Segmentation of Legal Documents

Eneldo Loza Mencía
Knowledge Engineering Group
TU Darmstadt, Germany
eneldo@ke.tu-darmstadt.de

ABSTRACT

An overwhelming number of legal documents is available in digital form. However, most of the texts are usually only provided in a semi-structured form, i.e. the documents are structured only implicitly using text formatting and alignment. In this form the documents are perfectly understandable by a human, but not by a machine. This is an obstacle towards advanced intelligent legal information retrieval and knowledge systems. The reason for this lack of structured knowledge is that the conversion of texts in conventional form into a structured, machine-readable form, a process called *segmentation*, is frequently done manually and is therefore very expensive.

We introduce a trainable system based on state-of-the-art Information Extraction techniques for the automatic segmentation of legal documents. Our system makes special use of the implicitly given structure in the source digital file as well as of the explicit knowledge about the target structure. Our evaluation on the French IPR Law demonstrates that the system is able to learn an effective segmenter given only a few manually processed training documents. In some cases, even only one seen example is sufficient in order to correctly process the remaining documents.

1. INTRODUCTION

In this age of digital communication, great efforts are being made to publish legal documents in digital form. Official gazettes are made available online and oftentimes paper is abandoned completely. Documents from the pre-digital era, too, are digitalized and published. An example are the almost 2 million U.S. court decisions that were recently released on the Internet.[1] These documents are normally provided in a human-readable format, i.e. the HTML or PDF documents contain formatting and text alignment that aids human users in recognizing the structure and organization of a document and facilitates navigation and finding concrete legal provisions within the text. This corresponds to the traditional way of using legal documents. However, the overwhelming number of documents as well as the advances in information retrieval and semantic processing in databases demand for more intelligent means of storage and processing, involving machine-readable and understandable formats. A first step are the different XML formats that were developed in recent times for the machine-readable storage of law documents and the interchangeability of legal knowledge [16]. Usually these conversions are done manually, which demands a great amount of time, e.g. with XML editing tools such as Vex or Jaxe for the MetaLex standard.[2] More advanced editors such as the Norma-Editor [21] include regular expression patterns in order to recognize certain specific parts of the document in an automatic way. However, these patterns are usually static and need to be defined by the user, so the process is still very cumbersome. In this paper we present a framework that allows to automatically or semi-automatically convert legal documents in a conventional text format into a machine-readable and structured format. More specifically, we concentrate on the task of separating a law document into its individual articles. In a way the task can be described as reconstructing the original structure of a document. We call this process *segmentation*.

We present a trainable approach based on techniques from the field of Information Extraction. These techniques are adapted and combined in order to make use of the peculiarities of segmentation of legal texts: We introduce techniques with the aim of taking advantage of the semi-structured source formats and the hierarchical organization of legislative texts in order to increase effectiveness and hence reduce the required human effort.

We applied our algorithms to the subset of the French Intellectual Property Code provided by the Legifrance platform.[3] The French IPR law was the starting point of the EU-funded ALIS project (Automated Legal Intelligent System), which aims at investigating automated reasoning in computational logic and game theory in the legal domain.[4] We showed that by providing only a few manually processed documents the system was able to effectively process the entire Code. Though we set our focus on this repository, the system is easily configurable for different sources and different target formats.

This document is organized as follows: We first give an extensive overview of Information Extraction and Segmentation in Sections 2 and 3. The segmentation of the Legifrance documents into articles is described in Section 4, followed by the experimental results and our conclusions in Sections 5 and 6.

2. INFORMATION EXTRACTION

Information Extraction (IE) is the task of identifying certain types

[1]`http://bulk.resource.org/courts.gov/`

[2]`http://legacy.metalex.eu/downloads`

[3]`http://www.legifrance.gouv.fr/`

[4]`http://www.alisproject.eu/`

of information in documents. When information extraction is applied to natural language text documents, a common task is e.g. Named Entity Recognition, which aims at finding names of people, organizations, places, temporal expressions or numerical expressions. Typically, for training an information extraction system or algorithm, one needs a large training set of documents that has been manually processed in the way the system is intended to do it automatically. This means e.g. that the relevant text segments are marked in the documents so that they can be identified by the processing system. This task is often called annotation and is done by the user, usually by domain experts.

The resulting system is then able to automatically extract the desired information in new, unseen documents and is therefore called extractor, recognizer or wrapper. The piece of information that the user has previously marked as relevant is commonly called annotation, target entity, mark-up or field. Consequently, the process of automatically extracting this information is called (automatic) annotation or marking up. We refer to [23] for a general introduction to Information Extraction.

2.1 Information Extraction as a Classification Problem

The first approaches that used methods from machine learning for information extraction were mainly rule based and applied finite state machines or statistical approaches such as Markov models or conditional random fields (cf. [23]). These approaches were usually strongly adapted to the specific extraction problem under consideration. In recent years, more and more approaches have appeared that translate the IE task into a classical classification problem, which is nowadways considered the most popular approach. Classification is the task of sorting instances or objects into a given set of classes. Learning algorithms automatically induce such classifiers from a set of training instances for which the class is known.

The advantage of this approach is twofold: on the one hand, the researchers can choose from a wide range of established and powerful state-of-the-art classification algorithms and therefore directly benefit from the most current advances achieved in the machine learning community. On the other hand, this allows for a greater focus on the actual task, the extraction task itself, since it is now decoupled from the algorithmic design. The scope is shifted from the development of a specialized algorithm to the extraction of useful information, exploration of the corpus to be analyzed and the design and architecture of the overall IE system.

The most common approach is to transform each text position, i.e. usually each text token in the document, into a classification example. The features or attributes of the generated classification example are derived from the underlying text position and context, where different characteristics can be chosen. The class information of the instance depends on whether the underlying text token is a part or boundary of the target annotation or not. The annotated documents are transformed in this manner to a set of classification training examples which are used to train a classifier. At extraction time, the new (non-annotated) documents are also transformed into feature vectors. The trained classifier is then used to receive class information for each of these instances. By means of this class information and the text position, it is possible to reconstruct the annotations. Different approaches and techniques exist for these steps, some of them are introduced in the following sections.

2.2 Boundary Classification

As already described earlier, boundary classification denotes the labeling of a text position as an annotation boundary or non-boundary. This information is used to train the classifier which is later used by the extractor to predict the boundaries on new text.

For our experiments we use the *Begin/End* approach, since it is simple and especially appropriate for the recognition of long text segments, making it the best choice for segmentation. The start and the end of each annotation, i.e. only the boundaries, are marked with the tag *START* or *END*, the rest is marked as negative examples with *NEG*. An example can be seen in Figure 1. When using the extractor on new texts, an annotation is started at a token with the predicted class *START* and ended at the next token with *END* as class. If there is an annotation which only includes one token, the token is classified as *UNIQUE*.

Of course an imperfect prediction can lead to an inconsistent tagging result and therefore to unresolvable cases in the reconstruction process. However, similar inconsistencies also appear during the reconstruction in other boundary tagging schemes. A simple approach is to ensure that an opening tag is always preceded by a closing tag and to eliminate the remaining inconsistent tags. This approach is often coupled with checking whether the length of the resulting annotation deviates from the seen examples [14].

Other boundary classification approaches include *Inside/Outside*, *Begin/Continue/End* or *Begin/Continue/Outside* (see also [3]). They have in common that they are not as appropriate for long text segments as the *Begin/End* scheme family, since a long sequence of text tokens between start and end have to be predicted correctly. Additionally, such positions and their contexts are less likely to contain information about the boundaries since they may not be close enough.

2.3 Feature Generation

The boundary classification step generates the class information for each training instance, but up to now these instances are empty, containing no information other than the ID and the class assignment. The instances lack features that could be used for learning. The simplest features which can be added are the occurrences of the different tokens themselves. Consider e.g. the first row of the token features in Figure 1: the field *the=1* denotes that the feature *the* is set for this instance, otherwise it is zero. This representation is called a sparse representation, because only values unequal to the default value zero are saved.

This approach is also called the set-of-words approach and generates very specific feature vectors. More sophisticated methods additionally analyze the token form, the capitalization of the words, or the token type. A simple approach is the one used by the Recommended.TokenFE[5] module of MinorThird, which analyzes the structure of the words in a very simple way. An example of the produced features is given in the character patterns row in Fig. 1. The following notation is used: X denotes a capital character, x a lower case character, and $+$ means that the preceding letter is present one or more times. In consequence Xxx denotes a capitalized word with three letters, $xxxxx$ a word with five letters in lower case, and $X+x+$ means that a word begins with one or more upper case letters and is followed by one or more lower case letters.

Though these character-based methods are very simple, they are very effective when used together with the normal token features and windowing since they enhance the very specific set-of-words approach by more general features. Note that although it can be expected that most of the generated features are uninformative, the classifier will automatically learn which of them are useful and ignore the rest. An additional advantage is that this approach is lan-

[5]`http://minorthird.sourceforge.net/javadoc/edu/cmu/minorthird/ui/Recommended.TokenFE.html`

token	The	quick	brown	fox	jumps	over	the	lazy	dog
position	1	2	3	4	5	6	7	8	9
class	NEG	START	NEG	END	NEG	NEG	NEG	NEG	NEG
token features	the=1 +1.quick=1	quick=1 +1.brown=1 -1.the=1	brown=1 +1.fox=1 -1.quick=1	fox=1 +1.jumps=1 -1.brown=1	jumps=1 +1.over=1 -1.fox=1	over=1 +1.the=1 -1.jumps=1	the=1 +1.lazy=1 -1.over=1	lazy=1 +1.dog=1 -1.the=1	dog=1 -1.lazy=1
character patterns	Xxx=1 X+x+=1	xxxxx=1 x+=1	xxxxx=1 x+=1	xxx=1 x+=1	xxxxx=1 x+=1	xxxx=1 x+=1	xxx=1 x+=1	xxxx=1 x+=1	xxx=1 x+=1
history features		-1.NEG=1	-1.START=1 -2.NEG=1	-1.NEG=1 -2.START=1	-1.END=1 -2.NEG=1	-1.NEG=1 -2.END=1	-1.NEG=1 -2.NEG=1	-1.NEG=1 -2.NEG=1	-1.NEG=1 -2.NEG=1

Figure 1: Transformation of a text sentence with the tagged text fragment *quick brown fox* into a classification problem. Each column shows a token and exemplarily the generated class information and attribute values of the corresponding classification instance.

guage independent, as it works on arbitrary character strings (except perhaps languages that use symbols such as Chinese).

Our system also allows using methods from natural language processing (NLP) that analyze the lemma of the token, part of speech (POS), semantic classes from a list of gazetteers, the named entity type, stemming etc. However, the output of automatic NLP tools are far from producing perfect results. Perhaps as a consequence of this, researchers have often determined that including linguistic features does not improve the performance of their IE algorithms in a significant way (e.g. [5, 13]). This applies also to standard text classification [17], although it is often stated that the more features a system uses, the better the performance it can achieve (e.g [15]). For further information, refer to the experimental study by [10] on the influence of POS, token orthography and token gazetteer category features on IE.

Finally, there can be a lot of information that can be extracted from the (partial) structure of the document, such as HTML-tag information, formatting, placements in tables, or simply the placement at the beginning or end of a line in the source code. This aspect is described in more detail in Section 3.1. Since we are dealing with partially structured documents and we also wanted to be language independent, we left out NLP for feature generation in our analysis.

2.4 Windowing

In boundary classification the context of the analyzed token is equally or even more important than the token itself. Therefore, the surrounding tokens are also included in the feature generation process. This method is called windowing because a window is opened around the tokens through which the surrounding tokens can also be observed. The window can involve a subsentence, a sentence, or even a paragraph, but usually a fixed number of preceding and following tokens are used. This number is referred to as the window size.

There exist different methods in order to merge the features of the neighborhood tokens. The approach which we use encodes the relative position into the features, so that it does matter where a word was seen in the window [22, 4]. This method can be seen in Figure 1 in the token features rows: *-1.* and *+1.* indicate the preceding resp. following token. Other approaches do not distinguish between the position of a token in the window, or take into account the distance from the center in the feature weight [14].

Usually the window size is set to a value higher than 3, more commonly a minimum value of 5. The highest values are 10, very rarely 15 [7, 8]. Note that a window size of n means to take into account information about $2n + 1$ words. The number of features and thus the computational costs grows very quickly with increasing window size. It has been also shown that a window size greater than a certain value does not improve the performance any further and may even deteriorate the results due to the increased noise, i.e. uninformative features.

2.5 Usage of Classification History

The goal in sequence predicting or sequence labeling is to predict the next sequence element using only information about the preceding elements in the sequence [20]. This problem often appears in genetics, but POS-Tagging can also be seen as a sequence labeling task (e.g. a noun is often preceded by a determiner). A common approach, as in boundary classification, is to represent each element as a feature vector and to include the information of the preceding elements as features. This approach is very similar to windowing, with the difference that we are only able to look back. Similarly to the window size a history size is set. The obtained features for our running example and a history size of 2 is presented in the history features row in Figure 1.

At classification time, the predicted class X for instance n is used to generate the corresponding features *-1.X=1* for yet unclassified instance $n + 1$ and *-2.X=1* for instance $n + 2$.

2.6 Selection of the Classification Algorithm

Support Vector Machines have shown to be very suitable for information extraction and are widely used in recent systems [5, 14, 15, 8, 7, 9, 10]. A number of advantages qualify them in order to be used for information extraction:

- The ability to deal with a relatively large number of features. Rule or tree learners are much more sensitive to this.
- The ability to find a discriminating boundary although the data is not linearly separable. In this context, linearly separable means that there is a hyperplane in the feature space of the instance that divides positive from negative examples. Though they are linear classifiers, SVMs are able to find such a hyperplane when the data is not linearly separable by means of the so called Kernel trick (cf. e.g. [1]). However in IE it is typically not necessary to use this trick.
- SVMs find the maximum-margin hyperplane. They are therefore considered to generalize very well.

We thus use Support Vector Machines as our classification algorithm, in particular the well-known LibSVM implementation [2] included in MinorThird. The standard settings are used, in particular a linear kernel.

3. SEGMENTATION

The typical tasks in information extraction involve retrieving small relevant pieces of information out of text documents (cf. Section 2). In comparison to this, segmentation aims at automatically dividing a document into parts. In an extended setting, it aims at automatically structuring a document, i.e. providing a text with a structure in an explicit way (in the sense that it is not only visually accessible by human beings as in text on paper or rendered on the screen but directly accessible electronically by a machine). This is done by dividing the text into text segments which have clearly defined relations between themselves such as *is-followed-by* or *is-contained-in*.

Consider the following scenario: a text book is usually divided into chapters, sections, subsections and paragraphs. A book's structure is clearly defined: a book contains a sequence of non-overlapping chapters, a chapter contains a sequence of non-overlapping sections, ..., a chapter is not allowed to directly contain subsections etc. Although book texts possess such a structure, it is not clearly reflected in their electronic representation and organization. A human being is able to naturally detect the structure with the aid of layout and formatting information, the use of spaces and line breaks, enumerations, etc. Automatic segmentation tries to imitate this behavior and to provide an unstructured book text with a structure.

Segmentation works similarly to information extraction, i.e. it is based on a specific algorithm that learns from given correctly preprocessed examples and tries to induce a system, usually called segmenter in this context, which is able to automatically process the examples in the previously presented way. We can therefore treat a segmentation task also as a classification problem (Section 2.1) and also use the same sub-techniques, however configuring and modifying them adaptively respecting the changed characteristics. The main characteristics of a text segmentation information extraction task are the following:

- The annotations, i.e. the segments, have clear restrictions and obligations regarding especially their encapsulation, i.e. which type can or has to be contained in another type of segment (e.g. subsection in section), and the ordering, i.e. which type can or has to follow another type. These requirements may be e.g. given by a XML Schema Definition (XSD file).[6]
- The underlying source files are often of a structured or at least semi-structured nature as opposed to raw natural language texts. In this context (semi-)structured text typically refers to HTML documents, since they contain information, mainly formatting information, about the underlying natural language text that helps to give the text a structure. HTML is e.g. used to mark up headings, emphasize text segments, define paragraphs, tables.
- The segments are usually large in comparison to e.g. named entity recognition tasks, where an annotation rarely encloses more than three words. Text segmentation normally begins at sentence length. However short segments can also exist, such as in the case of heading extraction and, even shorter, section numbering extraction.

3.1 Usage of structure information

As we are currently only dealing with HTML documents, we focus our analysis on this type of documents. HTML contains, apart from the underlying text, special elements that serve to denote certain characteristics of the content text. These elements are called mark-up tags. Let us mark-up our running example in HTML/XML, consider the following example:

> The <I>*quick brown* <B>***fox***</B></I> jumps over the lazy dog

In HTML, the *I* mark-up denotes an italic font formatting and *B* denotes bold text. There are different ways to preprocess this source text for information extraction. Their usage depends on the concrete characteristics and demands of the IE task. The first aspect is the tokenization of the document. One extreme is to completely ignore XML information and only regard the text between the tags. The resulting document would only contain the raw sentence text, making it impossible to exploit the additional structure information contained in the XML mark-up. The other extreme is to treat the XML code entirely as normal text. Depending on the tokenization variant used, this could result in a token sequence such as *The, <, I, >, quick, brown, <, B, >, fox, . . .* The usage of prefix and suffix string patterns is a very effective method to extract information from HTML texts, as has been shown by [12, 11] with his family of pre- and suffix wrappers WIEN. However this approach is problematic since it introduces a high number of XML-specific tokens and thus complicates the usage of text features. Remember the windowing technique described in Section 2.4. With the additionally introduced XML token, we have to significantly enlarge the window size in order to be able to detect a relationship between e.g. *fox* and *jumps* or even *brown*. Enlarging the window size has a significant impact on the number of features and hence on the effectiveness (computational costs) and efficiency (noisiness of the data) of the algorithm.

In order to find a compromise between the two extremes, we chose to decode whole tags as tokens, which results in the token sequence *The, <I>, quick, brown, <B>, fox . . .* The learning algorithm is able to use the XML-specific information if it is determined to be useful, while maintaining the ability to look at the surrounding text since the window size is not decreased too much. This setting achieved the best results in preliminary tests.

Actually, HTML mark-ups are simple annotations as we have been discussing them for information extraction, with the peculiarity that they mainly say something about the formatting of the annotated text fragment, or position or relevance for the document structure (as with headings, paragraphs, tables etc.). As done in Section 2.3 with language and token characteristics, we can transform this information into features for the token positions. We therefore add the feature *I=1* to token 2-4 and *B=1* to token 4. To differentiate between the relative positions inside the annotation, we can add features for starts and ends of annotations, token 2 would e.g. obtain the feature value *start_I=1* and token 4 *end_I=1*.

In a further step we plan to additionally make use of the hierarchical information included in XML-structured documents. Note that the *B* annotation is nested in the *I* mark-up. This can be used to indicate the hierarchical ordering of several mark-ups, not only the simple presence, by adding e.g. a feature *I.B=1* to the token *fox*. It is also possible to add the XPath query of the elements as features. XPath[7] is a language for accessing and selecting nodes in a XML document. Using these type of features allows us to intensively exploit the XML or HTML structure of a document as it is normally only known from a specialized wrapper. The wrapper algorithm STALKER [18, 19] in particular e.g. uses a query language very similar to XPath and is hence able to learn XPath-like queries for documents in XML format in order to extract certain elements. The SoftMealy system induces finite state transducers using XML tags as states [6].

[6]XSD, `http://www.w3.org/TR/xmlschema11-2/`

[7]`http://www.w3.org/TR/xpath`

3.2 Construction of combined extractors

Until now we have only analyzed the case of extracting one type of annotation. When the task is to extract several entities out of the same text, which is the case for the law articles extraction task, there are some aspects that may be used to improve the effectiveness and efficiency of the process.

Consider the case that we want to extend the extraction task in Figure 1, which we can consider as extracting the *actor* of a sentence, to additionally detecting the *action* and also entities describing *attributes* of the actor. We thus extend our running example by these annotations:

$$\text{The }\underbrace{\underbrace{\text{quick}}_{attribute}\ \underbrace{\text{brown}}_{attribute}\ \text{fox}}_{actor}\ \underbrace{\text{jumps}}_{action}\text{ over the lazy dog}$$

The simplest approach would consider to learn three independent extractors, one for each annotation type. Each learning algorithm would only have information about its own annotation. A step further, we could set the extractors in a concrete ordering, i.e. the extractors are trained and applied in a predetermined order. Having this, it is now possible for an extractor to exploit the results of the preceding extractors. Consider the ordering *actor, action, attribute*: after the extractor for *actor* has annotated a text, the *action* extractor can make use of the annotations that the previous annotator has provided, and assuming that actor entities appear near their actions, this is very useful information. The same applies for the *attribute* case. In order to be able to use this information in the extraction process, it has to be available during the training process, therefore the strict ordering of the extractors. In the example the training for the combined extractors would be the following: show text, train *actor* extractor, enrich text by *actor* annotations, train *action* extractor, enrich text by *action* annotations, train *attribute* extractor.

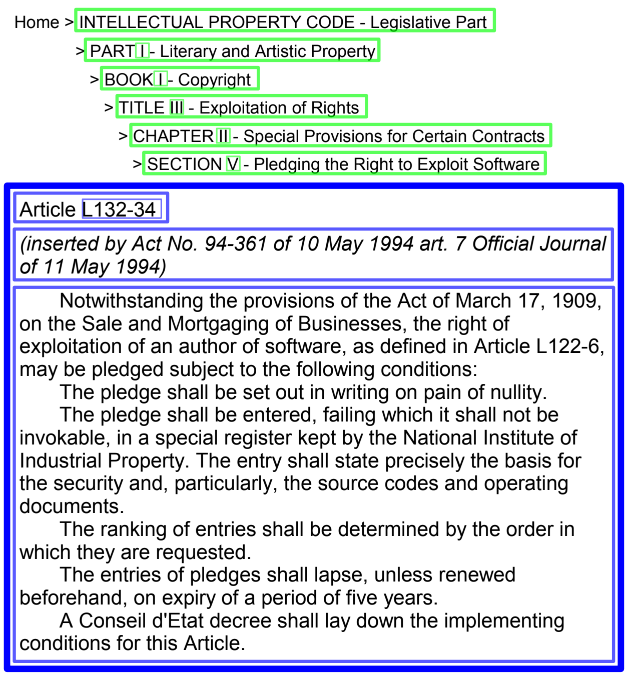
The main problem of this approach is the substantial influence of the preceding extractors' performance on the following extractors, and consequently the propagation of errors in the chain of extractors. Nevertheless by using a relatively error robust algorithm this approach has the potential to improve the performance in comparison to an ensemble of independently trained and applied extractors.

Hierarchically structured entities.

When the entities to be extracted are organized in some way, as is often the case in segmentation (cf. Section 3), this information can also be used in order to produce a more accurate and efficient ensemble of combined extractors. Consider again our running example, and assume the following constraints on the entities: an *actor* entity must not overlap with an *action* entity, an *attribute* entity has to be surrounded by an *actor* entity. These restrictions can be translated to the following training process: show text, train *actor* extractor, restrict text to *actor* text fragments, train *attribute* extractor, restrict text to non-*actor* text fragment, train *action* extractor. The extraction process has the same course.

This approach reduces the amount of text that has to be processed, both during training and extraction. Considering that the imbalance between negative and positive examples, i.e. having a lot of *NEG* examples and only a few other examples, is an important aspect regarding the performance of IE algorithms [5, 15, 8], one can expect an improvement in effectiveness due to the implicit instance selection, in addition to the efficiency gain.

However, the propagation of errors is even more dangerous than for the ordered extractor combination, since the subsequent extractors do not have the possibility to correct an error, as their field of vision is restricted by the result of the preceding extractors. A solution could be to relax the strict vision field limitations allowing to look at the surrounding text. This would be similar to the two-level approach of [5] where a second classifier is learned on the surrounding text of the output of the first classifier in order to correct or validate these predictions. However correcting the predictions of the preceding classifiers is not trivially solvable and a research topic itself.


Home > INTELLECTUAL PROPERTY CODE - Legislative Part
> PART I - Literary and Artistic Property
> BOOK I - Copyright
> TITLE III - Exploitation of Rights
> CHAPTER II - Special Provisions for Certain Contracts
> SECTION V - Pledging the Right to Exploit Software

Article L132-34

(inserted by Act No. 94-361 of 10 May 1994 art. 7 Official Journal of 11 May 1994)

Notwithstanding the provisions of the Act of March 17, 1909, on the Sale and Mortgaging of Businesses, the right of exploitation of an author of software, as defined in Article L122-6, may be pledged subject to the following conditions:
The pledge shall be set out in writing on pain of nullity.
The pledge shall be entered, failing which it shall not be invokable, in a special register kept by the National Institute of Industrial Property. The entry shall state precisely the basis for the security and, particularly, the source codes and operating documents.
The ranking of entries shall be determined by the order in which they are requested.
The entries of pledges shall lapse, unless renewed beforehand, on expiry of a period of five years.
A Conseil d'Etat decree shall lay down the implementing conditions for this Article.

Figure 2: Representation of the document `http://195.83.177.9/code/liste.phtml?lang=uk&c=36&r=2507` containing article L132-34. Text fragments are surrounded by colored rectangles. Green mark ups indicate annotations related to the classification while blue lines denote article text related entities.

In the case that the ensemble is trained without making use of hierarchical structures e.g. due to the issues just mentioned, it has to be ensured that the result is consistent with the existing structure, i.e. that none of the derived constraints are violated. An intuitive solution is to determine an extraction ordering beforehand, preferably respecting the hierarchy, and to eliminate every annotation that violates the constraints and previous annotations.

4. SEGMENTATION OF LEGAL DOCUMENTS INTO ARTICLES

We describe in this section the recognition of articles in the French IPR Law, however the described methodology can be applied to any other law corpus since we deal with a trainable system.

4.1 Legifrance Dataset

The site `http://www.legifrance.gouv.fr/` is maintained by the French government and hosts a database of French Law documents that ranges from the French constitution to the *Journal Officiel*. In addition to this, the site provides an English[8]

[8] `http://195.83.177.9/code/index.phtml?lang=uk`

```
<td width="100%" class="Texte1"><B><A name="art9844">
    Article L132-34</A></B><br>

<i>(inserted by Act No. 94-361 of 10 May 1994 art. 7
    Official Journal of 11 May 1994)</i><br>
                              &nbsp;&nbsp;&nbsp;&nbsp;&
    nbsp;&nbsp;&nbsp;Notwithstanding the provisions of
    the Act of March 17, 1909, on the Sale and
    Mortgaging of Businesses, the right of exploitation
    of an author of software, as defined in Article L122
    -6, may be pledged subject to the following
    conditions:<br>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&
    nbsp;The pledge shall be set out in writing on pain
    of nullity.<br>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&
    nbsp;The pledge shall be entered, failing which it
    shall not be invokable, in a special register kept
    by the National Institute of Industrial Property.
    The entry shall state precisely the basis for the
    security and, particularly, the source codes and
    operating documents.<br>&nbsp;&nbsp;&nbsp;&nbsp;&
    nbsp;&nbsp;&nbsp;The ranking of entries shall be
    determined by the order in which they are requested.
    <br>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;The
    entries of pledges shall lapse, unless renewed
    beforehand, on expiry of a period of five years.<br>
    &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;A <I>
    Conseil d'Etat</I> decree shall lay down the
    implementing conditions for this Article.</td>
```

Figure 3: Source code excerpt of the document depicted in Figure 2.

and Spanish[9] translation of the current French Law in force. The documents show the consolidated text of the law, i.e. changes to the wording of a particular law resolved later in a different law, as is common in legal systems, are already included.

We are especially interested in the French Intellectual Property Law (IPR-Law), which is divided into a Legislative and a Regulatory Part. We recursively retrieved in total 222 HTML files from the site[10], of which 112 belong to the legislative and 110 to the regulatory part. However only 181 are of particular use as the remaining 81 only exist for navigation purposes, i.e. actually do not contain any articles. An (edited) example document is shown in Figure 2.The documents contain in total 488 legislative articles that are numbered from L111-1 to L811-4 and 658 regulatory articles that range from R111-1 to R422-51-14. The goal is now to automatically retrieve these 1146 articles by only seeing a few of them. In addition to the raw article text, we have defined some interesting fields that will be also extracted.

French law is organized according to a classification system, and the classification of the contained articles is indicated in each document. Consequently each document contains the names of the following categories: code in which the article is included, part of the code, book, title, chapter and finally section. For each of these entities the name and the corresponding number is extracted, except for the code where no number exists. As part of the article body we have identified the following interesting fields: article heading, article ID, remarks to the consolidated version if available, and the consolidated article law text. At this point we do not stick with standard formats for legal sources such as MetaLex and Akoma Ntoso [16], which are certainly more precise and thourough, since the chosen granularity is sufficient for the needs of the ALIS project and for the demonstration of the framework. The structure of a Legifrance document and of each article is represented in Figure 4. Note that several articles can be included in one document while the other fields can only appear once. In addition, some of the category fields are sometimes not available. Figure 2 shows a document with the corresponding text fragments marked up. An excerpt of the source code is shown in Figure 3.

Since existing annotated documents were not available and manual annotation is too expensive, we developed a simple static segmenter using regular expressions for the extraction. We were able to accurately annotate the Legifrance corpus in this way so that we were able to train and evaluate our algorithms.

We are completing a tool that allows to manually annotate documents of any type with annotations of any type in a visually and interactive manner. This would permit to easily and quickly adapt the system to a new corpus or domain. It is also planned to extend the tool with active learning functionalities. The idea is that a user opens a document, adds new annotations and while this is being done, the system learns an extractor using the already available training data. The predictions of the extractor are presented to the user as suggestions which can be accepted or rejected, leading again to new training examples. This would accelerate the annotation process in a substantial way, limiting the interaction of the user to a minimum. We will see in Section 5 that sometimes only one manual annotation is necessary to correctly predict the following annotations. This functionality additionally permits the expert user to instantly and directly revise the performance of the learned extractor as the training process can directly be checked.

5. EVALUATION

The next sections describe the process and the experiments conducted in order to find the optimal parameters for training the segmenter of the French IPR law.

5.1 Algorithmic Setup

As already described, we use the Begin/End boundary classification scheme. The window size was varied between 5 and 15, the history size was always set to the same value. The learning algorithm used is LibSVM with its default settings. For the construction of the combined extractor we use the ordered variant where the extractors are put in a fixed ordering and the following extractors use the predicted annotation of the previous ones. The ordering was chosen according to the hierarchical structure of the law documents.

Experiments were in general conducted in the following way: The law files are ordered according their natural position in the law text, i.e. articles are sorted with ascending article number (articles from the legislative part first). Ten documents are taken out from the beginning of the list in order to be used as training data. The ten following documents are then selected for testing.

5.2 Evaluation Measures

As for classification and multilabel classification, there exist several measures for evaluating information extraction tasks. In particular, recall and precision are commonly used, since they provide meaningful percentage values which can be interpreted directly.

Since we are interested in exact matches, i.e. in completely correct extractions, we present only span based results that ignore partially correct extractions. Span recall/precision only accepts perfect matchings, i.e. the predicted starting and end boundaries have to exactly match the real ones. We present two equations that permit a direct and intuitive interpretation of the meaning of recall and precision for spans:

$$\textsc{Prec} \overset{\text{def}}{=} \frac{\textit{\# perfect span matchings}}{\textit{\# extracted spans}}$$

[9] `http://195.83.177.9/code/index.phtml?lang=esp`

[10] the files were retrieved on November 11, 2007

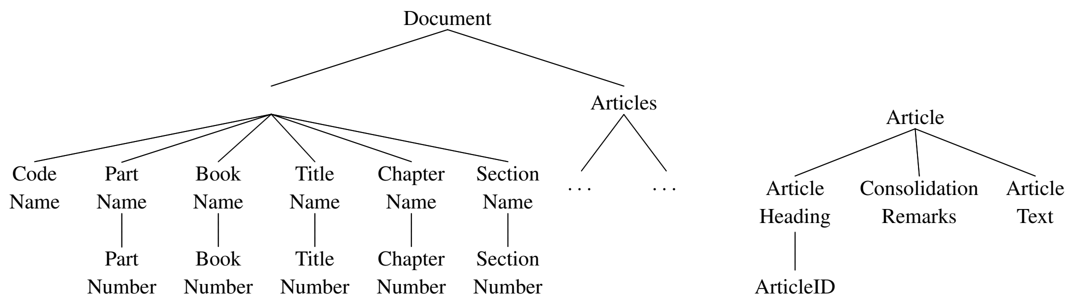


Figure 4: Structure of a Legifrance document.

$$\text{REC} \overset{\text{def}}{=} \frac{\#\ \textit{perfect span matchings}}{\#\ \textit{real spans}}$$

The number of span matchings and total number of spans are counted over all documents and not for each document individually.

5.3 Impact of Features

In order to discover the influence of the used features, we tried three different feature sets: the token itself, character pattern based features as produced by the Recommended.TokenFE module of MinorThird (cf. 2.3), and features generated from the HTML mark-ups (cf. 3.1). Mark-up start and end indicating features were generated for the predetermined window size, but inside-a-mark-up features were only generated for the current token. No XPath-like features were generated. We abbreviate the token features with *T*, the character patterns with *C* and the mark-up derived features *X*. A window size of 5 was chosen. The results for the different feature sets are shown in Tables 1, 2, 3 and 5.

The token feature set shows the best results, followed by char-pattern and XML-derived features sets. Note however that the T set produced 19.881 different feature sets, while C required only 3.682, and X was able to achieve the results with an impressive 697 features. Note also that the extractor trained on the XML features achieves perfect accuracy on the article text related annotations. There were more training examples available for these fields, since we found 94 articles in the first ten documents, while the law classification annotations were present only once in each document, or, as in the case of *Section*, only in the last three of the ten training documents. This explains why no fields were found at all for this annotation.

Extracting the category information seems to be in general a harder problem than extracting the article bodies. One explanation is of course the limited number of training examples. But another explanation could be the similar surroundings for the six annotation types, and the absence of landmarks especially for *Book* and *Title*, i.e. patterns in the surroundings of the boundaries that are easy to recognize. Note however that the achieved precision is 100%, i.e. the extractors (except for *X*) never predict a wrong annotation, they simply do not predict every annotation. We additionally repeated the experiment using the TX feature set, with the hope of benefiting from the informative XML features. The extractor combination achieved the same performance after 10 documents as with only using the T features.

5.4 Impact of Number of Training Examples

For analyzing the influence of the number of training examples we tested the extractor (using the same test set as before) after each of the ten training examples. This methodology was chosen because it indicates how fast a correct extractor is learned, and varying the parameters allows to find out which settings are responsible for the decrease or gain of performance. The convergence of a learning algorithm is an important aspect in IE because training documents usually have to be produced by humans (in a typical real world scenario), which is an expensive process.

Table 4 shows the performance for the experiment of Table 1 in dependency of both the number of documents and the number of actual annotations seen. Only the performance of those annotations are shown where recall and precision changed from using only the first to using all ten documents. This means in particular that a performance of 100% is achieved for the fields *Code Name, Part Name, Part Number, Article Body, Article Heading* and *Article ID* already after only one seen training example. In comparison *Article Remarks* needs (at most) 13 and the *Article Text* extractor (at most) 43 examples to reach a flawless prediction. It seems to be a difficult task to find a recognizable boundary between the article remarks and the article text itself since the end of *Article Heading*, which coincides with either the start of remarks or article text, and the boundary end of *Article Body*, which coincides with the end of the article text, are detected perfectly from the beginning. In fact, as can be seen in Figure 3, the space between the end of the remarks at *"11 May 1994"* and the start of the article text at *"Notwithstanding"* is filled up with standard whitespace and forced HTML whitespace "&nbsp;" used for formatting. This sequence of whitespace appears repeatedly in the source code after each line break "<br>", making it apparently difficult for the algorithm to recognize the right sequence. From an overall perspective, we can nevertheless observe a steep learning curve for the segmenters allowing to restrict the necessary human effort to a minimum and consequently decreasing costs.

5.5 Impact of Window Size

As already mentioned, window sizes of 5 to 15 are common. We therefore conducted experiments with window sizes of 5, 10 and 15. The TX feature set combination was used. Tables 5, 6 and 7 show the results for the different window sizes.

It is remarkable to see that increasing the window size from 5 to 10 does not improve the performance at all (except for the *Section* annotations), and that the further increase to 15 even deteriorates the recall of the *Chapter* predictions. Apparently the classification algorithm has difficulties to find relevant patterns within the increased amount of useless information due to the addition of more and more uninformative context. On the other hand, the additional features served the algorithm to find reasonable patterns in order to detect the *Section* annotation, and this after only three seen annotations.

span	PREC	REC
Code Name	100.00%	100.00%
Part Name	100.00%	100.00%
Part Number	100.00%	100.00%
Book Name	100.00%	20.00%
Book Number	100.00%	20.00%
Title Name	100.00%	20.00%
Title Number	100.00%	66.67%
Chapter Name	100.00%	90.00%
Chapter Number	100.00%	100.00%
Section Name	0.00%	0.00%
Section Number	0.00%	0.00%
Article Body	100.00%	100.00%
Article Heading	100.00%	100.00%
Article ID	100.00%	100.00%
Article Remarks	100.00%	100.00%
Article Text	100.00%	100.00%

Table 1: Features: T. Window size: 5

span	PREC	REC
Code Name	100.00%	100.00%
Part Name	100.00%	100.00%
Part Number	100.00%	100.00%
Book Name	100.00%	20.00%
Book Number	100.00%	20.00%
Title Name	100.00%	20.00%
Title Number	100.00%	66.67%
Chapter Name	100.00%	30.00%
Chapter Number	100.00%	33.33%
Section Name	0.00%	0.00%
Section Number	0.00%	0.00%
Article Body	100.00%	100.00%
Article Heading	100.00%	100.00%
Article ID	100.00%	100.00%
Article Remarks	100.00%	100.00%
Article Text	100.00%	100.00%

Table 2: Features: C. Window size: 5

span	PREC	REC
Code Name	0.00%	0.00%
Part Name	0.00%	0.00%
Part Number	0.00%	0.00%
Book Name	100.00%	20.00%
Book Number	100.00%	20.00%
Title Name	0.00%	0.00%
Title Number	0.00%	0.00%
Chapter Name	0.00%	0.00%
Chapter Number	0.00%	0.00%
Section Name	0.00%	0.00%
Section Number	0.00%	0.00%
Article Body	100.00%	100.00%
Article Heading	100.00%	100.00%
Article ID	100.00%	100.00%
Article Remarks	100.00%	100.00%
Article Text	100.00%	100.00%

Table 3: Features: X. Window size: 5

	1	2	3	4	5	6	7	8	9	10
Title Name	0 / 0%	0 / 0%	0 / 0%	100 / 20%	100 / 20%	100 / 20%	100 / 20%	100 / 20%	100 / 20%	100 / 20%
Title Number	0 / 0%	0 / 0%	0 / 0%	100 / 67%	100 / 67%	100 / 67%	100 / 67%	100 / 67%	100 / 67%	100 / 67%
Chapter Name	100 / 10%	100 / 80%	100 / 80%	100 / 80%	100 / 80%	100 / 80%	100 / 80%	100 / 90%	100 / 90%	100 / 90%
Chapter Number	100 / 11%	100 / 100%	100 / 100%	100 / 100%	100 / 100%	100 / 100%	100 / 100%	100 / 100%	100 / 100%	100 / 100%
	0	2	3	4	13	20	21	21	23	23
Article Remarks	0 / 0%	100 / 74%	100 / 68%	100 / 63%	100 / 100%	100 / 100%	100 / 100%	100 / 100%	100 / 100%	100 / 100%
	5	9	18	27	43	54	62	79	86	94
Article Text	100 / 51%	100 / 97%	100 / 92%	100 / 92%	100 / 100%	100 / 100%	100 / 100%	100 / 100%	100 / 100%	100 / 100%

Table 4: PREC / REC accuracy after each of the 10 training documents for selected annotation types. The row over the accuracy rows of the annotation types indicate the number of seen examples for this annotation. Features used: T. Window size: 5.

Note also that the feature set size increases significantly with a growing window size. The 20,164 features for window size 5 almost doubles to 38,430 when doubling the windows size, a window of 15 even produces 56,663 features. This high value is only manageable by the system because of the extreme sparseness of the feature vectors typical for boundary classification tasks, i.e. only a few of the total number of features are set in each instance. This allows an effective memory management and, in the case of SVMs and related algorithms, an effective exploitation of the gaps.

5.6 Combination of Topic Annotation Types

The preceding experiments have shown that the only serious problem of our extractor seems to be the law topic fields included in the head of the documents, especially the ones in the middle levels such as *Book*, *Title* and *Chapter*. Independently of the exact reason for this, the reduced number of training examples or the absence of clear landmarks, the following approach seems to be a reasonable solution: Given the fact that on the one hand the annotated text fragments for the different annotation types have a similar form, namely "> *category_type number - category_name*" (cf. Figure 2), and on the other hand the annotations are aligned in a predefined order, namely Code, Part, Book, Title, Chapter, Section, the idea arises to treat all annotation types as only one, namely to combine them to one annotation type called *Category*. The first point would provide the learning algorithms with more training examples per document, namely five to six, and possibly allow to find better landmarks. The second point would guarantee a correct reconstruction of the original annotations (cf. Section 3.2).

We conducted an experiment by replacing the original annotations with the annotations *Category* and *Category Number*. The window size was 10 and the feature set was TX. The results are depicted in Table 8. We can see a clear improvement in the accuracy of the prediction, reaching a perfect span extraction.

span	PREC	REC
Category Name	100.00%	100.00%
Category Number	100.00%	100.00%
Article Body	100.00%	100.00%
Article Heading	100.00%	100.00%
Article ID	100.00%	100.00%
Article Remarks	100.00%	100.00%
Article Text	100.00%	100.00%

Table 8: The different category annotation types were mapped to one single annotation type *Category Name* and the sub-annotation *Category Number*.

5.7 Final Configuration

In this experiment we tried to make use of the conclusions from the preceding experiments in order to train a segmenter that is able to accurately segment all presented IPR law documents and therefore can be used in production settings. Note that until now we have uniquely applied our extractor to the following ten documents after the initial ten training documents, with 161 remaining documents to evaluate on. In this experiment we therefore train on the first ten documents and apply the learned extractor on all the following documents.

Since the previous configuration of the algorithm resulted in such good results, we keep the parameters. However, there seems to be

span	PREC	REC
Code Name	100.00%	100.00%
Part Name	100.00%	100.00%
Part Number	100.00%	100.00%
Book Name	100.00%	20.00%
Book Number	100.00%	20.00%
Title Name	100.00%	20.00%
Title Number	100.00%	66.67%
Chapter Name	100.00%	90.00%
Chapter Number	100.00%	100.00%
Section Name	0.00%	0.00%
Section Number	0.00%	0.00%
Article Body	100.00%	100.00%
Article Heading	100.00%	100.00%
Article ID	100.00%	100.00%
Article Remarks	100.00%	100.00%
Article Text	100.00%	100.00%

Table 5: Features: TX. Window size: 5

span	PREC	REC
Code Name	100.00%	100.00%
Part Name	100.00%	100.00%
Part Number	100.00%	100.00%
Book Name	100.00%	20.00%
Book Number	100.00%	20.00%
Title Name	100.00%	20.00%
Title Number	100.00%	66.67%
Chapter Name	100.00%	50.00%
Chapter Number	100.00%	55.56%
Section Name	100.00%	100.00%
Section Number	100.00%	100.00%
Article Body	100.00%	100.00%
Article Heading	100.00%	100.00%
Article ID	100.00%	100.00%
Article Remarks	100.00%	100.00%
Article Text	100.00%	100.00%

Table 6: Features: TX. Window size: 10

span	PREC	REC
Code Name	100.00%	100.00%
Part Name	100.00%	100.00%
Part Number	100.00%	100.00%
Book Name	100.00%	20.00%
Book Number	100.00%	20.00%
Title Name	100.00%	20.00%
Title Number	100.00%	66.67%
Chapter Name	100.00%	20.00%
Chapter Number	100.00%	22.22%
Section Name	100.00%	100.00%
Section Number	100.00%	100.00%
Article Body	100.00%	100.00%
Article Heading	100.00%	100.00%
Article ID	100.00%	100.00%
Article Remarks	100.00%	100.00%
Article Text	100.00%	100.00%

Table 7: Features: TX. Window size: 15

span	PREC	REC
Category Name	100.00%	100.00%
Article Body	100.00%	100.00%
Article Heading	100.00%	100.00%
Article ID	100.00%	100.00%
Article Remarks	100.00%	99.28%
Article Text	100.00%	99.71%

Table 9: Performance for each annotation type. Trained on 10 first documents, tested on remaining, window size was 10 and used features were TX.

a bug in the routines of MinorThird that compute the precision and recall values so that the computation is aborted for the *Category Number* annotation after a few examples. Note however that this bug only affects the report of the results.

The results are shown in Table 9. We can see a nearly perfect performance of the extractors. E.g. for *Article Text* every recognized field was correct and only three of in total 820 were not retrieved. This is an excellent result for an information extraction task, since e.g. the state-of-the-art extractors in named entity extraction achieve around 90% accuracy.[11] However it has to be admitted that a named entity recognition task is of a different type than a segmentation task which can directly benefit from previously available structure information. But even for wrappers this is an excellent result.

6. CONCLUSIONS AND FUTURE WORK

We have presented a framework for the specialized IE task of segmentation, making use of structure information already present in the documents, including hierarchical organization of the desired information. The framework was configured in order to be applied to the Legifrance database, and the performance was evaluated in an extensive and detailed empirical study. We have shown that the presented system is perfectly usable for the automatic segmentation of law documents into articles. In some cases only one training example was sufficient in order to correctly learn a particular annotation.

[11] http://www.itl.nist.gov/iaui/894.02/related_projects/muc/info/whats_ie.html

Moreover, the framework can easily be adapted to other documents, e.g. to law corpora from other countries. A relevant deterioration of the effectiveness is not expected due to the language-independent methods used. However, if necessary, the modular design would permit the use of natural language processing.

Additionally, the aforementioned annotation tool, which is currently in development, will aid the user in providing the system with the necessary training information.

Acknowledgements

This work was supported by the EC 6th framework project ALIS (Automated Legal Information System).

7. REFERENCES

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

[2] C.-C. Chang and C.-J. Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[3] W. W. Cohen. Minorthird: Methods for identifying names and ontological relations in text using heuristics for inducing regularities from data, version from 2008-06-11, 2004-2008.

[4] A. Finn and N. Kushmerick. Information extraction by convergent boundary classification. In *In AAAI-2004 Workshop on Adaptive Text Extraction and Mining*, 2004.

[5] A. Finn and N. Kushmerick. Multi-level boundary classification for information extraction. In J.-F. Boulicaut, F. Esposito, F. Giannotti, and D. Pedreschi, editors, *ECML*, volume 3201 of *Lecture Notes in Computer Science*, pages 111–122. Springer, 2004.

[6] C.-N. Hsu and M.-T. Dung. Generating finite-state transducers for semi-structured data extraction from the web. *Inf. Syst.*, 23(8):521–538, 1998.

[7] N. Ireson and F. Ciravegna. Pascal challenge: The evaluation of machine learning for information extraction. In *Machine Learning for the Semantic Web Dagstuhl Seminar 05071*, Dagstuhl, Germany, feb 2005.

[8] N. Ireson, F. Ciravegna, M. E. Califf, D. Freitag, N. Kushmerick, and A. Lavelli. Evaluating machine learning for information extraction. In L. D. Raedt and S. Wrobel,

editors, *ICML*, volume 119 of *ACM International Conference Proceeding Series*, pages 345–352. ACM, 2005.

[9] J. Iria and F. Ciravegna. Relation extraction for mining the semantic web. In *Machine Learning for the Semantic Web Dagstuhl Seminar 05071*, Dagstuhl, Germany, feb 2005.

[10] J. Iria, N. Ireson, and F. Ciravegna. An experimental study on boundary classification algorithms for information extraction using svm. In *Proceedings of the EACL Workshop on Adaptive Text Extraction and Mining (ATEM)*, 2006.

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

[12] N. Kushmerick, D. S. Weld, and R. B. Doorenbos. Wrapper induction for information extraction. In *Proceeding of the International Joint Conference on Artificial Intelligence (IJCAI' 97)*, pages 729–737, 1997.

[13] Y. Li and K. Bontcheva. Hierarchical, perceptron-like learning for ontology-based information extraction. In C. L. Williamson, M. E. Zurko, P. F. Patel-Schneider, and P. J. Shenoy, editors, *WWW*, pages 777–786. ACM, 2007.

[14] Y. Li, K. Bontcheva, and H. Cunningham. Svm based learning system for information extraction. In J. Winkler, M. Niranjan, and N. D. Lawrence, editors, *Deterministic and Statistical Methods in Machine Learning*, volume 3635 of *Lecture Notes in Computer Science*, pages 319–339. Springer, 2004.

[15] Y. Li, K. Bontcheva, and H. Cunningham. Using uneven margins svm and perceptron for information extraction. In *Proceedings of Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*, 2005.

[16] C. Lupo, F. Vitali, E. Francesconi, M. Palmirani, R. Winkels, E. de Maat, A. Boer, and P. Mascellani. General xml format(s) for legal sources. ESTRELLA Project Deliverable D3.1 — IST-2004-027655, January 2006. `http://www.estrellaproject.org/doc/D3.1-General-XML-formats-For-Legal-Sources.pdf`.

[17] A. Moschitti and R. Basili. Complex linguistic features for text classification: A comprehensive study. In S. McDonald and J. Tait, editors, *ECIR*, volume 2997 of *Lecture Notes in Computer Science*, pages 181–196. Springer, 2004.

[18] I. Muslea, S. Minton, and C. Knoblock. A hierarchical approach to wrapper induction. In *AGENTS '99: Proceedings of the third annual conference on Autonomous Agents*, pages 190–197, New York, NY, USA, 1999. ACM.

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

[20] N. Nguyen and Y. Guo. Comparisons of sequence labeling algorithms and extensions. In Z. Ghahramani, editor, *ICML*, volume 227 of *ACM International Conference Proceeding Series*, pages 681–688. ACM, 2007.

[21] M. Palmirani and F. Benigni. Norma-system: A legal information system for managing time. In C. Biagioli, E. Francesconi, and G. Sartor, editors, *Proceedings of the V Legislative XML Workshop*, pages 205–224. European Press Academic Publishing, 2007.

[22] D. Roth and W. tau Yih. Relational learning via propositional algorithms: An information extraction case study. In B. Nebel, editor, *IJCAI*, pages 1257–1263. Morgan Kaufmann, 2001.

[23] J. Turmo, A. Ageno, and N. Català. Adaptive information extraction. *ACM Comput. Surv.*, 38(2), 2006.