Probabilistic models for the dynamics of tree-structured data

Nilesh Dalvi NDALVI@YAHOO-INC.COM
Fei Sha FEISHA@YAHOO-INC.COM
Philip Bohannon PLB@YAHOO-INC.COM

Abstract

Many information sources on the web are generated by scripts and are highly structured, e.g., a movie website like `IMDB`. While these documents share a common HTML tree structure (or XML schema), the structure is not static and changes over time. The temporal changes tend to break many information extraction tools such as Wrappers which depend on precise knowledge of the structures.

In this paper, we investigate how to model the temporal changes. We view the changes to the tree structure as suppositions of a series of edit operations: deleting nodes, inserting nodes and substituting labels of nodes. The tree structures evolve by choosing these edit operations stochastically. We give algorithm to learn the probabilistic model from training examples that contain pairwise collections of past and present website HTML documents. The learnt probabilistic models can be used to evaluate the robustness of different information extraction tools. We demonstrate the effectiveness of this strategy in choosing robust wrappers on synthetic and real HTML document examples.

1. Introduction

A large fraction of information available on the Web is generated by scripts and is highly structured, e.g. academic repositories, product catalogs and entertainment sites. In many cases, the structure can be represented as an XML document tree. Figure 1 displays a simplified example of such trees for movie pages on a popular movie database site. Tree structures can be greatly exploited for information extraction. For instance, to extract the director names from each movie,

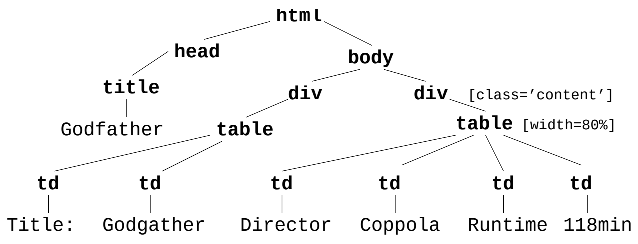

Figure 1. An HTML Webpage

we can use the following XPath expression,

$$W_1 \equiv \texttt{/html/body/div[2]/table/td[2]/text()}$$

which is a specification on how to traverse trees. In particular, the XPath expression W_1 above starts from the root, follows the tags `html`, `body`, the second `div` under `body`, `table` under it, the second `td` under `table` and then the `text`() under it.

The path expression W_1 is often called a *wrapper*. Using wrappers to extract information from structured pages has been an effective and dominant strategy. Wrappers can be handcrafted. However, in many cases, wrappers are inducted from a small number of labeled examples. Wrapper induction techniques have been extensively studied (Kushmerick et al., 1997; Muslea et al., 1998; Hsu & Dung, 1998; Crescenzi et al., 2001; Han et al., 2001; Sahuguet & Azavant, 1999; Baumgartner et al., 2001; Myllymaki & Jackson, 2002; Anton, 2005).

While wrapper induction is an effective way to extract information, it suffers from a fundamental problem: the underlying webpages frequently change, often very slightly, but causing the wrapper to break and requiring them to be relearnt. For instance, consider the wrapper W_1 above. It breaks if the structure of the underlying webpages change in any of the following way: a new `div` section gets added before the *content* section, or the first `div` is deleted or merged with the second `div`, a new `table` or `tr` is added under the second *div*, the order of *Director* and *Runtime* is changed, a new font element is added, and so on. Websites are

Appearing in the 6th *International Workshop on Mining and Learning with Graphs*, Helsinki, Finland, 2008.

constantly going through small edits and the breaking of wrappers is often an annoying issue.

Fortunately, there are often several wrappers that can extract the same information. We can choose among them a robust one with respect to small website changes. For example, each of the following XPath expressions extracts the director names:

$$\begin{aligned} W_2 &\equiv //\texttt{div}[@\texttt{class} =' \texttt{content}']/*/\texttt{td}[2]/\texttt{text}() \\ W_3 &\equiv //\texttt{table}[@\texttt{width} =' 80\%'/\texttt{td}[2]/\texttt{text}() \\ W_4 &\equiv //\texttt{text}()[\texttt{preceding}-\texttt{sibling} :: *[1] \\ &\quad [\texttt{text}() =' \texttt{Director}'] \end{aligned}$$

Each of them can be argued to be more robust than W_1. For example, the wrapper W_2 works even if new `div` tags are added or the *content* `div` is moved to a different part of the tree. However, which one among W_2, W_3 and W_4 should we choose?

One selection criteria is to choose the wrapper that has the least expected failure. Namely,

$$W_S^* = \arg\max_{W \in \mathcal{W}_S} \sum_T \texttt{succ}(W,T) P_S(T) \qquad (1)$$

where $\mathcal{W}_S$ is the set of candidate wrappers learned on the current tree S, $P_S(T)$ is the probability of the current tree S getting modified to tree T and the predicate `succ` is 1 if the wrapper W can successfully extract information from T. In this paper, we address the problem of learning probabilistic models of tree structure changes. In particular, we propose a model of tree structure change. The model is defined with three elementary edit operations, tree node deletion, insertion and substitution, each with a probability. Edit operations are carried out step-wize and form a memoryless Markov chain.

2. XML data model

Webpages can be viewed as *XML documents*, which we define next. Let $\mathcal{N}$ be the set of *nodes* which is infinite so as to accommodate arbitrarily large documents. Let Σ denote a finite set of *labels*. An XML document T is an ordered tree with nodes from $\mathcal{N}$ along with a function $\mathcal{L}$ that maps nodes of T to elements from Σ.

A forest F is a finite ordered list of trees. Let $[u]$ denote the subtree of F rooted at u and let $\lfloor u \rfloor$ denote the forest obtained from $[u]$ by deleting u.

3. Probabilistic model of XML structure change

Webpages undergo constant but generally small changes over time. These change reflect updates to either information (adding, removing or editing) or stylistic elements such as font change. In either case, the tree structures of the new documents are different from old ones, identifiable by adding and removing subtrees, changing node labels and etc. Note that what kinds of changes to be made at any given time are not deterministic, not only varying among websites but also varying among types of contents.

We propose to characterize the stochastic process that changes a tree S and generates a new tree T with a memoryless transducer. We begin describing this transducer in details in below, followed by a discussion on the inference problem in the transducer.

3.1. The XML tree edit transducer

While we are interested in XML tree structures, we find that it is more convenient to describe the stochastic process of tree changes in the more general term of forests. Let π be the process of changing a forest F to another forest G. The process π is defined recursively using two subprocesses π_{ins} and π_{ds} as follows.

Let $F_1, F_2, \cdots, F_K$ be the trees in F. Then

$$\pi(F) = \pi_{ins}(\pi_{ds}(F_1) \cdots \pi_{ds}(F_k)) \qquad (2)$$

where π_{ins} recursively maps a forest U to another forest V

$$\pi_{ins}(U) = \begin{cases} U & \text{with probability } p_{stop} \\ \pi_{ins}(e_1(U)) & \text{with the rest probability} \end{cases}$$

where $e_1(U)$ is an insert operation that adds a node at the top of U chosen randomly from all such operations. Specifically, $e_1(U)$ first chooses randomly a label $l \in \Sigma$ with probability $p_{ins}(l)$ and creates a new node. Then it chooses uniformly a random subsequence of $[1..K]$ as children of the new node. Furthermore, the probability $p_{ins}(l)$ is normalized $\sum_{l\in\Sigma} p_{ins}(l) = 1$.

The operator π_{ds} maps a *tree* S to a forest. It either deletes or substitutes the root of the tree and recursively transforms the subtrees of the tree S. Given a tree S with root s, we have

$$\pi_{ds}(S) = \begin{cases} \pi(\lfloor s \rfloor) & \text{with probability } p_{del}(\mathcal{L}(s)) \\ e_2(\pi(\lfloor s \rfloor)) & \text{with the rest probability} \end{cases}$$

where $e_2(U)$ is an insertion operation that creates a new root node whose children are all trees returned by $\pi(\lfloor s \rfloor)$. The label l of the new root is chosen randomly with probability $p_{sub}(\mathcal{L}(s), l)$. Notes that, we require $\sum_l p_{sub}(\mathcal{L}(s), l) = 1$.

To summarize, the generative process π is characterized by following parameters $\Theta =$

$(p_{stop}, \{p_{del}(l)\}, \{p_{ins}(l)\}, \{p_{sub}(l_1, l_2)\})\}$ for $l, l_1, l_2 \in \Sigma$ along with the following conditions:

$$\begin{array}{ccc} 0 < & p_{stop} & < 1 \\ 0 \leq & p_{del}(l) & \leq 1 \\ p_{ins}(l) \geq 0, & & \sum_l p_{ins}(l) = 1 \\ p_{sub}(l_1, l_2) \geq 0, & & \sum_{l_2} p_{sub}(l_1, l_2) = 1 \end{array} \quad (3)$$

Let $P_F(G)$ denote the probability that the process π applied to forest F stops and results in forest G. It is easy to show that:

Theorem 1 *If Θ satisfies all the conditions in eq. (3), then $P_F(G)$ is a probability distribution on the set of all forests, i.e. $\sum_G P_F(G) = 1$.*

3.2. Inference

For the XML tree edit transducer described in the above section, one important inference problem is to compute $P_S(T) = Pr[\pi(S) = T]$, ie, the probability of the current tree S changing into the tree T. We show below that this probability can be computed by dynamic programming.

Let F_s and F_t be subforest of S and T respectively. Let $\mathtt{DP}_1(F_s, F_t)$ denote the probability that $\pi(F_s) = F_t$. Let u and v denote the roots of the rightmost trees in F_t and F_t respectively. Note that every node in F_t is either newly created by some π_{ins} operator or is the result of a substitution by some π_{sub} operator from some node in F_S. Let $\mathtt{DP}_2(F_s, F_t)$ denote the probability that $\pi(F_s) = F_t$ and v was generated by a substitution under π.

We next show how to compute $\mathtt{DP}_1$ and $\mathtt{DP}_2$ recursively. Consider $\mathtt{DP}_1(F_s, F_t)$. There are two cases: (i) The node v was the result of an insertion by π_{ins} operator. Let p be the probability that π_{ins} inserts the node v in $F_t - v$ to form F_t. Then, the probability of this case is $\mathtt{DP}_1(F_s, F_t - v) * p$. (ii) The node v was the result of a substitution. The probability of this case is $\mathtt{DP}_2(F_s, F_t)$. Hence, we have

$$\mathtt{DP}_1(F_s, F_t) = \mathtt{DP}_2(F_s, F_t) + p * \mathtt{DP}_1(F_s, F_t - v) \quad (4)$$

Now consider $\mathtt{DP}_2(F_s, F_t)$. Again, there are two cases: (i) v was substituted for u. In this case, we must have $F_s - [u]$ transform to $F_t - [v]$ and $\lfloor u \rfloor$ transform to $\lfloor v \rfloor$. Denoting $p_{sub}(label(u), label(v))$ with p_1, the total probability of this case is $p_1 * \mathtt{DP}_1(F_s - [u], F_t - [v]) * \mathtt{DP}_1(\lfloor u \rfloor, \lfloor v \rfloor)$. (ii) v was substituted for some node other than u. Then, it must be the case that u was deleted. Denoting $p_{del}(label(u))$ with p_2, the total probability of this case is $p_2 * \mathtt{DP}_2(F_s - u, F_t - v)$. Hence,

$$\begin{aligned} \mathtt{DP}_2[F_s, F_t] &= p_1 \mathtt{DP}_1(F_s - [u], F_t - [v]) \mathtt{DP}_1(\lfloor u \rfloor, \lfloor v \rfloor) \\ &+ p_2 \mathtt{DP}_2(F_s - u, F_t - v) \end{aligned} \quad (5)$$

The functions $\mathtt{DP}_1$ and $\mathtt{DP}_2$ can be computed using a dynamic programming using Equation (4) and (5). We do not need to compute these functions for all pairs of subforests of S and T. We only need to compute them for *special subforests* as defined by Zhang and Shasha (Zhang & Shasha, 1989), who also show that the number of such subforests for a tree T is bounded by $|T| \min(D(T), L(T))$. Thus, we have:

Theorem 2 *Given trees S and T, the probability $Pr[\mu(S) = T]$ can be computed in time $O(|S||T| \min D(S), L(S) \min(D(T), L(T))$.*

3.3. Parameter estimation

To estimate the parameter Θ of the transducer from labeled data, we seek to maximize the log-likelihood $\ell(\Theta)$

$$\Theta^* = \arg\max_{\Theta} \sum_d \log Pr[\pi(S_d) = T_d] \quad (6)$$

where S_d and T_d are a pair of past and present documents. We use simple gradient-based algorithms to optimize the log-likelihood. The gradient with respect to Θ can be computed also in dynamic programming. The iterative update to Θ takes the form of

$$\Theta \leftarrow \Theta + \eta \frac{\partial \ell}{\partial \Theta} \quad (7)$$

where η is the learning step. Note that Θ is constrained in eq. (3). We can project the updated Θ back to the constraints. The update still converges to (local) stationary points.

Related Work There has been work on probabilistic tree transducers (Graehl & Knight, 2004) for natural language processing, but these models are not suitable for modelling changes in HTML webpages. More recently, there have been efforts to define a stochastic edit distance between trees (Boyer et al., 2007; Bernard et al., 2006). However, these models fail to define a valid probability distribution between trees as they do not normlize correctly.

References

Anton, T. (2005). Xpath-wrapper induction by generating tree traversal patterns. *LWA* (pp. 126–133).

Baumgartner, R., Flesca, S., & Gottlob, G. (2001). Visual web information extraction with lixto. *VLDB* (pp. 119–128).

Bernard, M., Habrard, A., & Sebban, M. (2006). Learning stochastic tree edit distance. *ECML* (pp. 42–53).

Boyer, L., Habrard, A., & Sebban, M. (2007). Learning metrics between tree structured data: Application to image recognition. *ECML* (pp. 54–66).

Crescenzi, V., Mecca, G., & Merialdo, P. (2001). Roadrunner: Towards automatic data extraction from large web sites. *VLDB* (pp. 109–118).

Graehl, J., & Knight, K. (2004). Training tree transducers. *HLT-NAACL* (pp. 105–112).

Han, W., Buttler, D., & Pu, C. (2001). Wrapping web data into XML. *SIGMOD Record*, *30*, 33–38.

Hsu, C.-N., & Dung, M.-T. (1998). Generating finite-state transducers for semi-structured data extraction from the web. *Information Systems*, *23*, 521–538.

Kushmerick, N., Weld, D. S., & Doorenbos, R. B. (1997). Wrapper induction for information extraction. *IJCAI* (pp. 729–737).

Muslea, I., Minton, S., & Knoblock, C. (1998). Stalker: Learning extraction rules for semistructured.

Myllymaki, J., & Jackson, J. (2002). *Robust web data extraction with xml path expressions* (Technical Report). IBM Research Report RJ 10245.

Sahuguet, A., & Azavant, F. (1999). Building light-weight wrappers for legacy web data-sources using w4f. *VLDB* (pp. 738–741).

Zhang, K., & Shasha, D. (1989). Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Comput.*, *18*, 1245–1262.