Multilingual Word Segmentation: Training Many Language-Specific Tokenizers Smoothly Thanks to the Universal Dependencies Corpus

Erwan Moreau[1], **Carl Vogel**[2]
[1] Adapt Centre, [2] Trinity Centre for Computing and Language Studies, [1,2] Trinity College Dublin
Trinity College Dublin, Dublin 2
{moreaue, vogel}@scss.tcd.ie

Abstract

This paper describes how a tokenizer can be trained from any dataset in the Universal Dependencies 2.1 corpus (UD2) (Nivre et al., 2017). A software tool, which relies on Elephant (Evang et al., 2013) to perform the training, is also made available. Beyond providing the community with a large choice of language-specific tokenizers, we argue in this paper that: (1) tokenization should be considered as a supervised task; (2) language scalability requires a streamlined software engineering process across languages.

Keywords: Universal Dependencies, Word Segmentation, Tokenization, Multilinguality, Interoperability

1. Introduction

This paper describes how a tokenizer can be trained from any dataset in the Universal Dependencies 2.1 corpus (UD2) (Nivre et al., 2017). A software tool, which relies on Elephant (Evang et al., 2013) to perform the training, is also made available. Beyond providing the community with a large choice of language-specific tokenizers, this paper explores two ideas:

- **A new perspective on word segmentation.** We argue that tokenization does not depend only on language and genre,[1] but includes conventions and choices related to ambiguous cases within the same language-genre pair. As a consequence, using some predefined tokenizer might cause the loss of some information, especially when combining multiple language resources. It follows that, in general, tokenization should be seen as a supervised task, where textual data is tokenized following a given "model": by making this model explicit, one can enforce model consistency when connecting various pre-tokenized language resources together, hence avoiding errors which might otherwise go undetected.[2]
- **Towards language scalability**. Major progress has been achieved in multilingual language technology in the recent years. In NLP applications, scalability in terms of data size has been addressed for the most part, but scalability in terms of language diversity is still a significant challenge. The Universal Dependencies 2.1 corpus includes 102 annotated datasets and 59 distinct languages, thanks to the authors' and contributors' great effort (Nivre et al., 2017). Packaging such a diversity of languages in a uniform format is a major step towards the ability to process multiple languages in an homogeneous way, which is the cornerstone of language-wise scalability. Bikel and Zitouni (2012, xxi) mention that *"Previously, to build robust and accurate multilingual natural language processing (NLP) applications, a researcher or developer had to consult several reference books and dozens, if not hundreds, of journal and conference papers."* One might add that said researcher or developer would also have to find, test and integrate multiple language-specific software tools. Thus, language scalability also requires a more streamlined engineering process: it becomes impractical to find a specific software tool for every language to process, let alone the best tool for every language. This is why we argue that the evaluation of software tools should progressively shift the focus from accuracy in a specific language to robustness and adaptability to a wide range of languages. We see the tool presented in this paper as a modest step in this direction.

This paper is organized as follows. First, we explain in section §2. why the need to tackle increasingly complex multilingual tasks in NLP makes it necessary to have robust and flexible pre-processing tools available, like tokenizers. In §3. we present the state of the art in word segmentation, together with the existing tools available. In §4. we present a new tool which satisfies the aforementioned criteria of robustness and flexibility. Finally we demonstrate the interest of the tool in three experiments in §5.

2. Motivations

Many NLP shared tasks nowadays provide datasets annotated with the linguistic information relevant to the task, so that participants can focus on the core aspects of the task rather than spend time on non-essential pre-processing steps. For example, the CoNLL format and its variants are widespread among the NLP community; a dataset provided in this format usually contains word and sentence segmentation, POS tagging, syntactic dependencies and possibly

[1] E.g. tokenizing tweets versus tokenizing The New York Times.

[2] This idea can be compared to the case of POS tags, for instance: it is clear that mixing datasets annotated with different labeling schemes can lead to errors. In the case of tokenization, the scheme consists in the guidelines that the annotators follow for choosing which units represent tokens (otherwise stated, where the annotators choose to put the boudaries between tokens).

other relevant features. This kind of annotation is particularly helpful in the case of multilingual shared tasks, since participants are unlikely to be familiar with all the languages to process.

All these efforts of the community aim at tackling more and more sophisticated problems on more and more diverse languages and types of data. As this natural trend to address high-level tasks progresses, the need to work with multiple datasets from various origins and in various formats grows. For example, it is common to collect raw data from the Internet in order to increase the coverage of a ML model, rather than relying solely on the annotated training data. But combining heterogenous datasets comes with challenges, and unsurprisingly the first one is tokenization. Tokenization errors can be costly performance-wise, as these errors may propagate through the whole processing chain. While sometimes these problems can be diagnosed and fixed manually, typically when dealing with only a few familiar languages, the multiplicity of languages makes manual diagnosis impractical. Some generic tokenization methods can be used as a fallback, but this is rarely optimal. As a consequence, such cases can only be solved by training a tokenizer on the provided input data, in order to apply the same tokenization choices in the third-party corpus.

3. Related Work

Tokenization has traditionally been addressed with rule-based systems (e.g. by Dridan and Oepen (2012)), but supervised ML approaches are more and more common due to their flexibility when tackling new languages (see e.g. (Frunza, 2008)). Dridan and Oepen (2012) and Fares et al. (2013) analyse the question of tokenization ambiguities and the resulting diversity of tokenization conventions for the English language, emphasizing the fact that many tokenization schemes co-exist in practice. In this context, both (Mark and Bojar, 2012) and (Evang et al., 2013) propose a supervised approach, considering tokenization as a sequence labeling problem. They both use Conditional Random Fields (Lafferty et al., 2001) to solve it, but in (Evang et al., 2013) the CRF features are enriched with the top N most active neurons of a Recurrent Neural Network (RNN) language model, based on the work of Chrupala (2013) for character-level language modeling. This approach shows significant improvement over the simple CRF one, as the authors show with three datasets in English, Dutch and Italian. (Mark and Bojar, 2012) validate their system on English and Chinese.

In this work we use the Elephant[3] tokenizer training software, described in (Evang et al., 2013). This software itself relies on the Wapiti implementation (Lavergne et al., 2010) for sequence labeling with CRFs, and on the work of (Mikolov et al., 2010) for the RNN language modeling. Although Elephant is capable of segmenting sentences as well, in this work we focus on word segmentation only.

[3]The authors named their system Elephant because *"like an elephant in the living room, it is a problem that is impossible to overlook whenever new raw datasets need to be processed or when tokenization conventions are reconsidered."* (Evang et al., 2013, 1422).

4. Tool: An Elephant Wrapper

4.1. Motivations

The authors of Elephant (Evang et al., 2013) made their system available at `http://gmb.let.rug.nl/elephant`,[4] and deserve our gratitude for making the effort to provide a clear documentation, including how to reproduce their experiments. However, although a training script is provided, this script only allows training the CRF model. Hence the user can either use one of the three pre-trained models, or train a CRF-only model, without the RNN language model features.[5] Additionally, the user must provide a CRF template file which describes the features that the sequential model uses. This means that the user has either to pick a template at random, or proceed by trial and error in order to find a suitable template.

Finally, for users who simply need to tokenize some data and cannot (or do not want to) train a model, Elephant contains pre-trained tokenizers but for only three languages.

The tool that we propose is available at `https://github.com/erwanm/elephant-wrapper`.[6]

It aims to fill the aforementioned usability gaps in Elephant, together with providing users with a more universal tool for word segmentation, in terms of technical usage (training and testing capabilities) as well as diversity of languages. The latter is made possible thanks to the availability of the Universal Dependencies 2.1 corpus (Nivre et al., 2017), which contains 102 datasets in 59 distinct languages.

4.2. Implementation

In the perspective of providing universal tokenizer training capabilities, we propose several scripts intended to make this part of the training more straightforward. Additionally, our wrapper provides a convenient way to generate the required IOB-labeled sequences[7] of characters from a pre-tokenized corpus. In particular, the `.conllu` format used in UD2 (Nivre et al., 2017) is accepted as input, as well as similar formats such as the one used in the 2017 Shared Task on Identifying Verbal Multi-Word Expressions (see §5.4.). This is intended to streamline the process of training a tokenizer by making the required internal formats transparent to the user, thus improving greatly the usability of the system. The conversion to the IOB format is implemented in the following way: following the general good practice of preserving the form of the original data, the UD2 datasets contain annotations which indicate for every token whether a space follows the token or not.[8] Thus, it is possi-

[4]Last retrieved: 21/02/2018.

[5]Remark: training the RNN language model so that it can be used with the existing CRF training script is not entirely trivial because it requires converting the training data to Unicode characters codes presented as "tokens".

[6]The tool can also be found at `https://www.scss.tcd.ie/clg/elephant-wrapper`.

[7]The IOB format consists of labeling every character with: B for the beginning of a token, I for subsequent characters inside a token, and O for characters outside any token (whitespaces).

[8]This indication is provided in column 10 of the `.conllu` file: this column contains the parameter/value pair `SpaceAfter=No` if and only if no space follows the token. Remark: the `.txt` file could also be used, but this would require

ble to rebuild the original text, together with the appropriate IOB label for every character. Additionally, our conversion script takes care of ignoring tokens corresponding to expanded contractions when they appear: several languages contain contracted tokens, e.g. *"du" (of the)* in French; such cases are represented as follows in UD2: (1) the surface token is given on the first line with a range of indexes as value in the first column (e.g. `18-19 du`), instead of a single token index; (2) the tokens on the following lines correspond to the expansion, and their indexes belong to the previously seen range (e.g. `18 de` followed by `19 le`). The expanded form is necessary for the morpho-syntactic analysis, but is irrelevant for the tokenization part.

The tool also provides scripts which generate CRF template files for Wapiti and run k-fold cross-validation with every possible template in a predefined set. This way the best performing template can be selected for the final training. The template indicates which features are used in the sequential model:

- Value n of n-grams features;
- Length of the window of characters;
- Using characters Unicode code point and/or Unicode category;
- Using the top 10 RNN features or not.

The set of patterns to test is simple to configure, so that the user can choose how thorough the search should be. Of course, the time required to run the cross-validation process depends on the number of templates to apply, but an option is supplied to stop the search when the performance shows no progress anymore.[9]

Finally, the tool provides 102 tokenizer models trained from the UD2 corpus (see §5.1.). For the convenience of the user, the language model to use by the tokenizer can be given in different forms: as a custom model directory, as the name of a UD2 dataset, or as the ISO639 standard code for the language.[10]

5. Experiments

Below we present several experiments made with the Elephant Wrapper tool introduced in §4.2., available at `https://github.com/erwanm/elephant-wrapper`. The tool contains the scripts required to reproduce these experiments. It was designed to make batch processing of multilingual experiments as simple as possible, with the idea of language scalability in mind.[11]

5.1. Training Multiple Tokenizers from UD2

The first experiment is essentially intended to demonstrate the feasibility and effectiveness of training multiple language-specific tokenizers in an homogeneous way, thus making the engineering design process straightforward. This is only made possible thanks to the consistency in the annotation format across datasets offered by the UD2 corpus.

The experiment consists in training a tokenizer for every dataset in the UD2 corpus using the file `<lang>-ud-train.conllu` as training data, then testing the tokenizer on the `<lang>-ud-test.conllu` file. The method generalizes the one described in (Evang et al., 2013) (see §3.): a CRF model is trained, which might include features from a RNN language model previously trained with the same training data. However, instead of using only one specific template for the CRF model, the cross-validation stage described in §4.2. is run over the training set in order to find the optimal template among a large set of possibilities (96 templates). Then the final training is performed on the whole training set, using the selected template. A simple generic tokenizer is used as a baseline for the sake of comparison: it relies on whitespaces and strips any sequence of punctuation signs from the word. Both the trained tokenizer and the baseline tokenizer are applied to the test set.

Table 1 gives the performance obtained by both the generic tokenizer (baseline) and the trained tokenizer for every dataset in UD2. For the evaluation we follow Shao et al. (2017): performance is measured using the token-based recall, i.e. the proportion of tokens correctly identified among the gold-standard tokens.[12] For all the datasets but two,[13], the Elephant-trained system performs as well or better than the baseline; in many cases the former dramatically outperforms the latter. The mean performance of the trained tokenizer is 99.23%; in average it improves the performance by 86% compared to the baseline, but the mean poorly reflect the diversity of the cases: for 71% of the datasets the performance increases by less than 5%, whereas for the top 13% it increases by more than 100%; the median improvement is 2.7%.

Due to the large number of datasets and the authors' ignorance of the vast majority of languages, it is completely impractical to investigate every case individually. However we investigated the cases where the tokenizer obtains a perfect score. Many such cases are due to the lack of annotation indicating whether a token is followed by a space or not (see 4.2.): Coptic, Danish, Finnish-FTB, Gothic, Marathi, Norwegian-NynorskLIA, Swedish_Sign_Language, Slovenian-SST, Telugu; in a few cases this might be due to the nature of the data (e.g. for the Swedish Sign Language); otherwise this is an error which prevents reconstructing the non-tokenized text from the data.[14] Some ancient languages do not contain any

accessing the .conllu file anyway in order to identify the tokens' boundaries.

[9] By default the template search space is explored from the simplest templates to the most complex.

[10] In the latter case, if there are several UD2 datasets for the same language, the default UD2 one (with no extension) is used.

[11] Feedback and contributions are welcome.

[12] The software also allows evaluation using character-based accuracy or error rate.

[13] The performance decreases very slightly for Kazakh (-0.65%) and Latin (-0.02%). In the case of Kazakh, this might be due to the small size of the training set (only 511 tokens) causing the model to overfit slightly; in the other case the difference is too small to be significant.

[14] Since the annotations indicate the absence of space after a token (see 4.2.), the system assumes that there are spaces everywhere between the tokens, in both the training and test set. This

UD 2.1 Dataset name	Training set size (tokens)	Baseline Recall (%)	Trained tokenizer Recall (%)	UD 2.1 Dataset name	Training set size (tokens)	Baseline Recall (%)	Trained tokenizer Recall (%)
Afrikaans	33894	95.73	99.93	Italian [E]	252631	95.65	99.82
Ancient_Greek	159895	96.35	99.97	Italian-ParTUT	45477	97.05	99.76
Ancient_Greek-PROIEL	184382	99.29	100.00	Italian-PoSTWITA [E]	49478	88.63	99.60
Arabic	191869	97.13	99.96	Italian-PUD	17746	96.46	99.87
Arabic-NYUAD	502991	100.00	100.00	Japanese	161900	13.09	93.01
Arabic-PUD	16601	70.83	99.09	Japanese-PUD	21454	15.11	96.45
Basque [E]	72974	97.54	99.98	Kazakh [E]	511	96.27	95.64
Belarusian	5217	95.15	99.35	Korean	52328	93.53	99.64
Bulgarian [E]	124336	97.52	99.80	Kurmanji	7957	97.50	99.67
Buryat	8026	98.77	99.92	Latin [E]	8018	99.98	99.96
Cantonese	552	27.39	96.81	Latin-ITTB	270403	99.43	99.76
Catalan	416659	93.35	99.94	Latin-PROIEL	147044	99.90	100.00
Chinese	98608	18.23	92.08	Latvian	62397	97.45	99.13
Chinese-CFL	5805	12.98	94.45	Lithuanian	3210	95.94	99.06
Chinese-HK	1497	20.63	93.21	Marathi	2730	29.79	100.00
Chinese-PUD	17132	16.76	96.33	North_Sami	16835	98.64	99.56
Coptic	4238	100.00	100.00	Norwegian-Bokmaal	243887	98.54	99.89
Croatian	169283	97.82	99.89	Norwegian-Nynorsk	245330	98.73	99.90
Czech [E]	1171190	98.47	99.95	Norwegian-NynorskLIA	8020	98.54	100.00
Czech-CAC [E]	471594	99.96	99.96	Old_Church_Slavonic	37432	91.78	100.00
Czech-CLTT	26225	92.97	99.54	Persian [E]	119945	92.91	100.00
Czech-FicTree	133137	96.27	99.99	Polish	61906	99.42	99.94
Czech-PUD	14852	98.63	99.88	Portuguese	191410	96.44	99.64
Danish	80378	98.01	100.00	Portuguese-BR	238334	98.04	99.88
Dutch [E]	186046	97.52	98.96	Portuguese-PUD	17511	98.72	99.77
Dutch-LassySmall	81243	95.48	99.84	Romanian	185113	96.87	99.49
English	204607	95.36	98.78	Romanian-Nonstandard	10352	95.87	98.28
English-LinES	50095	97.27	99.96	Russian	75964	96.06	99.89
English-ParTUT	43491	99.21	99.71	Russian-PUD [E]	15559	97.39	99.88
English-PUD	16941	97.60	99.89	Russian-SynTagRus	871082	96.82	99.58
Estonian	85567	96.10	99.81	Sanskrit	1158	8.22	99.59
Finnish	162827	97.22	99.53	Serbian	65764	96.58	99.88
Finnish-FTB	127359	99.08	100.00	Slovak [E]	80575	98.95	99.65
Finnish-PUD	12650	96.33	99.79	Slovenian	112530	98.99	99.87
French	346855	92.47	99.37	Slovenian-SST	9487	95.02	100.00
French-FTB	442228	66.57	100.00	Spanish	375149	98.47	99.95
French-ParTUT	23324	94.87	99.84	Spanish-AnCora [E]	443087	97.85	99.91
French-PUD [E]	19310	93.57	99.87	Spanish-PUD [E]	18258	99.16	99.96
French-Sequoia	49198	92.79	99.73	Swedish	66645	98.44	99.95
Galician	71928	97.96	99.83	Swedish-LinES	48325	99.08	99.95
Galician-TreeGal	4447	98.54	99.62	Swedish-PUD	15259	98.18	99.86
German	258927	97.56	99.14	Swedish_Sign_Language	644	59.22	100.00
German-PUD	16798	98.61	99.80	Tamil	5734	4.01	99.21
Gothic	35024	99.95	100.00	Telugu	5082	26.21	100.00
Greek [E]	41212	98.37	99.67	Turkish	36970	97.59	99.86
Hebrew	98348	97.34	99.93	Turkish-PUD	13228	94.90	99.83
Hindi	281057	16.27	99.97	Ukrainian	75054	96.18	99.63
Hindi-PUD	19063	13.74	99.92	Upper_Sorbian	8589	96.74	100.00
Hungarian	20166	96.79	99.82	Urdu	108690	98.55	100.00
Indonesian	97531	98.50	99.97	Uyghur	8264	98.47	99.95
Irish	3183	97.01	99.43	Vietnamese	20285	79.37	85.42

Table 1: Token-based recall (percentages) for the 102 datasets in UD 2.1, for both the baseline tokenizer and the Elephant-trained tokenizer. For the latter, the optimal model was selected by using cross-validation on the training set (see §4.2.); datasets marked with "[E]" show cases for which a model containing the Elman features (top 10 RNN most active neurons) is selected.

punctuation signs at all or very few, which makes the tokenization trivial: Ancient_Greek-PROIEL, Latin-PROIEL, Old_Church_Slavonic. A couple of datasets seem to be corrupted in version 2.1 of the UD corpus: Arabic-NYUAD and French-FTB (all the tokens are replaced with underscore signs). Finally the cases of Persian, Upper_Sorbian and Urdu show no sign of error. We decided to keep all these possibly erroneous cases in the results, because we cannot distinguish easily which of them are actual errors and have no way either to know if other datasets with apparently regular performance contain errors.

makes the task of the tokenizer trivial.

The case of CJKV languages (Chinese, Japanese, Korean and Vietnamese) is worth observing: although the CRF approach was originally designed for Indo-European languages, it performs decently on these Asian languages, considering their specificities. The recall ranges from 85.42% for Vietnamese to 99.64% for Korean; while the baseline tokenizer performs very poorly with these languages with a mean of 33.01%, the mean performance of the Elephant-trained tokenizer is 94.15%.

In (Evang et al., 2013), experiments are performed on three European languages: English, Dutch and Italian. Besides the number of languages, the experimental setup also dif-

fers by the number of CRF templates tested: in (Evang et al., 2013) the characteristics determining the template are tested with a few predefined values for all the languages. For instance, the first experiment concludes that the use of both Unicode code points and categories gives the best performance overall, even though using only the former performs better for English; then the following experiments use both Unicode code points and categories for all the languages including English, for which this is suboptimal according to the previous experiment. By contrast, our approach determines the best template for each individual language among a large set of possibilities (96 in this experiment) using cross-validation. Although this requires more training time, it will generate a more accurate model in general. From the perspective of language scalability, it should be emphasized that the requirement to streamline multilingual processes does not entail overlooking language-specific features: the design of the process should be as uniform as possible, but without "standardizing" the languages themselves.

One of the consequences of our approach is that the RNN features are selected or not based on the results of the cross-validation stage on the training data. It turns out that most datasets in the UD2 corpus do not benefit from these features: they were selected as part of the optimal template in only 16 cases (16% of the datasets). In particular, none of the CJKV languages benefits from them. In (Evang et al., 2013), these features were reported to provide a significant improvement in all three languages studied. Besides the difference in the datasets, we suppose that this difference might be caused by the more thorough search for an optimal template in our experiment: by exploring many more templates (with or without the RNN features), the process is more likely to reach a optimal level of performance, equivalent to the one that could have been reached with a less fine-grained template containing RNN features.

5.2. Training on a Different Dataset in the Same Language

The second experiment aims to illustrate the fact that tokenization follows a particular scheme, and different schemes lead to different ways to tokenize a text even within the same language. Consequently, when a task relies on matching tokens from a text with a given pre-tokenized language resource, it is important to ensure the consistency of the tokenizer with the "scheme" which was used to generate the resource. We illustrate this point by running the following experiment with five languages in UD2 for which several distinct datasets are provided: a tokenizer is trained with the training set of every dataset for the language, and applied to the test set of every dataset as well. Table 2 shows the performance in every case. As expected in any similar supervised ML setting, the best results are consistently achieved when the training and test set are drawn from the same corpus (with only one exception in Italian). This experiment shows that token boundaries do not only depend on the language, and therefore that applying a certain tokenizer for this sole reason is not always optimal. In particular, the fact that traditional tokenizers are rule-based does not imply that there is a unique way to tokenize a language. This is why we think that word segmentation should be considered a supervised task.

Training set	Chinese	Chinese CFL	Chinese HK	Chinese PUD
Chinese	**92.08**	71.32	75.09	90.46
Chinese-CFL	60.96	**94.45**	72.31	65.71
Chinese-HK	57.57	70.05	**93.21**	60.76
Chinese-PUD	77.42	70.40	73.12	**96.33**

Training set	Czech	Czech CAC	Czech CLTT	Czech FicTree	Czech PUD
Czech	**99.95**	99.96	93.73	**99.99**	99.69
Czech-CAC	96.38	**99.96**	95.78	94.30	97.33
Czech-CLTT	95.79	99.37	**99.54**	90.95	97.14
Czech-FicTree	99.16	99.88	94.41	**99.99**	99.38
Czech-PUD	98.79	99.89	94.90	99.43	**99.88**

Training set	English	English LinES	English ParTUT	English PUD
English	**98.78**	99.35	99.24	99.77
English-LinES	95.40	**99.96**	98.18	96.93
English-ParTUT	95.45	98.45	**99.71**	98.98
English-PUD	96.37	98.97	99.32	**99.89**

Training set	French	French ParTUT	French PUD	French Sequoia
French	**99.37**	99.36	97.69	99.36
French-ParTUT	98.66	**99.84**	96.79	98.55
French-PUD	97.88	99.36	**99.87**	98.22
French-Sequoia	99.00	99.36	96.74	**99.73**

Training set	Italian	Italian ParTUT	Italian PoSTWITA	Italian PUD
Italian	**99.82**	**99.82**	96.19	99.65
Italian-ParTUT	99.65	99.76	90.56	99.68
Italian-PoSTWITA	97.81	99.13	**99.60**	98.76
Italian-PUD	99.44	99.73	89.35	**99.87**

Table 2: Token-based recall (percentages) when applying a tokenizer trained on a given dataset (rows) to another dataset (columns) in the same language. The highest performance for each test set is highlighted in bold. Example: when applying the model trained on the Italian-ParTUT training set to the Italian-PUD test set, the recall is 99.68%.

5.3. Impact of the Size of the Training Set

Since tokenization should be seen as a supervised task, it is important to know how much data is needed to train an accurate model. This is why in this section we study the impact of the size of the training set on the perfomance of the tokenizer. The experiment simply consists in training a model using only a subset of the training set instead of the whole data, for various sizes of the subset; then the model is applied to the regular test set. For this experiment we select a group of datasets for which a large training set is provided, in order to collect the results for a large range of sizes.

Figure 1 shows the impact on performance of linearly increasing the size of the training set from 890 sentences to 8900, for the 20 largest datasets. Some datasets obtain a decent level of performance with as little as 890 sentences: for instance the Spanish one reaches 99.94%. However the variance is high, with most datasets far from their maximum performance. In fact, the main difference when increasing the size of the training set is that the variance decreases: the mean progresses as well, but the most important observation that can be made from Figure 1 is that, as the size of the training set increases, all the languages reach a high level of performance. In other words, even if a small training set might suffice to obtain an accurate tokenizer,

training it with a large number of sentence makes it more likely to be accurate. In terms of performance, Figure 2 shows that the performance mostly follows a logarithmic progression with respect to the size of the training set. This means that when the performance reaches a high level, increasing it more requires to add a much larger amount of data.

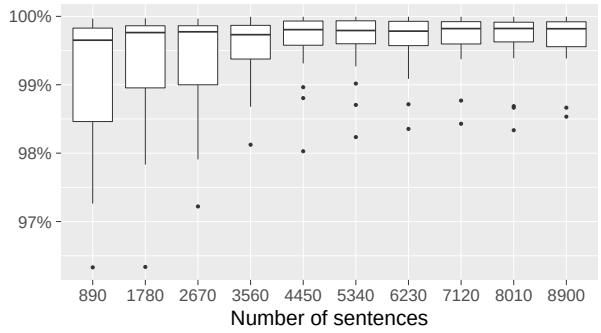

Figure 1: Token-based recall by number of sentences in the training set for the 20 largest datasets in UD 2.1 (by size of the full training set in number of sentences). Each boxplot represents the performance on the test set of a model trained with n sentences, for all 20 datasets.

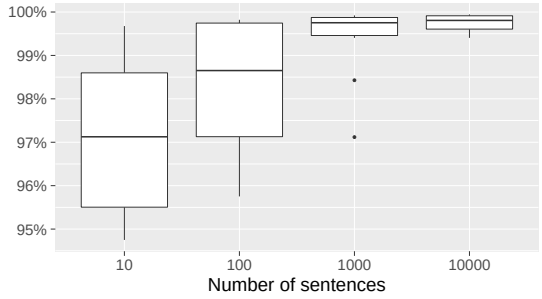

Figure 2: Token-based recall by number of sentences in the training set with exponential progression on the X axis, for the 10 largest datasets in UD 2.1.

5.4. Use Case: Detecting Multi-Word Expressions

As an example of a complex multilingual task in which supervised tokenization can help (see §2.), the authors of this paper participated in the 2017 Shared Task on Identifying Verbal Multi-Word Expressions (VMWE) (Savary et al., 2017). The task consists in identifying VMWEs in 18 different languages. The input data is provided in the CoNLL format, annotated with tokens, POS tags and dependencies.[15] In the approach described by Maldonado et al. (2017) and Moreau et al. (2017), we propose to leverage third-party raw text corpora in order to calculate semantic context vectors for the candidate expressions. Since computing context vectors requires a large resource, we opted for using the Europarl corpus (Koehn, 2005), which is available in 12 among the 15 required languages.[16] The candidate expressions are provided by the first component of the system, a sequence labeling system using CRF. The CRF component provides 10 candidate labeled sequences, the top one being the default choice; the second component (reranker) aims to replace the default choice with one of the other candidates if needed, based on semantic similarity. In order to make the semantic vectors as reliable as possible (i.e., as representative of the meaning of the expression as possible), the system must match all the occurrences of a candidate VMWE (extracted from the input data) in the third-party corpus. But Europarl might not contain every possible VMWE found in the VMWE17 data, or might contain too few occurrences for some VMWEs. Moreover, the discrepancies between the tokenized input data and the third party data (when tokenized using a generic tokenizer) might prevent matching correctly VMWEs which contain words tokenized in a different way. In particular, VMWEs frequently include function words which are susceptible to tokenization errors, like *"c'est" (it is)* in French: if the tokenizer does not properly recognize the apostrophe as part of the first token *"c' "*,[17] expressions which contain *"c'est"* cannot be matched. Moreover, even using a language-specific tokenizer might not always solve this problem, because some tokenization ambiguities cannot be solved other than by an arbitrary choice depending on interpretation; for instance, the documentation about the way tokenization is carried out for French in the UD2 corpus mentions that: *"This tokenizing and segmentating choice is arbitrary and other French treebanks could choose to do otherwise."*[18] As a consequence, a dataset-specific tokenizer should be trained on the provided input data in order to tokenize the third-party corpus accurately.

	Baseline	Trained tokenizer	Improvement
Precision	81.39	81.27	-0.15
Recall	22.43	22.83	+1.77
F-score	35.17	35.64	+1.35

Table 3: Sentence-level micro precision, micro recall and micro F-score of the reranker over all the datasets (i.e., considering all the instances from all the datasets together). (VMWE17 Shared Task data, §5.4.)

Table 3 shows the performance of the reranker on its own, first when tokenizing Europarl with a generic tokenizer (baseline) and then when tokenizing with a specific model trained on the appropriate training data. It can be observed that the precision decreases slightly, but the recall increases more strongly so that the F-score also increases.

[15] All the datasets but the following three are provided in the CoNLL format: Bulgarian, Hebrew and Lithuanian (these languages are discarded in our experiments).

[16] The following three languages are discarded for this reason: Farsi, Maltese and Turkish.

[17] One may notice that this is the opposite in English: the apostrophe should normally be assigned to the token on its right hand-side, like "'s" in "it's".

[18] `https://github.com/UniversalDependencies/docs/blob/pages-source/_fr/tokenization.md`; last retrieved: 19/02/2018.

Language	Baseline	Trained tokenizer	Improvement
CS	19.59	19.59	0.00
DE	23.65	23.73	+0.34
EL	08.72	08.72	0.00
ES	14.45	14.45	0.00
FR	17.46	19.68	+12.71
HU	36.39	36.39	0.00
IT	19.18	19.18	0.00
PL	16.12	16.22	+0.61
PT	10.73	10.58	-1.43
RO	06.38	06.38	0.00
SL	12.02	12.02	0.00
SV	10.56	10.56	0.00

Table 4: Proportion of VMWEs from the VMWE17 test data found in Europarl by dataset (percentages).[20]

In order to understand these results, one has to look at the different ways in which the tokenization of Europarl can impact the performance of the reranker. First, a candidate expression which was not identified before in Europarl might be identified thanks to the more accurate tokenization. Table 4 shows the proportion of expressions which are identified in Europarl: while for most languages using the appropriate tokenization model does not impact this proportion or only slightly, the French dataset shows a large improvement. We think that this is due to the case of the apostrophe, which is ubiquitous in French and cannot be handled properly by the baseline tokenizer. Second, a more accurate tokenization can affect the context words of an expression, and thus improve the accuracy of the context vectors, i.e. the vectors can be more representative of the meaning of the VMWE. This effect can be observed in Table 5, which gives the recall by dataset for the expressions covered in Europarl: the recall increases by 8% overall, with many languages showing a significant improvement up to double the baseline recall (for Hungarian). The F-score for these instances follows a similar trend, with an average improvement of 6.96%.

Thus the effect of a better tokenization varies greatly depending on the language, but in general it gives the reranker more and/or better information through the context vectors; thanks to this, the reranker can then take more risk in proposing an alternative labeled sequence, hence the increase in recall. This explanation is confirmed by breaking down the performance by number of expressions in the sentence, as shown in Table 6: when using the Elephant-trained model for Europarl, the performance decreases for the sentences which contain no VMWE at all, but increases noticeably for all the cases where the sentence contains at least one expression. It is worth noticing that the improvement includes the cases containing multiple expressions, which are the hardest to identify.

Finally, considering the whole system (CRF and reranker components together) and the official evaluation measure used in the VMWE17 Shared Task, the advantage of supervised tokenization translates into a modest 0.3% F-score improvement at the level of expressions, as shown in Table 7. This can be explained for the most part by the higher risk that the reranker takes, which increases the recall at the cost of decreasing the precision. Additionally, the official evaluation measure does not particularly reward the fact the system captures more difficult cases, as shown in Table 6.

[20] Counting the VMWEs in the gold standard test set, but only if the CRF component returns the gold labeled sequence as one of the 10 candidates (since otherwise the reranker cannot select the right answer).

Language	Baseline	Trained tokenizer	Improvement
CS	58.03	58.55	+0.89
DE	27.27	34.33	+25.87
EL	37.70	42.62	+13.04
ES	32.84	34.33	+4.55
FR	34.09	34.48	+1.15
HU	12.77	27.66	+116.67
IT	9.76	9.76	0.00
PL	32.43	32.43	0.00
PT	45.00	49.15	+9.23
RO	57.14	57.14	0.00
SL	36.96	34.78	-5.88
SV	6.56	6.56	0.00
All (macro)	32.55	35.15	+8.00

Table 5: Recall by dataset for the 8,544 sentences which contain at least one expression which appears in Europarl (percentages).

# exprs. / sentence	Proportion data	Baseline Macro	Baseline Micro	Trained tokenizer Macro	Trained tokenizer Micro
0	81.0%	**53.8**	**52.6**	53.1	51.7
1	16.4%	23.5	26.8	**25.6**	**28.1**
2	2.2%	30.4	25.6	**36.6**	**25.8**
3	0.3%	31.0	16.7	**34.3**	**18.2**
All	100%	29.1	35.2	**29.9**	**35.6**

Table 6: Macro and micro F-score (percentages) at sentence level by number of expressions in the sentence, for all the datasets.

6. Conclusion

In this paper we proposed a method to train multiple tokenizers using multilingual resources, like the Universal Dependencies 2.1 corpus. We showed that word segmentation can and should be seen as a supervised process, as opposed to a language-uniform process. More generally, this approach is in line with the view that language scalability is one of the most important challenges in NLP tasks. As a consequence, NLP tasks have to evolve into a more streamlined software engineering process across languages. We see the software tool that we propose for tokenization as a contribution to this idea.

Language	Baseline	Trained tokenizer	Improvement
CS	70.64	70.69	+0.07
DE	28.70	30.99	+7.98
EL	37.57	37.72	+0.40
ES	49.05	49.68	+1.28
FR	58.40	57.92	-0.82
HU	68.47	67.95	-0.76
IT	16.28	15.87	-2.52
PL	72.42	72.19	-0.32
PT	65.22	66.06	+1.29
RO	83.53	83.53	0.00
SL	44.02	43.63	-0.89
SV	31.91	31.91	0.00
All	52.18	52.34	+0.31

Table 7: VMWE17 official evaluation measure: F-score at the MWE level (percentages).

7. Acknowledgements

The ADAPT Centre for Digital Content Technology is funded under the SFI Research Centres Programme (Grant 13/RC/2106) and is co-funded under the European Regional Development Fund.

The graphics in this paper were created with `R` (R Core Team, 2012), using the `ggplot2` library (Wickham, 2009).

8. Bibliographical References

Bikel, D. and Zitouni, I. (2012). *Multilingual Natural Language Processing Applications: From Theory to Practice*. IBM Press, 1st edition.

Chrupala, G. (2013). Text segmentation with character-level text embeddings. *CoRR*, abs/1309.4628. Workshop on Deep Learning for Audio, Speech and Language Processing, ICML 2013.

Dridan, R. and Oepen, S. (2012). Tokenization: Returning to a long solved problem a survey, contrastive experiment, recommendations, and toolkit. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers - Volume 2*, ACL '12, pages 378–382, Stroudsburg, PA, USA. Association for Computational Linguistics.

Fares, M., Oepen, S., and Zhang, Y. (2013). Machine Learning for High-Quality Tokenization Replicating Variable Tokenization Schemes. In Alexander Gelbukh, editor, *Computational Linguistics and Intelligent Text Processing: 14th International Conference, CICLing 2013, Samos, Greece, March 24-30, 2013, Proceedings, Part I*, pages 231–244, Berlin, Heidelberg. Springer.

Frunza, O. (2008). A Trainable Tokenizer, Solution for Multilingual Texts and Compound Expression Tokenization. In Nicoletta Calzolari (Conference Chair), et al., editors, *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*, Marrakech, Morocco, may. European Language Resources Association (ELRA). http://www.lrec-conf.org/proceedings/lrec2008/.

Lafferty, J. D., McCallum, A., and Pereira, F. C. N. (2001). Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 282–289, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Maldonado, A., Han, L., Moreau, E., Alsulaimani, A., Chowdhury, K. D., Vogel, C., and Liu, Q. (2017). Detection of Verbal Multi-Word Expressions via Conditional Random Fields with Syntactic Dependency Features and Semantic Re-Ranking. In *Proceedings of The 13th Workshop on Multiword Expressions*, pages 114–120, Valencia.

Moreau, E., Alsulaimani, A., Maldonado, A., Han, L., Vogel, C., and Chowdhury, K. D. (2017). Semantic Re-Ranking of CRF Label Sequences for Verbal Multi-Word Expression Identification. *To appear*.

R Core Team, (2012). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0.

Savary, A., Ramisch, C., Cordeiro, S., Sangati, F., Vincze, V., QasemiZadeh, B., Candito, M., Cap, F., Giouli, V., Stoyanova, I., and Doucet, A. (2017). The PARSEME Shared Task on Automatic Identification of Verbal Multiword Expressions. In *Proceedings of the 13th Workshop on Multiword Expressions*, Valencia.

Shao, Y., Hardmeier, C., and Nivre, J. (2017). Recall is the Proper Evaluation Metric for Word Segmentation. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing*, volume 2, pages 86–90.

Wickham, H. (2009). *ggplot2. Elegant Graphics for Data Analysis*. Springer-Verlag New York.

9. Language Resource References

Evang, Kilian and Basile, Valerio and Chrupała, Grzegorz and Bos, Johan. (2013). *Elephant: Sequence Labeling for Word and Sentence Segmentation*. Association for Computational Linguistics.

Koehn, Philipp. (2005). *Europarl: A parallel corpus for statistical machine translation*.

Lavergne, Thomas and Cappé, Olivier and Yvon, François. (2010). *Practical Very Large Scale CRFs*. Association for Computational Linguistics.

Ji Mark and Ondej Bojar. (2012). *TrTok: A Fast and Trainable Tokenizer for Natural Languages*.

Tomas Mikolov and Martin Karafiát and Lukás Burget and Jan Cernocký and Sanjeev Khudanpur. (2010). *Recurrent neural network based language model*. ISCA.

Nivre, Joakim and Agić, Željko and Ahrenberg, Lars and Aranzabe, Maria Jesus and Asahara, Masayuki and Atutxa, Aitziber and Ballesteros, Miguel and Bauer, John and Bengoetxea, Kepa and Bhat, Riyaz Ahmad and Bick, Eckhard and Bosco, Cristina and Bouma, Gosse and Bowman, Sam and Candito, Marie and Cebirolu Eryiit, Gülşen and Celano, Giuseppe G. A. and Chalub, Fabricio and Choi, Jinho and Çöltekin, Çar and Connor, Miriam and Davidson, Elizabeth and de Marneffe, Marie-Catherine and de Paiva, Valeria and Diaz de Ilarraza, Arantza and Dobrovoljc, Kaja and Dozat, Timothy and Droganova, Kira and Dwivedi, Puneet and Eli, Marhaba and Erjavec, Tomaž and Farkas, Richárd and Foster, Jennifer and Freitas, Cláudia and Gajdošová, Katarína and Galbraith, Daniel and Garcia, Marcos and Ginter, Filip and Goenaga, Iakes and Gojenola, Koldo and Gökrmak, Memduh and Goldberg, Yoav and Gómez Guinovart, Xavier and Gonzáles Saavedra, Berta and Grioni, Matias and Grūzītis, Normunds and Guillaume, Bruno and Habash, Nizar and Hajič, Jan and Hà M, Linh and Haug, Dag and Hladká, Barbora and Hohle, Petter and Ion, Radu and Irimia, Elena and Johannsen, Anders and Jørgensen, Fredrik and Kaşkara, Hüner and Kanayama, Hiroshi and Kanerva, Jenna and Kotsyba, Natalia and Krek, Simon and Laippala, Veronika and Lê Hng, Phng and Lenci, Alessandro and Ljubešić, Nikola and Lyashevskaya, Olga and Lynn, Teresa and Makazhanov, Aibek and Manning, Christopher and Mărănduc, Cătălina and Mareček, David and Martínez Alonso, Héctor and Martins, André and Mašek,

Jan and Matsumoto, Yuji and McDonald, Ryan and Missilä, Anna and Mititelu, Verginica and Miyao, Yusuke and Montemagni, Simonetta and More, Amir and Mori, Shunsuke and Moskalevskyi, Bohdan and Muischnek, Kadri and Mustafina, Nina and Müürisep, Kaili and Nguyn Th, Lng and Nguyn Th Minh, Huyn and Nikolaev, Vitaly and Nurmi, Hanna and Ojala, Stina and Osenova, Petya and Øvrelid, Lilja and Pascual, Elena and Passarotti, Marco and Perez, Cenel-Augusto and Perrier, Guy and Petrov, Slav and Piitulainen, Jussi and Plank, Barbara and Popel, Martin and Pretkalnia, Lauma and Prokopidis, Prokopis and Puolakainen, Tiina and Pyysalo, Sampo and Rademaker, Alexandre and Ramasamy, Loganathan and Real, Livy and Rituma, Laura and Rosa, Rudolf and Saleh, Shadi and Sanguinetti, Manuela and Saulīte, Baiba and Schuster, Sebastian and Seddah, Djamé and Seeker, Wolfgang and Seraji, Mojgan and Shakurova, Lena and Shen, Mo and Sichinava, Dmitry and Silveira, Natalia and Simi, Maria and Simionescu, Radu and Simkó, Katalin and Šimková, Mária and Simov, Kiril and Smith, Aaron and Suhr, Alane and Sulubacak, Umut and Szántó, Zsolt and Taji, Dima and Tanaka, Takaaki and Tsarfaty, Reut and Tyers, Francis and Uematsu, Sumire and Uria, Larraitz and van Noord, Gertjan and Varga, Viktor and Vincze, Veronika and Washington, Jonathan North and Žabokrtský, Zdeněk and Zeldes, Amir and Zeman, Daniel and Zhu, Hanzhi. (2017). *Universal Dependencies 2.0*.