Project Title:

INFRASTRUCTURE AND INTEGRATED TOOLS FOR PERSONALIZED LEARNING OF READING SKILL

Project Acronym:

Grant Agreement number:

731724 — iRead H2020-ICT-2016-2017/H2020-ICT-2016-1

Subject:

D5.3 Syntax Analysers

Dissemination Level:

PUBLIC

Lead Beneficiary:

DFKI

Project Coordinator:

UCL

Contributors:

DFKI, NTUA, UOI

Revision	Preparation date	Period covered	Project start date	Project duration
V2	June 2018	Month 1-18	01/01/2017	48 Months

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Grant Agreement No 731724

Table of content

Contents

1. EXECUTIVE SUMMARY

This deliverable contains a description of the DFKI MunderLine pipeline (see also http://iread.dfki.de/), which is used for the syntactic analysis of free text from the relevant natural languages covered in the iREAD project, viz. English, German, Greek, and Spanish. MunderLine was developed during the reported period of the iREAD project making use of existing technologies and tools previously developed by the LT-lab team of DFKI. This concerns mainly the dependency parser MDP and the flexible tokenizer JTOC. As part of this deliverable the main tasks were on the following issues:

- Adaptation and extending MDP for the requirements of the iREAD project
- Development of a general Machine Learning tagger – called GNT - for part-of-speech tagging, morphological tagging and Named Entity tagging
- Integration of JTOC, GNT and MDP into one common framework
- Providing a common method for training all relevant language core tools using the newly developed multilingual universal dependency grammars
- Providing a common method for testing and running the learned models
- Development of server and client methods based on REST API standard
- Providing a simple and flexible solution for compiling and packaging the whole system including the necessary language resources

2. INTRODUCTION

This deliverable contains a description of the DFKI MunderLine pipeline (see also http://iread.dfki.de/), which is used for the syntactic analysis of free text from the relevant natural languages covered in the iREAD project, viz. English, German, Greek, and Spanish. MunderLine was developed during the reported period of the iREAD project making use of existing technologies and tools previously developed by the LT-lab team of DFKI. This concerns mainly the dependency parser MDP (cf. Volokh & Neumann, 2011, 2012) and the flexible tokenizer JTOC (https://github.com/DFKI-MLT/JTok). As part of this deliverable the main tasks were on the following issues:

- Adaptation and extending MDP for the requirements of the iREAD project
- Development of a general Machine Learning tagger – called GNT - for part-of-speech tagging, morphological tagging and Named Entity tagging
- Integration of JTOC, GNT and MDP into one common framework
- Providing a common method for training all relevant language core tools using the newly developed multilingual universal dependency grammars
- Providing a common method for testing and running the learned models
- Development of server and client methods based on REST API
- Providing a simple and flexible solution for compiling and packaging the whole system including the necessary language resources

Before describing the main points in more detail, the next section will provide an overview of the complete MunderLine pipeline architecture.
MunderLine is an instance of a trainable parsing engine. Such trainable parsers receive as input a (usually very large) set of sentences (also called a "treebank") each of which is already (manually or semi-automatically) annotated with the correct syntactic dependency tree. A parsing engine together with a statistical-based learning algorithm is applied to automatically learn a model of all possible syntactic decisions that can be induced from the treebank. This acquired model is then used to determine the syntactic structure of any new sentence. Since the annotation schema is basically the only language-specific parameter, such a trainable parsing system is inherently multilingual, because it can process treebanks of any language.
The syntactic annotation schema of a treebank usually follows a linguistic theory or formalism. However, dependency theory in particular has shown recently to be very suitable for achieving the necessary degree of robustness and efficiency in such a learning environment. The dependency structure of a sentence is a rooted tree (more precisely, a rooted acyclic graph) where the nodes are labelled with the words of the sentence and the directed edges between the nodes are labelled with the grammatical relations that hold between pairs of words. Dependency structures are appealing because they already represent a "shallow" semantic relationship. They are a suited data structure for many semantic applications, e.g. semantic search and relation extraction (Neumann et al., 2014).

3. SYNTAX ANALYSERS SECTIONS

3.1 MUNDERLINE ARCHITECTURE

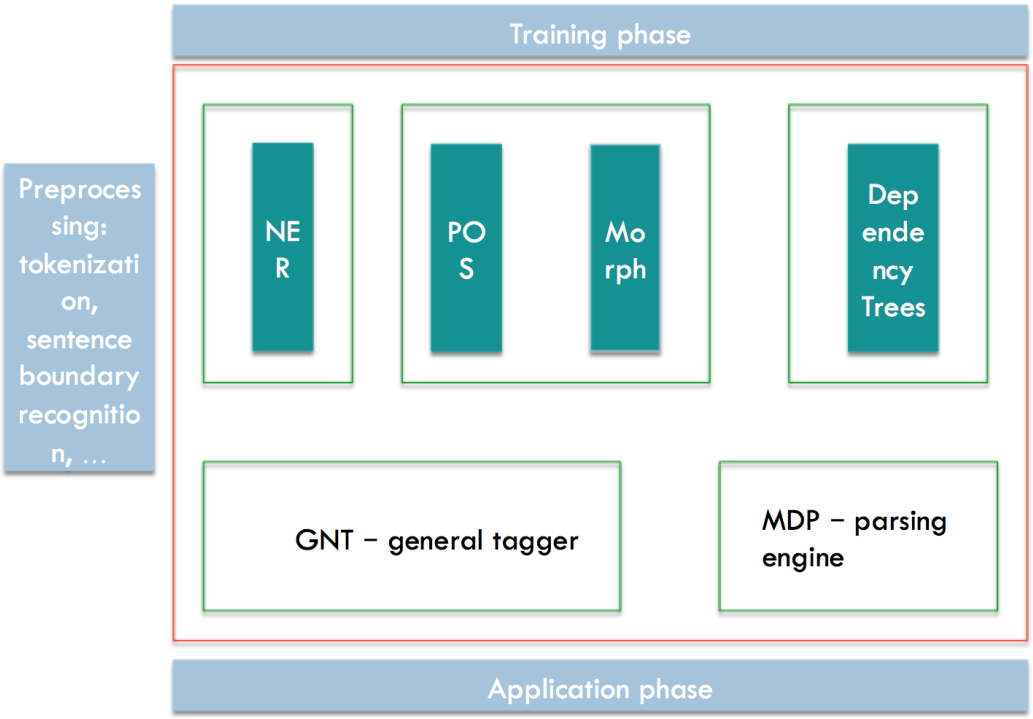


Figure 1: Blueprint of the MunderLine architecture. We distinguish a training phase and an application phase.

Figure 1 displays a blueprint of the MunderLine architecture. We distinguish a training phase, in which the language models for the different languages are learned by means of available training examples, and an application phase, in which new text is analysed using the trained language models. Both phases consist of the same core components and follow the same main processing steps. The main processing steps are:

1. Preprocessing: This step receives a natural language input in form of a free text and performs tokenization and sentence boundary recognition.
2. Named Entity recognition (NER): This step performs recognition and labeling of named entities, which covers single word and multi-word entries of kinds of person, location, originations and others. We follow a Machine Learning approach using standard open annotated data to learn class specific models.
3. Part-of-speech tagging (POS): This step performs tagging of words according to their part-of-speech, e.g., verb, noun etc.
4. Morphological tagging (Morph): This step performs tagging of words according to morphological features as they are defined and provided through the universal dependency treebanks. This covers information about number (singular or plural) or time related information of verbs (e.g., present form, past form).
5. Dependency analysis: This step performs a syntactic analysis of the dependency structure of a sentence using a specialized parsing engine that is realized through the MDP parser engine. The outcome of this step is a graph like structure for each sentence

representing the head-modifier relationships between all words and the specific labeling of it, e.g., grammatical relations like subject, object, sub-clause etc. In MunderLine we are using the universal dependency treebanks and the CONLL-U data annotation format for learning language specific models. An important characteristic of the universal dependency grammars is that they are based on a uniform tag set for part-of-speech and grammatical functions labels (more details are below).

The MunderLine pipeline is used for training language specific models and for applying them on new text. The core components for both phases are the generic tagger GNT and MDP, the parsing engine. And both again are using the linear classifier tool LibLinear (Fan et al., 2008) for performing automatic classification. Given training examples encoded in the CONLL-U format, we first train models for tagging, i.e., tagging models for POS, Morph and NER. We then train models for the dependency analysis using MDP.
In the training phase, preprocessing is not necessary because the training examples are already properly tokenized and sentence split. In the application phase, however, new text has to be firstly preprocessed, and then transformed into the CONLL-U format, before the different models can be applied. Because in the current version of MunderLine the different taggers do not depend on each other, NER, POS and Morph tagging can be done in parallel, in principle. However, in case of the dependency models, training and application of a dependency model requires POS-tagged tokens. Thus, POS tagging has to be done first, before the MDP parser can start.

Before describing more details of GNT and MDP, more details on the CONLL-U data format shall be provided.

Linguistic Data Annotation

In order to facilitate ease of integration of components, we are using the widely used standard text annotation format called CONLL-U (http://universaldependencies.org/format.html) in all linguistic processing components (but not for NER, because here we are using a different standard annotation scheme, which we nevertheless integrate with the CONLL-U format; see notes below). The CONLL-U format is part of the specification of the universal dependency treebanks, and hence, can be considered as a language independent annotation scheme.

Annotations are encoded in plain text files (UTF-8, using only the LF character as line break, including an LF character at the end of file) with three types of lines:

- Word lines containing the annotation of a word/token in 10 fields separated by single tab characters; see below.
- Blank lines marking sentence boundaries.
- Comment lines starting with hash (#).

Sentences consist of one or more word lines, and word lines contain the following fields:

1. ID: Word index, integer starting at 1 for each new sentence; may be a range for multi-word tokens; may be a decimal number for empty nodes.
2. FORM: Word form or punctuation symbol.
3. LEMMA: Lemma or stem of word form.
4. UPOS: Universal part-of-speech tag.
5. XPOS: Language-specific part-of-speech tag; underscore if not available.

6. FEATS: List of morphological features from the universal feature inventory or from a defined language-specific extension; underscore if not available.
7. HEAD: Head of the current word, which is either a value of ID or zero (0).
8. DEPREL: Universal dependency relation to the HEAD (root iff HEAD = 0) or a defined language-specific subtype of one.
9. DEPS: Enhanced dependency graph in the form of a list of head-deprel pairs.
10. MISC: Any other annotation.

Currently, we provide no values for the columns 3, 5, 9, 10 and use the dummy value _ (underscore) instead. In order to integrate the information from the NER tagger, we have added a column 11 and use the BILOU scheme (cf. Ratinov, L., & Roth, D. (2009)) for representing the span and label of recognized Named Entities. For consistency reasons we have added an additional column 12 which is filled with the dummy value (we are planning to use it for encoding additional semantic information in the future). Thus, in MunderLine we are using 12 columns instead of 10 as used in CONLL-U. Here is an example output for the English sentence “Angela Merkel works in Berlin.”

1	Angela	_	PROPN	_	Number=Sing	2	compound	_	B-PER	_	
2	Merkel	_	PROPN	_	Number=Sing	3	nsubj	_	L-PER	_	
3	works	_	VERB	_	Mood=Ind|Number=Sing|Person=3|Tense=Pres|VerbForm=Fin	0	root	_	O	_	
4	in	_	ADP	_	_	5	case	_	_	_	
5	Berlin	_	PROPN	_	Number=Sing	3	obl	_	U-LOC	_	
6	.	_	PUNCT	_	_	3	punct	_	_	_	

For example, the computed information for the token “Angela” can be read as: The token id is 1, the word form is “Angela”, the universal part of speech tag is PROPN (proper noun), the morphological feature is Number=Sing (singular form of number feature), the head of “Angela” is the token with id 2 (which is basically “Merkel”), the grammatical relation between “Angela” and “Merkel” is compound, and the NE tag is B-PER, which means, that “Angela” is the beginning token of a multi-word NE entity of type person (in the example this is the token sequence “Angela Merkel”).
The whole CONLL-U output of this sentence basically represents uniformly the POS tag sequence of a sentence, the Morph sequence, NER sequence and the dependency structure of the sentence, which is formally encoding as a labeled directed acyclic graph (dag).

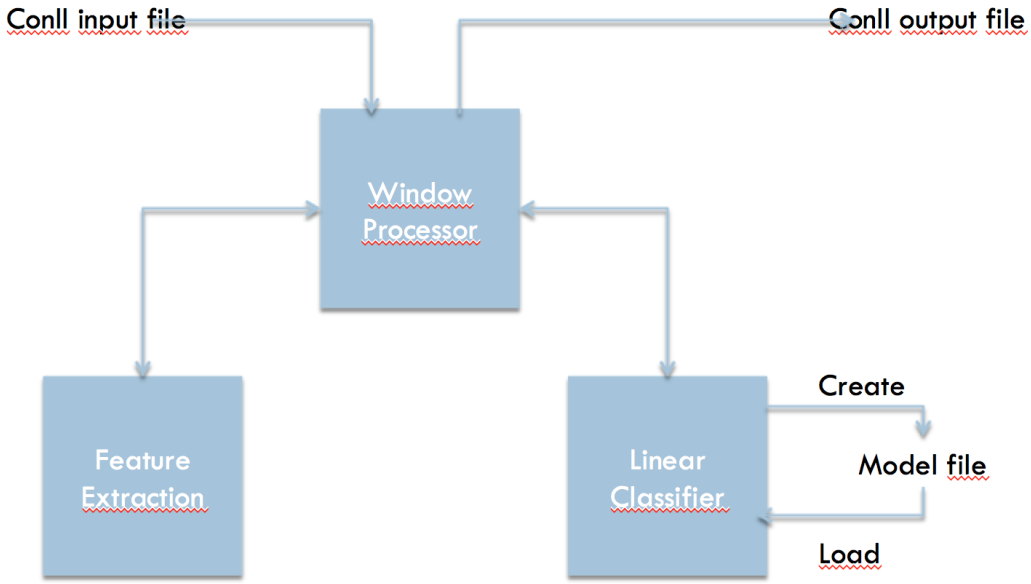
One important property of the CONLL-U format is that it is currently the standard annotation format for defining and implementing dependency treebanks for many languages cf. http://universaldependencies.org/. It is a framework for cross-linguistically consistent grammatical annotation and an open community effort with over 200 contributors producing more than 100 treebanks in over 60 languages.

3.2 GNT

A major motivation for the development of GNT was to provide a lightweight, uniform and fast classifier environment for a number of different classification tasks. As part of the MunderLine pipeline we are using GNT for learning classifier models for part-of-speech tagging,

morphological feature tagging and NER tagging. However, we also have recently used a variant of GNT for extracting and classifying relations (Tyler & Neumann, 2018). GNT was developed at the LT-lab of DFKI. As core Machine Learning backbone we are using the LibLinear tool, which is a simple package for solving large-scale regularized linear classification and regression. It currently supports a number of specific linear classifiers, e.g., classifiers based on Support Vector Machines. LibLinear provides easy to use interfaces and provides very fast training and prediction.

We are using LibLinear to define a token-level and window-based feature extraction architecture, which is roughly displayed in Figure 2. Given a dependency Treebank containing n tokens, we create for each CONLL token a window of size l, which results in a set of n windows. A window contains of l left and l right tokens of the current token x_i, which gives us n windows of form (w_i-l, …, w_i, ..., w_i+l). For example, for l=2, we have windows of form (w_i-2, w_i-1, w_i, w_i+1, w_i+2). If we have sentence boundaries, we are using so called padding elements for start/end positions of sentences. For example, for the token sequence “Der Mann sieht die Frau .” (‘ The man sees the woman.”), we obtain windows like (PAD, PAD, Der, Mann, sieht), (PAD, Der, Mann, sieht, die), (Der, Mann, sieht, die Frau), and (die, Frau, ., PAD, PAD). Each window is then encoded as a feature vector by defining a feature F for each token of the window by combining k different basic features f_k(w_j) of each window element f(w_j): F(w_j) = f_1(w_j) + ... + f_k(w_j). The whole feature vector of a window is then a concatenation of all feature vectors of the window elements. The feature vectors are then passed as input to LibLinear to perform either training of a model or the application of an already trained model.



1

Figure 2: Structure of the general tagger GNT

A particular property of GNT is that we are using the same set of basic feature functions f_k(w) for all tagging tasks in the MunderLine pipeline. In particular we are using a small set of lightweight features, which are directly extracted from the token string and one feature that is based on external knowledge. Beside this, we are using an additional feature that is based on

predicted labels of the window elements left from the window center element. More precisely, we define:

- f_suffix(x): This feature function computes all possible suffixes of a word and uses each one as an individual feature.
- f_shape(w): This feature determines shape information of a token and encodes it in form of a binary string that is then used as a feature. Shape information covers properties like uppercase of characters, subgroups of characters etc.
- f_cluster(w): This feature function assumes the existence of a word cluster dictionary and uses the cluster id of a word as feature.
- f_left_context_label(w): This feature functions uses the predicted label of a left adjacent word as feature.

Each individual basic feature can be switched off and on, which supports training of different classifiers using different combinations of basic features.

3.3 MDP

MDP is a transition-based parsing engine and it is based on a slightly modified version of Covington's parsing strategy. The module responsible for choosing the best operation in every step is called oracle. During the training phase the perfect oracle is simulated by using the training data in such a way that MDP is directly controlled by the already given gold standard result (the dependency tree of a training example). During this phase the system learns which operation is likely to be chosen in which situation. These situations depend on the current state of the system and its auxiliary data structures, and are called configurations. The result is a model, which is then used to make predictions about the most likely action, when the gold standard result is not available, which happens in the application phase. Figure 3 summaries the information flow of the two phases.

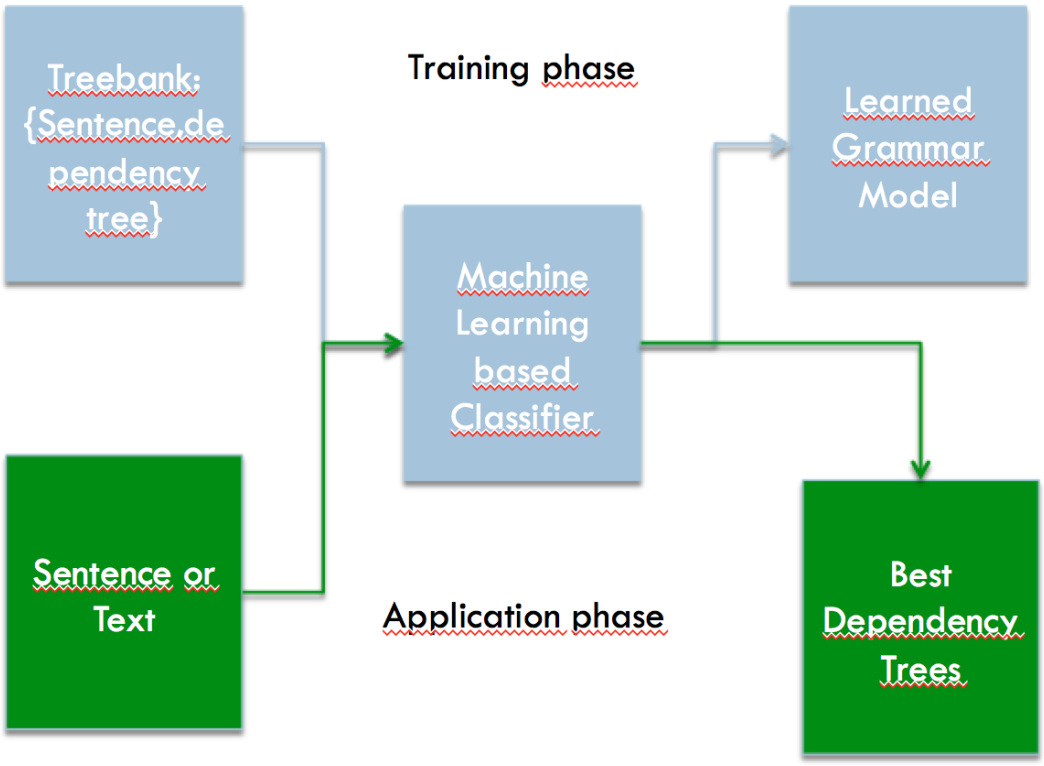


Figure 3: Structure of the MDP parser engine

The whole architecture of MDP is similar to GNT such that training and application data is given in CONLL-U format in form of dependency treebanks and that LibLinear is used to learn the classifier models for MDP and the specific languages considered. The major differences between GNT and MDP are that MDP uses a parser-based approach to extract and apply features instead of the window-based approach used in GNT, and that the nature and source of the features are different. Note that the major goal of the parser (and hence of the oracle) is to decide for a given pair of words from a sentence, whether to link them or not (called shift), and if they should be linked the direction of the link has to be determined as well as the label of the link. The different possibilities are considered as configurations of parser transitions or states, and hence, the sequence of configurations determined during the parsing step of a sentence defines the output dependency structure. The possible set of configurations can be considered as classes, and hence the main task for LibLinear is to learn combinations of features for predicting the best next configuration to use.

MDP parses the sequence of tokens of a sentence from left to right, and hence, also the best sequence of configurations is determined following this direction. During the parsing step, for each current position j, all possible word pairs (j,i) are determined (where 0 <= i <= (j-1)), and checked which configuration should be best applied. This check consists of two steps: first it is checked whether it makes sense to apply a configuration for a candidate word pair based on conditions of well-formedness of dependency trees. For example, a valid dependency tree is one where each modifier has only one head. So, if linking the current word pair would falsify this condition for the currently computed partial dependency tree, it would not make sense to predict a link (and consequently we apply a shift). Thus only word pairs that are proven to be permissible are further considered in the current step. This second step consists of creating a

feature vector representation for the current word pair and its context. In the training phase this feature vector is combined with the known configuration, and in the application phase the configuration is predicted using an existing model. Following a similar approach as known from GNT, a feature vector also consists of a conjunction of basic features. In MDP specific basic features are generated by applying predefined feature templates. As part of MDP we have defined about 27 different templates that are used for all languages. For example, pj is the name of a template, that when applied to a current word pair (j,i), returns the part of speech of the token j. See (Volokh & Neumann, 2011, 2012) for more details.

3.4 PERFORMANCE OF IREAD MODELS

We used the MunderLine pipeline for training and testing language models for the following linguistic levels:

1. Part-of-speech (POS)
2. Morphology (Morph)
3. Dependency structure (UD)
4. Named Entities (NER)

For all relevant iREAD languages – English, German, Greek, Spain – we have used existing dependency treebanks of the universal dependency version 2.0, cf. http://universaldependencies.org/ to obtain models for the first 3 levels mentioned above. For training and testing models for NE, we used existing NE annotated data from the CoNLL 2003 challenge (https://www.clips.uantwerpen.be/conll2003/ner/), mainly because the universal treebanks do not contain annotations of NEs. However, in that case, only data for English, German and Spain are available. We are not aware of any publicable available NE corpus for Greek.

The universal dependency treebanks for the different languages are using the same tagging inventory for the all three levels (POS, Morph, UD), but differ in size. In all cases, the treebanks are split into three data sets: train, dev, test. The following table summarises the size of each data set for each iREAD language (number of annotated words)

Language# words	Train	Dev	Test	Total
English	204,585	25,148	25,096	254,829
German	269,637	12,348	16,268	298,253
Greek	46,904	11,428	11,871	70,203
Spain	382,455	37,153	12,000	431,608

We are using this data split for training and testing in order to be able to compare our results with those reported by others. This means that we train all models only on the train set, and do testing of the models only on the test data. In principle, it would also be possible to extend the training set by adding the development set and then do the testing, which would mean that we would have larger training sets. However, we decided to follow the standard training and testing procedure also followed by others as close as possible in order to be able to compare our results with others, cf. (CoNLL, 2017).

Training and testing of POS, Morph using GNT

We used the same settings and feature sets for all languages, i.e., window size l=2, suffix feature and shape feature. As we can see, this is a very basic and lightweight use of features, because it means that we are only using features that we can directly extract from the training data, and no external knowledge is used. So, we can consider this as a true baseline case for GNT and its performance on the different tasks. Note that we do not use the word cluster features, because currently we only have them for English and German.

Currently, we obtain the following results for the test data. The next tables show the result of GNT for the POS and Morph tagging task. Note that we use the output of the MunderLine evaluation script directly. Acc means accuracy of all tokens per language and averaged on all languages; OOV refers to the out of vocabulary accuracy, that is the accuracy of words which have not been seen during training; INV refers to the accuracy of the words seen during training phase.

POS tagging result: Complete testing for 4 languages:

Lang	Acc	OOV	INV
English	92.72	73.87	94.61
German	91.55	84.85	92.45
Greek	95.26	82.39	97.93
Spanish	95.06	81.35	96.14
Avg	93.65	80.62	95.28

POS tagging result: Complete testing for 4 languages:

Lang	Acc	OOV	INV
English	94.21	81.81	95.46
German	81.95	66.49	84.02
Greek	89.17	73.62	92.39
Spanish	96.08	84.32	97.01
Avg	90.35	76.56	92.22

Compared to other published results this is very promising, cf. (CoNLL, 2017). Note that we have a very lightweight approach and that we are using neither language specific external knowledge sources nor any language specific parameter tuning.

Beside this positive performance, GNT also showed a very promising run-time behavior. For example, POS tagging can process between 91909 words/second (for English) and 150000 words/second (for Spanish). Morph tagging may process between 51644 words/second and

86537 words/second. The differences in accuracy and run-time between POS and Morph tagging are mainly due to the fact that the tag set for POS is smaller than that of Morph.

Training and testing of UD using MDP

Also for training the models for the different syntactic universal dependency models, we used the same settings for all languages in order to obtain a kind of language neutral baseline. The next table shows the result of the MDP parser on the test sets of the different languages. UAS means Unlabeled Accuracies and refers to the number of correct unlabeled dependency relations, whereas LAS refers to the number of correct labeled dependency relations. The number of classes for the latter case is much higher than the number of the unlabeled case, and hence, the performance for LAS is usually lower than for UAS.

UD tagging result: Complete testing for 4 languages:

Lang	UAS	LAS
English	83.75	81.13
German	76.46	71.24
Greek	83.12	79.85
Spanish	84.54	81.09
Avg	81.97	78.33

The results are also very promising if we remind us that we use no external knowledge sources other than the training data and no language specific tuning, cf. (CoNLL, 2017).

Training and testing of NE using GNT

We also have started implementing statistical models for Named Entity recognition using our general GNT tagger. We are using basically the same system and feature set as for POS and Morph tagging with the notable difference that only for NE tagging we are using the context feature. Remember that this feature uses the predicted label of a left context word as feature for predicting the label of the current word. The main reason for defining the feature is that it helps ensuring that consistent sequences of labels are predicted. This makes certainly sense for NE tagging because in general a NE consists of a sequence of tokens. We have run initial training and testing of GNT for NE tagging using the standard CoNLL-2003 NE tag set, cf. (https://www.clips.uantwerpen.be/conll2003/ner/).

They provide training, development and test data for English and German, and hence, we report our current results just for these languages. Using only the training data for inducing NE based models, we obtain:
For English: Development: F1=91.75 %, Test: F1=87.17% and
For German: Development: F1=79.8%, Test: F1=78.2%

These results are promising, although not optimal, since we are about 4,5% F1 behind the best reported results for the test data sets, cf. (Chiu & Nichols, 2016). But compared to these more

complex systems (using complex deep neural architecture and external word embeddings and word lists), we are using a rather simple baseline approach, and hence, are convinced we will be able to improve our approach in near future.

3.5 MUNDERLINE: TECHNICAL ASPECTS

MunderLine is completely developed and implemented in Java and packaged using the Maven tool. MunderLine can run from shell or as a server using REST API interface.

Configuration files exist for the pipeline configuration and the server configuration. The MunderLine configuration in `*_pipeline.conf` contains the pipeline definition as well as pointers to the models to be used by the GNTagger and MDParser instances of the pipeline. The MunderLine server configuration in `server.conf` contains pairs of languages and MunderLine configurations. The GNTagger configuration in `gnt.conf` only contains settings for a training tagger model. It is not used when running MunderLine. `GNT_*.conf` configuration files are also only required in training. The MDParser configuration in `mdp.conf` mostly contains settings for training a model. The only setting relevant for runtime is `parsing.threads` which sets the numbers of threads to be used when parsing.

MunderLine runs as a REST service that accepts form-data inside the body of an HTTP POST request that is send to http://iread.dfki.de/munderline/<lang>. Replace <lang> with one of the following language ids to use language specific models:

English: en
German: de
Spanish: es
Greek: el

Replace <lang> with one of the following language ids to use universal dependency models:

English: en_ud
German: de_ud
Spanish: es_ud
Greek: el_ud

The server can be tested using the curl tool. For analyzing some short text with MunderLine, run

```
curl -X POST -H "Content-Type: application/x-www-form-urlencoded; charset=utf-8" \
--data-urlencode input="This is a test" \
http://iread.dfki.de/munderline/en
```

For analyzing a plain text file with MunderLine, run

```
curl -X POST -H "Content-Type: application/x-www-form-urlencoded; charset=utf-8" \
--data-urlencode input@my_input_text.txt \
http://iread.dfki.de/munderline/en
```

For analyzing a plain text file with one sentence per line with MunderLine, run

```
curl -X POST -H "Content-Type: application/x-www-form-urlencoded; charset=utf-8" \
--data-urlencode input@my_sentences.txt \
--data "linewise=true" \
http://iread.dfki.de/munderline/en
```

How to run MunderLine from shell or how to start your own server is described in a README.md file that comes with the software package.

The MunderLine package also includes a client in form of a Java class that receives a string (or list of strings) encoding the dependency analysis of a sentence in form of a CONLL-U annotation and returns a two dimensional Java array that can be used within Java to further process the results.

4. NEXT STEPS

The next steps include improving the performance of the current language models, especially the dependency model. We already have started to create a new dependency Treebank for German containing about 1.000 sentences from children texts. We will use this corpus for testing and improving the current models on documents from the iREAD domain. We are also planning to add additional features in form of word clusters, word embeddings and lemma information.
In order to increase robustness, we will also explore trainable tokenizers for all languages starting from the tokenized universal dependency treebanks.

5. CONNECTION TO WP 5 AND THE IREAD PROJECT

Knowble as the WP5 leader has tested and confirmed acceptance of the syntactic parsers technology. Knowble will in the next stage integrate the syntactic parsers into the Content Classification Component from deliverable 5.2.

More specifically, these syntactic parser will be used to provide input for D5.2 Content Classification Component. This Content Classification Component will classify the syntactically parsed content based on the user-model driven content classification metrics provided by D5.3, which is informed by work of the work package 4 - user modelling, adaptivity and learning analytics. The user-model driven content matrix heavily rely on syntactic information to evaluate content difficulty.

6. CONCLUSIONS

We introduced and described MunderLine, a multilingual dependency based pipeline for syntactic analysis in iREAD. The pipeline was developed as part of WP 5, subtask 5.3 and consists of language adaptive tools for preprocessing, POS tagging, Morph tagging, NE tagging and dependency parsing using universal dependencies. MunderLine has been trained and tested for all relevant languages of the iREAD project and shows promising results with respect to performance and speed. We will continue working on MunderLine during the cause of the project to further increase its performance and robustness.

7. REFERENCES

CoNLL (2017): Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies. August 3-4, 2017, Vancouver, Canada, http://universaldependencies.org/conll17/proceedings/K17-3.pdf

Jason P. C. Chiu, Eric Nichols (2016): Named Entity Recognition with Bidirectional LSTM-CNNs. Transactions of the Association for Computational Linguistics, Volume 4, 357-370.

R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin (2008): LIBLINEAR: A library for large linear classification Journal of Machine Learning Research 9(2008), 1871-1874.

Günter Neumann, Gerhard Paaß, and David van den Akker (2014): Linguistics to Structure Unstructured Information. Towards the Internet of Services: The THESEUS Program, in Wolfgang Wahlster; Hans-Joachim Grallert; Stefan Wess; Hermann Friedrich; Thomas Widenka (eds), Springer International Publishing Switzerland, ISBN 978-3-319-06755-1, pp. 383-392, 2014.

Ratinov, L., & Roth, D. (2009): Design challenges and misconceptions in named entity recognition.

Tyler Renslow and Günter Neumann (2018): LightRel at SemEval-2018 Task 7: Lightweight, Fast and Robust Relation Classification. In proceedings of SemEval-2018 - International Workshop on Semantic Evaluation, June, 2018.

Alexander Volokh and Günter Neumann (2011): Automatic Detection and Correction of Errors in Dependency Treebanks. The 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies (ACL).

Alexander Volokh and Günter Neumann (2012): Transition-based Dependency Parsing with Efficient Feature Extraction, 35th German Conference on Artificial Intelligence (KI-2012), Saarbrücken, Germany, September, 2012.