some situations, the answer is "yes". For instance, Kushmeric *et al* [20] described ways in which automatic but imperfect entity recognizers could be used to drive wrapper learning; Embley *et al* [12] described accurate and page-independent heuristics for recognizing strings that separate one data record from another in web pages; and other researchers [13, 22] have described techniques for finding logically meaningful facts from HTML lists and tables without training data. Below we will summarize some of our own work on automatically discovering useful structure.

One difficulty with evaluating such a discovery system is that, in general, it is unclear what structures are "useful". One definition of "useful" is "structure that could be used to guide web-site wrapping". In previous work [5], we evaluated the performance of a structure-discovery algorithm by using it to propose structures for 82 previously-wrapped web pages, and measuring the fraction of the time that the discovered structure coincided with the wrapper for that page.

The algorithm proposes two types of structures, called *simple lists* and *simple hotlists*, which have the property that only a polynomial number of possible structures can appear on any given web page; this means that finding the best structure can be reduced to generating all simple lists or hotlists, and then ranking them. One simple ranking scheme is to score a structure by the number of elements extracted by the the structure; on this dataset, this heuristic ranked the correct structure highest about 20% of the time. If more information is available, then more powerful heuristics can be used. For instance, if a large list of "seed" items of the correct type are available, then ranking lists according to the an aggregate measure of the distance from extracted items to "seeds" (using an appropriate similarity metric) ranks the correct structure highest about 80% of the time.

This algorithm could thus be used as the basis for a wrapper "learning" algorithm that uses no explicit user examples. Instead wrappers would be constructed as follows:

- The system generates, ranks, and presents to the user an ordered list of possible structures. For instance, given a set of person-name seeds and the web page of Figure 1 as input, the system might produce these candidate lists:

  1. "Bullwinkle Moose", "Boris Badinov", "Mr. Peabody", ...
  2. "Bullwinkle Moose, President and CEO", "Boris Badinov VP, International Sales", ...
  3. "Home", "Products", "Customers", "Careers", ...
  4. "President and CEO", "VP, International Sales", "Chief Scientist", ...

  The ranking shown is reasonable, as structures containing words like "Mr." and "Boris" (which are likely to appear in some seed) should be ranked higher than structures containing only phrases like "Home" and "Chief Scientist".

- The user picks a candidate structure from the ranked list (say, the first one) that contains the items to be extracted.

- The user specifies semantically how these items will be stored in the database being populated; e.g., by specifying that they are the set of "people employed by TD-Lambada, Inc".

4 Discovering simple structure for indirect use

We emphasize that the 80% figure reported in the section above is relative to the set of 82 wrappers used in these experiments, all of which could actually be represented as simple lists or hotlists. Of course, many wrappers are more complex. For appropriately simple wrappers, less user intervention is required using the method described above than with than learning techniques like WL^2; however, the user is still needed to (a) verify the correctness of a proposed structure and (b) ascribe the appropriate semantics to the structure.

One way to exploit structure without involving the user at all is to use discovered structure indirectly. As an example, consider the task of classifying musical artists by genre given only their names: thus "John Williams" would be classified as "classical" and "Lucinda Williams" as "country". This is difficult to do accurately without some background knowledge about the artists involved. Such background knowledge might be obtained by manually training (and then executing) wrappers for some appropriate set of web sites—can one obtain similarly useful background knowledge using *automatically* discovered structure?

In another set of experiments [6], we used structure discovery algorithms to find features useful for learning. One task we explored was learning to classify musical artists by genre. We took a large list of musical artists and labeled a subset to use as training data. We then used the complete list of artists as seeds for (a variant of) the structure-discovery algorithm described above. Applied to a large set of web pages, this produces many possible structures. Without user filtering, many of these structures are meaningless, and other structures correspond to sets that are completely uncorrelated with genre (like "artists with names starting with 'A' that have CDs on sale at Bar.com"). However, may of the discovered structures *are* correlated with genre, and hence can be exploited by a statistical learner.

Specifically, we associated each discovered structure D with a matching subset S_D of the complete artist list. We then introduced, for every structure D, a new binary feature f_D which was true for every example $x \in S_D$ and false for every other example. Finally we applied traditional feature-based statistical learning algorithms to the augmented examples. In our experiments, the new structure-based features improved performance on several benchmark problems in a wide variety of situations, and performance improvements were sometimes dramatic: on one problem, the error rate was decreased by a factor of ten. Similar features can be used for other tasks—for instance, we also experimented with an analogous system that uses features based on discovered structure to guide a collaborative recommendation system [8].

5 Discovering complex structure for indirect use

Using structure indirectly (rather than directly) avoids many of the problems associated with discovering (rather than learning) structure. Meaningless structures can be filtered out by the statistical learning system—which is, after all, designed to handle irrelevant and noisy features. Importantly, structures can be used even if their precise semantics are unknown—meaningful structures can be exploited whenever they are correlated with the class to be predicted. In the settings described above, discovery algorithms that consider only a few types of structures are also less problematic; in principle, one need only run the discovery algorithm on more web pages to compensate for limited coverage.

For other applications, however, algorithms that can only discover simple structures from a limited class are much less useful. Consider the task of learning to identify executive biography pages (like the one shown in Figure 1) on company sites. One approach would be to train a classifier that uses as features the words in the page. Such a classifier might be trained by labeling pages from a number of representative company web sites, and then used to identify executive biography pages on new sites, such as the one of Figure 1.

This approach, however, ignores possibly useful structural information. For instance, on the web site for Figure 1, it might be that the executive biography pages are *exactly* those pages linked to by the anchors in the second column: in other words, the site might contain a structure that accurately identifies the target pages. Unfortunately, this structure cannot be used as a feature of the classifier, since it does not appear in the training data at all (recall the training data is taken from other web sites); it can only be used if it is somehow discovered "on the fly" when the pages on this site are encountered by the classifier.

Notice that in this setting, a algorithm that discovers only simple structures is of limited use: since only a few structures on a site will be informative, it is essential to be able to find all of them. This suggests adapting wrapper-learning machinery for finding complex structures to this task. In previous work [7], we proposed the following method for processing a web site.

First, all pages on a site are classified using a word-based classifier, and all anchors that point to an "executive biography" page are marked as positive. These markings are shown in Figure 1 with "'+" signs, and some errors are likely to be made. In the figure, the link to "Customers" is mistakenly marked positive, and the link to "Boris Badinov" is mistakenly unmarked.

Next, all sets of at most k nearby positive anchors are generated, for $k = 1, 2, 3$, and each such set is passed as input to each of the builders of WL^2. This produces a large number of structures D: some simple structures, produced from a single positive examples, and some more complex ones, produced from two or three nearby examples. Each discovered structure D is then used to generate new a feature f_D, which is true for all pages pointed to by some anchor in structure D. Finally, a new classifier is learned for this site, using the predictions provided by the original word-based classifier as labels, and using the new structure-based features to represent a page. The overall effect of this process is to smooth the predictions made by the word-based classifier toward sets of pages that are easily described by structures on the site. In our experiments, this smoothing process decreased the error rate of the original word-based page classifier by a factor of about half, on average.

In previous work, a number of other researchers have used hyperlink structure (hubs) to improve page classifiers [10, 17, 27] in a similar manner. Blei *et al* [1] also used the same structure-discovery algorithms in conjunction with a more complex probabilistic scheme for combining structural features with word-based predictions, and achieved a significant reduction in error rate on entity extraction tasks.

6 Conclusion

‘

Techniques that automatically learn and discover web-page structure are important for efforts to "understand" web pages. Techniques for learning web page structure are crucial for wrapper-learning, an important and active research area. They can also be beneficial for tasks as diverse: as classification of entities mentioned on the web; collaborative filtering for music; web page classification, and entity extraction from web pages.

References

[1] David M. Blei, J. Andrew Bagnell, and Andrew K. McCallum. Learning with scope, with application to information extraction and classification. In *Proceedings of UAI-2002*, Edmonton, Alberta, 2002.

[2] Boris Chidlovskii. Information extraction from tree documents by learning subtree delimiters. In *Proceedings of the 2003 Workshop on Information Integration on the Web (IIWeb-03)*, Acapulco, Mexico, August 2003. On line at http://www.isi.edu/info-agents/workshops/ijcai03/proceedings.htm.

[3] William Cohen, Andrew McCallum, and Dallan Quass. Learning to understand the web. *IEEE Data Engineering Bulletin*, 23:17–24, September 2000.

[4] William W. Cohen. Reasoning about textual similarity in information access. *Autonomous Agents and Multi-Agent Systems*, pages 65–86, 1999.

[5] William W. Cohen. Recognizing structure in web pages using similarity queries. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, Orlando, FL, 1999.

[6] William W. Cohen. Automatically extracting features for concept learning from the web. In *Machine Learning: Proceedings of the Seventeeth International Conference*, Palo Alto, California, 2000. Morgan Kaufmann.

[7] William W. Cohen. Improving a page classifier with anchor extraction and link analysis. In *Advances in Neural Processing Systems 15*, Vancouver, British Columbia, 2002.

[8] William W. Cohen and Wei Fan. Web-collaborative filtering: Recommending music by crawling the web. In *Proceedings of The Ninth International World Wide Web Conference (WWW-2000)*, Amsterdam, 2000.

[9] William W. Cohen, Lee S. Jensen, and Matthew Hurst. A flexible learning system for wrapping tables and lists in HTML documents. In *Proceedings of The Eleventh International World Wide Web Conference (WWW-2002)*, Honolulu, Hawaii, 2002.

[10] David Cohn and Thomas Hofmann. The missing link - a probabilistic model of document content and hypertext connectivity. In *Advances in Neural Information Processing Systems 13*. MIT Press, 2001.

[11] M. Craven, D. DiPasquo, D. Freitag, A. McCallum, T. Mitchell, K. Nigam, and S. Slattery. Learning to extract symbolic knowledge from the world wide web. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, Madison, WI, 1998.

[12] D.W. Embley, Y.S. Jiang, and W.-K. Ng. Record-boundary discovery in web documents. In *SIGMOD'99 Proceedings*, 1999.

[13] D.W. Embley, C. Tao, and S.W. Liddle. Automatically extracting ontologically specified data from html tables with unknown structure. In *Proceedings of the 21st International Conference on Conceptual Modeling*, Tampere, Finland, October 2002.

[14] D. Freitag and N. Kushmeric. Boosted wrapper induction. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000)*, Austin, TX, 2000.

[15] Chun-Nan Hsu, Chia-Hui Chang, Harianto Siek, Jiann-Jyh Lu, and Jen-Jie Chiou. Reconfigurable web wrapper agents for web information integration. In *Proceedings of the 2003 Workshop on Information Integration on the Web (IIWeb-03)*, Acapulco, Mexico, August 2003. On line at http://www.isi.edu/info-agents/workshops/ijcai03/proceedings.htm.

[16] Lee S. Jensen and William W. Cohen. Grouping extracted fields. In *Proceedings of the IJCAI-2001 Workshop on Adaptive Text Extraction and Mining*, Seattle, WA, 2001.

[17] T. Joachims, N. Cristianini, and J. Shawe-Taylor. Composite kernels for hypertext categorisation. In *Proceedings of the International Conference on Machine Learning (ICML-2001)*, 2001.

[18] Craig A. Knoblock, Steven Minton, Jose Luis Ambite, Naveen Ashish, Pragnesh Jay Modi, Ion Muslea, Andrew G. Philpot, and Sheila Tejada. Modeling web sources for information integration. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, Madison, WI, 1998.

[19] N. Kushmeric. Wrapper induction: efficiency and expressiveness. *Artificial Intelligence*, 118:15–68, 2000.

[20] Nicholas Kushmeric, Daniel S. Weld, and Robert Doorenbos. Wrapper induction for information extraction. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, Osaka, Japan, 1997.

[21] Steve Lawrence, C. Lee Giles, and Kurt Bollacker. Digital libraries and autonomous citation indexing. *IEEE Computer*, 32(6):67–71, 1999.

[22] Kristina Lerman, Craig A. Knoblock, and Steven Minton. Automatic data extraction from lists and tables in web sources. In *Proceedings of the Automatic Text Extraction and Mining Workshop (ATEM-01)*, Seattle, WA, August 2001. Held in conjunction with IJCAI-01.

[23] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Query answering algorithms for information agents. In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI-96)*, Portland, Oregon, august 1996.

[24] A. K. McCallum, K. Nigam, J. Rennie, , and K. Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval Journal*, 3:127–163, 2000.

[25] Steven N. Minton, Sorinel I. Ticrea, and Jennifer Beach. Trainability: Developing a responsive learning system. In *Proceedings of the 2003 Workshop on Information Integration on the Web (IIWeb-03)*, Acapulco, Mexico, August 2003. On line at http://www.isi.edu/info-agents/workshops/ijcai03/proceedings.htm.

[26] Ion Muslea, Steven Minton, and Craig Knoblock. Wrapper induction for semistructured information sources. *Journal of Autonomous Agents and Multi-Agent Systems*, 16(12), 1999.

[27] S. Slattery and T. Mitchell. Discovering test set regularities in relational domains. In *Proceedings of the 17th International Conference on Machine Learning (ICML-2000)*, June 2000.