
Minoru Yoshida and Hiroshi Nakagawa
Information Technology Center, University of Tokyo,
7-3-1 Hongo, Bunkyo-ku, Tokyo 113-0033
{mino, nakagawa}@dl.its.u-tokyo.ac.jp

Abstract

We propose a novel approach for extracting semantic structures from web documents. Our task is to extract trees that describe the hierarchical relations in documents. We developed an algorithm for this task by using the Stochastic Context Free Grammar (SCFG) framework. Experiments showed that our approach effectively worked showing performance improvement through the parameter estimation.

1 Introduction

This paper proposes a novel approach for extracting semantic structures from web documents by focusing on headers. Most web documents contain headers that are crucial since they represent hierarchical aspects of semantic structures. Titles, headlines, and attributes are examples of headers. The left part of Figure 1 shows an example web document. In this document, the headers are About Me, which is a title, and NAME and AGE, which are attributes. We also regard the page title “John’s profile” as a header. They are usually presented without any linguistic clue that indicates they are headers. Note that, for example, NAME and John Smith are not related by any linguistic fragments, but by certain visual appearances such as “NAME is presented just above John Smith.” The headers can be regarded as tags annotated to other parts of the document. (For example, NAME can be seen as a tag annotated to John Smith.) Our research aims to extract the hierarchical relations embedded in web documents by extracting such headers.1

Our final goal is to extract semantic structures from HTML documents in the form of header trees. Header trees can be seen as a variant of XML trees. Further, each internal node is not an XML tags but a header that appears in the given HTML document. The left part of Figure 2 shows a header tree for the document shown in Figure 1. It should be noted that each node is labeled with parts of HTML documents; not labeled with abstract categories such as XML tags. The header trees extracted from web documents can be used to create various intelligent applications including the transformation of HTML to XML and audio-browsable web content [7].

The remainder of this paper is organized as follows. Section 2 defines the terms used in this paper. Section 3 provides the details of our algorithm. Section 4 lists the experimental results and Section 5 discusses about related work. Section 6 concludes this paper.

2 Definitions

Our system decomposes an HTML document into a list of blocks. Blocks are used as basic units of HTML documents in our algorithm. A block is defined as a part of the web document that is separated by a separator. A separator is a sequence of HTML tags and symbols. Symbols are

1Such relations are also embedded in sentences, such as “The name is John Smith and the age is 25.” This study focuses on supplementing text mining systems developed for sentences by extracting hierarchical relations that cannot be obtained from texts only.
defined as characters in texts that are neither numbers nor letters. The right part of Figure 1 shows an example of the conversion of an HTML document to a list of blocks.

A header is defined as a block that modifies subsequent blocks. In other words, a block that can be tagged annotated to subsequent blocks is defined as a header. Examples of headers are: titles (e.g., ‘About Me”), headlines (e.g., “Here is my profile”), attributes (e.g., “Name,” “Age,” etc.), dates (e.g., “2005/12/16”, etc.). On the other hand, blocks that do not modify any other block are called contents.

The system takes web documents as inputs, decomposes them into block lists, and produces header trees. The next section describes how to produce header trees from given block lists.

3 Web Page Modeling

We use stochastic context-free grammars (SCFGs) to model web documents. Each subtree in header trees is associated with nonterminal symbols in SCFGs, and each node is associated with terminal symbols. The right hand side of Figure 2 shows a parse tree corresponding to the header tree in the left hand side. Conversion from parse trees to header trees is performed by straightforward heuristic rules.

3.1 SCFGs

Formally, a stochastic context-free grammar is a four-tuple $G = (N,T,S,R)$ where $N$ is a set of nonterminal symbols, $T$ is a set of terminal symbols, $S$ is a start symbol and $R$ is a set of production rules. Production rules are of the form $X \rightarrow \gamma$ where $X$ is a nonterminal symbol (i.e., $X \in N$) and $\gamma$ is a string of symbols (i.e., $\gamma \in (N \cup T)^*$). Each rule has its probability $P(X \rightarrow \gamma)$ where $\sum_{X \rightarrow \gamma} P(X \rightarrow \gamma) = 1$. We define an SCFG grammar for document parsing as follows.

Rule-0 (Start rule): $S \rightarrow H_e$

RuleSet-1 (Rules for headed-contents):
$H_e \rightarrow H_0 H_e | H_0 C | H_e | H_0 H_L$

RuleSet-2 (Rules for contents): $C \rightarrow CC|C_0$

RuleSet-3 (Rules for header-lists): $H_L \rightarrow H_0 H_L | H_0$

The symbol $|$ is used to represent disjunctive rules. For example, $C \rightarrow CC|C_0$ means that there are two rules, say, $C \rightarrow CC$ and $C \rightarrow C_0$. Here, $H_e$ stands for parts of documents consisting of headers and contents (called headed-contents) and $C$ stands for parts of documents without headers (i.e., consisting only of contents). $H_L$ stands for header lists, which are the list of blocks each of which is appropriate to be regarded as a header rather than a content. $H_0$ is a header that directly produces a terminal symbol (i.e., not further transformed into other nonterminals) and $C_0$ is a content that directly produces a terminal symbol.

We refer to each rule as “rule $x,y$” when the rule is the $y$-th rule in the rule set $x$. (For example, $H_e \rightarrow H_0 H_e$ is referred to as rule 1.1 and $H_e \rightarrow H_e H_e$ is as rule 1.3.) In short, rule 1.1 and rule 1.2 are for contents modified by a header, rule 1.3 is for parallel headed-contents and rule 1.4 indicates the start of header lists. Rule 2.1 and 2.2 are for expansion of contents, and rule 3.1 and 3.2 are for expansion of header lists.

In addition, the grammar has block emission rules which emit blocks from nonterminal $H_0$ or $C_0$. All of these rules has its production probabilities $P(r)$. Note that probability values of all rules having the same left symbol sum up to 1 (e.g., the probabilities of all rules in RuleSet-2 must sum up to 1.)

For example, the production of document in Figure 1 is as follows. Given a start symbol $S$, a headed-content $H_e$ is produced by rule 0. From the $H_e$, two symbols $H_0$ and $H_e$ are produced by rule 1.1. After that, from the $H_0$, the block “John’s profile” is produced, etc.

We use $T$ to denote parse trees. For every $T$, the list $tr(T)$ indicates the list of rules (except for block emission rules) used to produce $T$, and $n(T)$ indicates the list $(N_1, N_2, \ldots, N_n)$ of nonterminal symbols, in which $N_i$ is a nonterminal symbol that produce the $i$-th block $b_i$. (Thus, $N_i$ must be $H_0$ or $C_0$.) Each rule $r \in tr(T)$ is represented by $(n,l,m,r)$ where $n$ is the rule number, $(l,m-1)$ is a span of the first nonterminal in the right hand side, and $(m,r-1)$ is a span of the second nonterminal in the right hand side.

Examples of such header lists include list of web page titles with link tags.
3.2 Model Parameters

A document is specified by its list of blocks $B = (b_1, b_2, \ldots, b_n)$ and its layout $L$. The probability of parse tree $T$ given document $d$ is defined as follows:

$$P(T|B, L) = \frac{P(T)P(B|T)p(L|T)}{\sum_T P(T)P(B|T)p(L|T)}$$

The system searches for the most appropriate parse tree $T$ for given document $d$:

$$\arg\max_T P(T|B, L) = \arg\max_T P(T)P(B|T)p(L|T)$$

The probability for $T$ is given as $P(T) = \prod_{r \in \text{tr}(T)} P(r)$ where $\text{tr}(T)$ is a list of rules used to produce tree $T$. The probability for $B$ conditioned on tree $T$ is given in the following way:

$$P(B|T) = \prod_i P(b_i|T) = \prod_i P(N_i \rightarrow b_i)$$

where $N_i$ is the nonterminal symbol that produces $b_i$ in parse tree $T$. We use a suffix for each block to avoid a data sparseness problem. We use a character unigram, bigram, or trigram (changed according to character types such as “trigrams for alphabets and numbers”, “unigrams for Kanji characters”, etc.)

3.3 Layouts

Next, we give the definition for layout $L$. $L$ is specified by several features: margin $m_i$ (the number of lines between $b_i$ and $b_{i+1}$), margin difference $md_i$ (the sign\(^7\) of $m_{i+1} - m_i$), similarity $sim_{ij}$ (the similarity value between $b_i$ and $b_j$), font size $fs_i$ (the font size of $b_i$ defined by heuristic rules\(^8\)), font size difference $fd_{ij}$ (the sign of $fs_j - fs_i$), sentence indicators $si_i$ (the value from $\{1, 0\}$ that indicate whether periods, $\cdot$, or \(\ldots\) follows $b_i$ (1) or not (0)).

The likelihood function $p(L|T)$ is defined as

$$\prod_i P(md_i|T)P(si_i|T)\prod_{j>i} P(fd_{ij}|T)p(sim_{ij}|T).$$

\(^5\)Another strategy is to use word $n$-grams instead of character $n$-grams. We chose character $n$-grams because we performed studies on Japanese web documents that cannot be easily split into words. The use of character $n$-grams guarantees the robust extraction of frequent suffixes, and the system can be applied to multilingual documents without any word-segmentation module.

\(^6\)We use pre-defined rules to assign the margin value to each pair of neighboring blocks according to HTML tags between them.

\(^7\)Signs are elements from $\{-, 0, +\}$

\(^8\)Similarities are defined as visual similarities, as the cosine similarity between the separators surrounding the two blocks.

$P(md_i|T)$ is determined according to the value of $N_i$, that is, $P(md_i|T) = P(md_i|N_i)$. We assume that a header represents a top of some region and intra-region margins are often smaller than the inter-region margins, resulting in high probability values for $P(md_i = -1|N_i = H_0)$. Currently, we set $(P(-|H_0), P(0|H_0), P(+|H_0)) = (0.8, 0.12, 0.01)$ and $(P(-|C_0), P(0|C_0), P(+|C_0)) = (0.1, 0.9, 0.1)$.

$P(si_i|T)$ is estimated based on the assumption that sentence indicators are likely to be emitted from successive $C_0$. Here, successive means that two or more $C_0$ continue on frontiers of trees. Then, $P(si_i|T) = P(si_i|N_{i-1}, N_i, N_{i+1})$, and $P(si_i|N_{i-1}, N_i, N_{i+1})$ is defined as $(P(?|\text{succeed}), P(0|\text{succeed}), P(1|\text{succeed}), P(0|\text{non-succeed}) = (0.5, 0.5, 0.9, 0.1))$, where the condition is succeed when $N_{i-1} = N_i = C_0$ or $N_i = N_{i+1} = C_0$ and non-succeed otherwise.

$P(fd_{ij})$ is uniform ($P(fd_{ij}) = 1/3$) except for some $(i, j)$ where $i$ and $j$ are indexes in rules for header-content relations, that is, $(i, j) = (l, m)$ for rules $(n, l, m, r) \in \text{tr}(T)$ where $n = 1.1$ or $1.2$. We assume that font size of headers are likely to be larger than their contents. This means that the value of $P(fd_{im})$ is likely to be large if $N_i$ is a header for $N_m$. Currently, we set $P(fd_{ij}|k = l, j = m)$ to $(P(-), P(0), P(+)) = (0.65, 0.25, 0.1)$.

$p(sim_{ij})$ is estimated based on the assumption that headers in parallel structures are likely to have similar visual appearances. The probability density function for similarities is uniform ($p(sim_{ij}) = 1$) except for some $(i, j)$ where $i$ and $j$ are indexes in rules for parallel structures, that is, $(i, j) = (l, m)$ for rules $(n, l, m, r) \in \text{tr}(T)$ where $n = 1.3, 1.4, 3.1$. $p(sim_{im})$ is a step function whose (proportion of) values are $0.0 : 0.1 : 0.35 : 0.65$ for $[0.0, 0.1), [0.1, 0.8), [0.8, 0.9), [0.9, 1.0]$, respectively.

3.4 Implementation Issues

3.4.1 Chart Parsing

We use a CKY-style chart parsing algorithm for SCFGs, in which $chart[N, i, j]$ represents a likelihood of the subtree that is rooted by $N$ and spans from $b_i$ to $b_{j-1}$. Each $chart[N, i, j]$ can be filled by $O(n)$ time complexity by taking rules $(n, i, m, j)$ for all $i < m < j$ and multiplying $P(n)$, $P(fd_{im})$, and $p(sim_{im})$ according to the rule number $n$.

3.4.2 Parameter Estimation

Parameters of our model include grammar parameters (probabilities for rule selection: $P(r)$), layout parameters

\(^{10}\)We use $P(N_i \rightarrow b_i)$, $P(md_i)$, and $P(si_i)$ are multiplied in $chart[N, i, i+1]$ in initialization steps.
(likelihood for layout emission: $p(L|T)$), and linguistic parameters (probabilities for block emission: $P(N \rightarrow b)$). The first two are set in a heuristic manner. However, it is difficult to do that for linguistic parameters. $P(N \rightarrow b)$ is estimated in an unsupervised manner by using the EM algorithm where $P(N \rightarrow b)$ is reestimated in each iteration while other parameters are left unchanged, by using the inside-outside style algorithm.

3.4.3 Preprocessing

Before model estimation described above, the system calls preprocessing routines to perform some data cleaning on input web documents. The main task here is to distinguish symbols in a block (intra-block symbols) and symbols between blocks (inter-block symbols). For example, symbol : in string 12:00 is a intra-block symbol, while the same symbol in age: 25 is inter-block symbol. The discrimination is performed by using heuristic rules based on the assumption that intra-block symbols do not appear with white spaces.

We use a decision-list based algorithm for preprocessing. In short, if (symbol, left-block, right-block) triples whose ratio of the number of appearances with white spaces to the total number of appearances is greater than a threshold, the symbol is regarded as an inter-block symbol and defined as a separator.

4 Experiments

We collected a set of web documents from www.infoweb.ne.jp with a moderate file size; these documents were chosen not to include “src” or “script” tags. The former criteria is based on the observation that too small or too large documents are difficult to use for measuring the performance of algorithms, and the latter criteria is based on the fact that our system currently has no module to handle image files as blocks.

The performance of web-page structuring algorithms can be evaluated via the nested-list form of a tree by bracketed recall and bracketed precision [4]. Recall is the ratio of the number of correct bracketings (i.e., the number of machine-human agreements) to the number of all manual bracketings, while precision is the ratio of the number of correct bracketings to the number of all machine-made bracketings. F-measure is a harmonic mean of recall and precision; it is used as a combined measure. We randomly selected 50 documents as test documents. Each test document was manually bracketed to evaluate machine-made bracketings. The number of open/close bracket pairs was 1,388. This set also was used as a training set, so the test was a closed test. The training data for the preprocessing was a set of 3,000 documents and training data for the estimation of block emission parameters was a set of 100 documents.

Recall and precision were evaluated for each test document, and they were averaged across all test documents. We call these averaged values macro-averaged recall, precision, and f-measure on the analogy of the measure used in the information retrieval research area [10]. Note that macro-averaged f-measure is not calculated from macro-averaged recall and precision (i.e., macro-averaged f-measure is not a harmonic mean of macro-averaged recall and precision.) On the other hand, micro-averaged recall, precision, and f-measure are defined as recall, precision, and f-measure calculated on one document made by concatenating all documents in the test set.

Figure 3 shows the result (shown as “w rule 1.4”). We observed that the estimation of parameters for block emission probabilities improved the performance from (micro, macro) = (0.431, 0.418) to (0.477, 0.438). Improvement was due to, for example, detecting sentences without sentence indicators by learning sentence-like suffixes, etc. Performance reached the best at the 9th iteration to (0.476, 0.444). We also investigated the performance when a more simplified grammar, which consist only of rules 0, 1.1, 1.2, 1.3, 2.1, and 2.2. (i.e., when rules for header lists are omitted.) The result is also shown in Figure 3 (shown as “w/o rule 1.4”). The best micro-averaged f-measure was 0.449 and the best macro-averaged f-measure was 0.405. Although omitting rule 1.4 did not harm the performance so seriously, we observed that the rule contributed to the performance.

We also evaluated the performance when (a) no layout features, (b) only margins, (c) margins and similarities, (d) margins, similarities, and sentence indicators, were used. The best f-measures for each setting were: (a)(0.226, 0.220), (b)(0.306, 0.282), (c)(0.419, 0.386), and (d) (0.471, 0.440). We observed that font size features were not so effective, but other three types of features contributed the significant performance improvement.

5 Related Work

Several studies have addressed the problem of extracting logical structures from general HTML pages without labeled training examples. One of these studies used domain-specific knowledge to extract information used to organize logical structures [2]. However, their approach cannot be applied to domains without any knowledge. Another study employed algorithms to detect repeated patterns in a list of HTML tags and texts [11, 9], or more structured forms [7, 3, 1] such as DOM trees. This approach might be use-

---

11 Most of them were written in Japanese.
12 from 1,000 to 10,000 bytes
13 These tags indicate inclusion of image files, java codes, etc.
14 For example, the nested-list for the header tree in Figure 2 is [ John’s Profile, [About Me, [NAME, [John Smith]], [AGE, [25]]]].
ful for certain types of web documents, particularly those with highly regular formats such as www.yahoo.com and www.amazon.com. However, there are also many cases where HTML tag usage does not have significant regularity, or, HTML tag patterns does not reflect semantic structure (and instead symbol patterns do so); further, there are cases when headers don’t repeat at all. Therefore, this type of algorithm may be inadequate for the task of header extraction from arbitrary web documents.

In recent years, some studies on extracting titles or headlines have been reported.[6, 8] Our task differs from those in that their methods focus only on titles (and headlines) and ignore the other parts of Web documents, while our algorithm handles all parts of web documents and provides a tree structure of the entire document; this algorithm enables the system to extract various types of headers other than titles and headlines, such as attributes. Especially, our approach has the advantage that it can handle symbols as well as HTML tags, making the system applicable to many private (not so formal) web documents.

Use of grammars and parsing techniques for document understanding, including stochastic context-free grammar based algorithms, have actively been studied especially in the last decade. Although most of work in this line was about understanding of printed document images by using 2-dimentional grammars, [5] proposed an algorithm that uses 1-dimentional SCFGs for understanding of texts extracted from business card images. However, their method require pre-defined grammars and annotated training data for target domains and they report only about business card understanding. On the other hand, our method is for web documents in general, proposing several kinds of features useful for understanding of many kinds of web documents.

6 Conclusions and Future Work

This paper proposed an algorithm to extract the header trees that give hierarchical structures to web documents. Experiments showed that our approach effectively worked showing performance improvement through the parameter estimation. However, the performance of the algorithm was not sufficient to be used as a base for other applications. Also, further investigation about categorization errors in preprocessing routines is needed. We plan to improve the performance by, for example, applying state-of-the-art machine learning algorithms to preprocessing, or using larger amount of training examples and using more complex grammars, etc.

References