STAN: Structural Analysis for Web Documents

Johannes Goller
FAST Search & Transfer
Am Rindermarkt 7, 80331 Munich, Germany
Johannes.Goller@fastsearch.com

Abstract

In this paper we present STAN, a structural analysis tool used for classifying web documents while at the same time extracting meaningful information from them. The extraction and classification rules are defined in terms of a structrural grammar *operating on both layout properties and content properties of the document. Stan was designed to accept HTML as input and is able to process documents at a speed of several MB/s (depending on the complexity of the structural grammar used).*

1 Introduction

Traditional search on collections of web documents, as it is offered by major internet search engines, but also by site-specific portals or company-specific enterprise search engines, is generally based on keyword search, sometimes enhanced by phrase search, proximity search, or specific ranking boosts given to certain, easy to identify parts of the documents (e.g. the title given in terms of the title tag).

However, often there is need for giving boost to parts of the document that cannot be extracted in a trivial manner. Examples include the automatic extraction of author names from scientific web documents (when searching for papers authored by a particular person), or the automatic recognition of addresses (e.g. to enable search for *geographical closeness* of web documents in addition to purely text-based similarity measures).

Depending on the degree of uniformity among the documents of the collection, it might in many cases be possible to define the passages of the documents one wants to extract in terms of their HTML markup, or to manually mark up those passages and have some wrapper algorithm learn the extraction rules ([1, 5, 6, 2, 3, 4]).

With both approaches a question arises about how to represent the properties of the HTML markup surrounding the relevant portions of the text, as well as how to represent properties of the text itself. In some cases, the conditions imposed on both markup and text can be stated very precisely (e.g. product descriptions from the web pages of a specific company are likely to be marked-up all in precisely the same unique manner, surrounded by a very specific sequence of HTML tags), whereas in other cases text and layout properties are of a rather general nature (e.g. author names from randomly chosen scientific web documents).

The approach taken in this study aims at being applicable to a great variety of such extraction tasks, therefore it attempts to offer a means of description of the target text as flexible as possible. Furthermore, there is a natural extension of the basic information extraction feature, namely, the capability to classify the whole document as belonging to a certain type before carrying out the actual extraction of text; in STAN, this extension has been closely integrated with the general extraction principle.

In addition, STAN was designed to be part of large-scale internet search engines, hence extremely high processing speed was another indispensable goal. Finally, the tool was envisaged to be at some time able to automatically learn extraction rules, therefore we aimed at designing it in a way that wouldn't hamper such an extension.

2 General methodology

When processing an HTML document, STAN performs the following main steps:

Step 1 Parse the HTML and generate a representation of the text nodes of the document tree reflecting their layout and textual properties, as well as a representation of their relative positions;

Step 2 Apply a number of BLOCKTYPE rules to text nodes belonging together which operate on the text nodes' features and assign one or several text block types to each of them;

Step 3 Apply a number of COMPLEX BLOCKTYPE rules to the – now typed – text blocks, combining some of the text blocks to complex objects;

Step 4 Apply DOCTYPE classification rules to the network of typed (partly complex) text blocks and assign one or several types to the entire document;

Step 5 Print out of blocks that were assigned relevant types.

In the following, we give a brief description of each of these steps.

2.1 HTML Parsing

The HTML parsing algorithm used for STAN first goes through the document in one pass and generates a representation of each text node (i.e. each piece of text between two HTML tags) along with its layout features defined by the enclosing HTML tags; in addition, it looks up words and phrases in a finite state automaton of typed words and phrases. The words in the lexicon and the types associated with them have to be provided to STAN in advance. The purpose of the lexicon lookup is to obtain a notion of the textual properties of each text node by counting words of certain types. E.g. for an application that aims at extracting addresses, one would define a lexicon of street names, city names, zip codes etc. and have STAN count these types.

After the parser finished, each text node n is represented as a tuple (S^n, F^n, L^n) where

$$S^n = (s_c^n, s_w^n, s_s^n)$$

represents *size* properties of the node (number of characters, words, and sentences respectively),

$$F^n \subseteq \{\text{anch}, \text{high}_0, \text{high}_1, \text{high}_2, \text{high}_3, \text{title}, \text{meta}, \text{url}, \text{ext}\}$$

represents *format* properties, and

$$L^n = (l_1^n, \ldots, l_m^n)$$

represents *lexical* properties by indicating the count for each word type defined in the lexicon (we assume m word types). The format properties are given as a set of *format specifiers*, each referring to a certain layout feature text nodes may have: they can be anchor texts (anch), highlighted to a certain degree (high$_0, \ldots,$ high$_3$), they can be the title of the document (title), meta keywords (meta), the URL of the document (url, it's treated like a text node), or external reference (ext, e.g. external anchor text pointing to the document). The highlighting degree is computed based on an evaluation of font size, color (compared to the predominant color of the document), and boldness, which in turn are calculated from an interpretation of the enclosing HTML tags.

In addition to representing each single node, also the relative positions of the text nodes are computed, as well as a rough approximation to the distance between adjacent text nodes. To this end, all HTML tags that have a separating effect (such as `<br>`, `<p>`, `<li>` etc.) were assigned a score ranging $1-4$, indicating *line break* (1), *paragraph break* (2 or 2×1), *visual separator* (3, e.g. `<hr>`), *conceptual separation* (4, e.g. when one text block is part of the head, while the other is part of the body). During parsing, pointers are placed inside the text nodes, tying together those that are next to each other, along with an indication of in what sense they are adjacent (left-right vs. up-down) and what the distance is (if there is more than one separator tag in between, the maximum distance score among them is used). Note that the only way left-right associations can occur is through table structures – which are widely used and often nested and very complex.

Of course, determining the exact adjacency relation of the text nodes, i.e. finding out which text nodes are side by side with which other text nodes, is not possible unless one (i) knows the size of the pictures in the document, and (ii) assumes a window and font size and some other rendering parameters a browser might use. Both these factors are not taken into account: wherever the adjacency relation among text nodes in a table cannot be determined precisely, *all* text nodes that could potentially be neighbours of each other are assumed to be adjacent (which is the case e.g. in a table field that has a neighbour field split in two parts).

2.2 BLOCKTYPE rules

A *text block* is defined as the collection of all text nodes located on one line and not separated by visual vertical separators (as in many tables). Based on this definition, STAN transforms the network of text nodes into a network of text blocks by re-establishing the adjacency relation as induced by the adjacency relation among the text nodes and by computing for each text block b a representing tuple (S^b, F^b, L^b) according to the following rules; (b itself is formalized as the set of the nodes belonging to it.)

- Set $s_c^b = \sum_{n \in b} s_c^n$, and correspondingly for s_w^b, s_s^b;
- Set $F^n = \bigcup_{n \in b} F^n$, but remove all high$_i$ elements from the set, replacing them by only one new highlighting element, re-computed according to the average font size, color etc. score in b;
- $L^b = \sum_{n \in b} L^n$.

After that STAN attempts to assign types to each of the text blocks. The available types have to be defined by the user in advance in terms of BLOCKTYPE rules given in a configuration file. A BLOCKTYPE rule specifies a type (i.e. a semantic descriptor such as "author section", "street line of an address" etc.) and defines the conditions a text block has to meet in order to be assigned that type. The conditions operate on the representing tuple of the text block and are given in terms of a list of *model tuples* one of which a text block has to *match* in order to be assigned that type. A model tuple is basically a triple of conditions (C_S, C_F, C_L), one for each of the elements of the representing tuples. Instead of a lengthy precise definition, we only give one example here, which hopefully conveys sufficient information to understand how these rules work:

```
LEXTYPE lxStreet
  (wtStreet,1-3),
  (wtNumber,>=1)

BLOCKTYPE Street
  ((words<7),(0-2),lxStreet)
```

That is, C_S is `(words<7)`, indicating a restriction of the maximum number of words, C_F is `(0-2)`, meaning that the highlighting of the line should be high$_0$, high$_1$, or high$_2$, C_L is a reference to a definition given separately of the conditions imposed on the vector L^b. Note that we assume two word types here, `wtStreet` and `wtNumber`, the former referring to words that were identified to be street names (by virtue of a lexicon), the latter referring to strings that were recognized as numbers. As a whole, this BLOCKTYPE type is supposed to identify text blocks that are likely to be that part of an address that specifies the street (and house number) – it was one of several alternative rules for such lines used for extracting addresses. More complex combinations of the features accessible in (S^b, F^b, L^b) can be defined in the same fashion.

2.3 COMPLEX BLOCKTYPE rules

COMPLEX BLOCKTYPE rules are also specified in the configuration and enable STAN to merge certain combinations of typed text blocks into bigger blocks, also assigning a type to them. There is a variety of alternatives for how complex blocks can be defined: as pairs of adjacent blocks, as lists of blocks of a certain type, or with a minimum density of blocks of a certain type, or as sequences of particular block types. For instance, an address can be specified as a sequence `owner block`, `street block`, `city block`, provided that these types were defined in BLOCKTYPE rules. Of course, alternative sequences can be defined and wildcard block types can be used in the sequence in case not all parts of an address can be recognized. Note that, as yet, the complex rules focus on vertical, sequence-like block combinations. Horizontal structures cannot be defined very accurately at this time, but obviously, such extensions are possible.

2.4 DOCTYPE rules

In a way similar to COMPLEX BLOCKTYPE rules a type can be found for the entire document, if the order or density of certain textblock types meets restrictions specified in the configuration. See below for a simplified example of such a rule; it has been taken from an application for the automatic detection of scientist homepages:

```
DOCTYPE ScHomepage
  30,
  (NameTitle,10),
  (HpHeading,10),
  (HpHeading,200),
  {BiogrTxt,1}
```

This (being one of several alternative rules) describes a scientist homepage as consisting of (1) a title section with a proper name (recognized by a BLOCKTYPE rule using appropriate lexica of first and last names), (2) a characteristic heading ≤ 10 blocks after the title (e.g. "Research Interests", recognized by a corresponding BLOCKTYPE rule, based on a lexicon of typical phrases), (3) another such heading, ≤ 10 blocks after the first one. (4) It requires there to be ≤ 30 blocks (of any type) before the title, and ≤ 200 blocks after the second typical heading. The expression in curly brackets refers to a minimum-occurrence condition, requiring the document to have ≥ 1 biographical text block somewhere (no matter what its position). A biographical text block can be thought of as a rather big paragraph containing typical phrases, e.g. "I was born", "I studied", etc.).

Of course alternative rules for the same document type have to be specified to increase recall.

3 Tests and results

STAN has been applied to a number of classification and extraction tasks, undertaken for real-life search engines. Among them are (i) extraction of German postal addresses, (ii) extraction of bibliographic sections of articles, (iii) recognition of scientist homepages. Precision achieved for theses tasks was around 90%, 95%, and 80% respectively, measured only in a rough manner by checking about 150 randomly chosen extraction outputs. For (iii), a more thorough analysis was performed, measuring both precision and recall on a set of 2,000 documents (among them 360 homepages of scientists). Precision was 82.16%, recall 80.28%. For (i) 48 text block and complex text block rules were used, in (ii) there were about 50 rules[1], in (iii) 45 rules.

As for speed, the number and complexity of the rules has a major impact on the performance of the program. In the examples given above, a speed of about 1-1.5 MB/s of processed input was achieved.

Lexica used for these test cases encompass big proper noun lexica ($\sim$1M surnames) and lists of street and city

[1]This can only be estimated since it was tested with an earlier version of the program with the rules slightly different and hardwired into the code.

names ($\sim$110K), as well as hand-crafted lists of phrases typical for certain document types.

4 Machine learning extensions

In order to facilitate the generation of rules while at the same time improving accuracy, the capability to automatically learn rules from sample documents would clearly be a useful improvement. Only a few steps have been taken towards that goal, and a detailed description cannot be given here. To us it seems quite straightforward to apply different learning strategies to the various types of rules present in the system: For learning vocabulary lists (for the LEXTYPES), n-gram based terminology extraction algorithms are promising; for learning adequate restrictions on S and F, a good strategy could be to transform the bunch of size numbers in S into an element of a finite set of predefined values – such as "small", "middle-sized", "big" –, based on a simple a-priori heuristics, and use them as unary predicates for the text nodes. As for F, since the set of its potential elements is finite anyway, predicates could be defined for them as well (let t be a text node, some predicate $p_e(t)$ is true iff $e \in F^t$). Then constraints on S and F could be represented in terms of a decision tree, to be learned by a 0-order inductive learning algorithm. L however, being a classical document vector, would best be learned based on a vector space approach. Eventually, optimal combinations of the restrictions found for S, F and L, to be used as the final BLOCKTYPE rule, could be derived by virtue of a detector fusion principle.

For complex block types much depends on the degree of complexity one would like to achieve – but for lists and pairs, ad-hoc combinations of existing learning methodologies similar to those mentioned above are certainly quite suitable. The bigger problem will probably emerge from the fact that for tasks such as document type identification, though it might be easy to define a set of positive samples, it's likely to be a very hard task to find adequate negative samples – having a structure similar to but slightly different from that of the positives. We plan to design a user-interactive procedure, through which the rules would be learned in several iterative steps, after each of which the user would have to evaluate a number of positively classified documents from a big training corpus, thus pointing the learner to near misses.

5 Conclusion and future perspective

STAN as a program that provides structure-based classification and extraction, has proven to be capable of solving a variety of different problems in the area. Grammar-like rules are maintained, operating on various levels of "granularity", i.e. nodes, blocks, and complex structures. It is easy to include huge lexica of relevant words and phrases, and the grammar is defined in the form of structural rules which are simple enough for being likely to be automatically learnable, though some difficulties might emerge when it comes to selecting proper sample documents.

The main idea behind STAN is of course to provide a means of transforming a document into an abstract representation of its predominant, meaningful parts and to give a notion of what their meaning is (in terms of a type associated with them). However, since the system operates in close connection with an HTML parser, more precise specifications of the extraction task in terms of exact sequences of HTML tags can potentially be tied in with the current system without having to make substantial modifications to the overall design.

To sum up, basic goals set in the beginning were achieved without precluding the opportunity to further develop it with respect to machine learning and improvement of rule definition and matching functionality.

The program has been successfully integrated in several search engines, nonetheless being subject to constant development in the directions outlined above. Clearly, the STAN approach is only sensible when the class of target document types is fairly limited so that the task of specifying the grammar rules is still feasible. E.g. while the detection of scientist homepages was possible, the more general class of all (private) homepages might be considerably more difficult to describe in terms of STAN rules. Most probably, one would need to describe various subtypes, each of them giving rise to a large number of text block types and further rules. Hence, implementing some learning capability is among the most urgent next steps in the work to come.

References

[1] B. Adelberg. NoDoSE–a tool for semi-automatically extracting structured and semistructured data from text documents. In *Proceedings of SIGMOD-98*, pages 283–294, 1998.

[2] N. Ashish and C. Knoblock. Wrapper generation for semi-structured internet sources. In *Proc. Workshop on Management of Semistructured Data*, Tucson, 1997.

[3] M. E. Califf and R. J. Mooney. Relational learning of pattern-match rules for information extraction. In *Working Notes of AAAI Spring Symposium on Applying Machine Learning to Discourse Processing*, pages 6–11, Menlo Park, CA, 1998. AAAI Press.

[4] D. Freitag. Information extraction from HTML: Application of a general machine learning approach. In *AAAI/IAAI*, pages 517–523, 1998.

[5] N. Kushmerick. Wrapper induction for information extraction. Ph.D. Dissertation, 1997.

[6] I. Muslea, S. Minton, and C. Knoblock. Wrapper induction for semistructured web-based information sources. In *Third International Conference on Automatic Learning and Discovery CONALD-98*, Pittsburgh, June 1998.