Proceedings of the 12th Conference on Language Resources and Evaluation (LREC 2020), pages 7152–7160
Marseille, 11–16 May 2020

EstNLTK 1.6: Remastered Estonian NLP Pipeline

Sven Laur, Siim Orasmaa, Dage Särg, Paul Tammo
Institute of Computer Science, University of Tartu
Liivi 2, 50409 Tartu, Estonia
{sven.laur, siim.orasmaa, dage.sarg}@ut.ee, paul.tammo@gmail.com

Abstract

The goal of the EstNLTK Python library is to provide a unified programming interface for natural language processing in Estonian. As such, previous versions of the library have been immensely successful both in academic and industrial circles. However, they also contained serious structural limitations – it was hard to add new components and there was a lack of fine-grained control needed for back-end programming. These issues have been explicitly addressed in the EstNLTK library while preserving the intuitive interface for novices. We have remastered the basic NLP pipeline by adding many data cleaning steps that are necessary for analyzing real-life texts, and state of the art components for morphological analysis and fact extraction. Our evaluation on unlabelled data shows that the remastered basic NLP pipeline outperforms both the previous version of the toolkit, as well as neural models of StanfordNLP. In addition, EstNLTK contains a new interface for storing, processing and querying text objects in PostgreSQL database which greatly simplifies processing of large text collections. EstNLTK is freely available under the GNU GPL version 2 license, which is standard for academic software.

Keywords: natural language processing, Python, Estonian language

1. Introduction

Estonian natural language processing has seen steady advancements in recent years. As a part of the Universal Dependencies effort (Nivre et al., 2016), Estonian treebanks with standardized grammatical annotations have become available. The Universal Dependencies effort has fostered the experimentation with multilingual data-driven parsing models, many of which also cover Estonian (Zeman et al., 2018; Yusupujiang, 2018). In the wake of these developments, a new paradigm – data-driven neural models – has been introduced into Estonian morphological analysis (Tkachenko and Sirts, 2018). Still, rule-based linguistic analysis models are also being successfully implemented. Kaalep et al. (2018) introduced a finite-state-transducers-based morphological analyzer for Estonian which uses a revised morphological category system and provides a more detailed analysis for inflectional forms and compound words.

Universal Dependencies data sets have enabled the inclusion of automatic analysis of Estonian in multilingual NLP tools, such as StanfordNLP (Qi et al., 2018) and NLP-Cube (Boroş et al., 2018). Harnessing the power of deep neural networks, these tools provide end-to-end text analysis capabilities, starting from text segmentation into words and sentences, and leading up to morphological analysis and syntactic parsing. While these pipelines provide basic analysis for standard language, they offer no convenient ways for adapting to specific text domains – apart from amassing a large amount of training data and retraining the system. However, when dealing with real-life text analysis problems, cost-effective ways for pipeline adaptation are often needed. For instance, in the context of growing interest in applying NLP in Estonian digital humanities research (Laak et al., 2019), more fine-grained control over text normalization is required to adapt tools for analyzing old dialects (Pilvik et al., 2019).

In this paper, we present a remastered NLP pipeline for EstNLTK, the design of which allows data-driven methods to be combined with rule-based ones, enabling the end user to customize the pipeline to their needs.

The next important contributions are revised data representation and redesign of the programming interface. While the previous versions of the library were widely used, they were difficult to extend and customize. In this paper, we introduce EstNLTK version 1.6, which addresses these issues explicitly. The remastered basic NLP pipeline consists of many small cleaning steps that resolve common problems that occur in typical real-life texts. All of these steps are easily extendable and researchers can therefore focus on source-specific problems.

Our evaluation on unlabelled data shows that the remastered basic NLP pipeline outperforms both the previous version of the toolkit, as well as state-of-the-art neural models of StanfordNLP on full morphological processing of Estonian.

Finally, we also introduce a new interface for storing, processing and querying text objects in PostgreSQL database which greatly simplifies processing large text collections.

EstNLTK v1.6 is freely available under the GNU GPL version 2 license from `https://github.com/estnltk/estnltk`. The recommended way of installing EstNLTK is by using the precompiled Anaconda packages available at `https://anaconda.org/estnltk/estnltk`. The library runs on Linux, Windows, and Mac OS X platforms, and supports Python versions 3.5 and 3.6. EstNLTK's tutorials are available in the form of interactive Jupyter[1] Notebooks: `https://github.com/estnltk/estnltk/tree/version_1.6/tutorials`

[1] `https://jupyter.org` (accessed 2019-12-02)

2. Related Work

As Python is a popular programming language in NLP and data science communities, there are many Python libraries for NLP. Among them, EstNLTK is most closely related to NTLK (Bird and Klein, 2009) and TextBlob (Loria, 2014) .

NTLK library provides basic text processing, such as tokenization, lemmatization, part-of-speech tagging, and parsing, as well as more advanced tools for building classifiers, extracting information, or using semantic reasoning. EstNLTK uses NTLK's word tokenizer and Estonian sentence tokenization model as a basis, and improves their outcomes by rule-based post-corrections. While EstNLTK also provides a basic text analysis pipeline, the two toolkits diverge notably in the representation of (linguistic) annotations. By design, EstNLTK also allows ambiguities in annotations, which enables experimentation with different disambiguation strategies. We see it as crucial for morphologically rich languages like Estonian.

TextBlob library builds upon NLTK, and makes basic text processing functionalities (and also more advanced functionalities, such as translation) accessible through a central `TextBlob` object, which provides a convenient interface for both initiating text analysis (launching pipelines) and storing / accessing results of the analysis. EstNLTK shares these design principles with TextBlob: we use a central `Text` object, from which the programmer can call for text analyzers (taggers) and access their results (e.g. ask for words, sentences, lemmas or part-of-speech tags).

Most recent Python NLP libraries are built upon deep learning models. AllenNLP (Gardner et al., 2017) focuses on supporting NLP research and allows users to experiment with different NLP architectures, but it can be difficult for a beginner level user to start with. spaCy[2] offers a relatively easy user interface and a fast processing speed, making it suitable for large scale text analysis and product applications; however, it currently lacks support for processing Estonian. To our best knowledge, there are only two multilingual neural NLP pipelines that offer a relatively easy API (suitable for entry-level teaching) and also support Estonian. These are StanfordNLP (Qi et al., 2018) and NLP-Cube (Boroş et al., 2018). However, compared to EstNLTK, these libraries offer only NLP pipelines (and training utilities), but lack the support for customizing the pipeline (e.g. adding custom taggers and making rule-based post-corrections) and providing a convenient interface for database storage.

3. Revised Design Principles

Since the first version of the library (Orasmaa et al., 2016), the basic premise of EstNLTK has stayed the same. EstNLTK library is an extendable collection of NLP utilities which use `Text` objects to communicate with each other. However, practice showed that the original structure of `Text` objects was not easily extendable and we had to rethink how the information is stored and structured.

Text object consists of a raw text string, metadata, and a collection of layers. Layer is a collection of spans together with metadata. Each span has at least one annotation. All annotations in a layer have the same fixed list of attributes.

Annotations. In our library, an analysis component can only add annotations to the original text. Each annotation refers to a span that specifies a text region and a list of attributes. As sentences consist of words and some words consist of several tokens, spans must also reflect this structure. Hence, a span can be specified as a continuous text region or a list of other spans. There can be multiple spans that cover the same text regions. A span can have several annotations, e.g. alternative outcomes of morphological analysis. Such ambiguous annotations are represented as a list of attribute tuples. The latter is needed to capture the relation between attributes.

Layers. Layer is a collection of annotations with the same set of attributes. For instance, the outcome of morphological analysis can be stored by annotating words with lemma, part-of-speech, and form attributes. Different layers may share the same spans. For instance, words are underlying spans in morphological and syntax analysis. These annotations can be combined by indexing different layers with the same spans. However, the information of which annotation pairs are valid if both layers have ambiguous annotations is not captured. If such correspondence is relevant, a single layer with a joint attribute set must be created.

Taggers. All NLP utilities are taggers that take in some layers and create or update a layer. More than forty taggers are included in the EstNLTK library. Approximately a fifth of those are system level taggers which encapsulate code for common tasks, such as annotating text based on regular expression patterns. Users must provide only the configuration and the rest is done by the tagger.

Text collections. The EstNLTK library provides a mechanism for storing and analyzing text corpora. A collection is a set of texts that are stored in a database. The library provides an API for iterating over a subset of texts that match a filter criterion which can be specified in terms of attribute values. It is also possible to run a tagger over the entire collection and compare different layers. This allows the user to inspect how changes in a tagger affect its outcomes.

User interaction. The EstNLTK library tries to balance between two main use cases: interactive data exploration and back-end programming. Tight integration with the Jupyter environment makes it easy to explore the data. There are convenience methods for hiding the underlying complexity. Knowing just a few commands is enough for common tasks. At the same time, there is a parallel API that gives a precise control over the execution without surprises.

4. Pipeline for Standard NLP Tasks

The EstNLTK library can perform a wide range of standard analyses. The corresponding pipeline starts from tokenization and ends with syntax analysis and information extraction. Different steps depend on each other. The information flow is unidirectional and the outcomes of the analysis are not used to update the results of the preceding steps. Correction and disambiguation steps based on the re-

[2] `https://spacy.io` (accessed: 2019-12-01)

verse information flow are not part of the pipeline but they can be added by the user.

4.1. Text Segmentation

Tokens. The new EstNLTK version makes a distinction between words and tokens. Tokens are smallest units form which words are formed, but they also include punctuation separated from words. Most tokens are words but sometimes several tokens form one word, e.g. abbreviations, URLs, and proper names. As the tokenizer does not have to recognize multi-token words it can be simple, i.e. splitting tokens based on whitespace characters.

Words. After the input text has been tokenized, words are formed by using a rule-based combiner that first identifies compound tokens and then normalizes their spelling. For example, a compound token *m-mõ-mõtlema* 't-th-think' (prolonged pronunciation) is corrected to the word *mõtlema* 'think'. The corrected form will be stored as normalized form attribute.
By default, EstNLTK handles hyphenation, common abbreviations, numeric expressions, units of measurement, XML-tags, common emoticons, case and number markers of proper names and numerals. However, the system is easily expandable.

Paragraphs and sentences. Paragraphs are defined through double line-breaks as usual. A pre-trained NLTK `PunktSentenceTokenizer` (which works on raw text) is used to get sentences. But, unlike in previous versions, post-processing guarantees that no words are split between sentences. Additional rules are used to handle common errors caused by direct speech, abbreviations, emoticons, and missing punctuation marks.

4.2. Morphological Analysis

Standard analysis pipeline. EstNLTK uses C++ library Vabamorf (Kaalep and Vaino, 2001) for standard morphological analysis. The corresponding `VabamorfTagger` tagger provides analyses for words and solves the disambiguation when a word has several possible analyses. It works best on standard written language and can serve as off-the-shelf linguistic preprocessing step for applications and experiments. However, non-standard varieties of Estonian, such as Internet slang or transcripts of spoken language require more fine-grained control. Thus, `VabamorfTagger` can be split into three sub-taggers.
The first of them, `VabamorfAnalyzer` is responsible for analyzing all the words in the input, including analyses for unknown words and proper names. Its guessing component for unknown words can be switched off, revealing non-standard words, such as misspelled words or words from a dialect. By design, `VabamorfAnalyzer` provides analyses without disambiguation, so its output layer will be ambiguous. Approximately 45% of word tokens have more than one valid analysis in Estonian texts (Kaalep et al., 2010).
The second component, `PostMorphAnalysisTagger` fixes part-of-speech and case information of numerals (which is often incorrect in Vabamorf's output), and corrects analyses of compound tokens (e.g. fixes part-of-speech tags of emoticons and abbreviations). For instance, if a number has a case ending, such as '10e' in the phrase *10e euroga* 'with 10 euros', then Vabamorf incorrectly analyses it as an abbreviation, and does not assign a correct form (singular genitive) to it. `PostMorphAnalysisTagger` fixes this problem. In addition, the tagger can flag words as "to-be-ignored" by the morphological disambiguation. Words corresponding to emoticons and XML tags are flagged this way.
The third component, `VabamorfDisambiguator` finalizes the morphological analysis process with the Hidden Markov Model based morphological disambiguation, provided by the Vabamorf tool (Kaalep and Vaino, 2001; Kaalep et al., 2010).

Corpus-based morphological disambiguation. There are many word forms in Estonian that can have at least two valid lemmas even if the part-of-speech and grammatical form tags are fixed (Kaalep et al., 2012). For instance, the noun *teod*, which is in plural nominative case, has two valid lemmas: *tigu* 'snail' and *tegu* 'deed'. Kaalep et al. (2012) argue that looking at the local sentence context is not enough for resolving such ambiguity. They propose an algorithm that uses the idea of "one sense per discourse", which originates from word sense disambiguation (Gale et al., 1992) and can be summarized as follows: count all the instances of ambiguous lemmas in the context corpus and choose the most frequent reading (Kaalep et al., 2012). The algorithm is implemented in `CorpusBasedMorphDisambiguator`, which post-corrects morphological analysis and resolves lexical ambiguities based on the lemma frequency information gathered from a user-specified context (a list of texts).
The tagger `VabamorfCorpusTagger` combines `CorpusBasedMorphDisambiguator` and components from `VabamorfTagger` into a single pipeline, which includes analysis and both local context and corpus-based disambiguation. Unlike other taggers, `VabamorfCorpusTagger` operates on a list of `Text` objects, which is both the context used in corpus-based disambiguation for collecting lemma frequency information, as well as the target of disambiguation.

Pipeline based on finite-state transducers. In addition to the standard analysis pipeline, EstNLTK includes `HfstEstMorphAnalyser` which provides morphological analysis based on the Helsinki Finite-State Technology (Lindén et al., 2009; Kaalep et al., 2018). Compared to the standard pipeline, it provides following improvements:

- fine-grained analysis of the structure of compound and inflected words (e.g. lemma, part-of-speech and form information is available for each sub word);
- revised morphological category system, which provides more detailed analyses of verb categories;
- special tags that describe word form's usage characteristics, e.g. whether the form is rare, or commonly used, but not in normative dictionaries;

Fine-grained analysis of compound and inflected words' structure can be useful in information extraction. For instance, analysis of the common noun *kanadalane* 'Canadian' shows that the common noun has been derived from

the proper noun *Kanada* 'Canada', and this information can be used to enhance search of named entity mentions.
Vabamorf's morphological category system uses verb ending morphemes as category names. If the ending morpheme is (syntactically) ambiguous, a single category name is used, e.g. the category 'sid' stands for both 2nd person in singular and 3rd person in plural past indicative mood (Kaalep, 2015). `HfstEstMorphAnalyser`'s category system makes verb categories explicit, distinguishing between voice, tense, mood, person and polarity categories, and, in turn, enables straightforward building of syntactic analysis upon the morphological annotations. Word use characteristics tags can be used in stylometry, e.g the degree to which the word usage follows the language norm.
It must be noted that while `HfstEstMorphAnalyser` provides an alternative to `VabamorfTagger`, the model is still under development. Its guessing component for unknown words has a low coverage and morphological disambiguation has not yet been integrated. Both of these are serious limitations in practice and thus, at its current stage, `HfstEstMorphAnalyser` is an experimental tool.

4.3. Syntactic Analysis

While the first version of EstNLTK (Orasmaa et al., 2016) did not include syntactic analysis, the current version provides access to two dependency syntactic analyzers: rule-based VislCG3 (Karlsson et al., 1995; Müürisep et al., 2003) and statistical MaltParser (Nivre et al., 2006).
As both analyzers need a detailed morphological analysis, `MorphExtendedTagger` is used to add detailed information about verb forms (person, tense, mood, voice, number, polarity), and subtype information for verbs and other word classes (e.g. pronoun types). VislCG3 parser `VislTagger` can be applied directly on the resulting layer while MaltParser needs an additional conversion to CoNLL format (Buchholz and Marsi, 2006) performed by `ConllMorphTagger`.
Both parsers add a syntax layer to the text object that contains the morphological information together with the syntactic function labels and dependency information. We have also implemented several tools to analyze the correctness of automatic syntax labelling. For instance, it is possible to import syntactic information from several CoNLL files into different layers of one text object and calculate a LAS score between those layers.

4.4. Other Tools

EstNLTK also includes less commonly used linguistic analysis tools introduced in the first version: clause segmenter and verb chain detector (Orasmaa et al., 2016). The NLP pipeline ends with information extraction tools: temporal expression tagger (Orasmaa, 2012), named entity recognizer (Tkachenko et al., 2013), and a newly introduced component: grammar-based address recognizer. The library also contains neural morphological disambiguator (Tkachenko and Sirts, 2018) the performance of which is comparable to `VabamorfDisambiguator`.

5. Pipeline for Fact Extraction

Automatic fact extraction is useful only for large data sets where the initial setup cost is small compared to the resources needed for manual processing. Also, it is easy to achieve decent performance with simple methods but it becomes increasingly difficult to handle remaining corner-cases correctly. The latter is true even if we apply state of the art deep learning methods (Li, 2018).
To address these issues, EstNLTK offers taggers for basic tasks, an API for handling large text collections, and a robust framework for building rule-based fact extraction tools. To support iterative development, the library provides tools for regression testing and error analysis, and methods for highlighting differences between alternative algorithms.

5.1. Standard Taggers For Simple Tasks

Almost all rule-based fact extraction algorithms start from token detection. In this context, tokens are the smallest textual units from which phrases are constructed. These could be words, sub-phrases, or sequences of symbols. For instance, if we are interested in Estonian car registration numbers, tokens could be triples of numerals and letters, such as '145', '9 7 6', 'ABC', and 'K F C'. If we need to process a text that contains many typing errors, tokens can be joined with other words or even overlap with each other.
In many cases, `RegexTagger` that provides a simple way to identify tokens with regular expressions is sufficient. The amount of overlapping spans can be reduced by using pattern priorities and prepackaged conflict resolving strategies. The amount of false matches can be reduced by a decorator that uses additional program logic to validate the match and derive the values of annotation attributes.
In some cases, the set of relevant tokens can be specified as a finite list, such as common food item names. `PhraseTagger` fixes a particular attribute such as lemma, and each entry in the list is defined by a tuple of attribute values. Again, additional program logic can be added to filter out false positives and to define the values of annotation attributes.
Many documents consist of subsections that are separated by header lines. `TextSegmentsTagger` allows for the extraction and annotation of these subsections, provided that the headers have been identified and stored in a separate layer.

5.2. Fact Extraction With Finite Grammars

Rule-based fact extraction can be viewed as a compact way to describe all the possible text fragments below a certain length threshold that correspond to a particular fact. Finite grammars are particularly suitable for this as they are simple to write and comprehend, and efficient to parse. Moreover, fingerprint patterns for discarding texts that cannot contain facts can be generated automatically.

Grammar. The component `Grammar` allows the user to specify a finite grammar with two extensions. First, there is a special `SEQ`(α) extension that makes it possible to specify repetitions of the same grammar symbol α. The latter compacts the description of repeating patterns, such as rows in food recipes. Second, for each rule, it is possible to add program logic to validate and decorate matches. This reduces the size of the grammar and increases readability. For example, we do not have to define a special category for

foods that are measured in teaspoons, instead the validator can look up from a large table if a food item and a unit match. Decorators are useful for bottom-up propagation of information that is needed for validation and for computing final annotation attributes.

Ambiguous tokenization and parsing. The true complexity in phrase detection lies in correct tokenization. In theory, tokenization is assumed to produce a list of non-overlapping tokens. In practice, it is not possible to distinguish between numbers and dates without surrounding text, e.g. consider the token 22.03. Moreover, if we use machine-learning methods to identify complex tokens (such as named entities), we have no guarantees for what happens with overlaps. Similar problems arise if we want to treat non-conventional ways of writing numbers. To resolve an ambiguity, we can add disambiguation rules into the grammar and let the parser choose the most consistent tokenization. `GrammarParsingTagger` implements such a robust parsing algorithm by lifting the standard bottom-up CYK parser to the setting where the ordering of tokens is represented by a graph. More precisely, there is an arc $\mathtt{A} \to \mathtt{B}$ if there exists a valid disambiguation $\ldots, \mathtt{A}, \mathtt{B}, \ldots$ without overlapping tokens.

Conflict resolving strategies. To reduce the amount of superfluous phrases a grammar can match in a text, we use rule prioritization and phrase scoring. Rule prioritization allows for blocking some derivation rules when alternatives are present, e.g. to force that '2 kg' is parsed as *measurement* although '2' can be interpreted as a *number*. Phrase scoring allows pruning alternative interpretations of the same phrase. If the same set of tokens has several different parsing trees with the same root node, the parser keeps the one with the highest score. Even with these additions, the parser can still match overlapping phrases. Thus, the user can specify which phrases to keep. Normally, maximal phrases are kept.

5.3. Tools For Iterative Development

The best way to organize the development of a fact extraction pipeline is to store all the texts as an EstNLTK collection and then iteratively develop the set of necessary taggers. Each text object in a collection is stored as a searchable JSON object in a PostgreSQL database. The collection provides an API for applying taggers. When a tagger is applied, a new layer is generated and stored as a separate JSON object. Results can be retrieved by select queries that reassemble the text object from different JSON objects.

Such setup allows the user to iteratively generate all the necessary layers and to measure the progress on a dedicated test collection. For that, the output layers of different taggers can be compared using `DiffTagger` after which either the results can be summarized over the collection or examples of differences can be extracted. Collections have specific API calls for doing that. Regression tests for fixing the expected behaviour on selected examples can be generated in a similar manner.

6. Empirical Validation

We compared the new version of EstNLTK (v1.6.4b) with the old version of EstNLTK (v1.4.1) and with the StanfordNLP's neural NLP pipeline (Qi et al., 2018)[3] on the complete morphological processing, which involved word segmentation, sentence segmentation, and morphological analysis and disambiguation[4].

As the performance scores of all these systems are close to maximum values, it is impossible to use traditional test set for comparison. Statistical fluctuations are much bigger than true performance differences unless the test set size is enormous. To bypass this restriction, we evaluate the systems on unlabelled data and resolve manually a small set of randomly chosen differences. This allows us to estimate the relative differences in performance precisely.

Corpus. Evaluation data was initially taken from the Estonian National Corpus (ENC) (Kallas and Koppel, 2018), which is the largest published collection of Estonian texts so far. However, we discovered errors in one of its subcorpora. The Estonian Reference Corpus (ERC) was missing information about paragraph boundaries, which is crucial for sentence segmentation. Thus, we replaced the ERC with the original version by exporting texts from (Laur, 2018), in which paragraph endings are marked by double newlines.

For the evaluation, we chose documents randomly from the ENC focusing on 5 text types: periodicals, fiction, and science (from the ERC subcoprus), web texts (blogs and forums from the Estonian Web 2013 subcorpus), and Wikipedia articles (from the Estonian Wikipedia 2017 subcorpus). While four of these text types represent standard written language (the analysis of which is the main aim of our pipeline), texts from blogs and forums allow us to evaluate the pipeline's performance on non-standard language.

During the corpus selection process, we excluded documents belonging to the Estonian UD treebank (Muischnek et al., 2016) and the Estonian Web UD treebank (Särg et al., 2018)[5], because these have been used for training language models. For each text type, a subcorpus in size of approx. 2,100,000 tokens[6] was chosen. Table 1 describes corpus structure and statistics.

Evaluation setup. To compare StanfordNLP's linguistic analysis output with that of EstNLTK's, we converted annotations of both tools into a common format. This format uses 3 attributes in each annotation: lemma, part-of-speech tag, and form. In the attributes of part-of-speech and form, we use Vabamorf's categories[7], which are more geared towards expressing Estonian morphological features

[3] We used StanfordNLP's latest Estonian model (v0.2.0).

[4] The source code of our experiments is available at `https://github.com/estnltk/eval_experiments_lrec_2020` (accessed: 2020-02-27)

[5] `https://github.com/UniversalDependencies/UD_Estonian-EDT` (v2.4) and `https://github.com/UniversalDependencies/UD_Estonian-EWT` (v2.4)

[6] Here, we mean tokens that appear on the tokens layer of EstNLTK v1.6 . Tokens also include punctuation separated from words.

[7] For description of the tagset (in Estonian), see: `https://github.com/estnltk/estnltk/blob/version_1.6/tutorials/nlp_pipeline/A_02_morphology_tables.ipynb` (2019-11-21)

Corpus Structure and Statistics						
	Fiction	*Periodicals*	*Science*	*Blogs and forums*	*Wikipedia*	**Total**
Documents	53	5,917	230	3,016	9,270	18,486
Tokens	2,190,173	2,170,290	2,230,925	2,096,558	2,127,617	10,815,563
Word segmentation span similarity ratios						
EstNLTK v1.4 vs v1.6	0.9979	0.9892	0.9713	0.9715	0.9584	0.9780
EstNLTK v1.6 vs StanfordNLP	0.9977	0.9893	0.9638	0.9601	0.9690	0.9763
Sentence segmentation span similarity ratios						
EstNLTK v1.4 vs v1.6	0.8467	0.8845	0.8744	0.9014	0.9542	0.8904
EstNLTK v1.6 vs StanfordNLP	0.7496	0.8297	0.7338	0.8878	0.9125	0.8230
Morphological analysis annotation similarity ratios						
EstNLTK v1.4 vs v1.6	0.9954	0.9874	0.9710	0.9706	0.9595	0.9772
EstNLTK v1.6 vs StanfordNLP	0.8451	0.8268	0.8056	0.8024	0.7823	0.8131

Table 1: Statistics of the evaluation corpus and pairwise agreement scores for tool outputs. EstNLTK v1.6 was compared with EstNLTK v1.4 and with StanfordNLP in terms of word and sentence segmentation (marking segmentation spans) and morphological analysis (adding morphological annotations to words). Morphological analysis matches were calculated only over those words for which both tools found the same spans.

than Universal Dependencies' categories (used by StanfordNLP).

Agreement score. We first estimated the agreement between each pipeline pair. In case of segmentation annotations, we calculated span similarity ratio as

$$r = 2u/(n_1 + n_2) \quad (1)$$

where u is the number of spans same in both outputs and n_1 and n_2 are the span counts for both outputs. In case of morphological annotations, the same formula was used, but u was now defined as the number of annotations for which spans and attribute values were equal and $n_1 + n_2$ was defined as the total number of annotations. Micro-averaging was used: span and annotation counts were summarized over the corpus before calculating ratios.

The results are depicted in Table 1. All the tools have very similar performance on word segmentation. For sentence segmentation, there is a notable disagreement, especially between EstNLTK v1.6 and StanfordNLP. For morphological annotations, the difference between the two EstNLTK's versions is minor, but a notable difference emerges when comparing EstNLTK with StanfordNLP.

Manual evaluation. To obtain a relative ranking between pipelines, we collected all annotation differences and randomly chose 100 differences (20 from each text type) for manual evaluation. The process involved aggregation of differences to make human judgements easier. For segmentation tasks, we aggregated differences into gaps, defining a *gap* as any number of consecutive differences (missing, extra or partially overlapping spans) between two matching spans[8]. For morphological analysis, we grouped annotations by words, so that overlapping and differing morphological annotations of a word formed a single unit of evaluation. As a result, situations arose when, despite the difference, "both tools are correct", because EstNLTK can leave morphological analysis ambiguous. Ambiguous output triggers a "difference" even if there is a pair of matching (and possibly correct) annotations for the word.

Table 2 shows the corresponding results. The evaluation of segmentation differences shows that v1.6 performs notably better than both v1.4 and StanfordNLP on word and sentence segmentation tasks. This indicates that our rule-based improvements have met the target. Note that when comparing StanfordNLP to v1.6 on word segmentation, Table 1 shows only a minor difference between the two pipelines, and yet the manual evaluation (Table 2) reveals a large gap between the annotation quality of the two tools. Our rule-based corrections effectively address rare / infrequent tokenization cases, which would be infeasible to address by neural models due to the need for a large amount of training data. 9-23% uncertain cases in the segmentation evaluation indicate that tasks themselves need further clarifications, e.g. how to segment sentences in the context of the web language with irregular usage of punctuation.

EstNLTK v1.6 also performed better than the alternatives on the task of full morphological analysis. Table 1 shows only a minor difference between outputs of v1.6 and v1.4, and the difference in qualities (Table 2) is again due to addressing/fixing rare cases in v1.6. However, the situation is different when comparing v1.6 with StanfordNLP. The morphological analysis differences become notable already in the automatic comparison (Table 1), and the manual evaluation (Table 2) shows v1.6 outperforming StanfordNLP. When examining StanfordNLP's morphological analysis errors, in most cases there were problems with lemmatization – the neural model was suggesting a wrong

[8]... or between start of the document and the first matching span, or between the last matching span and end of the document.

Manual evaluation of differences between EstNLTK v1.4 and v1.6					
	Both correct	Only v1.6 correct	Only v1.4 correct	Both wrong	Uncertain
Segmentation: words	0	72	7	12	9
Segmentation: sentences	0	61	12	10	17
Morphological analysis	41	32	18	6	3
Manual evaluation of differences between EstNLTK v1.6 and StanfordNLP					
	Both correct	Only EstNLTK v1.6 correct	Only StanfordNLP correct	Both wrong	Uncertain
segmentation: words	0	69	17	2	12
segmentation: sentences	0	51	16	10	23
morphological analysis	42	24	12	4	18

Table 2: Results of the manual evaluation of 100 randomly picked annotation differences.

Morphological disambiguation similarity ratios						
	Fiction	*Periodicals*	*Science*	*Blogs and forums*	*Wikipedia*	**Total**
Default vs corpus-based disambiguation	0.9790	0.9804	0.9790	0.9882	0.9774	0.9807
Default vs neural disambiguation	0.9095	0.9186	0.9154	0.9099	0.9239	0.9154
Manual evaluation of 100 randomly picked differences						
	both correct	only default correct	only improved method correct	both wrong	uncertain	
Default vs corpus-based disambiguation	18	19	45	7	11	
Default vs neural disambiguation	65	18	16	0	1	

Table 3: Automatic comparison and manual evaluation of EstNLTK's disambiguation tools.

lemma, even if the grammatical tags were guessed correctly. Due to the Zipfian distribution of word lemmas, the neural model needs (possibly infeasibly) large amounts of data to learn correct the lemmatization of rare words. However, EstNLTK's Vabamorf-based lemmatizer, which combines a lexicon with derivation rules, maintains relatively stable performance even on rare words.

Enhanced morphological disambiguation. We also measured the performance of the two new disambiguation components: the corpus-based morphological disambiguator (`VabamorfCorpusTagger`) and the neural morphological disambiguator (`SoftmaxEmbCatSumTagger`). As the neural model outputs only grammatical information and uses Universal Dependencies' categories, annotations of the tools under comparison were again converted into a common format. In this format, an annotation included only 2 attributes: part-of-speech tag and form in Vabamorf's category system. Table 3 compares the standard morphological disambiguation (provided by `VabamorfTagger`) to alternative (corpus-based and neural) disambiguation methods. Results show that corpus-based disambiguation[9] introduces only minor differences compared to the standard one (very high similarity ratios), but still outperforms the standard one with its higher number of correct analyses. On the other hand, neural morphological disambiguation is notably dissimilar from the standard disambiguation, but if we consider the manual quality evaluation, the performance difference is insignificant. Moreover, the neural model outputs only grammatical information and does not resolve lemma ambiguities discussed by Kaalep et al. (2012). Hence, if obtaining correct lemmas is also important, then `VabamorfTagger` and its successor `VabamorfCorpusTagger` are still better choices.

7. Conclusion

We presented a remastered version of the Estonian NLP pipeline in EstNLTK library. We introduced library's revised design principles, modifications and enhancements to the standard pipeline, as well as new tools for fact extraction and iterative tagger development. The work concluded with an empirical evaluation, showing that our remastered morphological processing pipeline improves upon the previous version of EstNLTK, and outperforms the Stanford NLP pipeline for Estonian.

[9]The context used in the corpus-based disambiguation was the document under analysis.

8. Acknowledgements

ESTNLTK has been funded by Eesti Keeletehnoloogia Riiklik Programm (National Programme for Estonian Language Technology) under the projects EKT110 and EKTB14, and supported by Estonian Ministry of Education and Research (grant IUT 20-56 "Computational models for Estonian").

9. Language Resource References

Kallas, J. and Koppel, K. (2018). Eesti keele ühendkorpus 2017 (*Estonian National Corpus 2017*). https://doi.org/10.15155/3-00-0000-0000-0000-071E7L.

Laur, S. (2018). Eesti keele koondkorpus analüüsitud estnltk v1.6.b abil (*Estonian Reference Corpus analysed with EstNLTK v1.6.b*). https://doi.org/10.15155/1-00-0000-0000-0000-00156L.

10. Bibliographical References

Bird, Steven, E. L. and Klein, E. (2009). *Natural Language Processing with Python*. O'Reilly Media Inc.

Boroș, T., Dumitrescu, S. D., and Burtica, R. (2018). NLP-Cube: End-to-end raw text processing with neural networks. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 171–179.

Buchholz, S. and Marsi, E. (2006). CoNLL-X Shared Task on Multilingual Dependency Parsing. In *Proceedings of the tenth conference on computational natural language learning*, pages 149–164. Association for Computational Linguistics.

Gale, W. A., Church, K. W., and Yarowsky, D. (1992). One sense per discourse. In *Proceedings of the workshop on Speech and Natural Language*, pages 233–237. Association for Computational Linguistics.

Gardner, M., Grus, J., Neumann, M., Tafjord, O., Dasigi, P., Liu, N. F., Peters, M., Schmitz, M., and Zettlemoyer, L. S. (2017). AllenNLP: A Deep Semantic Natural Language Processing Platform.

Kaalep, H.-J. and Vaino, T. (2001). Complete morphological analysis in the linguist's toolbox. *Congressus Nonus Internationalis Fenno-Ugristarum Pars V*, pages 9–16. The corresponding C++ code is available from https://github.com/Filosoft/vabamorf.

Kaalep, H.-J., Muischnek, K., Uiboaed, K., and Veskis, K. (2010). The estonian reference corpus: Its composition and morphology-aware user interface. In *Baltic HLT*, pages 143–146.

Kaalep, H.-J., Kirt, R., and Muischnek, K. (2012). A trivial method for choosing the right lemma. In *Baltic HLT*, pages 82–89.

Kaalep, H.-J., Moshagen, S. N., and Trosterud, T. (2018). Estonian morphology in the giella infrastructure. In *Baltic HLT*, pages 47–54.

Kaalep, H.-J. (2015). Eesti verbi vormistik *(Estonian verb paradigm)*. *Keel ja Kirjandus*, 58(01):1–15. (in Estonian with English summary).

Karlsson, F., Voutilainen, A., Heikkilä, J., and Anttila, A. (1995). *Constraint Grammar: A Language-Independent System for Parsing Unrestricted Text*. 01.

Laak, M., Veskis, K., Gerassimenko, O., Kahusk, N., and Vider, K. (2019). Literary Studies Meet Corpus Linguistics: Estonian Pilot Project of Private Letters in KORP. In *DHN*, pages 283–294.

Li, H. (2018). Deep learning for natural language processing: Advantages and challenges. *National Science Review*, 5:24–26, 01.

Lindén, K., Silfverberg, M., and Pirinen, T. (2009). Hfst tools for morphology–an efficient open-source package for construction of morphological analyzers. In *International Workshop on Systems and Frameworks for Computational Morphology*, pages 28–47. Springer.

Loria, S. (2014). TextBlob: Simplified Text Processing. *TextBlob. Np*.

Muischnek, K., Müürisep, K., and Puolakainen, T. (2016). Estonian dependency treebank: From constraint grammar tagset to universal dependencies. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 1558–1565.

Müürisep, K., Puolakainen, T., Muischnek, K., Koit, M., Roosmaa, T., and Uibo, H. (2003). A new language for constraint grammar: Estonian. In *Proceedings of the International Conference Recent Advances in Natural Language Processing RANLP 2003*, pages 304–310.

Nivre, J., Hall, J., and Nilsson, J. (2006). MaltParser: A data-driven parser-generator for dependency parsing. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC'06)*, Genoa, Italy, May. European Language Resources Association (ELRA).

Nivre, J., De Marneffe, M.-C., Ginter, F., Goldberg, Y., Hajic, J., Manning, C. D., McDonald, R., Petrov, S., Pyysalo, S., Silveira, N., et al. (2016). Universal dependencies v1: A multilingual treebank collection. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 1659–1666.

Orasmaa, S., Petmanson, T., Tkachenko, A., Laur, S., and Kaalep, H.-J. (2016). EstNLTK - NLP Toolkit for Estonian. In Nicoletta Calzolari (Conference Chair), et al., editors, *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*, Paris, France, may. European Language Resources Association (ELRA).

Orasmaa, S. (2012). Automaatne ajaväljendite tuvastamine eestikeelsetes tekstides *(Automatic Recognition and Normalization of Temporal Expressions in Estonian Language Texts)*. *Eesti Rakenduslingvistika Ühingu aastaraamat*, (8):153–169. (in Estonian with English summary).

Pilvik, M.-L., Muischnek, K., Jaanimäe, G., Lindström, L., Lust, K., Orasmaa, S., and Türna, T. (2019). Möistus sai kuulotedu: 19. sajandi vallakohtuprotokollide tekstidest digitaalse ressursi loomine (*Creating A Digital Resource From 19th Century Communal Court Minute Books*). *Eesti Rakenduslingvistika Ühingu aastaraamat*, 15:139–158. (in Estonian with English summary).

Qi, P., Dozat, T., Zhang, Y., and Manning, C. D. (2018). Universal dependency parsing from scratch. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 160–170, Brussels, Belgium, October. Association for Computational Linguistics.

Särg, D., Muischnek, K., and Müürisep, K. (2018). Annotated clause boundaries' influence on parsing results. In *International Conference on Text, Speech, and Dialogue*, pages 171–179. Springer.

Tkachenko, A. and Sirts, K. (2018). Neural morphological tagging for estonian. In Kadri Muischnek et al., editors, *Human Language Technologies - The Baltic Perspective - Proceedings of the Eighth International Conference Baltic HLT 2018, Tartu, Estonia, 27-29 September 2018*, volume 307 of *Frontiers in Artificial Intelligence and Applications*, pages 166–174. IOS Press.

Tkachenko, A., Petmanson, T., and Laur, S. (2013). Named entity recognition in estonian. In *Proceedings of the Workshop on Balto-Slavic NLP*, page 78. Association for Computational Linguistics.

Yusupujiang, Z. (2018). Using Unsupervised Morphological Segmentation to Improve Dependency Parsing for Morphologically Rich Languages. `http://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1221345`, accessed: 2019-11-29.

Zeman, D., Hajič, J., Popel, M., Potthast, M., Straka, M., Ginter, F., Nivre, J., and Petrov, S. (2018). CoNLL 2018 shared task: multilingual parsing from raw text to universal dependencies. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 1–21.