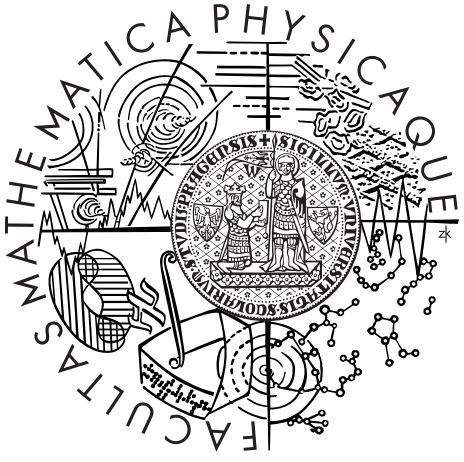
Charles University in Prague
Faculty of Mathematics and Physics

DOCTORAL THESIS



First and last name of the author

Title of the thesis

Name of the department or institute

Supervisor of the doctoral thesis: First and last name
Study programme: programme
Specialization: specialization

Prague YEAR

Acknowledgments

I wish to thank my supervisor ...

I declare that I carried out this doctoral thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In ..... date ..... signature of the author

Název práce: Název práce

Autor: Jméno a příjmení autora

Katedra: Název katedry či ústavu, kde byla práce oficiálně zadána

Vedoucí disertační práce: Jméno a příjmení s tituly, pracoviště

Abstrakt:

Klíčová slova:

Title:

Author: Jméno a příjmení autora

Department: Název katedry či ústavu, kde byla práce oficiálně zadána

Supervisor: Jméno a příjmení s tituly, pracoviště

Abstract:

Keywords:

Contents

1. Extraction Method Based on ILP Machine Learning

1.1 Introduction

Automated semantic annotation (SA) is considered to be one of the most important elements in the evolution of the Semantic Web. Besides that, SA can provide great help in the process of data and information integration and it could also be a basis for intelligent search and navigation.

In this chapter we present a method for classical and semantic information extraction and annotation of texts, which is based on a deep linguistic analysis and Inductive Logic Programming (ILP). This approach is quite novel because it directly combines deep linguistic parsing with machine learning (ML). This combination and the use of ILP as a ML engine have following benefits: Manual selection of learning features is not needed. The learning procedure has full available linguistic information at its disposal and it is capable to select relevant parts itself. Extraction rules learned by ILP can be easily visualized, understood and adapted by human.

A description, implementation and initial evaluation of the method are the main contributions of the present work.

1.2 Data Flow ?or Schema of the Extraction Process?

Figure 1.1

1.3 Implementation

Here we just briefly describe implementation of our system. The system consists of several modules, all integrated in GATE as processing resources (PRs).

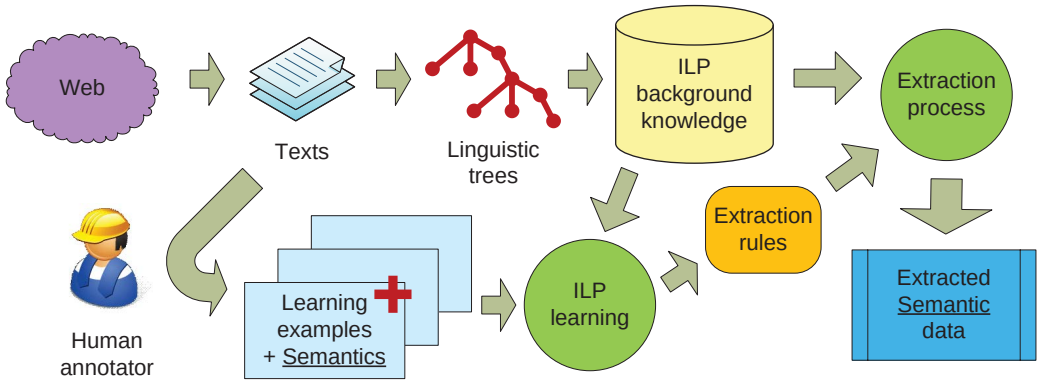


Figure 1.1: ILP data flow.

1.3.1 TectoMT Wrapper (Linguistic Analysis)

TectoMT wrapper is a GATE component (processing resource), which takes the text of a GATE document, sends it to TectoMT linguistic analyzers, parses the results and converts the results to the form of GATE annotations. The next section provides details about how PDT annotations are represented in GATE.

Because TectoMT has to run as a separate process (it is implemented in Perl) and the initialization of TectoMT analyzers usually takes significant amount of time it would be very inefficient to start a new TectoMT instance for each document. Therefore the implementation currently offers two modes of execution: batch (TectoMTBatchAnalyser) and online (TectoMTOnlineAnalyser).

The batch mode is implemented similarly to the Batch Learning PR[1]. Batch mode of execution in the context of GATE corpus pipelines is a deviation from the standard execution mode, when documents are processed one by one. During the execution as a part of a corpus pipeline, TectoMTBatchAnalyser only accumulates documents and the whole work is done as a batch when the last document is encountered. This also implies that TectoMTBatchAnalyser has to be the last PR in the pipeline because it produces no output in the time of execution (except for the last document when the whole batch is executed).

Client-server model of implementation is used in the online mode. A separate TectoMT server process is started at the time of initialization and during the execution, GATE documents are processed in ordinary way, one by one. This means that (similarly to the previous case) each document is converted to the TectoMT readable format, sent to TectoMT and the result is converted back to GATE. The online mode of execution is a bit slower than the batch mode because additional time is spent on client-server communication (XML-RPC[2]).

1.3.2 PDT Annotations in GATE

Although GATE annotations are just highlighted pieces of text (see also Section **??**) it is possible to use them to encode dependency tree structures. It is possible because each GATE annotation has a unique identifier (ID) and an arbitrary set of features (name-value pairs) can be assigned to it. The way how the PDT dependency trees are encoded in GATE is in fact the same as in the GATE wrapper for the Stanford Parser[3].

Three main constructs are used to capture an arbitrary configuration of a linguistic dependency tree:

tree nodes (usually corresponding to words (tokens) of a sentence)

edges (dependency relations between nodes)

node attributes (connected linguistic features like POS, gender, tense, case, etc.)

These constructs are encoded in GATE in the following way: tree nodes correspond to token annotations, node attributes are saved as token annotation features and edges are encoded as another special kind of annotations.

Two kinds of token annotations are used to represent two kinds of trees and tree nodes.

[1] `http://gate.ac.uk/userguide/sec:ml:batch-learning-pr`
[2] `http://www.xmlrpc.com/`
[3] `http://gate.ac.uk/userguide/sec:parsers:stanford`

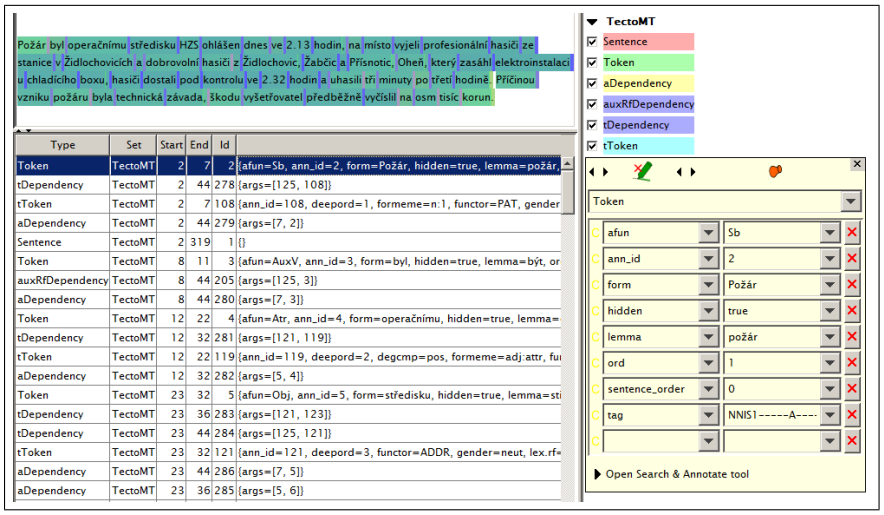


Figure 1.2: PDT annotations in GATE (screenshot).

"Token" annotation type is used for analytical tree nodes and "tToken" for tectogrammatical tree nodes.

Four kinds of edges (dependencies) are implemented by the TectoMT wrapper: analytical dependencies, tectogrammatical dependencies, aux.rf (auxiliary reference) and lex.rf (main lexical reference). The last two kinds (aux.rf and lex.rf) are used to connect tectogrammatical and analytical nodes. The implementation differs according to the cardinality of a dependency type. The first three kinds are of the cardinality one-to-many (one parent node can have many children nodes) and the last one (lex.rf) if of the cardinality one-to-one (one parent node has at most one child). Because of that lex.rf edges can be stored as features (with the name "lex.rf") of "tToken" annotations. Note that a GATE annotation feature can only have one value per annotation. In this case the annotation ID of the referenced "Token" annotation (referenced analytical node) is the value of the lex.rf feature.

One-to-many dependencies are stored as separate annotations (type names: "aDependency", "tDependency", "aux.rf") with a single feature called "args". Values of this feature are of Java type List<Integer> (list of integers). The list always contains just two items. The first one is the annotation ID of the parent annotation; the second one is the ID of the child annotation. Instead of using one list feature, two simple features (like "arg1", "arg2" or "parentID", "childID") could be used, but the implementation is consistent with the wrapper for the Stanford Parser (using the single list feature "args"), thus PDT dependencies are compatible with Stanford dependencies in GATE.

It is not simple to demonstrate the GATE representation of the dependencies in a static printed form; we can only show a GATE screenshot (Figure 1.2) that partly illustrates that.

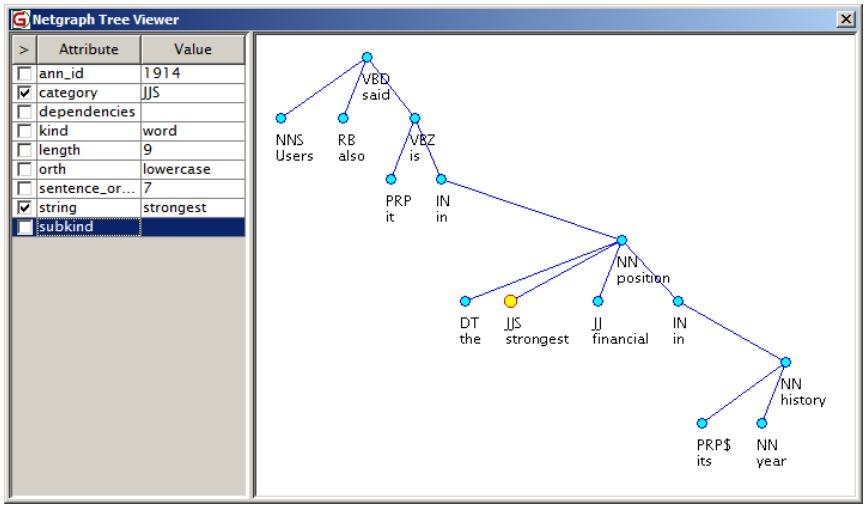
Netgraph Tree Viewer

Figure 1.3

1.3.3 ILP Wrapper (Machine Learning)

After a human annotator have annotated several documents with desired target annotations, machine learning takes place. This consists of two steps:

1. learning of extraction rules from the target annotations and



Sentence: Users also said it is in the strongest financial position in its 24-year history.

Figure 1.3: Netgraph Tree Viewer in GATE (for Stanford Dependencies, screenshot).

2. application of the extraction rules on new documents.

In both steps the linguistic analysis has to be done before and in both steps background knowledge (a logical database of facts) is constructed from linguistic structures of documents that are being processed. We call the process of background knowledge construction as *ILP serialization*; more details are presented below in Section 1.3.4.

After the ILP serialization is done, in the learning case, positive and negative examples are constructed from target annotations and the machine learning ILP inductive procedure is executed to obtain extraction rules.

In the application case a Prolog system is used to check if the extraction rules entail any of target annotation candidates.

The learning examples and annotation candidates are usually constructed from all document tokens (and we did so in the present solution), but it can be optionally changed to any other textual unit, for example only numerals or tectogrammatical nodes (words with lexical meaning) can be selected. This can be done easily with the help of *Machine Learning PR* (LM PR) from GATE[4].

ML PR provides an interface for exchange of features (including target class) between annotated texts and propositional learners in both directions – during learning as well as during application. We have used ML PR and developed our *ILP Wrapper* for it. The implementation was a little complicated because complex linguistic structures cannot be easily passed as propositional features, so in our solution we use the ML PR interface only for exchange of the class attribute and annotation id and we access the linguistic structures directly in a document.

1.3.4 ILP Serialization

In this section details about conversion of linguistic trees to ILP background knowledge (a Prolog logical database of facts) will be presented. Although the construction is quite strait forward it is worth describing because it makes it easier to understand the extraction rules found by the ILP learning procedure.

[4] *Machine Learning PR* is an old GATE interface for ML and it is almost obsolete but in contrast to the new *Batch Learning PR* the LM PR is easy to extend for a new ML engine.

As mentioned in Section 1.3.2: three main constructs are used to capture an arbitrary configuration of a dependency linguistic tree: nodes, edges and node attributes. During the process of ILP Serialization these constructs are rendered to Prolog in following way.

A unique identifier (node ID) is generated for every tree node. The identifier is based on document name and GATE annotation ID (sentence order and node deep order are used outside of GATE, see PML→RDF transformation in Section **??**.) These node IDs correspond to simple atoms and they represent tree nodes in the fact database. A node type (used by the ILP learning algorithm) is assigned to a node ID by predicates `Token(NodeID)` for analytical tree nodes and `tToken(NodeID)` for tectogrammatical tree nodes.

Tree nodes are connected by edges using binary predicates of the form:

```
dependency_type_name(ParentNodeID, ChildNodeID)
```

Note that the parent (governor) node always occupies the first argument and the child (dependant) node the second one. Predicate name *tDependency* is used for tectogrammatical dependencies and *aDependency* for analytical ones. There are also special kinds of dependencies that connect tectogrammatical and analytical nodes: *lex.rf* (main lexical reference) and *aux.rf* (auxiliary reference), in these cases tectogrammatical node occupies the first argument and analytical the second.

Node attributes are assigned to node IDs by binary predicates of the form:

```
attribute_name(NodeID, AttributeValue)
```

There are about thirty such predicates like *t_lemma* (tectogrammatical lemma), *functor* (tectogrammatical functor), *sempos* (semantic part of speech), *negation*, *gender*, etc. but minority of them is usually used in extraction rules.

Example of a serialized tectogrammatical tree is in Figure 1.4 it is the same tree as in Figure **??**.

1.3.5 ?Connecting Linear GATE Annotations with Tree Nodes? Intersection with Tree Nodes?

How to compare correctness of IE?

ILP identifies relevant tree nodes but manual gold standard annotations are put directly on the surface text.

A mapping of tree nodes and text surface has to be established.

1.3.6 Root/Subtree Preprocessing/Postprocessing

Sometimes annotations span over more than one token. This situation complicates the process of machine learning and this situation is often called as "chunk learning". Either we have to split a single annotation to multiple learning instances and after application we have to merge them back together, or we can change the learning task from learning annotated tokens to learning borders of annotations (start tokens and end tokens). The later approach is implemented in GATE in *Batch Learning PR* in the 'SURROUND' mode.

We have used another approach to solve this issue. Our approach is based on syntactic structure of a sentence and we call it "root/subtree preprocessing/postprocessing". The idea is based on the observation that tokens of a multi-token annotation usually have a common parent node in a syntactic tree. So we can

1. extract the parent nodes (in dependency linguistics this node is also a token and it is usually one of the tokens inside the annotation),

2. learn extraction rules for parent nodes only and

```
tToken(  id_jihomoravsky47443_243).
t_lemma( id_jihomoravsky47443_243, 'být'). %to be
functor( id_jihomoravsky47443_243, 'PRED').
sempos(  id_jihomoravsky47443_243, 'v').
tDependency( id_jihomoravsky47443_243, id_jihomoravsky47443_238).
tToken(  id_jihomoravsky47443_238).
t_lemma( id_jihomoravsky47443_238, ','). %comma
functor( id_jihomoravsky47443_238, 'APPS').
sempos(  id_jihomoravsky47443_238, 'n.denot').
gender(  id_jihomoravsky47443_238, 'nr').
tDependency( id_jihomoravsky47443_238, id_jihomoravsky47443_237).
tToken(  id_jihomoravsky47443_237).
t_lemma( id_jihomoravsky47443_237, 'vyčíslit'). %to quantify
functor( id_jihomoravsky47443_237, 'PAT').
sempos(  id_jihomoravsky47443_237, 'v').
tDependency( id_jihomoravsky47443_237, id_jihomoravsky47443_245).
tToken(  id_jihomoravsky47443_245).
t_lemma( id_jihomoravsky47443_245, 'předběžně'). %preliminarily
functor( id_jihomoravsky47443_245, 'MANN').
sempos(  id_jihomoravsky47443_245, 'adv.denot.grad.nneg').
tDependency( id_jihomoravsky47443_237, id_jihomoravsky47443_244).
tToken(  id_jihomoravsky47443_244).
t_lemma( id_jihomoravsky47443_244, 'vyšetřovatel'). %investigator
functor( id_jihomoravsky47443_244, 'ACT').
sempos(  id_jihomoravsky47443_244, 'n.denot').
gender(  id_jihomoravsky47443_244, 'anim').
tDependency( id_jihomoravsky47443_237, id_jihomoravsky47443_240).
tToken(  id_jihomoravsky47443_240).
t_lemma( id_jihomoravsky47443_240, 'osm'). %eight
functor( id_jihomoravsky47443_240, 'PAT').
sempos(  id_jihomoravsky47443_240, 'n.quant.def').
gender(  id_jihomoravsky47443_240, 'nr').
tDependency( id_jihomoravsky47443_240, id_jihomoravsky47443_242).
tToken(  id_jihomoravsky47443_242).
t_lemma( id_jihomoravsky47443_242, 'tisíc'). %thousand
functor( id_jihomoravsky47443_242, 'RSTR').
sempos(  id_jihomoravsky47443_242, 'n.quant.def').
gender(  id_jihomoravsky47443_242, 'inan').
tDependency( id_jihomoravsky47443_242, id_jihomoravsky47443_247).
tToken(  id_jihomoravsky47443_247).
t_lemma( id_jihomoravsky47443_247, 'koruna'). %crown
functor( id_jihomoravsky47443_247, 'MAT').
sempos(  id_jihomoravsky47443_247, 'n.denot').
gender(  id_jihomoravsky47443_247, 'fem').
tDependency( id_jihomoravsky47443_237, id_jihomoravsky47443_246).
tToken(  id_jihomoravsky47443_246).
t_lemma( id_jihomoravsky47443_246, 'škoda'). %damage
functor( id_jihomoravsky47443_246, 'PAT').
sempos(  id_jihomoravsky47443_246, 'n.denot').
gender(  id_jihomoravsky47443_246,'fem').
```

Figure 1.4: ILP serialization example

3. span annotations over the whole subtrees of root tokens found during the application of extraction rules.

We call the first point as *root preprocessing* and the last point as *subtree postprocessing*. We have successfully used this technique for the 'damage' task of our evaluation corpus (See Section 1.4 for details.)

1.3.7 Semantic Interpretation

Information extraction can solve the task "how to get documents annotated", but as we aim on the semantic annotation, there is a second step of "semantic interpretation" that has to be done. In this step we have to interpret the annotations in terms of a standard ontology. On a very coarse level this can be done easily. Thanks to GATE ontology tools [Bontcheva *et al.*, 2004] we can convert all the annotations to ontology instances with a quite simple JAPE [Cunningham *et al.*, 2000] rule, which takes the content of an annotation and saves it as a label of a new instance or as a value of some property of a shared instance. For example in our case of traffic and fire accidents, there will be a new instance of an accident class for each document and the annotations would be attached to this instance as values of its properties. Thus from all annotations of the same type, instances of the same ontology class or values of the same property would be constructed. This is very inaccurate form of semantic interpretation but still it can be useful. It is similar to the GoodRelation [Hepp, 2008] design principle of *incremental enrichment*[5]:

> "...you can still publish the data, even if not yet perfect. The Web will do the rest – new tools and people."

But of course we are not satisfied with this fashion of semantic interpretation and we plan to further develop the semantic interpretation step as a sophisticated "annotation → ontology" transformation process that we have proposed in one of our previous works [Dědek and Vojtáš, 2008].

1.3.8 How to Download

The project website[6] provides several ways how to get all the presented tools running. A platform independent installer, Java binaries and source codes are provided under the GPL license.

1.4 Evaluation

1.4.1 Dataset

We have evaluated our state of the art solution on a small dataset that we use for development. It is a collection of 50 Czech texts that are reporting on some accidents (car accidents and other actions of fire rescue services). These reports come from the web of Fire rescue service of Czech Republic[7]. The labeled corpus is publically available on the

[5] http://www.ebusiness-unibw.org/wiki/Modeling_Product_Models#Recipe:_.22Incremental_Enrichment.22

[6] http://czsem.berlios.de

[7] http://www.hzscr.cz/hasicien/

web of our project[8]. The corpus is structured such that each document represents one event (accident) and several attributes of the accident are marked in text. For the evaluation we selected two attributes of different kind. The first one is 'damage' – an amount (in CZK - Czech Crowns) of summarized damage arisen during a reported accident. The second one is 'injuries', it marks mentions of people injured during an accident. These two attributes differ. Injuries annotations always cover only a single token, while damage annotations usually consist of two or three tokens – one or two numerals express the amount and one extra token is for currency.

These two attributes differ in two directions:

1. Injuries annotations always cover only a single token while damage usually consists of two or three tokens - one or two numerals express the amount and one extra token is for currency.

2. The complexity of the marked information (and the difficulty of the corresponding extraction task) differs slightly. While labeling of all money amounts in the corpus will result in 75% accuracy for damage annotations, in the case of injured persons mentions there are much more possibilities and indications are more spread in context.

1.4.2 Comparison with Paum classifier

To compare our solution with other alternatives we took the Paum propositional learner from GATE [Li *et al.*, 2002]. The quality of propositional learning from texts is strongly dependent on the selection of right features. We obtained quite good results with features of a window of two preceding and two following token lemmas and morphological tags. The precision was further improved by adding the feature of *analytical function* from the syntactic parser (see the last row of Table 1.1).

Because we did not want to invest much time to this and the feature setting of the Paum learner was quite simple (a window of two preceding and following token lemmas and morphological tags). We admit that looking for better features could further improve the results of the Paum learner.

1.4.3 Cross validation

We used the 10-fold cross validation in the evaluation. Thanks to this technique the evaluation is simple. After processing all the folds each document is processed with some of the ten learned models such that the particular document was not used in learning of that model, so all documents are unseen by the model applied on them. At the end we just compare the gold standard annotations with the learned ones in all documents.

1.4.4 Results

Results of a 10-fold cross validation are summarized in Table 1.1. We used standard information retrieval performance measures: precision, recall and F_1 measure and also theirs lenient variants (overlapping annotations are added to the correctly matching ones, the measures are the same if no overlapping annotations are present).

In the first task ('damage') the methods obtained much higher scores then in the second ('injuries') because the second task is more difficult. In the first task also the root/subtree

[8] `http://czsem.berlios.de/`

task/method	matching	missing	excessive	overlap	prec.%	recall%	F1.0%
damage/ILP	14	0	7	6	51.85	70.00	59.57
damage/ILP – lenient measures					74.07	100.00	85.11
dam./ILP-roots	16	4	2	0	88.89	80.00	84.21
damage/Paum	20	0	6	0	76.92	100.00	86.96
injuries/ILP	15	18	11	0	57.69	45.45	50.85
injuries/Paum	25	8	54	0	31.65	75.76	44.64
inj./Paum-afun	24	9	38	0	38.71	72.73	50.53

Table 1.1: Evaluation results

```
%[Rule 1] [Pos cover = 14 Neg cover = 0]
damage_root(A) :- lex_rf(B,A), has_sempos(B,'n.quant.def'), tDependency(C,B),
   tDependency(C,D), has_t_lemma(D,'investigator').

%[Rule 2] [Pos cover = 13 Neg cover = 0]
damage_root(A) :- lex_rf(B,A), has_functor(B,'TOWH'), tDependency(C,B),
   tDependency(C,D), has_t_lemma(D,'damage').


%[Rule 1] [Pos cover = 7 Neg cover = 0]
injuries(A) :- lex_rf(B,A), has_functor(B,'PAT'), has_gender(B,anim),
   tDependency(B,C), has_t_lemma(C,'injured').

%[Rule 8] [Pos cover = 6 Neg cover = 0]
injuries(A) :- lex_rf(B,A), has_gender(B,anim), tDependency(C,B),
   has_t_lemma(C,'injure'), has_negation(C,neg0).
```

TODO: `damage_root -> mention_root`

Figure 1.5: Examples of learned rules, Czech words are translated.

preprocessing/postprocessing improved results of ILP such that afterwards, annotation borders were all placed precisely. The ILP method had better precision and worse recall than the Paum learner but the F_1 score was very similar in both cases.

Statistical Significance

The term statistical significance refers to the result of a pair-wise comparison of learning engines using the corrected resampled (two tailed) T-Test [Nadeau and Bengio, 2003], which is suitable for cross validation based experiments. The Weka implementation[9] is used. Test significance is 0.05 in all cases.

1.4.5 Examples of learned rules

In Figure 1.5 we present some examples of the rules learned from the whole dataset. The rules demonstrate a connection of a target token with other parts of a sentence through linguistic syntax structures. For example the first rule connects a root numeral (*n.quant.def*) of 'damage' with a mention of 'investigator' that stated the mount. In the last rule only a positive occurrence of the verb 'injure' is allowed.

Experience with human-designed rules.

[9] `http://www.cs.waikato.ac.nz/ml/weka/`

1.5 Conclusion and Future Work

From our experiments can be seen that ILP is capable to find complex and meaningful rules that cover the intended information. But in terms of the performance measures the results are not better than those from a propositional learner. This is quite surprising observation because Czech is a language with free word order and we would expect much better results of the dependency approach than those of the position based approach, which was used by the propositional learner.

Our method is still missing an intelligent semantic interpretation procedure and it should be evaluated on bigger datasets (e.g. MUC, ACE, TAC, CoNLL) and other languages. So far we also do not provide a method for classical relation extraction (like e.g. in [Bunescu and Mooney, 2007]). In the present solution we deal with relations implicitly. The method has to be adapted for explicit learning of relations in the form of "subject predicate object".

Our method can also provide a comparison of linguistic formalisms and tools because on the same data we could run our method using different linguistic analyzers and compare the results.

Bibliography

K. Bontcheva, V. Tablan, D. Maynard, and H. Cunningham (2004), Evolving GATE to Meet New Challenges in Language Engineering, *Natural Language Engineering*, 10(3/4):349—373.

Razvan Bunescu and Raymond Mooney (2007), Extracting Relations from Text: From Word Sequences to Dependency Paths, in Anne Kao and Stephen R. Poteet, editors, *Natural Language Processing and Text Mining*, chapter 3, pp. 29–44, Springer, London, ISBN 978-1-84628-175-4, doi:10.1007/978-1-84628-754-1_3, URL `http://dx.doi.org/10.1007/978-1-84628-754-1_3`.

Hamish Cunningham, Diana Maynard, and Valentin Tablan (2000), JAPE: a Java Annotation Patterns Engine, Technical report, Department of Computer Science, The University of Sheffield, URL `http://www.dcs.shef.ac.uk/intranet/research/resmes/CS0010.pdf`.

Jan Dědek and Peter Vojtáš (2008), Computing aggregations from linguistic web resources: a case study in Czech Republic sector/traffic accidents, in Cosmin Dini, editor, *Second International Conference on Advanced Engineering Computing and Applications in Sciences*, pp. 7–12, IEEE Computer Society, ISBN 978-0-7695-3369-8, URL `http://dx.doi.org/10.1109/ADVCOMP.2008.17`.

Martin Hepp (2008), GoodRelations: An Ontology for Describing Products and Services Offers on the Web, in Aldo Gangemi and Jérôme Euzenat, editors, *EKAW*, volume 5268 of *Lecture Notes in Computer Science*, pp. 329–346, Springer, ISBN 978-3-540-87695-3.

Yaoyong Li, Hugo Zaragoza, Ralf Herbrich, John Shawe-Taylor, and Jaz S. Kandola (2002), The Perceptron Algorithm with Uneven Margins, in *ICML '02: Proceedings of the Nineteenth International Conference on Machine Learning*, pp. 379–386, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, ISBN 1-55860-873-7.

Claude Nadeau and Yoshua Bengio (2003), Inference for the Generalization Error, *Mach. Learn.*, 52:239–281, ISSN 0885-6125, doi:10.1023/A:1024068626366, URL `http://portal.acm.org/citation.cfm?id=779909.779927`.

List of Tables

Nomenclature

API	Application Programming Interface
CRISP	Cross Industry Standard Process (for Data Mining) http://en.wikipedia.org/wiki/CRISP_DM
DL	Description Logic http://en.wikipedia.org/wiki/Description_logic
FGD	Functional Generative Description (of Language), see Section ??
FS	Feature Structure, see Section ??
GAP	Generalized Annotated Programs, see Section ??
GATE	General Architecture for Text Engineering, see Section ??
GPL	GNU General Public License http://www.gnu.org/licenses/gpl.html
GRDDL	Gleaning Resource Descriptions from Dialects of Languages http://www.w3.org/TR/grddl/
GUI	Graphical User Interface
IE	Information Extraction
ILP	Inductive Logic Programming
JAPE	Java Annotation Patterns Engine http://gate.ac.uk/userguide/chap:jape
ML	Machine Learning
MST	Minimum-Spanning Tree
OWL	Web Ontology Language http://www.w3.org/TR/owl-primer/
PAUM	Perceptron Algorithm with Uneven Margins
PDT	Prague Dependency Treebank, see Section ??
PEDT	Prague English Dependency Treebank
PML	Prague Markup Language, see Section ??
POS	Part of Speech
PR	Processing Resource (in GATE) http://gate.ac.uk/userguide/chap:creole-model
PTB	Penn Treebank http://www.cis.upenn.edu/~treebank/

RDF	Resource Description Framework http://www.w3.org/RDF/
RDFa	Resource Description Framework - in - attributes http://www.w3.org/TR/xhtml-rdfa-primer/
RPC	Remote Procedure Call
RSS	Really Simple Syndication, or Rich Site Summary, but originally RDF Site Summary http://www.rssboard.org/rss-specification
SPARQL	SPARQL Query Language for RDF http://www.w3.org/TR/rdf-sparql-query/
SQL	Structured Query Language http://www.iso.org/iso/catalogue_detail.htm?csnumber=45498
SWRL	Semantic Web Rule Language http://www.w3.org/Submission/SWRL/
URL	Uniform Resource Locator (RFC1738, http://www.ietf.org/rfc/rfc1738.txt)
W3C	World Wide Web Consortium http://www.w3.org/
XHTML	Extensible HyperText Markup Language http://www.w3.org/TR/xhtml1/
XML	Extensible Markup Language http://www.w3.org/XML/
XSLT	Extensible Stylesheet Language Transformations http://www.w3.org/TR/xslt
ÚFAL	Institute of Formal and Applied Linguistics, Prague, Czech Republic (Ústav formální a aplikované lingvistiky) http://ufal.mff.cuni.cz/

Attachments