RACAI's Natural Language Processing pipeline for Universal Dependencies

Stefan Daniel Dumitrescu, Tiberiu Boros and Dan Tufis
Research Institute for Artificial
Intelligence, Romanian Academy
Bucharest, Romania
sdumitrescu@racai.ro, tibi@racai.ro, tufis@racai.ro

Abstract

This paper presents RACAI's approach, experiments and results at *CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*. We handle raw text and we cover tokenization, sentence splitting, word segmentation, tagging, lemmatization and parsing. All results are reported under strict training, development and testing conditions, in which the corpora provided for the shared tasks is used "as is", without any modifications to the composition of the train and development sets.

1 Introduction

This paper describes RACAI's entry for the CONLL Shared Task on Universal Dependencies parsing. We represent the Research Institute for Artificial Intelligence, in Bucharest, Romania. The shared task refers to processing raw text with the goal of automatically inferring word dependencies. While some approaches require only segmented (tokenized) text, parsing methods that depend on rich feature sets (which is our case), implicitly require that the text is tokenized, POS tagged and lemmatized. The Universal Dependencies (UD) corpus (Nivre et al., 2016, 2017a) uses 3 distinct layers of analysis: (a) a Universal Part-of-Speech layer (UPOS) (Petrov et al., 2011); (b) a language specific part-of-speech layer (XPOS) and (c) a list of language-dependent morphological attributes Zeman (2008). Also, in the current version of UD, tokenization and word-segmentation require different handling strategies (see section 3.2 for details). In what follows we will provide an overview of our system's architecture (section 2) and a detailed description of each module (section 3) used in our processing pipeline, followed by its evaluation (section 4) (Nivre et al., 2017b) on the TIRA platform (Potthast et al., 2014). Though Syntaxnet (Weiss et al., 2015) models were also available, some discrepancies in the token and word-segmentation methodology made the comparison impossible (mainly because the Syntaxnet's output was incompatible with the evaluation script). This work is focused on presenting our system's technical details and individual results. The full comparison between competing systems, as well as the baseline values obtained by UDPipe v1.1 (Straka et al., 2016) are available in Zeman et al. (2017).

2 System Architecture

Figure 1 presents a bird's eye view on the individual modules of the system and how they interconnect.

At runtime, the input of the system is a raw text file. As the file is not sentence split, all new line characters are removed and passed as a single long string to the first module: the **Tokenization and Sentence Splitting (Tok/SS)** (section 3.1). Depending on language, some files are already tokenized, for which we perform only sentence splitting; however, for most languages, we perform both tokenization and sentence-splitting in a single pass. At this point, we obtain a file in the conllu format, passed to the **Compound Word** module (section 3.2). This module looks for words that can (and should) be split in two or more tokens. For example, in German "im" becomes "in dem", or in Polish: "kupiłbym" becomes "kupił by m". Compound words are added as new lines in the conllu file. Next, language independent part of speech (UPOS) tags are added by the **Tag.UPOS** module (section 3.3). Based on the available information so far, the following modules all run in parallel: The **Lemma** (section 3.4), **Tag.XPOS**

Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies, pages 174–181,
Vancouver, Canada, August 3-4, 2017.

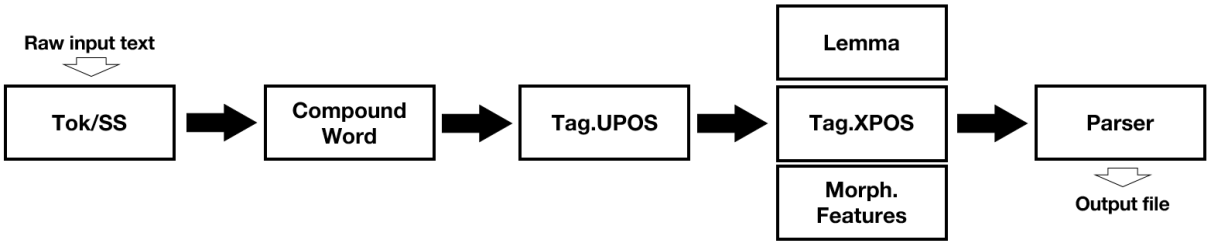

Figure 1: System architecture

and **Morphological Features** modules add lemmas, XPOSes and Morphological Features. The XPOS and Morphological Features modules are language dependent and are described together in section 3.5. Finally, the **Parser** module (section 3.6) adds the final dependencies in the output conllu file.

For the training process we used the available training data in the conllu format. The conllu files provided by the organizers contained "detokenization" information (a SpaceAfter=No flag added to every token that in the original text had no space) and morphological analysis layers: word, lemmas, part-of-speech, language-dependent part-of-speech, morphological attributes and word dependencies (index of the head of each word, as well as the relationship type).

Further details about each information layer will be provided later in the paper, when we introduce the individual processing modules and describe our feature extraction and labeling strategies. During training we use a separate script responsible for preparing the custom training and development sets. Depending on the memory and CPU requirements we trained either the models sequentially (e.g. the parser is memory expensive and the linear classifier for part-of-speech tagging is multi-threaded) or multiple models in parallel (the morphological analysis require far less memory and CPU time - practically the decision trees for all languages in the competition were built and pruned in parallel).

3 System Description

Before we proceed with the description of the modules we must note that some of our methods rely on decision trees (DT) that are built using a custom designed algorithm that relies on the constituency matrix to speed-up computation for the Information Gain (IG) (Equation 1). Also, after the initial trees are built, we use the available development sets to prune them to reduce possible train-data overfitting, leading to improved performance.

$$IG_i(S) = H(S) - \sum_{t \in T}(P(t) \cdot H(t)) \quad (1)$$

where i represents an input feature, $H(S)$ (Equation 2) is the entropy of the initial set S, $H(t)$ the entropy of subset t, and $P(t)$ is the fraction of elements in t over the entire $|S|$. We note $|S|$ as the number of instances in S.

$$H(S) = -\sum_{x=1}^{N} P_x \cdot \log_2 P_x \quad (2)$$

We note this because the above mentioned methodology was not presented elsewhere and we feel the it is an important aspect of our approach (see section 4 for details regarding the model sizes and section 5 for comments).

3.1 Tokenization and Sentence Splitting

The first module in the pipeline is the Tokenizer and Sentence Splitter. Depending on the training data, we actually have 4 distinct tokenizers/sentence splitters merged into our module: the standard Tok/SS for corpora that had both punctuation and was not already tokenized (this is the case for most languages), the character level Tok/SS for Japanese and Chinese, and two versions of the sentence splitter for training data with pre-tokenized sentences that had (e.g. Danish, Finish FTB and Slovene SST), and had not punctuation (e.g. Gothic, Latin Proiel, Ancient Greek Proiel and Old Church Slavonic).

The main tokenizer and sentence splitter is based on DTs. Tokenization and sentence splitting are independent models, and are run in parallel. Based on a number of features (described below), the DT tokenizer model chooses between 4 classes: SPLIT_LEFT, SPLIT_RIGHT,

SPLIT_LEFTRIGHT and NONE, meaning it should split to the left of the current character, right, to put two spaces around the character or not make a split. The DT sentence splitter has only 2 classes: SPLIT and NONE. Training is done in the following manner: (1) initially, we look for specific characters where we might have a word split, marked in the conllu train file by "SpaceAfter=No"; we then normalize the frequency of these characters; iterating again on the training data, we choose only the character that is most probable to initiate a split, based on the normalized frequency; for sentence splitting we perform the same process, the only difference being we pre-seed the character list with a number of punctuation characters like: *.-!?* etc., because we had cases where *?* for example was not frequent enough to remain in the split list, though it was a valid sentence splitter; (2) we extracted the following features for each split character: current letter, 3 characters before and after (4 for sentence splitting), and for the previous and next words a marker whether or not the word is punctuation only, if it ends with punctuation, if it contains punctuation, if it is uppercase, if it is capitalized and the number of periods in the word; (3) we trained the DT model and pruned the tree based on the dev set. Another small optimization worth mentioning is that we replaced all digits with zeroes to reduce variability in the training data.

The symbol based tokenizer is targeted for Japanese and Chinese where we have to look after each symbol and decide whether to split. The features are simply the current symbol and one symbol to the left and right, with two possible classes: SPLIT (split after the current symbol) and NONE.

The last two Tok/SS address languages that are already tokenized, performing only sentence splitting. For the languages that had no punctuation (e.g. Gothic), the features are 5 characters to the left and 3 to the right (including the current character). For the languages that had punctuation (e.g. Danish) the features are token based: the current token plus 2 tokens before and after; for each token we mark all the features the main tokenizer has for words (if the token is punctuation only, etc.)

Overall, we compared our results on the dev sets against UDPipe's, obtaining good results. However, for a few languages it seems that the decision tree approach is not optimal, yielding low performances. For English, Bulgarian, Korean, Portuguese and other 14 smaller languages, at runtime we directly used the UDPipe tokenization and sentence split conllu files, bypassing this module. This is the only place in the system where we use data that was not generated by us.

3.2 Compound Words

The compound words module has the task of word expansion. For example, in German "*im*" is the contraction of "*in dem*"; in Turkish, "*muhabbetliydi*" is for "*muhabbet li ydi*". While word expansion could be relatively well solved by using a dictionary (search for the key and replace with expansion tokens), it would fail for unseen words as well as ignoring split-no split decisions depending on POS tags. Our intuition was that we need to represent generated word expansions as parts of the original string (longest common substring or LCS) plus new terminations, either before and/or after the LCS. For example, currency tokens like "*7000€*" should always expand as the first variable part plus the last symbol separately if that last symbol is "*€*", while the opposite example "*€7000*" should be represented as a static first symbol followed by a variable string (here the numeric amount).

Needless to say, the process of token decomposition into words carries a great weight over the accuracy of the system, because all other modules depend on it: tagging, morphological analysis and parsing. Obviously preserving the head or tail of the original token and concatenating strings at the beginning or end requires different labeling strategies, because any of the words in the decomposition can be written either by keeping the head of the original token and concatenating a suffix or by a prefix and concatenating the tail of the original token. More often than not, one strategy will likely yield a larger number of unique output labels in the training data than the other. As such, the actual difficulty is determining which labeling strategy would be more accurate. We attempted to determine the labeling strategy as follows:

- First, take the training data and generate output labels using all (desired) tagging strategies, generating multiple label sets;
- Then, measure the system entropy for each of the previously generated label sets;
- Finally, use the tagging strategy which gen-

erated the lowest-entropy system for training the classifier.

In our approach the labeling scheme was in the form of "*n*+<*string*>" where *n* is a number and <*string*> is a string to keep. Consider the following example, where *a* is the word that will be expanded in two words *b c*. There are four different output encodings we can choose from: FS_KS, FS_KE, FE_KS and FE_KE. FS means "from start" and denotes that the number n is an **absolute index**, which measures the character span from the beginning of the word, FE is "from end" and denotes that n is a **relative index** and it will express a character span relative to the size of the original token. The meaning of KS is "keep start" and means that the head of the token should be preserved, while KE stands for "keep end" and means that the tail of the original token will be used in building the decomposed token. Now, suppose we write word *a* as letters $a_1a_2a_3a_4a_5$ and its first expansion *b* as $b_1b_2b_3b_4$, and a_{1-2} is equal to b_{1-2} and a_5 is equal to b_4. To obtain the first encoding FS_KS we find the longest common substring *from start* of *a* and *b* which is b_{1-2}, of length 2, keeping the rest of b; so, the FS_KS label encoding is $2+ < b_{3-4} >$. Encoding FS_KE means finding the longest common substring from start, while keeping the end: $3 + b_{4-n-3}$.

Our algorithm has to choose between FS and FE labeling schemes, the decision between KS and KE subordinated depending only on the LCS criterion (for each word in the decomposition, we chose to keep the head or tail of the original token depending on which would provide a higher character overlapping).

After determining the best labeling strategy we generated a decision tree using the following features: first four letters, last four letters, wordform (if occurrence frequency was higher than 10 in the training data).

To show how important choosing the appropriate labeling strategy is, on the Hebrew development set we obtained 93% F-score using the FE notation (automatically selected by the system), versus 88% when we forced the system to use FS.

3.3 POS Tagging

The part-of-speech inventory used in this step refers to UPOS tags, an inventory which contains only 17 unique labels. Our tagging methodology is fairly standard: we use a Conditional Random Field to estimate the probability of the i-th tag (t_i), based on the previous tag and a rich set of features (f_i) (Equation 3). During runtime, we use Viterbi to obtain the optimal sequence of labels.

$$P(t_i|t_{i-1}, f_i) \tag{3}$$

The set of features is composed of (a) the lower-cased wordform, (b) a large number of letter n-grams, with n ranging from 2 to 5 and (c) a feature which we refer to as "writing style". All the features are extracted from a window of 3 words (centered on the current word). The "writing style" feature takes 4 values:

- ALL-CAPS - the word is written in CAPS;
- ALL-LOWER - the word contains only lower-cased symbols;
- F-UPPER - the word starts with a capital letter and all other symbols are lower-cased;
- F-UPPER-START - similar with F-UPPER, only this time the word is also the first token of the sentence.

We use a large number of character combinations (90), which includes **cross-word letter n-grams** and was manually obtained using a trial-and-error process.

To prevent overfitting and obtain a robust model for out-of-vocabulary (OOV) words, we only include a wordform as a feature, if that word's occurrence frequency is higher than a threshold (k) in the training data[1].

3.4 Lemmatizer

Lemmatization is done in two steps. First, the surface wordform and its UPOS (language independent part of speech) is searched in a dictionary created at train time. If the surface form & UPOS match, the corresponding lemma is used (if there is more than one lemma for the surface&UPOS pair, we prefer the most frequent). If not found, we attempt to create the new lemma using a DT. Given a word, we extract the UPOS, the first 4 letters and the last 4 as features. The first and last letters may overlap if the word is smaller than 8 letters, or can be null (encoded as "_") if the word is smaller than 4 letters. The output classes are

[1]In our experiments we observed that $k = 10$ is a good choice for many of the languages we used for tunning

Model	Type	Count	Min	Max	Average	St. Dev.
Tokenization	DT	57	0.66 KB	394 KB	18.09 KB	71.09 KB
Sentence Split	DT	64	0.91 KB	345 KB	23.90 KB	54.07 KB
Compound Words	DT	22	0.34 KB	484 KB	50.53 KB	126.68 KB
Lemmatization	DT	56	2.79 KB	370.53 KB	56.36 KB	68.23 KB
Tag.UPOS	CRF	64	2.97 MB	612 MB	121.26 MB	100.09 MB
Morph. Feats.	DT	64	10.46 KB	1.34 MB	170.21 KB	219.05 KB
Tag.XPOS	DT	64	10.64 KB	1.08 MB	111.73 KB	200.39 KB
Parsing	RBG	64	43.41 MB	3.16 GB	1.01 GB	755.5 MB

Table 1: Model types and sizes

strings looking like "*n*+<*string*>". The *n* represents how many letters to cut from the surface form of the word, and <*string*> means the string to append to the word. For example, given the word *forgotten*, which is a *Verb*, with features *f o r g* and *t t e n*, its output class would be *5*+*et*, meaning that we need to cut the last 5 letters (to obtain the largest common prefix) and add *et* to obtain the lemma *forget*.

We note that the number of output classes varies between a few hundreds to several thousands depending on the language, but, even for this large number, the results of the DT seem accurate.

3.5 Language Specific Morphological Analysis

Language-specific morphological analysis is a two-fold process that refers to the resolution of (a) the language-dependent part-of-speech (XPOS) tag and (b) a structured set of morphological attributes (in the form of key-value pairs), which are used to encode important information such as gender, number, case etc.

As a rule-of-thumb, the XPOS tag used in morphologically rich languages is a compact representation of the morphological attributes. For instance, the Romanian corpus from the UD data uses a standardized compact representation, which is composed of morphosyntactic descriptors (MSDs) (Erjavec, 2004). Given the similarities between language-specific tags and UD morphological-attributes, in our approach we used the same feature sets for both tasks. The features are composed of: (a) the UPOS tag, (b) the first four characters of the word; (c) the last four characters of the word and (d) the previously mentioned "writing-style" feature. To capture local-dependencies between words we used a context window of 5 centered on the current word.

In this case, we preferred to use decision trees, mainly because of reduced computational requirements and the small-footprint of the output models.

3.6 Parsing

Once the morphological analysis is completed, our processing pipeline relies on RBGParser[2] which is a greedy hill-climbing parser, well described in Zhang et al. (2014a,b); Lei et al. (2014). In our approach, we used branch 1.1.2 of RBG, which we modified in order to be compliant with the current UD version.

The main incompatibility was generated by the presence of multiword tokens. During training we modified the data adapter of the RBGParser to skip multiword tokens and, for the runtime version, we filtered the input for RBG to exclude multiword tokens and we re-aligned the output of RBG with the unparsed dataset, to restore multiword tokens and provide an output compatible with the current UD standard.

The RBG models were built using the default parameters for the "standardModel" predefined configuration, on which we added automatically extracted word embeddings (Mikolov et al., 2013), obtained using word2vec[3]. The word embeddings were computed by applying the Continuous Bag of Words (CBOW) model on the permitted raw-text resources.

Depending on the language, for the computation of word vectors we compiled monolithic corpora composed of Wikipedia Dumps (whenever available) and raw text from UD training.

[2]https://github.com/taolei87/RBGParser - accessed 2017-05-24

[3]https://github.com/dav/word2vec - accessed 2017-05-24

Language	Tok	SS	Words	Lemma	UPOS	XPOS	Morpho	UAS	LAS
ar	99.98	60.50	95.50	85.51	90.01	81.96	80.76	76.35	69.32
ar_pud	80.81	98.80	93.34	0	73.54	0	0	59.80	48.73
bg	99.91	92.83	99.91	92.14	97.20	91.45	90.00	87.66	82.47
bxr	99.35	91.81	99.35	81.40	84.12	99.35	78.08	38.46	21.66
ca	99.96	98.95	99.95	84.26	97.60	97.60	95.46	87.98	83.94
cs	99.96	82.83	99.96	95.03	98.07	88.31	85.15	85.97	81.14
cs_cac	100.00	100.00	99.97	96.27	98.38	85.22	81.94	87.18	81.95
cs_cltt	99.82	80.14	99.820	94.56	95.78	83.24	83.00	76.17	68.36
cs_pud	97.98	93.23	97.97	92.00	95.40	83.93	80.01	84.13	77.71
cu	99.96	36.05	99.96	86.38	94.06	94.28	80.76	74.04	67.12
da	100.00	76.85	100.00	94.14	94.62	100.00	90.46	77.15	72.29
de	99.44	76.80	99.45	90.82	91.02	91.27	71.30	76.21	69.14
de_pud	96.72	89.87	96.47	2.71	83.12	19.45	1.36	74.65	65.24
el	99.83	87.56	99.83	91.78	95.78	95.78	84.98	84.03	79.08
en	98.67	73.22	98.67	93.43	92.76	91.67	89.75	78.34	74.44
en_lines	99.92	82.11	99.92	88.36	93.86	90.90	88.98	76.84	70.97
en_partut	99.51	97.51	99.49	94.81	92.93	92.60	90.22	78.11	72.69
en_pud	99.66	97.13	99.66	95.00	93.65	91.81	88.36	82.59	77.79
es	99.94	94.17	99.77	94.17	95.21	99.75	92.04	84.43	79.97
es_ancora	99.95	98.06	99.93	78.56	97.60	97.60	95.56	86.12	82.07
es_pud	99.53	95.27	99.36	3.32	87.55	1.70	0	84.60	76.64
et	99.85	92.53	99.85	79.28	87.53	89.48	77.32	68.69	58.74
eu	99.98	99.83	99.98	88.89	92.44	99.98	80.72	77.31	70.40
fa	99.99	98.01	99.34	96.58	94.92	94.50	93.75	82.69	77.45
fi	99.46	86.05	99.46	81.87	92.56	93.77	85.05	78.23	72.59
fi_ftb	99.90	83.83	99.87	85.58	89.77	85.74	82.94	78.21	72.09
fi_pud	99.39	91.91	99.39	81.06	93.74	0.01	0.01	80.85	75.44
fr	99.72	92.12	98.85	95.41	94.95	98.85	92.18	82.64	78.38
fr_partut	99.75	99.12	98.89	92.86	94.30	90.15	84.69	81.42	76.75
fr_pud	97.43	91.71	96.62	4.82	87.23	2.41	0	77.50	72.18
fr_sequoia	99.77	83.75	99.14	95.66	95.41	99.14	91.94	81.52	77.64
ga	99.73	96.69	99.73	83.33	88.95	83.25	65.64	76.27	65.65
gl	99.92	95.92	99.92	96.94	96.11	95.15	94.88	82.43	78.34
gl_treegal	98.91	84.80	97.97	90.94	90.51	85.60	83.99	71.74	65.22
got	100.00	27.85	100.00	88.80	93.39	93.13	80.18	70.18	62.30
grc	99.98	98.70	99.98	72.35	86.48	72.68	71.44	68.53	60.48
grc_proiel	100.00	43.11	100.00	90.17	95.70	96.03	83.01	74.89	69.47
he	99.98	100.00	88.55	75.64	83.24	83.24	78.17	66.23	60.72
hi	100.00	99.20	100.00	87.09	95.94	95.05	84.93	89.90	85.33
hi_pud	92.27	94.45	92.27	0	78.12	33.23	4.67	57.33	45.57
hr	99.84	95.91	99.84	93.36	95.80	99.84	79.17	84.13	77.03
hsb	99.84	90.69	99.84	87.70	90.30	99.84	72.43	54.24	43.74
hu	99.70	89.75	99.70	80.02	90.52	99.70	67.59	72.34	64.76
id	99.99	88.41	99.99	85.47	92.71	99.99	92.67	80.89	73.86
it	99.92	98.45	99.82	89.40	96.56	95.74	94.19	87.98	84.45
it_pud	99.60	94.56	99.16	88.22	92.50	2.47	2.47	86.73	82.48
ja	87.57	93.18	86.57	86.00	84.82	87.57	84.82	69.14	67.64
ja_pud	89.87	96.21	89.87	88.55	87.78	6.17	5.69	74.92	74.06
kk	94.52	81.50	93.89	51.48	56.49	54.94	35.81	43.01	29.22
kmr	99.01	97.02	98.85	89.76	90.04	89.84	80.62	32.05	14.73
ko	99.73	93.05	99.73	53.80	91.21	81.02	81.02	69.45	62.79
la	99.97	99.20	99.97	31.42	84.58	63.18	62.79	59.15	46.77
la_ittb	99.91	83.10	99.91	94.32	96.68	88.49	84.56	79.61	74.45
la_proiel	100.00	25.80	100.00	88.08	93.73	93.91	80.29	67.68	60.80
lv	99.30	93.30	99.30	84.86	88.62	71.51	69.55	68.54	60.08
nl	99.85	74.52	99.85	77.02	91.19	85.67	83.82	77.32	68.23
nl_lassy	99.93	78.62	99.93	97.13	96.88	99.93	93.82	84.05	79.54
no_bokmaal	99.88	93.11	99.88	95.55	96.04	99.88	90.91	83.84	79.84
no_nynorsk	99.85	91.23	99.85	93.30	95.51	99.85	90.47	82.33	77.83
pl	99.97	99.59	99.89	92.23	95.45	77.72	75.94	85.56	78.29
pt	99.64	89.79	99.49	94.05	96.01	73.24	71.81	85.81	81.92
pt_br	99.95	95.51	99.84	81.13	96.81	96.81	96.80	86.92	83.54
pt_pud	99.29	95.65	99.40	11.87	88.12	0	0	79.85	73.27
ro	99.54	92.60	99.54	95.90	96.02	94.11	93.88	85.75	79.44
ru	99.94	95.30	99.94	74.14	95.15	94.38	77.71	81.06	75.54
ru_pud	97.22	96.93	97.21	0	85.33	78.49	33.85	77.32	68.96
ru_syntag	99.51	98.01	99.51	93.55	97.70	99.51	89.04	89.41	85.41
sk	100.00	83.53	100.00	86.18	94.11	71.55	68.68	80.77	74.26
sl	99.96	99.24	99.96	93.95	96.24	83.08	80.99	84.67	79.61

Language	Tok	SS	Words	Lemma	UPOS	XPOS	Morpho	UAS	LAS
sl_sst	99.82	16.72	99.82	90.64	89.42	77.15	72.58	55.84	48.13
sme	99.88	98.79	99.82	81.86	86.81	88.98	77.76	47.51	35.47
sv	99.59	89.87	99.59	91.62	94.63	87.70	86.42	79.36	73.56
sv_lines	99.90	84.96	99.90	83.82	93.48	89.42	88.02	76.95	70.72
sv_pud	98.39	94.46	98.39	81.73	90.87	83.63	68.71	74.81	68.40
tr	99.76	95.77	97.52	84.83	89.30	87.31	76.01	64.09	55.74
tr_pud	97.61	91.79	95.10	0	68.66	0	0	53.78	33.20
ug	98.57	68.17	98.57	36.73	72.79	74.73	72.41	57.61	38.76
uk	99.65	94.25	99.65	85.16	88.58	65.35	61.26	72.45	63.54
ur	100.00	96.83	100.00	69.78	92.12	89.64	75.35	82.72	75.17
vi	82.47	92.59	82.47	78.18	71.23	50.35	50.25	39.98	34.19
zh	88.91	98.19	88.91	88.38	82.81	82.50	81.14	62.56	57.75

Table 2: Results per language

4 Evaluation

Table 1 presents the types of the individual models and their sizes. We use decision trees for most of the tasks, a CRF model for UPOS tagging and the models RBG Parser creates for the last task of parsing. We can directly see that the DT models are very small, even if they are written in text mode, with the largest average for morphological features of 170 KB. We also note that while there are 64 models for the 64 languages we had training data for, we only created 22 models for the languages that actually had compound words and 56 for lemmatization. The 57 tokenization models do not include the symbol tokenization models for Japanese and Chinese, which are even smaller; also, there were other languages that were pre-tokenized, so no models were created for them (details in section 3.1). The CRF models used for UPOS tagging are significantly larger, with the average of 121MB. Still, with a standard deviation of 100MB, we can say that most models are smaller than 250MB. The largest models are created by RBG, with the model for Czech reaching an impressive 3.16GB. On average, RBG creates models of around 1GB.

Moving on to the system results, we evaluate each task incrementally, starting from tokenization. We obtained a macro-averaged F1 score of 98.58, with a 0.37 difference to the first place. The decision tree approach used, while simple, brought interesting results, like first place for Czech (CLTT), Italian, Irish or Russian.

For sentence splitting, also based on DT, we obtained an average F1 score of 87.52 versus the top score of 89.10. We obtained first place on a number of languages like Hebrew, Basque or Latin. However on Latvian we obtained last place with an F1 of 93.30 versus 98.90. We note that no language-dependent tuning was performed, neither for tokenization nor for sentence splitting. The same features were chosen for all languages. While we did perform tree pruning based on the dev sets, we did not vary and choose the best feature set for each language (e.g. tokenization on some languages was better with a context of 2,2, while for others with context 4,1; we used 3,3 for every language that had punctuation and was not pre-tokenized).

In word segmentation we obtained a score of 98.39 vs 98.81, a difference of only 0.42 percent. Using the DT classifier brought us top places in several languages like Czech (CLTT), Danish, Norwegian (Bokmaal) or Russian.

Lemmatization, also based on decision trees, unfortunately worked really well on only a small number of languages. For example on Farsi we were the first with a 1.5 point difference over second place. Overall, we obtained an F1 score of 77.45 versus the top performer that obtained 83.74.

The morphological features average F1 score of 70.8 brings us relatively close to the top score of 73.92. Again, while not a best performer, the decision tree algorithm we used has shown very good performance compared to more complex algorithms in the competition.

On the language independent parts of speech (UPOSes) we obtained 90.71 vs 93.09. On language dependent parts of speech (XPOSes) we have an average F1 of 78.20 vs the top performer 82.27, a larger difference that for UPOSes.

Finally, on parsing we obtained an UAS of 74.67 vs 81.30 (11th place), and on LAS an F1 score of 67.71 vs 76.3, placing us on the 18th place.

5 Conclusions

The *CoNLL 2017 UD Shared Task* has been a learning experience for us. Considering that so far we only worked mostly on Romanian and English, and only up to the level of POS tagging, we managed to draw a number of conclusions, a few outlined below:

• while the DT algorithm has, on average, below state-of-the-art performance, it is very close to top performers. It sometimes achieves first place on tasks like tokenization or word expansion which follow a simpler and more predictable set of rules. We used a decision tree model because it is a predictable and understandable model, that, for this initial set of experiments allowed us to obtain significant insight on how we should create features and output labels, something that using a neural network would not allow.

• sticking with a method and trying out variations can lead to noticeable improvements. For example, pruning the character list on which to attempt a word split for tokenization based on normalized frequency yielded a more balanced training set. This has led to better results than simply asking whether to split on each character in the unpruned list (an unbalanced training set with most examples being "no split").

• sometimes intuition does not work. Initially, we hypothesized that for the morphological feature prediction task it was natural to attempted to predict each feature individually: we would predict, for every word irrespective of its part of speech, all available features separately. Each feature had an extra class of NONE meaning that it was not appropriate for that particular word, so it would not show up to the final composition of features. The results were actually significantly worse that trying to predict all features at once, as a single class output label, even if the number of such labels was much higher as it contained all combinations seen in the training data for morphological features.

As we viewed the *CoNLL 2017 UD Shared Task* as a learning experience, we attempted all tasks sequentially, even though the main goal of the challenge was the last task: parsing. The only place we used baseline UDPipe files was in the tokenization and sentence splitting where our decision tree approach with no tuning produced results significantly below the baseline. However, we kept our Tok/SS module

even for languages where we were 5 points below the baseline, to see what would be the results on the test data. That basically meant that any error in the initial task would be partially propagated in the next one in our processing chain, as each module relies on information from any number of the preceding modules, marginally explaining some of the lower scores in later tasks.

Regarding the system itself we already created a fully functioning on-line version available at our NLP Tools Website[4]. During the last weeks after the shared task ended, we have replaced the decision tree algorithm with our own implementation of a linear classifier, and have obtained superior results. However there the footprint of the model obtained using the linear classifier is, in some cases, 1000 times larger than that of the decision tree classifier (i.e. the Ancient Greek XPOS linear model size is 4.5GB, whereas the DT model is only 4MB). Experiments using a deep neural network (DNN) architecture, trained to predict attributes and XPOS based on character-level features were also performed. Though this approach provided state-of-the art results for some languages, we found it difficult to tune hyper-parameters for all languages. However, for the DNN approach, the model footprint and performance figures (accuracy and computational time) were very appealing.

While one might consider that training independent models for each morphological attribute would provide better results, decision trees, Linear Classifier and DNN performed significantly better, when trained to output all the morphological features at once (softmax one-of-n encoded, not multitask learning). Additionally, we experimented with multitask learning (i.e.: using a common network structure, followed by multiple softmax layers) (Collobert and Weston, 2008) and observed that it did not improve the learning process, at least on the corpora and feature sets we used. Further tuning will be done and performance figures evaluated by the UD evaluation script will be reported on the above mentioned website.

Acknowledgments

This work was supported by UEFISCDI, under grant PN-II-PT-PCCA-2013-4-0789, project "Assistive Natural-language, Voice-controlled System for Intelligent Buildings" (2013-2017).

References

Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*. ACM, pages 160–167.

Tomaz Erjavec. 2004. Multext-east version 3: Multilingual morphosyntactic specifications, lexicons and corpora. In *Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC 2004)*.

Tao Lei, Yuan Zhang, Regina Barzilay, and Tommi Jaakkola. 2014. Low-rank tensors for scoring dependency structures. Association for Computational Linguistics.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. pages 3111–3119.

Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajič, Christopher Manning, Ryan McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, Reut Tsarfaty, and Daniel Zeman. 2016. Universal Dependencies v1: A multilingual treebank collection. In *Proceedings of the 10th International Conference on Language Resources and Evaluation (LREC 2016)*. European Language Resources Association, Portoroz, Slovenia, pages 1659–1666.

Joakim Nivre et al. 2017a. Universal Dependencies 2.0. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics, Charles University, Prague, http://hdl.handle.net/11234/1-1983. http://hdl.handle.net/11234/1-1983.

Joakim Nivre et al. 2017b. Universal Dependencies 2.0 – CoNLL 2017 shared task development and test data. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics, Charles University, Prague, http://hdl.handle.net/11234/1-2184. http://hdl.handle.net/11234/1-2184.

Slav Petrov, Dipanjan Das, and Ryan McDonald. 2011. A universal part-of-speech tagset. *arXiv preprint arXiv:1104.2086* .

Martin Potthast, Tim Gollub, Francisco Rangel, Paolo Rosso, Efstathios Stamatatos, and Benno Stein. 2014. Improving the reproducibility of PAN's shared tasks: Plagiarism detection, author identification, and author profiling. In Evangelos Kanoulas, Mihai Lupu, Paul Clough, Mark Sanderson, Mark Hall, Allan Hanbury, and Elaine Toms, editors, *Information Access Evaluation meets Multilinguality, Multimodality, and Visualization. 5th International Conference of the CLEF Initiative (CLEF 14)*. Springer, Berlin Heidelberg New York, pages 268–299. https://doi.org/10.1007/978-3-319-11382-1_22.

Milan Straka, Jan Hajič, and Jana Straková. 2016. UDPipe: trainable pipeline for processing CoNLL-U files performing tokenization, morphological analysis, POS tagging and parsing. In *Proceedings of the 10th International Conference on Language Resources and Evaluation (LREC 2016)*. European Language Resources Association, Portoroz, Slovenia.

David Weiss, Chris Alberti, Michael Collins, and Slav Petrov. 2015. Structured training for neural network transition-based parsing. *CoRR* abs/1506.06158. http://arxiv.org/abs/1506.06158.

Daniel Zeman. 2008. Reusable tagset conversion using tagset drivers. In *Proceedings of the 6th International Conference on Language Resources and Evaluation (LREC 2008)*.

Daniel Zeman, Filip Ginter, Jan Hajič, Joakim Nivre, Martin Popel, Milan Straka, and et al. 2017. CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*. Association for Computational Linguistics, pages 1–20.

Yuan Zhang, Tao Lei, Regina Barzilay, and Tommi Jaakkola. 2014a. Greed is good if randomized: New inference for dependency parsing .

Yuan Zhang, Tao Lei, Regina Barzilay, Tommi Jaakkola, and Amir Globerson. 2014b. Steps to excellence: Simple inference with refined scoring of dependency trees. Association for Computational Linguistics.

[4] http://slp.racai.ro/index.php/mlpla-new/