Unsupervised Learning of Tree Alignment Models for Information Extraction

Philip Zigoris, Damian Eads, and Yi Zhang
Department of Computer Science
University of California, Santa Cruz
1156 High Street
Santa Cruz, CA 95064
{zigoris,eads,yiz}@soe.ucsc.edu

Abstract

We propose an algorithm for extracting fields from HTML search results. The output of the algorithm is a database table– a data structure that better lends itself to high-level data mining and information exploitation. Our algorithm effectively combines tree and string alignment algorithms, as well as domain-specific feature extraction to match semantically related data across search results. The applications of our approach are vast and include hidden web crawling, semantic tagging, and federated search. We build on earlier research on the use of tree alignment for information extraction. In contrast to previous approaches that rely on hand tuned parameters, our algorithm makes use of a variant of Support Vector Machines (SVMs) to learn a parameterized, site-independent tree alignment model. This model can then be used to deduce common structural and textual elements of a set of HTML parse trees. We report some preliminary results of our system's performance on data from websites with a variety of different layouts.

1 Introduction

There is a proliferation of research in the field of Knowledge Discovery in Databases (KDD) aiming to derive high value conclusions from information stored in a database[7, 9]. Many of the advances in the field rely on the presence of highly structured data. Unfortunately, a vast amount of content on the Internet is in the form of semi-structured HTML search results, making the data unusable to many algorithms. The field of information extraction (IE) tries to address this problem by developing tools for transforming semi-structured text into highly-structured database content[17, 1, 11, 2, 18, 6, 3]. Thus, successes in IE will enable the exploitation of a broad class of KDD and Data Mining (DM) algorithms on the largest source of information in the world–the Internet.

The main intuition driving our approach to information extraction is that search results will often contain a high degree of repetition and this indirectly yields information about the structure of the data. In order to identify the repetitive elements we use a variety of parameterized *tree alignment models*. Simply put, a tree alignment model assigns a cost of pairing vertices from two trees. By finding a minimum cost pairing between vertices we presumably identify the common structural, and possibly textual, elements of the trees. These models are introduced formally in Section 2.

One of the major contributions of our work is an unsupervised method for learning the tree alignment parameters. With well-tuned parameters these models are resilient to structural variation in dynamic HTML returned by the web site and a well-informed alignment model will often have too many parameters to effectively tune by hand. Specifically, we explore the use of Support Vector Machines (SVM), a popular machine learning algorithm, for learning these parameters. The details of our approach are presented in Section 3.

The other novel aspect of our work is a simple method for generating and representing schema. Rather than inducing a set of rules for processing unseen data, our method works by simply comparing new data against that which has already been seen; analogous to nearest neighbor classification. The details of this method are presented in Section 4.

In Section 5 we present a preliminary evaluation of our work.

2 Tree Edit Distance

This section introduces two ideas which are central to our approach: tree alignments and tree edit distance, both originally due to Tai [15]. They are, respectively, analogous to the well studied concepts of string alignment and string edit-distance. Instead of strings, however, we are con-

cerned with *vertex labeled trees*, a tree coupled with a labeling function $l(v)$ that maps vertices to a *label*. In our work we study HTML parse trees where vertices are labeled with tag identifiers or free text. We will often refer to vertices labeled with text as *textual vertices* to distinguish them from vertices labeled with HTML tags.

Intuitively, a *tree alignment* is an association between vertices in two labeled trees, T_1 and T_2. The *tree edit-distance* between the two trees corresponds to the minimal cost of transforming T_1 into T_2. The edit-distance provides a measure of similarity between trees (and their sub-trees) and the alignment provides a way to identify the common elements of each tree, with respect to both structure and the vertex labeling.

In this work we concern ourselves with *rooted ordered labeled trees*. This is a special case of labeled trees where a vertex r is identified as the root and the children of every node have a fixed ordering. We can, therefore, speak of the 'left' and 'right', as well as the i^{th}, child of a node. Throughout this paper we will assume all trees are rooted, ordered, and labeled. With a fixed ordering and designated root, we can formally define an *alignment* between trees T_1 and T_2 as a set $\mathcal{A} \subset 2^{T_1 \times T_2}$ such that for all $(a, b), (a', b') \in \mathcal{A}$

- $a = a' \Leftrightarrow b = b'$,
- a is to the left of $a' \Leftrightarrow b$ is to the left of b', and
- a is an ancestor of $a' \Leftrightarrow b$ is an ancestor of b'

In other words, an alignment is a list of pairs of nodes, one from each tree, such that each vertex is paired with at most one other vertex and there are no "crossovers". An example of an alignment between two HTML parse trees is shown in Figure 1.

Associated with an alignment is a set of editing operations for transforming one tree into the other: Every pair (a, b) in the alignment corresponds to a relabeling/copying of a vertex, an operation we denote by $R(a, b)$. A vertex a from T_1 that does not occur in the alignment corresponds to a *deletion*, denoted by $D(a)$. Similarly, a vertex $b \in T_2$ not occurring in the alignment corresponds to an *insertion*, denoted by $I(b)$.

In order to discuss tree edit distance we require a function c_θ that assigns a cost to each operation. We will often refer to the parameter θ as the *operation costs*, although the terminology is somewhat imprecise. It is natural, given c_θ, to define the cost of an alignment, $C_\theta(\mathcal{A})$, as the sum of costs of its associated operations:

$$\sum_{(a,b)\in\mathcal{A}} c_\theta(R(a,b)) + \sum_{\substack{a\in T_1:\\ \forall b\in T_2\\ (a,b)\notin\mathcal{A}}} c_\theta(D(a)) + \sum_{\substack{b\in T_2:\\ \forall a\in T_1\\ (a,b)\notin\mathcal{A}}} c_\theta(I(a))$$

Finally, the tree edit distance between trees T_1 and T_2, $d_\theta(T_1, T_2)$ is defined to be the minimum cost over all alignments. We will denote by $\mathcal{A}_\theta(T_1, T_2)$ the minimizing alignment.

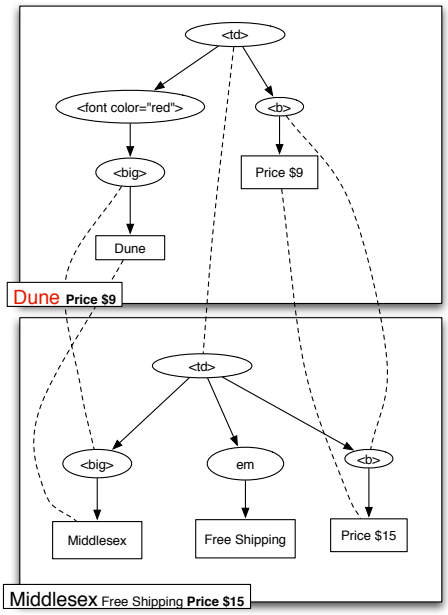

Figure 1. An example of an alignment between two HTML parse trees. The rendered HTML text is illustrated at the bottom left of each box. Vertices labeled with HTML tags are shown as ellipses and *textual vertices* are shown as squares.

Finding the minimum cost alignment can be done in, at best, cubic time with dynamic programming [8, 4]. However, for large trees even cubic running time can be prohibitive. To alleviate this, other work in information extraction has relied on approximate alignment algorithms such as *partial tree alignment* [17] and *restricted top down mappings* [14]. In our work the trees were small enough that resorting to such methods was unnecessary. However, incorporating our methodology into a practical system would probably require their use.

2.1 Cost functions

The task of information extraction requires a high degree of specificity in the cost function. A field will typically contain different strings with similar semantics (e.g. prices, dates, ISBN). In order for the vertices in a field to align well, the cost function must assign a low cost to aligning strings with similar content. For instance, consider aligning the text "Price $4.99" with "$100". Despite the large syntactic differences between them, both strings have a similar

function (i.e. to convey the price of an item).

In order to study the sensitivity to these issues we developed three different cost functions with varying degrees of specificity. The first, referred to as the *Simple* cost function, is parameterized by only three numbers: θ_M is the cost of copying any vertex, θ_{ID} is the cost of inserting/deleting any vertex, and θ_R is the cost of relabeling any vertex. The Simple function completely ignores the semantics of the labeling and so aligning "$100" with "$5" will have the same cost as aligning "$100" with "July 4th".

The second cost function we developed differs from the Simple function only in the cost of aligning textual vertices. We refer to this as the *string edit distance* or *sed* cost function and it includes one extra parameter, θ_S. The cost of aligning two text labeled vertices is θ_S times the normalized string edit distance of those two strings.

The third cost function incorporates some simple rules for determining the semantic relationship between two strings of text and so we refer to this as the *Semantic* cost function. Here the cost of aligning two vertices labeled with strings s_1 and s_2, respectively, is $\bar{\theta}_S \cdot f(s_1, s_2)$ where $f(\cdot, \cdot)$ is a feature vector and $\bar{\theta}_S$ are the weights associated with each feature. The features include whether or not both strings represent a(n): street, email or web address, date, phone number, or price. In total the Semantic cost function has 24 parameters.

3 Learning Operation Costs with SVM^{struct}

In the previous section we presented parameterized tree alignments. Here we present an algorithm for learning these parameters from what is, effectively, unlabeled data. It is an extension of work by Tsochantaridis, et al. [16] on using Support Vector Machines (SVMs) for learning structured labels. In their work they outline a very general framework that accommodates settings such as multi-class learning, grammar learning, and learning sequence alignment parameters.

Assume we are given a collection of HTML parse trees $\mathcal{T}$ that are each labeled with their site of origin $s \in S$, where S is the set of web sites (e.g., google.com, amazon.com, soe.ucsc.edu). In our work each parse tree $T \in \mathcal{T}$ corresponds to one data record from a search results page. Denote by $\mathcal{T}_s$ all trees originating from site s.

The constraint we impose is that two trees from one site must be closer to one another, in terms of tree-edit distance, than to a tree from another website, illustrated in Figure 2. Note that the labeling, i.e. the site of origin, is provided by the system that fetches search results and so no manual labeling is required.

Formally, we seek θ (the cost function parameters) such that $\forall s_1, s_2 \in S$, $\forall T_1, T_1' \in \mathcal{T}_{s_1}$, $\forall T_2, \in \mathcal{T}_{s_2}$ the following holds:

$$d_\theta(T_1, T_1') \leq d_\theta(T_1, T_2)$$

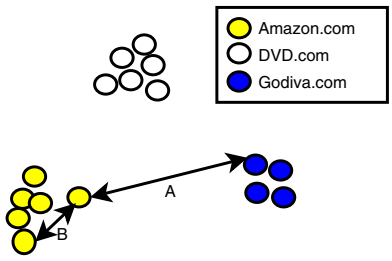

Figure 2. The points represent records from three different sites. The task is to find operation costs such that preserves this grouping under tree-edit distance. A represents inter-site distance and B represents site width.

We refer to the maximum distance between any two trees from site $s \in S$ as the *width* of s. We refer to the minimum distance between any two trees in sites s_1 and s_2 as the *inter-site distance*.

It may be the case that no θ satisfies the above constraints. Accordingly, we can relax the constraints by introducing slack variables ξ_s for $s \in S$ and penalize a solution by the sum of the slack variables. Our optimization problem becomes

$$\begin{aligned} \min_{\theta, \xi \geq 0} \quad & \sum_{s \in S} \xi_s \\ \forall s_1, s_2 \in S & \\ \forall T_1, T_1' \in \mathcal{T}_{s_1} \quad & d_\theta(T_1, T_1') - d_\theta(T_1, T_2) \leq \xi_{s_1} \\ \forall T_2, \in \mathcal{T}_{s_2} & \end{aligned}$$

Note that ξ_s corresponds to the maximum inter-site distance between s and any other site.

In order to introduce a notion of *margin* we require that the distance function scales linearly with the parameter values. That is

$$\forall \theta, \alpha > 0,\ d_{\alpha\theta}(a, b) = \alpha d_\theta(a, b)$$

All of cost functions discussed previously satisfy this requirement. In this way we can specify a unique solution by giving favor to parameters settings that are small in magnitude. This has been shown to improve generalization error in the case of linear separators[5]. We balance the cost of the slack variables with the parameter magnitude with the parameter C by adding the term $\frac{1}{2}||\theta||^2$ to the objective function, giving us a quadratic program. The effect of maximizing the margin is to not only maximize the inter-site distances but also minimize the widths of the sites.

The above optimization problems are difficult to solve directly, because the distance between two trees is a func-

tion of a hidden variable, namely the minimum cost alignment. Our solution is an *EM-like* algorithm[1], the format, of which, should be familiar to most readers:

- Set θ_0 to be all zeros and $t = 0$
- Do

  Expectation-Like Step Find the minimum cost alignment, $\mathcal{A}_{\theta_t}(\cdot,\cdot)$, between pairs of trees.

  Maximization-Like Step Solve the above optimization problem but replace the constraints with

  $$C_{\theta_t}(\mathcal{A}_{\theta_t}(T_1, T_1')) - C_{\theta_t}(\mathcal{A}_{\theta_t}(T_1, T_2)) \leq \xi_{s_1} - 1$$

  Store the result as θ_{t+1}

  $t \leftarrow t + 1$

  While$(\theta_t \neq \theta_{t-1})$

We implemented our algorithm using the SVM^{struct} package, which is built on the popular SVM^{light} software [10].

4 Schema Generator

Our approach differs from previous work in schema induction in that we do not explicitly construct rules for extracting fields [14, 13]. Instead, incoming records are aligned against a set of stored and labeled HTML parse trees (i.e records), referred to as *keys*. This is akin to nearest neighbor classification. A benefit of this approach, besides its simplicity, is that when a website changes its formatting all that is necessary to update the schema is to cull a new set of keys from the website. In particular, there is no need to label data or retrain.

Our algorithm for generating schema takes two arguments: an alignment model and a set of keys, which are HTML parse trees. We assume that all fields occur as text (i.e., we do not extract accompanying images or formatting tags). The initial field labeling attempts to label the keys' textual vertices with numeric abstract field ids. The rule we follow is that a vertex should have the same field id as all of the vertices with which it aligns. The algorithm for doing this, illustrated in Figure 3, amounts to finding the connected components of graph:

- Initialize graph $G = (V, E)$ such that the edge set E is empty and the vertex set V is the set of text vertices from the keys.
- For every pair of keys find their minimum cost alignment $\mathcal{A}$.
  - For every $(u, v) \in A$, if u and v are both text vertices then add (u, v) to E

[1]Technically it is not EM since we there is no explicit concept of probability or expectation in our framework

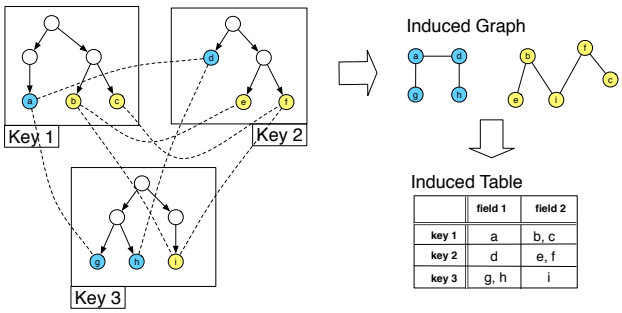

Figure 3. Key labeling algorithm. The first step is to align all pairs of keys. This induces a graph, of which we find the connected components. Each component is labeled with a field identifier, which is interpreted as a table.

- Assign a unique field label to every connected component of G containing more than one vertex. Every vertex assumes the field label of the component to which it belongs

The process for extracting fields from a new record is similar to the schema generation process. We simply align the record with each of the keys in the schema. Every textual vertex in the new record assumes the field label of the vertices with which it aligns; in the event of a conflict the vertex assumes the majority field label.

5 Preliminary Experiments

In this section we briefly describe our experimental results.

Search results were gathered from 13 popular websites. On each website the results for between 1 and 4 queries were collected. The search results were then partitioned into data records using the MDR tool developed by Liu et. al [12]. Eight sites (24 queries) were chosen as a training set. This set contained: Bed, Bath & Beyond, Epicurious, Google, Jet Blue, Metro North Schedule, RIT Course Schedule, Tiger Direct, and Zagats. The test set included 5 sites (16 queries): Bank Rate (BR), Choice Hotels(CH), Deep Discount DVD(DDD), Olive Garden(OG), and Walmart(WM).

To evaluate our methods we generated four separate schema for every query on every test site. The four schema included each of the tree alignment models described earlier as well as a baseline generator. As a baseline we used the following simple schema generator: align tow keys by traversing the in parallel and pairing two textual vertices that occur simultaneously in the traversal. We measured the

Query	# Queries	Baseline	Simple	SED	Semantic
BR	5	0.897	0.955	0.955	0.955
CH	2	0.838	0.960	0.956	0.946
DDD	2	0.796	0.867	0.845	0.859
OG	4	0.710	1.000	1.000	0.994
WM	4	0.883	0.849	0.901	0.835
Avg	3.4	0.830	0.930	0.939	0.923

Figure 4. Comparison of performance of different tree alignment models. Performance is measured by the percentage of correctly grouped textual vertices.

performance of each method as the percentage of pairs of text fields that are correctly grouped together.

The results of the experiments are given in the table in Figure 4. In general, the schema generators that rely on tree alignments outperform the baseline generator. However, there are cases where the baseline schema generator performs surprisingly well. This is likely due to the fact that some web sites have very little variation in the formatting of their search results.

The String Edit Distance (SED) model almost always outperforms all the other models. One possible explanation is that many web sites' search results contain a number of fixed strings like "Price" or "by". The vertices associated with these strings will align very well under the SED model since their edit distance is 0; these vertices essentially act as anchor points for the alignment.

It is disappointing that the Semantic model performed worse than even the Simple model. We believe this is due to overfitting in the features used. For example, one of the results contained the address "475 Ohio Pike". The features corresponding to addresses did not detect the word "Pike" and so that particular field was labeled incorrectly. Including the string edit distance as a feature would probably alleviate these kinds of issues.

6 Conclusion

We have presented an unsupervised framework for information extraction that enables higher-level data mining. Work is underway to demonstrate the power of our approach on a larger data set. An experimental comparison is needed to understand the relative benefits and weaknesses of our approach compared to other approaches. We have demonstrated the use of machine learning to learn tree alignment parameters and the results are promising. Our preliminary work also demonstrates how our framework can include a rich set of extensible cost functions. This feature makes our approach to learning tree alignment models broadly relevant since it can be accommodate any tree-type phenomena; in particular, natural and programming language grammars.

References

[1] A. Arasu, H. Garcia-Molina, and S. University. Extracting structured data from web pages. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 337–348, New York, NY, USA, 2003. ACM Press.

[2] L. Arlotta, V. Crescenzi, and G. Mecca. Automatic annotation of data extracted from large web sites, 2003.

[3] R. Baumgartner, S. Flesca, and G. Gottlob. Visual web information extraction with lixto. In *The VLDB Journal*, pages 119–128, 2001.

[4] P. Bille. A survey on tree edit distance and related problems. *Theor. Comput. Sci.*, 337(1-3):217–239, 2005.

[5] C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[6] C.-H. Chang and S.-C. Lui. IEPAD: information extraction based on pattern discovery. In *WWW '01: Proceedings of the 10th international conference on World Wide Web*, pages 681–688, New York, NY, USA, 2001. ACM Press.

[7] M.-S. Chen, J. Han, and P. S. Yu. Data mining: an overview from a database perspective. *Ieee Trans. On Knowledge And Data Engineering*, 8:866–883, 1996.

[8] E. D. Demaine, S. Mozes, B. Rossman, and O. Weimann. An $o(n^3)$-time algorithm for tree edit distance. Technical report, MIT, 2005.

[9] U. M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. The kdd process for extracting useful knowledge from volumes of data. *Commun. ACM*, 39(11):27–34, 1996.

[10] T. Joachims. Making large-scale support vector machine learning practical. pages 169–184, 1999.

[11] N. Kushmerick. Wrapper induction: efficiency and expressiveness. *Artif. Intell.*, 118(1-2):15–68, 2000.

[12] B. Liu, R. Grossman, and Y. Zhai. Mining data records in web pages. pages 601–606, 2003.

[13] I. Muslea, S. Minton, and C. A. Knoblock. Hierarchical wrapper induction for semistructured information sources. *Autonomous Agents and Multi-Agent Systems*, 4(1/2):93–114, 2001.

[14] D. Reis, P. Golgher, A. Silva, and A. Laender. Automatic web news extraction using tree edit distance, 2004.

[15] K.-C. Tai. The tree-to-tree correction problem. *J. ACM*, 26(3):422–433, 1979.

[16] I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun. Support vector machine learning for interdependent and structured output spaces.

[17] Y. Zhai and B. Liu. Web data extraction based on partial tree alignment. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 76–85, New York, NY, USA, 2005. ACM Press.

[18] J. Zhu, Z. Nie, J.-R. Wen, B. Zhang, and W.-Y. Ma. 2d conditional random fields for web information extraction. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*, pages 1044–1051, New York, NY, USA, 2005. ACM Press.