Learning for Text Categorization and Information Extraction with ILP

Markus Junker, **Michael Sintek**, and **Matthias Rinck**
German Research Center for Artificial Intelligence (DFKI) GmbH
P.O. Box 2080, D-67608 Kaiserslautern, Germany
e-mail: {markus.junker|michael.sintek|matthias.rinck}@dfki.de

Abstract

Text Categorization (TC) and Information Extraction (IE) are two important goals of Natural Language Processing. While handcrafting rules for both tasks has a long tradition, learning approaches gained much interest in the past. In the present paper we try to provide a solid basis for the application of ILP methods to these learning problems. We propose to introduce three basic types (namely a type for text, one for words and one for text positions) and three simple predicate definitions over these types which enable to write text categorization and information extraction rules as logic programs. Based on the proposed representation, we present the key concepts of our approach to the problem of learning rules for TC and IE in terms of ILP. We conclude the paper by comparing our approach of representing texts and rules as logic programs to others.

1 Introduction

Classifying texts into content-defined categories and extracting pieces of information of a document are important goals when dealing with documents. As a guiding example we use our efforts in office automation [4, 5]. When a new business document comes in, one of the first problems is document categorization, i.e., the filing of documents into categories such as invoices, confirmation of order, etc. Afterwards, we are interested in extracting the process relevant information from the document. In case of an invoice we want to know the sender, the item we should pay for, the amount and similar information. This information can then be used to automate the document handling by triggering the right processes, e.g., by electronically remitting money to some bank account.

There are many more applications to document categorizations and information extraction. While the more traditional approaches work with hand-crafted categorization and extraction rules there is an increasing interest in applying learning approaches. The idea is that a learning system only has to be provided with positive and negative examples of some document category (or important text fragments, respectively) and then learns a classifier (an extraction rule, respectively) by itself. In text categorization most approaches use propositional learners with only a few exceptions [3, 9]. Approaches to learning rules

for information extraction generally go beyond propositional learning but mostly rely on proprietary formalisms (for a survey see [10] and [13]).

The tasks of learning rules for TC and IE are very closely related. In both cases, some characteristic patterns for text fragments have to be found. Compared to text categorization, in learning rules for information extraction an additional problem arises. While in text categorization the boundary of the fragment is always given (which is the document to be classified), information extraction rules have to locate the fragment boundaries by themselves. Nevertheless both tasks are very similar and thus also rely on very similar pattern languages. For this reason, it seems reasonable to treat both learning problems in a single learning framework.

Before going into details, we first want to argue why we are interested in making pattern learning for text categorization and information extraction an ILP problem. Our experiences in both domains show that it is relatively easy to learn TC and IE rules with some more or less reasonable effectiveness. But going beyond that effectiveness seems pretty hard as results reported in literature also illustrate. Extensions in various directions we made to learning algorithms only turned out to improve the results in some rare cases (i.e., in some specific category in a domain or in some specific information type to be extracted). Motivated by this observation, we concluded that at least one direction of further improvement may be the combination of several of these extensions within a single learning framework. For this framework, ILP seems to be the right choice, since it allows a sound and communicable formulation of the problem and its solution approaches.

In the remainder of the paper we first briefly introduce common constructs of pattern languages used for text categorization and information extraction. We then propose the basic types (like *word* or *text position*) and predicate definitions we use in mapping the functionalities of such pattern languages into logic programs. Having a means to formulate TC and IE rules as logic programs, the next section demonstrates how we apply standard ILP learning techniques to the problem of learning such programs.

2 Mapping Text Pattern Languages to Logic Programs

In this section we first describe some typical constructs of pattern languages for text categorization and information extraction. We then propose a minimum set of three basic predicates needed for both tasks. Additional constructs of text pattern languages can be easily incorporated which we show only by example.

There are some obvious constructs a pattern language suited for TC and IE should have, which include (for easier reference, we have marked the different constructs by **A**-**E**):

A Testing on the occurrence of specific words (does the word "invoice" occur in the document to be classified?)

B Tests on words occurring in some order and/or some restricted distance range. A special case of this kind of test is the test on word sequences.

C Boolean combinations of tests.

D Tests on properties of words or word sequences (e.g.: Is a "word" a number? Is it uppercase? Does it have a specific syntactic word category, such as noun? Is a word/word sequence a noun phrase? Does it denote a person or company?)

E Tests whether some patterns occur within some specified environment (e.g.: Does the pattern occur in one sentence? In the title?). These tests are particularly useful when dealing with HTML or XML documents.

All of the above tests should be combinable in any reasonable way. For information extraction there must be some additional means to specify some fragment within a pattern which can be bound to the information to be extracted.

Our mapping of text pattern languages relies on the representation of a text "w_1 w_2 ... w_n" as a *word list* or *text* `[`$\mathtt{w}_1$`,` $\mathtt{w}_2$`, ... ,` $\mathtt{w}_n$`]`. Texts in this sense can be used to represent a whole document, but they can also be used to describe information in form of word sequences extracted from a document. A *word* is also a special type and implemented as a list of characters. For easier readability, we do not write a word in form of a comma separated list of characters. Another type we need is the type *text position* which is used to locate single words within texts.

Our transformation of text patterns to logic programs relies on the predicates `wordpos(Text,Word,Pos)`, `fragment(Text,P1,P2,F)` and `next(P1,Min,Max,P2)`. The predicate `wordpos` is used to describe words and their positions in a text. Note that these texts are not necessarily documents, they can also describe some arbitrary word sequences within a document. The predicate `fragment` provides locations of word sequences within a larger text. A location of a word sequence `F` is given by its starting position `P1` and its ending position `P2` within the larger text `Text`. Here too, the larger text is not necessarily a whole document. In addition, we need some form of position arithmetics to describe the relation between word positions. The predicate `next(P1,Min,Max,P2)` is true iff the two positions `P1` and `P2` have a distance of at least `Min` words and at most `Max` words (i.e., `Min`$\leq$`P2-P1`$\leq$`Max`). For easier readability we also use the abbreviations `next(P1,X,P2)` and `next(P1,P2)` with `next(P1,X,P2)=next(P1,X,X,P2)` and `next(P1,P2)=next(P1,1,P2)`. In the appendix we give a definition of the above predicates in PROLOG.

Using only these predicates, we are already able to write complex categorization and extraction rules which map the constructs **A**, **B**, **C**, and partly **E**. For easier readability, we write `wordpos(Doc, Word, Pos)` as `Word@Doc:Pos`. The following four rules describe document categories:

```
invoice(Doc) :- invoice@Doc:P.

invoice(Doc) :- payment@Doc:P1, next(P1,P2), within@Doc:P2.

offer(Doc) :- thank@Doc:P1, you@Doc:P2, for@Doc:P3,
              next(P1,P2), next(P2,1,3,P3).
```

The first rule assigns category `invoice` to a document if the word "invoice" occurs somewhere in the document. The second rule indicates that category `invoice` is also assigned if the words sequence "payment within" occurs. The next rule for the category

`offer` tests on the word sequence "thank you" and the word "for" with up to two words in between. This also allows the formulation "thank you very much for" and similar ones.

The following categorization rule for `inquiry` also returns some text fragment within the input document:

```
inquiry(Doc, Interest) :- interested@Doc:P1, in@Doc:P2,
                          next(P1,P2), next(P2,P3),
                          fragment(Doc,P3,P3,Interest).
```

The rule tests on "interested in" and extracts the following word, which indicates the subject of interest. This capability of extracting word sequences from documents is the basis for information extraction, as some more examples illustrate:

```
payment(Doc, Days) :- within@Doc:P1, days@Doc:P3,
                      next(P1,P2), next(P2,P3),
                      fragment(Doc,P2,P2,Days).

cash_discount(Doc, Percent) :-
            cash@Doc:P1, discount@Doc:P2, of@Doc:P3, '%'@Doc:P6,
            next(P1,P2), next(P2,P3), next(P3,P4),
            next(P5,P6), next(P4,1,4,P5),
            fragment(Doc,P4,P5, Percent).

cash_discount(Doc, Days, Percent) :-
             days@Doc:P2, '%'@Doc:P5,
             next(P1,P2), next(P2,P3), next(P4,P5), next(P3,1,4,P4),
             fragment(Doc,P1,P1,Days), fragment(Doc,P3,P4,Percent).
```

The rule for `payment` searches for "within" and "days" with exactly one word in between and returns exactly this word. The next rule for `cash_discount` tests on the word sequence "cash discount" and the word "%" with a maximum distance of three words in between and it extracts these words. The second rule for `cash_discount` illustrates that even so-called multi-slot rules can be expressed. The rule extracts corresponding pairs of days and discounts.

It is very easy to extend the basic language given by the primitives `wordpos`, `fragment`, and `next`. For instance, it is possible to introduce a unary predicate which tests whether a "word" is a number or a binary predicate which tests whether two words have the same stem. Another example for a built-in predicate is one that tests for a minimum syntactic similarity of two words given by the Levenshtein distance [12]. Even more elaborated things such as testing whether some or all words within a word sequence have some property is pretty simple. By introducing these built-ins, we can map the constructs **D**.

To model word sequence features given by the structure of a document, it is possible to introduce predicates which tell, for instance, whether some word occurs within the title. This is the simple solution we currently pursue to map the constructs **E**. In parallel we work on implementing a more sound mapping of document structures by extending the notion of word positions.

3 The Learning Algorithm

For learning text categorization and information extraction rules in form of logic programs we have designed a learning algorithm which implements the widely used separate-and-conquer strategy [7]: First a rule is searched which explains parts of the positive examples, then the covered examples are separated, and the algorithm recursively conquers the remaining positive examples. This repeats until no good rule for the remaining examples can be found anymore. The set of negative examples remains unchanged during learning and is used in the evaluation of rule hypotheses.

The algorithm can be seen an a special ILP learner with a focus on textual data. Textual data is supported by providing the three fundamental predicates described in the section before as built-ins. To achieve a reasonable efficency when evaluating the built-ins, various indexing structures borrowed from work in information retrieval are used. In addition to the basic predicates, many more built-ins for text handling are provided such as predicates which test on word or word sequence properties or relations between words/word sequences.

In addition to the support of text handling by built-ins, we also have support for learning text categorization and information extractions rules by providing special refinement operators. We will only introduce some of these operators by example. The simplest refinement operator is one that adds a literal to a rule which directly tests on the occurrence of some word *word* (the l_i denote literals, we have omitted the head of the rules in our representation):

$$\frac{l_1,..,l_n}{l_1,..,l_n,\ word@Doc{:}Q}$$

For this type of refinement we do not allow all possible words for *word*, but only a subset having a minimum positive correlation to the positive examples. The selection is done to accelerate learning and does not —as experiments show— harm the effectiveness of the learned rules too much. Two more refinement operators are used to test on word occurrences to the left or to the right of some already given word.

$$\frac{l_1,..,word@Doc{:}Q,..,l_n}{l_1,..,word@Doc{:}P,next(P,1,1,Q),word'@Doc{:}Q,..,l_n}$$

$$\frac{l_1,..,word@Doc{:}Q,..,l_n}{l_1,..,word@Doc{:}Q,next(Q,1,1,P),word'@Doc{:}P,..,l_n}$$

Note that in particular there are no refinement operators which replace position variables by constants. This reflects the heuristics that —at least in our experience— the absolute position of a word within a text fragment is not important. While the above refinement operators defined specializations of the current rule, we also have generalization operators. These generalizations are initiated heuristically only by the current rule hypothesis. An example is the following generalization operator, which increases the maximum distance required between two word occurrences:

$$\frac{l_1,..,next(Q,1,x,P),..,l_n}{l_1,..,next(Q,1,y,P),..,l_n \quad with\ y{=}x{+}1}$$

Another set of generalization operators replaces a test of some word occurrence by a test requiring just a word with some specific property (e.g., the property of being a number, or being uppercase).

$$\frac{l_1,..,word@Doc{:}P,..,l_n}{l_1,..,Word@Doc{:}P,some\_property(Word),..,l_n}$$

Generalization steps are initiated heuristically only by the current rule hypothesis.

In the current implementation, there is no declarative language to control the application of the refinement operators. The search strategy is hard-wired in the learning algorithm. Depending on the state of the learner, beams with various refinement operators and of various widths are used to investigate the space of rule hypotheses. Implementing and experimenting with a hard-wired strategy was a conscious decision, since we first wanted to explore which kind of expressiveness of a declarative strategy definition is needed for our purpose. To prevent from overfitting, we use standard pre-pruning techniques such as the Laplace estimate as optimization criterion and the the likelihood estimate [2].

After having described the general features of our rule learner, we now turn to the concrete learning of text categorization and information extraction rules as we implemented it.

For text categorization, the input to the learner is a set of positive and negative example documents for each category as in the following example (`+` indicates a positive example, `-` a negative example, the dots "`..`" indicate more text):

```
offer+([.. thank, you, very, much, for, you, inquiry, ..]).
offer+([.. thank, you, for, your, letter, of, 10, February, ..]).
offer-([.. payment, within, 20, days, ..]).
offer-([.. we, are, interested, in, ..]).
```

Learning rules for text categorization is straight forward: In each conquer step we successively refine the initial rule "`offer+(Doc) :-`" by applying our refinement operators.

In the case of information extraction, the positive examples are a set of text fragments correctly extracted from some larger piece of text, typically a whole document.

```
cash_discount+([.. cash, discount, of, 2, %, ..],232,232,[2]).
cash_discount+([.. cash, discount, of, 2, ., 5, %, ..],143,145,[2, ., 5]).
cash_discount+([.. 2, %, for, cash, ..],198,198,[2]).
cash_discount+([.. 3, ., 5, %, for, cash, ..],101,103,[3, ., 5]).
```

In contrast to text categorization the negative examples are not given as ground facts. Instead, we provide a set of text fragments which are guaranteed to not contain any information to be extracted besides the informations given by the positive examples.

```
t-([.. we, offer, a, discount, of, 30, %, to, all, these, goods, ..]).
t-([.. is, 20, %, faster, .., we, offer, a, cash, discount, of, 2, %, ..]).
```

Using these texts, we define the negative examples for `cash_discount` by:

```
cash_discount-(Doc,X,Y,F) :- t-(Doc), ∄ cash_discount+(Doc,X,Y,F).
```

The way we introduced the negative examples reflects the typical situation when collecting training material for an information extraction task: For a set of documents all information of some type is extracted by hand. While these extracts serve as positive examples, the remaining text implicitly defines the negative information.

We require information extraction rules to return the exact location and length of the interesting text fragments. As a heuristics, we assume that for determining the beginning of this information, either the word position just before the information or the first word position within the information has to be located. Similarly, for determining the end of the information, either the last word of the information or the first word after the information has to be found. This heuristics results in four initial rules. Each of these rules is refined separately and the best rule found is the result of the respective conquer step:

```
x+(Doc,Pos1,Pos2,Text) :- Word1@Doc:Pos1, Word2@Doc:Pos2,
                          fragment(Doc,Pos1,Pos2,Text).
x+(Doc,Pos1,Pos2,Text) :- Word1@Doc:Pos1', Word2@Doc:Pos2,
                          next(Pos1',Pos1),
                          fragment(Doc,Pos1,Pos2,Text).
x+(Doc,Pos1,Pos2,Text) :- Word1@Doc:Pos1, Word2@Doc:Pos2',
                          next(Pos2,Pos2'),
                          fragment(Doc,Pos1,Pos2,Text).
x+(Doc,Pos1,Pos2,Text) :- Word1@Doc:Pos1', Word2@Doc:Pos2',
                          next(Pos1',Pos), next(Pos2,Pos2'),
                          fragment(Doc,Pos1,Pos2,Text).
```

It is important to note that in information extraction, we have additional refinement operators. These allow to refine the initial extraction rules based on the occurring word variables:

$$\frac{l_1,..,Word@Doc{:}P,..,l_n}{l_1,..,word@Doc{:}P,..,l_n}$$

$$\frac{l_1,..,Word@Doc{:}P,..,l_n}{l_1,..,Word@Doc{:}P,some\_property(Word),..,l_n}$$

4 Related Work

As already mentioned in the introduction, ILP has been rarely used for text categorization and information extraction in its "pure" form. In [3] an ILP approach to the problem of document categorization is described. Cohen represents a document by a set of facts of the form `word`$_i$`(doc,pos)`. The predicate `word`$_i$ indicates that a word `word`$_i$ occurs in a document `doc` at position `pos` with `pos` being a natural number. The main difference of this document representation to our own representation is that in Cohen's approach words are predicates while we have an own type for words. This allows to test on word properties

without leaving 1st order logic. Cohen's approach also allows for phrases. This is done by the predicates `near1(`p_1`,`p_2`)`, `near2(`p_1`,`p_2`)`, `near3(`p_1`,`p_2`)` which denote the maximum difference between two word positions. For instance, `near1(`p_1`,`p_2`)` is true if $|p_1 - p_2| \leq 1$. An additional binary predicate `after(`p_1`,`p_2`)` is used to test whether a position p_1 is before a position p_2, i.e., it is true if $p_2 > p_1$ holds. The problem of information extraction was not addressed by Cohen.

In recent work Freitag proposes an ILP-like formalism for information extraction [6], called *SRV*. Freitag informally describes the examples as a set of annotated documents. Without going into the details of his rule language constructs, we just want to give a raw impression of how Freitag's information extraction rules look like. The example shows a rule for extracting course numbers of a university's web page:

```
coursenumber :- length(= 2),
                every(in_title false),
                some(?A [] all_upper_case true),
                some(?B [] tripleton true).
```

The rule extracts every text fragment which satisfies the following conditions: the fragment contains two words (`length(= 2)`), no word within the title is part of the fragment (`every(in_title false)`), one word of the fragment consists only of upper-case characters (`some(?A [] all_upper_case true)`), and the other word of the fragment consists of three characters (`some(?B [] tripleton true)`). The relative positions of words in Freitag's approach are captured by two constructs. Using so-called relational paths a position relative to the current one can be addressed. For instance, `some(?A [prev_token prev_token] capitalized true)` requires a word within a fragment which is preceeded by a capitalized word two tokens back. Similar to our predicate `next`, another predicate `relpos(Var1 Var2 Relop N)` allows to specify distances and ordering of word occurrences. The variables `Var1` and `Var2` denote word occurrences, `Relop` is a comparison operator and `N` is some natural number.

Freitag did not address the problem of text categorization explicitly. As potential disadvantages of Freitag's approach we see:

- Freitag's rules are not represented in a standard logic language.
- There are no variables whose bindings provide the document the information was found in, the position it was found at and the information itself.
- Only single value extraction rules can be formulated. This is because all predicates implicitly relate to one text fragment.

There are a a number of other systems that learn information extractions rules and do not use any logic programming formalism such as *AutoSlog* [11], *LIEP* [8], *WHISK* [13], and *RAPIER* [1]. More complete and detailed overviews are given in [10] and in [13].

5 Summary

We have proposed our mapping of typical text patterns to logic programs which is based on types for text, words, and text positions and three fundamental predicates. Based on this representation we presented the main concepts of our own rule learner for text categorization and information extraction.

We see the contribution of this paper in the mapping of text pattern languages to standard logic expressions. We think that this mapping may help to see TC and IE as two interesting and stimulating ILP applications.

It was not the goal of the paper to compare the effectiveness of our learning algorithm to the effectiveness of other algorithms. Nevertheless, first experiments with an initial version of our rule learner indicate that results are comparable to those reported in literature for standard problems.

Appendix - Predicate Definitions in PROLOG

```
wordpos([Word | Rest], Word, 1).
wordpos([_ | Rest], Word, P) :- wordpos(Rest, Word, Q), P is Q + 1.

fragment([_ | Rest], P1, P2, F) :-  P1 > 1, P11 is P1-1, P21 is P2-1,
                                    fragment(Rest, P11, P21, F).
fragment(Text, 1, P2, F) :- fragment(Text, P2, F).
fragment([W | R], P, [W | F]) :- P > 0, Q is P-1, fragment(R, Q, F).
fragment(Text, 0, []).

next(P, P1) :- next(P, 1, P1).
next(P, X, P1) :- number(P), number(X), P1 is P + X.
next(P, X, P1) :- var(P), number(X), number(P1), P is P1 - X.
next(P1, Min, Max, P2) :-  number(P1), number(Min), number(Max), number(P2),
                           D is P2 - P1, D >= Min, D =< Max.
```

Note that not all of the above predicates are completely invertible.

References

[1] M. Califf and R. Mooney. Relational Learning of Pattern-Match Rules for Information Extraction. *Working Papers of the ACL 97 Workshop on Natural Language Learning*, pages 9–15, 1997.

[2] P. Clark and R. Boswell. Rule Induction with CN2: Some Recent Improvements. In *Machine Learning – EWSL-91. European Working Session on Learning*, pages 151–163, Porto, Portugal, March 6-8 1991.

[3] W.W. Cohen. Learning to Classify English Text with ILP Methods. In *Advances in Inductive Logic Programming*, pages 124–143. IOS Press, 1996.

[4] A. Dengel and K. Hinkelmann. The Specialist Board — A Technology Workbench for Document Analysis and Understanding. In *Proceedings of the 2nd World Conference on Integrated Design and Process Technology (IDPT '96)*, pages 36–47, Austin, TX, USA, December 1996.

[5] A. Dengel, R. Hoch, F. Hönes, M. Malburg, and A. Weigel. *Techniques for Improving OCR Results.* World Scientific Publishers Company, 1997.

[6] D. Freitag. Toward general-purpose learning for information extraction. In *17th International Conference on Computational Linguistics (COLING-ACL 98)*, pages 404–408, Montreal, Canada, August 10-14 1998.

[7] J. Fürnkranz. Separate-and-Conquer Rule Learning. *Artificial Intelligence Review*, 13(1):3–54, 1999.

[8] S. B. Huffman. Learning information extraction patterns from examples. In *Connectionist, Statistical, and Symbolic Approaches to Learning for Natural Language Processing*, volume 1040 of *LNAI*, pages 246–260. Springer Verlag, Berlin, 1996.

[9] M. Junker and A. Abecker. Learning complex patterns for document categorization. In *15th National Conference on Artifical Intelligence (AAAI 98)*, Madison, Wisconsin, July 26-30 1998. AAAI98/ICML98 Workshop on Learning for Text Categorization, (to appear).

[10] I. Muslea. Extraction Patterns: from Information Extraction to Wrapper Generation. http://www.isi.edu/~muslea/PS/epiReview.ps, 1998.

[11] E. Riloff. Automatically Generating Extraction Patterns from Untagged Text. In *13th National Conference on Artifical Intelligence (AAAI 96)*, pages 1044–1049, Portland, Oregon, USA, August 4-8 1996.

[12] D. Sankoff and J.B. Kruskal. *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison.* Addison-Wesley, 1983. (out of print).

[13] S. Soderland. Learning Information Extraction Rules for Semi-Structured and Free Text. *Machine Learning*, 34(1-3), 1999.