XML Document Transformation with Conditional Random Fields

Rémi Gilleron, Florent Jousse, Isabelle Tellier, Marc Tommasi

INRIA Futurs and Lille University, LIFL, Mostrare Project
first.last@univ-lille3.fr — http://www.grappa.univ-lille3.fr/mostrare

Abstract. We address the problem of structure mapping that arises in XML data exchange or XML document transformation. Our approach relies on XML annotation with semantic labels that describe local tree editions. We propose XML Conditional Random Fields (XCRFs), a framework for building conditional models for labeling XML documents. We equip XCRFs with efficient algorithms for inference and parameter estimation. We provide theoretical arguments and practical experiments that illustrate their expressivity and efficiency. Experiments on the Structure Mapping movie datasets of the INEX XML Document Mining Challenge yield very good results.

1 Introduction

Semi-structured documents in XML are omnipresent in today's computer science applications, since XML has become the standard format for data exchange. The essence of XML documents is their tree structure. Machine learning tasks dealing with XML structures should account for this fact. This is why *tree labeling* has become one of the most basic tasks in the XML context. It arises in information extraction from structured documents on the Web. The basic idea is to label nodes that are selected for some role positively and all others negatively. Another application is learning-based schema matching [1], where XML data for a source schema are labeled w.r.t. a target schema. In this case, matches are to be elaborated into mappings to enable XML data translation. In this paper, we consider the problem of XML document transformation which is ubiquitous in XML document engineering. For the INEX structure mapping task, the problem is to learn to transform layout-oriented HTML documents into content-oriented XML documents. The transformation task can be modeled by a labeling task, by assigning an operational semantic (local deletion, local inversion, etc.) to every element of the labeling alphabet. Thus we reduce the problem of learning to transform in the problem of learning to label.

The labeling task can be described as follows: given an observable $\mathbf{x}$, the problem consists in finding the most likely labeling $\mathbf{y}$, of the same dimension. Solutions to this problem based on graphical generative models try to evaluate from the labeled examples the joint probability distribution $p(\mathbf{y}, \mathbf{x})$. As we are only interested in labeling, we prefer to use conditional models that model the conditional distribution $p(\mathbf{y}|\mathbf{x})$ directly. Thus, for learning to label trees, we propose to extend on Conditional Random Fields (CRFs) introduced in [2]. Until now, CRFs have mainly been applied to sequence labeling tasks occurring in computational linguistic applications such as part-of-speech tagging, shallow parsing, but also to information extraction [3–7]. For an overview, see [8].

In this paper, we develop XCRFs, a new instance of CRFs that properly accounts for the inherent tree structure of XML documents. As a matter of fact, in an XML document, every node has an unlimited number of ordered children, and a possibly unbounded number of unordered attributes. Independence conditions in CRFs are governed by an undirected graph over random variables for labelings. The undirected graph for XCRFs is defined by: for ordered (parts of the) trees, the maximal cliques of the graph are all triangles consisting of a node and two adjacent children; for unordered (parts of the) trees, the maximal cliques are edges consisting of a node and one child. With such an undirected graph, in XCRFs, $p(\mathbf{y}|\mathbf{x})$ can be decomposed into a product of potential functions, each applying on a one-node clique, or an edge clique or a triangular clique. And these potential functions are themselves defined thanks to feature functions and parameters.

A contribution of this paper is to adapt the technical apparatus associated with CRFs to this new kind of dependence relationships. We define efficient algorithms for the inference problem and the parameter estimation problem in XCRFs. Because of the unranked property of XML trees, algorithms for XCRFs implement two recursions: a vertical recursion following the child ordering and an horizontal recursion following the sibling ordering using both forward-backward variables and inside-outside variables.

We have implemented XCRFs in a freely available system that allows to (learn to) label XML documents. In the experiments section, we evaluate our model on the Movie datasets from the Structure Mapping task of the INEX XML Document Mining challenge. This task consists in transforming layout-oriented HTML documents into data-oriented XML documents. To perform this task, we model such a transformation by a labeling of the HTML documents. We evaluate both the quality of the labeling of the HTML documents and the complete transformation. Results show that XCRFs perform very well on this task.

Related work

The idea to define CRFs for tree structured data has shown up recently. Basically, works differ in the graphical structure of CRFs. In [9], output variables are independent. Other approaches such as [10, 11] define the graphical structure on rules of context-free or categorial grammars. [12] have considered discriminative context-free grammars, trying to combine the advantages of non-generative approaches (such as CRFs) and the readability of generative ones. All these approaches apply to ordered ranked rather than unranked trees. As far as we know, their graphical models are limited to edges, not accounting for father-child-next-sibling triangles as in XCRFs.

XML data translation takes place in the domain of semantic integration. An overview can be found in [1]. For schema mapping, it has been shown [13] that

XCRFs can be compared with LSD [14]. But, applications of CRFs to more complex tasks and a comparison with recent systems for schema matching remain to be done. For information extraction, systems [15–20] deal with semi-structured input, mainly HTML data, but, to the best of our knowledge, structured output have not been considered so far by machine learning techniques.

For XML document transformation, Chidlovskii and Fuselier [21] address the problem of semantic annotation of HTML documents according to a target XML schema. They use a two-step procedure: terminals (leaves of the output document) are predicted by a maximum entropy classifier, then the most likely output tree is generated using probabilistic parsing for probabilistic context-free grammars. They suppose that leaves are in the same order in the input and the output document. The complexity of probabilistic parsing is cubic in the number of leaves, and therefore the system is not appropriate for large XML trees. Gallinari *et al* have considered generative stochastic models in [22]. Such models need to model input documents and to perform the decoding of input documents according to the learned model. The complexity of the decoding procedure could be prohibitive for large XML documents.

2 Conditional Random Fields for xml Trees

2.1 Conditional Random Fields

We refer to [8] for a complete introduction to CRFs. A CRF is a conditional distribution with an associated graphical structure. Let $\mathbf{X}$ and $\mathbf{Y}$ be two random fields, let $\mathcal{G}$ be an undirected graph over $\mathbf{Y}$. Let $\mathcal{C}$ be the set of all cliques of $\mathcal{G}$. The conditional probability distribution is of the form:

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{c \in \mathcal{C}} \psi_c(\mathbf{y}_c, \mathbf{x})$$

where ψ_c is the potential function for the clique c and $Z(\mathbf{x})$ is a normalization factor. Each potential function has the form:

$$\psi_c(\mathbf{y}_c, \mathbf{x}) = \exp\Big(\sum_k \lambda_k f_k(\mathbf{y}_c, \mathbf{x}, c)\Big)$$

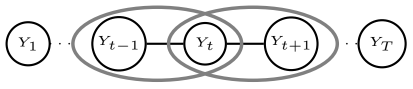
for some real-valued parameter vector $\Lambda = \{\lambda_k\}$, and for some set of real-valued feature functions $\{f_k\}$. This form ensures that the family of distributions parameterized by Λ is an exponential family. The feature function values only depend on $\mathbf{y}_c$, *i.e.* the assignments of the random variables in the clique c, and the whole observable $\mathbf{x}$.

The two main problems that arise for CRFs are:

Inference: given a new observable $\mathbf{x}$, find the most likely labeling $\hat{\mathbf{y}}$ for $\mathbf{x}$, *i.e.* compute $\hat{\mathbf{y}} = \arg\max_{\mathbf{y}} p(\mathbf{y}|\mathbf{x})$.

Training: given a sample set S of pairs $\{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}$, learn the best real-valued parameter vector Λ according to some criteria. In this paper, the criterion for training is the maximum conditional penalized log-likelihood.

Linear chain crfs



In first-order linear chain CRFs, the maximal cliques of the graph are pairs of consecutive nodes as depicted on the right. Thus, there are feature functions over node labels called node features, and feature functions over pairs of labels called edge features. For ease of notation, node features and edge features are merged and f_k denotes the k^{th} feature. The conditional probability can be written as:

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp\Big(\sum_{t=1}^{T} \sum_{k} \lambda_k f_k(y_{t-1}, y_t, \mathbf{x}, t)\Big) \tag{2.1}$$

In first-order linear chain CRFs, the inference task can be performed efficiently and exactly by the standard dynamic-programming Viterbi algorithm. For training linear chain CRFs, the problem is, given an input sample S, to learn the parameter vector Λ which maximizes the log-likelihood. The parameter vector can be learnt using traditional log-likelihood maximization methods. Since the optimal parameters cannot be found analytically, gradient ascent techniques are used. [23] has experimentally shown that the most effective technique in the case of linear-chain CRFs is the limited memory BFGS algorithm (L-BFGS) [24]. Both the function $Z(\mathbf{x})$ in the likelihood and the marginal distributions in the gradient can be computed by forward-backward techniques.

2.2 XCRFs: Conditional Random Fields for xml Trees

XML documents are represented by their DOM tree. We only consider element nodes, attribute nodes and text nodes of the DOM representation. Other types of nodes[1] are not concerned by labeling tasks. Attribute nodes are unordered, while element nodes and text nodes are ordered. We identify a node by a position which is an integer sequence n and we denote by x_n the symbol in a tree $\mathbf{x}$ in position n. The k ordered children of a node in position n are identified by positions $n.1$ to $n.k$. As a running example, consider the two XML trees $\mathbf{x}$ (on the top) and $\mathbf{y}$ (on the bottom) in Figure 1. The set of nodes for both trees is $\{\epsilon, 1, 1.1, 1.2, 2, 2.1, 2.2, 2.3, 2.4\}$, ϵ being the root. The symbol in position 2.1 in $\mathbf{x}$ is `td`; in its labeling $\mathbf{y}$, the label in position 2.1 is `name`. $\{2.1, 2.2, 2.3, 2.4\}$ is the set of children of 2. Ordered children (2.1, 2.2 and 2.3) are listed before unordered children (2.4).

With every set of nodes, we associate a random field $\mathbf{X}$ of observable variables X_n and a random field $\mathbf{Y}$ of output variables Y_n where n is a position. The realizations of X_n will be the symbols of the input trees, and the realizations of Y_n will be the labels of their labelings. In the following, we freely identify realizations of these random fields with ordered unranked trees.

For their ordered parts, the structure of XML trees is governed by the next-sibling and the child orderings. We translate this structural property into XCRFs defining maximal cliques of the undirected graph over an ordered unranked tree

[1] comments, processing instructions...

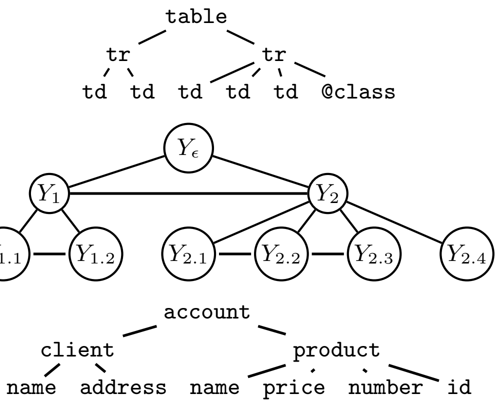


Fig. 1. An ordered unranked tree, its graph and its labeling.

$\mathbf{y}$ to be triangles $(Y_n, Y_{n.i}, Y_{n.(i+1)})$, for $i < o$, where o is the number of ordered children of n. For unordered parts of XML trees, the graph only includes pairs $(Y_n, Y_{n.i})$ because the next-sibling ordering is meaningless. In Fig. 1, we show the graph for our running example. Feature functions are thus defined over nodes (node features), pairs of nodes (edge features) and triples of nodes (triangle features). Triangle feature functions have the form: $f_k\big(y_n, y_{n.i}, y_{n.(i+1)}, \mathbf{x}, (n, n.i, n.(i+1))\big)$. Their arguments are the labels assigned to the node n and to two of its consecutive children $n.i$ and $n.(i+1)$, the whole observable $\mathbf{x}$, and the identifier of the clique in the tree $(n, n.i, n.(i+1))$. In fact, a triangular clique $(n, n.i, n.(i+1))$ can be shortly identified by $n.i$. We denote by $\mathcal{C}$ the set of cliques in the dependency graph. Every feature function f_k is associated with a real-valued parameter λ_k, defining the vector $\Lambda = \{\lambda_k\}$. It is worth pointing out that our model uses the same set of feature functions with the same parameters for every clique in the graph.

The introduction of triangle features is the main difference between linear chain CRFs and XCRFs. Training and inference algorithms need to be adapted because these triangle features bring both a horizontal and a vertical recursion in the graph structure. Therefore for ease of notation, we only consider such triangle features in the formal definition of the algorithms. The conditional probability distribution for an XCRF can be written as:

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{n.i \in \mathcal{C}} \psi_{n.i}(y_n, y_{n.i}, y_{n.(i+1)}, \mathbf{x}) \tag{2.2}$$

where

$$\psi_{n.i}(y_n, y_{n.i}, y_{n.(i+1)}, \mathbf{x}) = \exp\Big(\sum_k \lambda_k f_k(y_n, y_{n.i}, y_{n.(i+1)}, \mathbf{x}, n.i)\Big) \tag{2.3}$$

and

$$Z(\mathbf{x}) = \sum_{\mathbf{y}} \Big(\prod_{n.i \in \mathcal{C}} \psi_{n.i}(y_n, y_{n.i}, y_{n.(i+1)}, \mathbf{x})\Big) \tag{2.4}$$

2.3 Algorithms for XCRFs

Inference Algorithm for $\boldsymbol{Z(\mathbf{x})}$ The normalization factor $Z(x)$ must be computed to compute the normalized conditional probability $p(\mathbf{y}|\mathbf{x})$ and to compute the likelihood in training CRFs. In XCRFs, $Z(\mathbf{x})$ is defined in Eq. (2.4) as a sum over all possible labelings. It can be efficiently computed using dynamic programming. To do so, for every node n and every label $y \in \mathcal{Y}$, we define the *inside variable* $\beta_n(y)$ as the sum of the unnormalized probabilities over all the possible labelings of the subtree rooted in node n in which n is labeled with y. The recursive definition of $\beta_n(y)$ is 1 if n is a leaf, and $\beta_n(y) = \sum_{(y_1,\dots,y_m)\in\mathcal{Y}^m} \left(\prod_{i=1}^{m} \beta_{n.i}(y_i)\psi_{n.i}(y, y_i, y_{i+1}, \mathbf{x})\right)$ if n has m children. Clearly, the sum over all possible label assignments for the children of n again leads to a combinatorial explosion. Therefore we also use a dynamic programming technique for the horizontal recursion. For every node n with m children, for every $k \le m$, and for every pair of labels $(y, y') \in \mathcal{Y}^2$, we define the *backward variable* $\beta'_{n,k}(y, y')$ as the sum of the the unnormalized probabilities over all labelings of the subtree rooted in the node n whose $k-1$ first subtrees are deleted and where n is labeled with y and $n.k$ with y'. If $k = m$ we have $\beta'_{n,k}(y, y') = \beta_{n.m}(y')$, and otherwise

$$\beta'_{n,k}(y, y') = \beta_{n.k}(y') \sum_{y''\in\mathcal{Y}} \left(\psi_{n.k}(y, y', y'', \mathbf{x})\beta'_{n,k+1}(y, y'')\right)$$

Therefore, $\beta_n(y) = \sum_{y, y_1 \in \mathcal{Y}\times\mathcal{Y}} \beta'_{n,1}(y, y_1)$

Thus, $Z(\mathbf{x}) = \sum_{y\in\mathcal{Y}} \beta_\epsilon(y)$ can be computed in $O(N \times M^3)$ where N is the number of nodes of $\mathbf{x}$ and M is the number of labels in $\mathcal{Y}$.

Inference for XCRFs When computing $Z(\mathbf{x})$, the aim was to compute the sum of the unnormalized conditional probabilities for all labelings. Here, we want to compute the most likely labeling. The Viterbi recursion is obtained by replacing the `sum` function by the `max` function in the inside and backward recursions of the computation of $Z(\mathbf{x})$. Finding the most likely labeling $\hat{\mathbf{y}}$ then consists in the memorization of the Viterbi path associated with the maximum unnormalized conditional probability.

Training XCRFs Training an XCRF means learning its parameter vector Λ. We are given iid training data S of pairs of the form (observable tree, labeled tree). Parameter estimation is typically performed by penalized maximum likelihood. The conditional log-likelihood, defined as $\mathcal{L}_\Lambda = \sum_{(\mathbf{x},\mathbf{y})\in S} \log p(\mathbf{y}|\mathbf{x}; \Lambda)$, is used. This function is concave and the global optimum is the vector of parameters with which the first derivative is null. However, finding analytically this derivative with respect to all the model parameters is impossible. A gradient ascent (L-BFGS), which requires the calculation of the partial derivatives of $\mathcal{L}_\Lambda$ for each parameter, is therefore used. Replacing $p(\mathbf{y}|\mathbf{x};\Lambda)$ by its definition (*c.f.* equation (2.2)), $\mathcal{L}_\Lambda$ becomes:

$$\mathcal{L}_\Lambda = \sum_{(\mathbf{x},\mathbf{y})\in S} \sum_{n.i\in\mathcal{C}} \sum_{k} \lambda_k f_k(y_n, y_{n.i}, y_{n.(i+1)}, \mathbf{x}, n.i) - \sum_{(\mathbf{x},\mathbf{y})\in S} \log Z(\mathbf{x}) \quad . \tag{2.5}$$

Thus partial derivatives can be written as:

$$\frac{\partial \mathcal{L}_\Lambda}{\partial \lambda_k} = \sum_{(\mathbf{x},\mathbf{y})\in S} \sum_{n.i\in\mathcal{C}} f_k(y_n, y_{n.i}, y_{n.(i+1)}, \mathbf{x}, n.i) - \sum_{(\mathbf{x},\mathbf{y})\in S} \sum_{n.i\in\mathcal{C}} \sum_{y_1,y_2,y_3} P(y_1,y_2,y_3) f_k(y_1,y_2,y_3,\mathbf{x},n.i)$$

where $P(y_1,y_2,y_3) = p(Y_n = y_1, Y_{n.i} = y_2, Y_{n.i+1} = y_3|\mathbf{x};\Lambda)$. The computation of the first term is relatively straightforward. On the opposite, calculating the second one, *i.e.* the marginal probability for a given clique, is more difficult since it requires to sum over all possible labelings outside the clique. To make these computations tractable, we introduce a dynamic programming algorithm using both forward-backward variables and inside-outside variables.

For every node n and every label y, the *outside variable* $\alpha_n(y)$ is the sum of all the unnormalized probabilities over all the possible labelings of the context of the subtree rooted at the node n, in which n is labeled with $y \in \mathcal{Y}$. We have $\alpha_n(y) = 1$ if n is the root and $\alpha_n(y) = \sum_{y'\in\mathcal{Y}} \alpha_{n'}(y') \frac{\beta_{n'}(y')}{\beta_n(y)}$ if $n = n'.i$.

The *forward variable* $\alpha'_{n,k}(y,y')$ is introduced for horizontal recursion. It is defined as the sum of the unnormalized probabilities over all labelings of the subtree rooted in the node n whose $(m-k+1)$ last subtrees are deleted and where n is labeled with y and $n.k$ with y'. We have $\alpha'_{n,k}(y,y') = 1$ when $k=1$, otherwise

$$\alpha'_{n,k}(y,y') = \sum_{y''\in\mathcal{Y}} \Big(\beta_{n.(k-1)}(y'')\psi_{n.k-1}(y,y'',y',\mathbf{x})\alpha'_{n,k-1}(y,y'')\Big)$$

Then, the marginals can be computed by:

$$P(y_1,y_2,y_3) = \frac{1}{Z(\mathbf{x})}\alpha_n(y_1)\psi_{n.i}(y_1,y_2,y_3,\mathbf{x}) \alpha'_{n,n.i}(y_1,y_2)\beta_{ni}(y_2)\beta'_{n,n.(i+1)}(y_1,y_3) \tag{2.6}$$

Complexity Issues and Discussion We have shown that $Z(\mathbf{x})$ can be computed in $O(N\times M^3)$ where N is the number of nodes of $\mathbf{x}$ and M is the number of labels in $\mathcal{Y}$. This result can be extended to the computation of the marginal probabilities in the gradient. This leads to an overall complexity for training in

$O(N \times M^3 \times G)$ where N is the total number of nodes of the trees in the input sample S, M is the number of labels in $\mathcal{Y}$, and G is the number of gradient steps. For linear chain CRFs only a factor M^2 occurs.

We have presented the inference algorithms for XCRFs as extensions of the algorithms for linear chain CRFs. An alternative approach would be to consider XCRFs as a particular case of general CRFs. Indeed, the treewidth of undirected graphs for XCRFs is 2. For every graph associated with an XCRF, a junction tree can be computed in linear time. Then the belief propagation algorithm can be applied [25, 7]. The inference algorithms for XCRFs, given in the previous section, can be considered as inference algorithms for general CRFs using the knowledge of the tree-shaped graphical structures associated with XCRFs.

3 Experiments with the XCRF System

3.1 The XCRF System

The XCRF model is implemented by a freely available JAVA library [2]. For training, parameters are estimated by maximizing the penalized log-likelihood. The L-BFGS gradient ascent package from the "RISO Project"[3] is used. The system allows to label element, attribute and text nodes of XML trees. An XCRF is specified by an XML file. Feature functions are 0-1 valued functions defined by XPATH expressions. There are node features, edge features, attribute features (edge features for unordered children), and triangle features. An example of triangle feature is given in Fig. 2.

```
<Feature name="f_title" weight="7.46" xsi:type="TriangleFeature">
 <Y value="0" />
 <Yi value="title" />
 <Yj value="year" />
 <TestX value="parent::*/name() = 'tr'"/>
 <TestX value="name() = 'td'"/>
 <TestX value="name(following-sibling::*[1])='td'"/>
</Feature>
```

Fig. 2. XML definition of a feature function of weight 7.46 defined by $f_{\text{title}}(y_n, y_{n.i}, y_{n.(i+1)}, \mathbf{x}, n.i) = 1$ if $y_n = \perp$, $y_{n.i} =$'title', $y_{n.(i+1)} =$'year', $x_n =$'tr, $x_{n.i} =$'td', $x_{n.(i+1)} =$'td'

3.2 Feature Generation

In the following experiments, the set of feature functions we used were automatically generated from a set of labeled documents, typically the learning set.

[2] http://treecrf.gforge.inria.fr/

[3] http://riso.sourceforge.net/

Attribute	Description
nbChildren	number of children of the node
depth	depth of the node
childPos	node is the i^{th} child of its father

Table 1. Structure Attributes computed during the preprocessing

Attribute	Description
containsComma	text contains a comma
containsColon	text contains a colon
containsSemiColon	text contains a semi-colon
containsAmpersand	text contains an ampersand
containsArobas	text contains an arobas
isUpperCase	text is in upper case
firstUpperCase	first letter is in upper case
onlyDigits	text is made of digits
oneDigit	text is a single digit
containsDigits	text contains digits
rowHeader	text value of the row header in a table (HTML only)
columnHeader	text value of the column header in a table (HTML only)

Table 2. Text Attributes computed during the preprocessing

There are essentially two kinds of generic feature functions: structure features and attribute features.

Before the automatic generation of the feature functions, a first step consists in preprocessing additional attributes. These attributes are purely informative. Therefore, during the experiments, the XCRFs do not consider them as regular nodes and do not attempt to label them. These preprocessed attributes give additional information on the structure of the trees and basic information on the content of the text nodes. Tables 1 and 2 show the different kind of information given by these preprocessed attributes.

The first kind of automatically generated feature functions are structure features. These feature functions are node features, edge features and triangle features testing solely the node symbols (nodes can be element, attribute or text nodes) and labels. Let 1_p be 1 if and only if predicate p is true. If the tree in Figure 3 is part of the learning set, the following features are generated:

– Node features testing the label of node n its the node symbol, *e.g.* $1_{(y_n=\texttt{title})(x_n=\texttt{td})}$
– Node features similar to the previous one, but testing if the node symbol is different from the one in the learning set, *e.g.* $1_{(y_n=\texttt{title})(x_n\neq\texttt{td})}$
– Edge features testing the labels of a node n and one of its children i and the node symbols, *e.g.* $1_{(y_n=\perp)(y_i=\texttt{title})(x_n=\texttt{tr})(x_i=\texttt{td})}$
– Triangle features on the same principle as the edge features above, but testing on two consecutive children i and j of node n:
$1_{(y_n=\perp)(y_i=\texttt{title})(y_j=\texttt{year})(x_n=\texttt{tr})(x_i=\texttt{td})(x_j=\texttt{td})}$

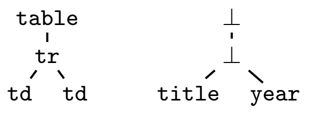


Fig. 3. Feature generation example

The second kind of feature functions that are generated are attribute features. These feature functions are based on attribute values. The attributes used to generate these features are both the ones originally in the corpus, for instance the `class` attribute of a `div` element in HTML, or attributes resulting from the preprocessing performed earlier. With all these attributes, for each node in the tree we generate feature functions testing the label assigned to this node and one or two attributes of the node itself, its father, grandfather, great-grandfather, previous sibling and next sibling. For instance, on the example in Figure 3, the following feature functions are generated:

$$1_{(y_n=\texttt{title})(father(x_n)@nbChildren=2)(x_n@childPos=1)}$$

$$1_{(y_n=\texttt{title})(x_{@}ndepth=2)}$$

With such a generation procedure, we get feature functions that are mostly domain independent. However, they prove to be a very relevant set of feature functions when dealing with well-structured XML documents.

3.3 Experiments on the Structure Mapping Task

This second experiment is on a bigger scale. It was conducted as part of the Structure Mapping task of the XML Document Mining Challenge [26]. The Structure Mapping task consists in transforming layout-oriented HTML documents into content-oriented XML documents. The dataset is made of HTML descriptions of movies taken from the website allmovie[4] and their XML counterpart taken from the IMDB repository[5]. Each document gives thorough information about a single movie such as the title, the director, the cast and crew, related movies, *etc.* Since the DTD of the output XML documents was not given as part of the challenge, we used the algorithm given in [27] to build it. The DTD contains 63 elements, among which 39 contain textual information.

There are two tasks with two different input HTML datasets, called html1 and html2. Both HTML datasets describe the same movies, but the documents in html1 contain only the HTML table describing the movie, whereas the documents in html2 also contain useless information. Documents from the html1 dataset are therefore subtrees of the documents in the html2 dataset. This makes the task a bit harder for html2. The average number of nodes in a document is 470 in the html1 dataset, and 530 in the html2 dataset.

[4] `http://www.allmovie.com/`
[5] `http://www.imdb.com`

Method We chose to model a transformation from an HTML document into an XML document by a labeling of the input HTML document.

In the HTML documents, a text node can contain several distinct information, which will appear in different text nodes in the output XML document. For instance, the release date, the movie format and the movie length are all located in the same text node of the HTML document. Therefore, as a preprocessing step, text nodes are tokenized in several nodes.

The labels used in the labeling task are: the names of the 39 elements of the output DTD which contain textual information, *e.g.* `director`, `synopsis`; the labels indicating that a node is part of a set of consecutive nodes corresponding to a single element of the output DTD,*e.g.* `director_continued`, `synopsis_continued`; the label $\perp$ for useless nodes. This leads to a total number of 79 labels. However, only 66 of them are useful, since some information are never split over several text nodes in the HTML documents (*e.g.* the year of release, the duration, *etc.*).

A very simplified example of a labeled HTML tree is given in Figure 4. The XML output documents are then computed using a simple post-processing function which takes a labeled HTML document and the output DTD as parameters.

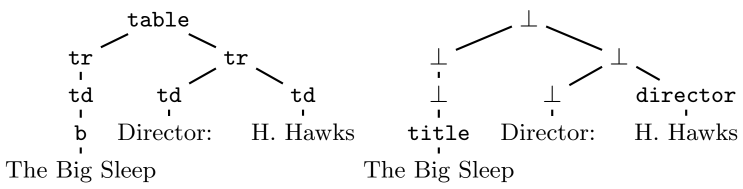


Fig. 4. a (simplified) HTML input tree (left) and its labeling (right) for the Structure Mapping task of the XML Mining challenge

We assume that the text nodes appear in the same order in the input HTML document and in its output XML counterpart, which is the case in this Structure Mapping task. With this assumption, one can easily build the learning set of labeled HTML documents from the original learning set of (HTML, XML) pairs of documents. Indeed, automatically finding the labeling of the HTML document corresponding to the transformation is straightforward. One only has to parse both the HTML and the XML documents at the same time and label the text leaves of the HTML document which occur in the XML document.

For the testing phase of our experiments, the assumption we made also gives us the ability to easily transform an HTML document into an XML document. Indeed, we first have to label the HTML document using XCRFs. Then, from this labeled document, and knowing the DTD of the output XML document and the fact that the text leaves will appear in the same order, we can build this output using a very simple XSLT stylesheet or an easily writable script.

An XCRF is defined with more than 7000 feature functions over the set of 66 labels. It should label HTML trees of average size 950 nodes (after tokenization).

A naive application should fail for complexity reasons: $7000 \times 950 \times 66^3 \approx 2.10^{12}$. Therefore, we had to use a technique called "sequential partition composition" of XCRFs to perform this task.

This technique consists in breaking the complexity factor M^3, where M is the number of possible labels in the XCRF, by combining several XCRFs where M is considerably smaller than the original one. Let $\mathcal{Y}$ be the original set of labels. First, we need to build a partition $\mathcal{Y}_1, \ldots, \mathcal{Y}_k$ of $\mathcal{Y}$. The choice of this partition is guided by the DTD of the output XML documents. For instance, all the information about an award (name, location and year) are in a single part, whereas the title of the movie is alone, since it is not directly related to any other information. With this policy, we obtained a partition of $k = 30$ subsets of labels containing from 2 to 6 labels.

Then, we define k XCRFs over the k parts of the partition. To label a document, these k XCRFs are applied sequentially, following the order on the subsets of labels. At step i, the XCRF labels the HTML document with the labels in $\mathcal{Y}_i$. Since the XCRFs are applied sequentially, labeling at step i can use the labelings performed at the previous steps j, where $j < i$. Previous labeling are encoded in the HTML document. This allows for long distance dependencies between these labels. Once the HTML documents have been labeled by the k XCRFs, for some nodes, two or more labels might have been predicted by different XCRFs. In this case, a choice needs to be made. We decided to choose the label with the higher marginal probability.

The training of the k XCRFs is also performed sequentially. When training the i^{th} XCRF, the learning set is composed of the HTML documents enriched with the labels of the previous subsets of labels $\mathcal{Y}_j$, where $j < i$.

Using this method, the XCRFs were trained on a learning set of 692 labeled documents. We evaluated them on a testing set of 693 documents. We first evaluate the quality of the labeling performed by the XCRF. Then, we evaluate the performance of our method on the overall HTML to XML transformation task.

Evaluation of the xcrf for the labeling task To evaluate the quality of the labeling performed by the XCRFs, we measure precision, recall and F1-measure for all the 66 labels in the task. In Table 3, we only show the micro-average of these results. Since these results might be biased by the great proportion of nodes labeled with $\perp$, *i.e.* nodes which are not used in the final transformation, we also give the micro-average without this insignificant label.

Dataset	Average method	Rec.	Prec.	F1
html1	Micro	93.00	94.11	93.55
	Micro (without $\perp$)	94.96	77.09	85.10
html2	Micro	92.41	93.10	92.76
	Micro (without $\perp$)	94.10	69.95	80.24

Table 3. Evaluation of the labeling on the Structure Mapping task

First, it is worth noticing that the micro-averaged results over all the labels are very good on both versions of the dataset. Both the recall and the precision are above 92%, which proves the quality of the labeling. When averaging over all the labels except $\perp$ (*i.e.* only on the significant labels), two behaviours occur. On the one hand, the recall increases, meaning that most of the significant information in the HTML documents are correctly identified. On the other hand, there is a drop in precision. The explanation is that some useless information sometimes occur in the HTML documents in a structural context very similar to that of significant information and the XCRFs can not make the difference between them. This drop is slightly more important with the html2 dataset. This is not a surprise since this dataset contains more useless information than html1. This drop could be avoided by using feature functions which uses longer distance dependencies on the observation.

Evaluation of the html to xml transformation Now, we evaluate the overall quality of the complete transformation from the HTML documents to their XML counterpart. This transformation is performed in two steps: first, the HTML documents are labeled with XCRFs; then a simple transformation using the output DTD and the labeled HTML documents produces the predicted XML documents. The evaluation is therefore a comparison between the predicted XML documents and the correct ones. To compare them, we compute the F1-measure according to two different criteria:

– **Paths:** for each text node in the predicted XML document, we build a couple made of the text content and the path to the text node. These couples are compared to those in the correct XML document.
– **Subtrees:** for each node in the XML document, we consider its subtrees, and the subtrees are compared to those in the correct XML document.

The main difference between these two criteria is that the first one does not take into account the order in which the text leaves appear. Overall, the second criteria is more precise and gives a better idea of the accuracy of the predicted XML documents.

Table 4 shows the average F1-measure over all the documents in the test dataset for the four criteria. First, we notice that, as when we evaluated the labeling, results show that our system performs very well on the html1 dataset. The results are very similar with both criteria. This is partly due to the text nodes appearing in the same order in both the HTML and the XML documents. Moreover, since the construction of the XML documents is guided by the output

Dataset	F1 on paths	F1 on subtrees
html1	91.81	89.76
html2	79.60	71.79

Table 4. Evaluation of the transformation on the Structure Mapping Task

DTD, our system always predicts XML documents conform to the DTD. Therefore, it is very unlikely that an undesired node is inserted. This explains the stability between the two measures.

With the html2 dataset, results are a bit lower, which is consistent with the results we observed when evaluating the labeling. Still, the F1-measure on the paths of the XML documents is good and close to 80. The drop of the F1-measure on the subtrees can be explained by the nature of the html2 dataset. Indeed, with this dataset, the XCRF sometimes failed to identify useless information. Therefore, these information are in the predicted XML documents. This results in several subtrees being incorrect, and a drop of the F1-measure, although the correct information are present in the predicted XML documents.

4 Conclusion and Future Work

We have introduced XCRF that are conditional models for labeling XML trees. We tackle the problem of structure mapping presented in this INEX challenge as a labeling problem for XML data. Experimental results show that XCRFs perform very well and confirm that XCRFs are well suited for such applications.

Our main line of future research involves extending our system to handle more sophisticated applications. Datasets we used in the challenge assume that document order is preserved during the transformation. To overcome this limitation, it can be necessary to extend the set of allowed editions operations. Another point is to integrate the transformation step in the XCRF training phase for a better parameter estimation. Other improvements include the combination of linear chain CRFs and XCRFs, and the introduction of ontologies, NLP outcomes, and domain knowledge to take advantage of both the structure and content of XML documents.