UNIVERSITY OF TARTU
Institute of Computer Science
Software Engineering Curriculum

Nusaeb Nur Alam

The Comparative Evaluation of Dependency Parsers in Parsing Estonian

Master's Thesis (30 ECTS)

Supervisor(s): Kairit Sirts

Tartu 2017

The Comparative Evaluation of Dependency Parsers in Parsing Estonian

Abstract:

Natural Language Processing (NLP) technology has been constantly developing and has seen a vast improvement in the last couple of decades. One key task in NLP is dependency parsing that oftentimes is a pre-requisite for many other tasks such as machine translation, Named Entity Recognition (NER) and so on. The idea of dependency parsing is to perform a syntactic analysis of a sentence and extract the grammatical relations among the words in that sentence. Most research on dependency parsing has been focusing on English text parsing. In this thesis, an effort has been made to evaluate and compare the performance of some of the state-of-the-art dependency parsers in parsing Estonian. The dependency parsers chosen for evaluation are: MaltParser, spaCy, Stanford neural network dependency parser (nndep), SyntaxNet and UDPipe. The comparison is done using mainly Labelled Attachment Score (LAS), Unlabelled Attachment Score (UAS) and Label Accuracy (LA). New models for Estonian were trained for the spaCy, Stanford nndep and UDPipe parsers while pre-trained models for the MaltParser and SyntaxNet were used in the experiments.

Keywords:

Estonian Dependency Parsing, Natural Language Processing (NLP)

CERCS: P170 - Computer science, numerical analysis, systems, control

[Kommentaarid]

Sõltuvusüntaksi analüsaatorite võrdlus eesti keele süntaksi analüüsimiseks

Lühikokkuvõte:

Loomuliku keele töötluse (LKT) tehnoloogia on pidevalt arenemas, viimastel kümnenditel on selles valdkonnas toimunud väga suured edasiminekud. Üks LKT põhiülesanne on sõltuvussüntaksi analüüs, mis on sageli aluseks ka paljudele teistele ülesannetele, näiteks masintõlkele, nimeolemite tuvastamisele jne. Sõltuvussüntaksi analüüsi eesmärgiks on leida lause süntaktiline struktuur ja tuvastada sõnadevahelised grammatilised seosed. Enamik sõltuvussüntaksi analüüsi uuringuid on keskendunud inglise keele analüüsimisele. Antud magistritöö eesmärgiks on hinnata ja võrrelda erinevate süntaksianalüsaatorite tulemuslikkust eesti keele analüüsimisel. Võrdlusesse valitud sõltuvussüntaksi analüsaatorid on: MaltParser, spaCy, Stanford'i neuroanalüsaator (nndep), SyntaxNet ja UDPipe. Hindamiseks kasutati peamiselt märgendatud seoste täpsust (*Labelled Attachment Score*), märgendamata seoste täpsust (*Unlabelled Attachment Score*) ning märgenduse täpsust (*Label Accuracy*). Magistritöö käigus treeniti spaCy, Stanfordi neuroparseri ning UDParseri mudelid eesti keele süntaksi analüüsimiseks, MaltParseri ja SyntaksNet'i jaoks kasutati eksperimentides olemasolevaid eeltreenitud mudeleid.

Võtmesõnad:

Eesti keele sõltuvussüntaksi analüüs, loomuliku keele töötlus

CERCS: P170 - Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Table of Contents

Glossary

DEPENDENT - In a dependency relation between two words, DEPENDENT is normally the modifier, object or complement of the HEAD of the pair

DEPREL - Dependency relation between HEAD and its DEPENDENTs

HEAD - In a dependency relation between two words, HEAD is the word that gets modified by the DEPENDENT and usually determines the behaviour of the word pair

LA - Label Accuracy; percentage of tokens with DEPREL being assigned correctly

LAS - Labelled Attachment Score; percentage of tokens with both HEAD and DEPREL being assigned correctly

NER - Named Entity Recognition

Nndep - Neural network dependency parser

SD - Stanford Dependencies

UAS - Unlabelled Attachment Score; percentage of tokens with HEAD being assigned correctly

UD - UniversalDependencies; treebank annotation scheme available cross-linguistically consistent for many languages

1 Introduction

Natural Language Processing (NLP) has seen an enormous progress in the recent decades. A combined effort and contribution from individual researchers, numerous research groups as well as big technology companies are driving the success of NLP development. Work on natural language text processing started in the 1950s when Alan Turing published his paper "Computing Machinery and Intelligence" introducing the idea of Turing Test for the first time (Turing, 1950). Then, in 1954, the Georgetown-IBM experiment was performed in which sixty Russian sentences were machine translated into English (Dostert, 1955). Natural language processing was revolutionised in the 1980s with the introduction of machine learning algorithms in language modelling for speech recognition (Bahl et al., 1983).

The focus of this thesis is on dependency parsing which is an important task in natural language processing. In dependency parsing, a syntactic analysis of a sentence is performed to find the grammatical relations between the words within the sentence and a parse tree is generated which is a directed graph showing the relationships among the words. Dependency parsing is required to understand the true meaning of the sentence as a sentence could be interpreted in multiple ways. Dependency parsing is a preliminary step for many NLP tasks, such as machine translation (Hutchins and Somers, 1992), Named Entity Recognition (NER) (Nadeau and Sekine, 2007), Relation Extraction (Agichtein and Gravano, 2000). Language translators, chatbots, and similar software products are real-life examples where dependency parsing lies in the core of the functionality.

Most research on dependency parsing has typically been done in English. In terms of parsing Estonian, there have been two efforts made in the past in developing a syntactic dependency parser, one is based on Constraint Grammar (CG) framework (Karlsson et al., 1995) and the other one is statistical parser using MaltParser (Nivre et al., 2006). The success of the CG based parser contributed in the development of the first version of Estonian Dependency Treebank (Muischnek et al., 2014b). Considering all the advancements achieved in English language parsing, one natural question is whether Estonian language technology can gain some benefits from these advancements or not. This also constitutes the research question of this thesis stated below:

Can the state-of-the-art dependency parsers be used off-the-shelf to parse Estonian text while maintaining high performance in terms of parsing accuracy?

To find an answer to this question, the performance of the five different parsers listed below in parsing Estonian is being evaluated.

- MaltParser (Nivre, Hall, and Nilsson, 2006)
- spaCy (Honnibal, Goldberg, and Johnson, 2013)
- Stanford neural network dependency parser (nndep) (Chen and Manning, 2014)
- SyntaxNet parser (Andor et. al, 2016)
- UDPipe (Straka et al., 2016)

The reason Stanford neural network dependency parser (nndep) and spaCy systems were chosen is because these are widely used in the NLP industry and state-of-the-art in English. For SyntaxNet, it is a novel NN-based model that is competitive on parsing English and provides pre-trained models for many languages. MaltParser was picked because it has been trained on Estonian before using optimised configuration particularly suitable for parsing Estonian.

All these parsers are data-driven and assume the presence of an annotated training set. In principle, these parsers can be trained on any language even though most of these are heavily

tested on English along with few other languages. In particular, MaltParser was initially evaluated on Swedish, English, Czech, Danish, and Bulgarian. The latest version of spaCy (version 1.8) supports English, German and French[1]. The Stanford parser supports Chinese besides English[2]. SyntaxNet has pre-trained models for over 40 languages, including Estonian, which can be used off-the-shelf for parsing text of the respective language[3].

Two of these five parsing systems have already been trained on Estonian but their parsing accuracies have not been systematically compared before. Thus, the models for the other three parsers were trained and the results of all five parsers are compared in this thesis. All models are trained on the same UD treebank training set and evaluated on the same test set, so that the results of all models, both those that were pre-trained and those that are trained by the author, are directly comparable.

This thesis is structured in the following manner:

Chapter 2 first presents an overview of dependency parsing (in particular, transition-based dependency parsing), then describes the five parsers used in this work and finally gives the background history of dependency parsing of Estonian text.

Chapter 3 describes the datasets and evaluation measures used in training the new models, using the pre-trained models and evaluating their performance.

Chapter 4 documents the procedure of training the spaCy, Stanford neural network dependency parser (nndep), and UDPipe models, and provides a guide to use the existing MaltParser and SyntaxNet models.

Chapter 5 reports an evaluation and analysis of the parsing results obtained from the four parsers and discusses the result in the light of the posed research question.

Chapter 6 draws the conclusion of the whole work.

[1] https://github.com/explosion/spaCy

[2] https://nlp.stanford.edu/software/nndep.shtml

[3] https://github.com/tensorflow/models/tree/master/syntaxnet

2 Background

This chapter briefly describes dependency parsing of natural languages, in particular, transition-based dependency parsing. This description is followed by an overview of the five dependency parsers used for this thesis work. The configuration, transition systems, algorithms are explained shortly. The chapter ends with a background history of dependency parsing of the Estonian language.

2.1 Dependency Parsing

There are two methods of parsing text which are quite popular. One is dependency parsing, which focuses on representing grammatical relations between words in a sentence (Kübler et al., 2009) and the other is constituency parsing which breaks down a sentence into sub-phrases and generates a phrase-structure tree where nodes represent the phrases and leaves are the words in the sentence (Charniak, 1997). The focus of this thesis is on dependency parsing to evaluate some of the state-of-the-art dependency parsers in parsing Estonian text.

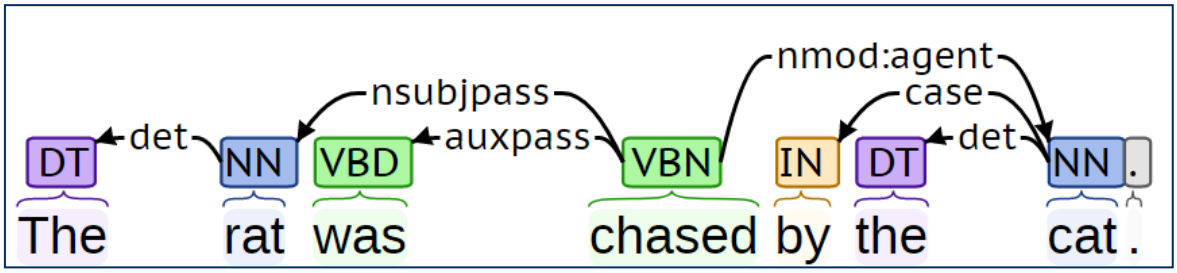

Figure-1: An example of a dependency graph generated using the online Stanford CoreNLP Demo[4]

All the words in a sentence are connected to each other with some grammatical relations like 'subject', 'modifier', 'determiner', and so on. These relations are known as dependency relations as these express how one word is dependent on another word. In a dependency relation between two words, one is called DEPENDENT, which generally acts as modifier, object or complement of the other word, known as HEAD. Figure-1 gives an example of a dependency tree (which is basically a directed acyclic graph with arcs pointing from the HEAD to the DEPENDENT). The arc-labels (also known as attachments) represent the dependency relations. For example, in the sentence *'The rat was chased by the cat'*, determiner (DT) 'The' modifies the noun (NN) 'rat'. Thus, 'rat' is the HEAD and 'The' is the DEPENDENT in the dependency relationship of 'det'(determiner) between the word pair. Generally, the principal verb of the sentence serves as the root of the tree, which is 'chased' in the given example.

[4] http://nlp.stanford.edu:8080/corenlp/process

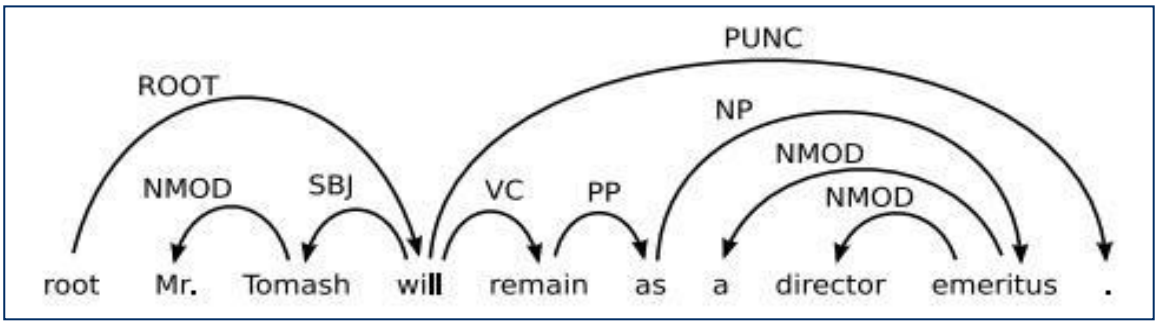

Figure-2: A projective dependency graph (McDonald and Satta, 2007)

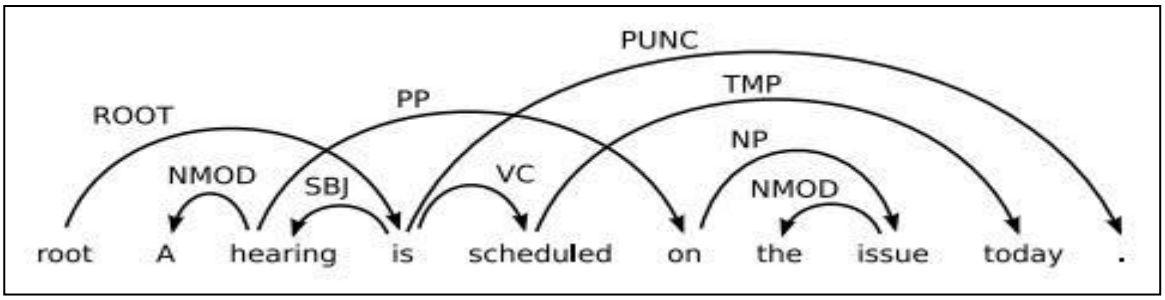

Figure-3: A non-projective dependency graph (McDonald and Satta, 2007)

Dependency trees can be of two types, projective and non-projective. In a projective graph, if the words are put in their linear order with the root in the very beginning, then the edges can be drawn in the plane above the sentence without two edges crossing each other. This property does not hold for non-projective trees. Long distance dependencies or free word order of a language could contribute in non-projectivity (McDonald and Satta, 2007). The projectivity of a sentence is of importance as the transition-based dependency parsers can parse only projective sentences, so parses of any non-projective sentence probably would contain some error and influence the performance of the parsers.

Figure-2 and Figure-3 give examples of a projective dependency graph and a non-projective dependency graph, respectively. Both the sentences are adopted from (McDonald and Satta, 2007).

2.1.1 Transition-based Dependency Parsing

A popular method for dependency parsing is transition-based parsing which aims to derive a dependency parse tree by predicting a transition sequence from an initial configuration to some terminal configuration. In every step during parsing, the most probable transition is chosen based on the current configuration available to the parser.

The algorithms for transition systems can be categorised into two families- stack-based, and list-based. Stack-based algorithms are restricted to projective dependency structures while list-based algorithms can work with both projective and non-projective dependency structures. All parsers used for this thesis employ either of the two popular stack-based transition systems- arc-standard transition system (Nivre, 2004) and arc-eager transition system (Nivre, 2003).

Adopting the notation from Nivre (2008), a stack-based configuration for a sentence $x = (w_0, w_1, \ldots, w_n)$ is a triple $c = (\sigma, \beta, A)$, where

1. σ is a stack of tokens $i \leq k$ (for some $k \leq n$),
2. β is a buffer of tokens $j > k$,

3. A is a set of dependency arcs such that $G = (\{0, 1, \dots, n\}, A)$ is a dependency graph for x.

An initial configuration for the sentence *x* would look like the following:

$\sigma = [ROOT]$
$\beta = [w_0, w_1, \dots, w_n]$
$A = \{\}$

Here, $ROOT$ is an artificial node that represents the root of the graph. Both the stack and buffer are represented as lists. Thus, $\sigma|i$ represents a stack with top i and tail σ, and $j|\beta$ represents a buffer with head j and tail β[5].

A stack-based transition system is a quadruple $S = (C, T, c_s, C_t)$, where

1. C is the set of all possible stack-based configurations;

2. $c_s\big(x = (w_0, w_1, \dots, w_n)\big) = ([0], [1, \dots, n], \emptyset)$, is the initial configuration for the sentence x where $w_1, \dots, w_n$ denote the *n* number of words the sentence is consisted of and w_0 represents the artificial ROOT node. [0] is the initial stack containing only the artificial root node 0, the buffer [1, …, n] contains all the words in linear order and $\emptyset$ denotes an empty set of the dependency arcs;

3. T is a set of possible transitions, each of which is a function $t : C \rightarrow C$, where the function *t* takes a configuration C_{in} as input and outputs the configuration C_{out} resulting from performing the transition;

4. $C_t = \{c \in C | c = ([0], [\,], A)\}$, is the terminal configuration where the stack only contains the artificial ROOT node, the buffer is empty and the set of dependency arcs contains the labelled dependency arcs.

Three kinds of data are always available to the parser; a partial parse built so far, a stack containing already processed words and a buffer of words yet to be processed. The transitions are being applied to the parser's states until the buffer is empty and a complete parse is being generated.

2.1.1.1 Arc-standard Transition System

Following transitions can be applied in the arc-standard system:

- **LEFT-ARC:** For a dependency label l, add a dependency arc (j, l, i) to A, where i is the node on top of the stack σ and j is the first node in the buffer β; then, pop the stack σ.
  Pre-condition: Token i cannot be the dummy ROOT node and must not have been assigned a HEAD yet.

- **RIGHT-ARC:** For any dependency label l, add a dependency arc (i, l, j) to A, where i is the node on top of the stack σ and j is the first node in the buffer β; then, pop the stack σ and replace j by i at the head of β.
  Pre-condition: Token j must not have been assigned a HEAD yet.

[5] The operator | is taken to be left-associative for the stack and right-associative for the buffer.

Table-1: Transitions and preconditions of the arc-standard and the arc-eager transition systems

	Arc-standard Transition System	**Arc-eager Transition System**
Transitions		
LEFT-ARC	$(\sigma\|i, j\|\beta, A) \Rightarrow (\sigma, j\|\beta, \mathrm{A} \cup \{(\mathrm{j}, \mathrm{l}, \mathrm{i})\})$	$(\sigma\|i, j\|\beta, A) \Rightarrow (\sigma, j\|\beta, \mathrm{A} \cup \{(\mathrm{j}, \mathrm{l}, \mathrm{i})\})$
RIGHT-ARC	$(\sigma\|i, j\|\beta, A) \Rightarrow (\sigma, i\|\beta, \mathrm{A} \cup \{(\mathrm{i}, \mathrm{l}, \mathrm{j})\})$	$(\sigma\|i, j\|\beta, A) \Rightarrow (\sigma\|i\|j, \beta, \mathrm{A} \cup \{(\mathrm{i}, \mathrm{l}, \mathrm{j})\})$
SHIFT	$(\sigma, i\|\beta, A) \Rightarrow (\sigma\|i, \beta, \mathrm{A})$	$(\sigma, i\|\beta, A) \Rightarrow (\sigma\|i, \beta, \mathrm{A})$
REDUCE	-	$(\sigma\|i, \beta, A) \Rightarrow (\sigma, \beta, \mathrm{A})$
Preconditions		
LEFT-ARC	$\neg[i = 0]$ $\neg\exists k \exists l'[(k, l', i) \in A]$	$\neg[i = 0]$ $\neg\exists k \exists l'[(k, l', i) \in A]$
RIGHT-ARC	$\neg\exists k \exists l'[(k, l', j)$	$\neg\exists k \exists l'[(k, l', j) \in A]$
REDUCE	-	$\exists k \exists l[(k, l, i) \in A]$

- **SHIFT:** Removes the first node i in the buffer β and pushes it on top of the stack σ.

2.1.1.2 Arc-Eager Transition System

Arc-eager system employs the same configuration and follows a similar method of working as the arc-standard system except that, it stores both the HEADs and the DEPENDENTs in the stack for further processing and the DEPENDENT word is being popped later by the REDUCE transition.

- **LEFT-ARC:** For any dependency label l, add a dependency arc (j, l, i) to A, where i is the node on top of the stack σ and j is the first node in the buffer β; then, pop the stack σ.
  Pre-condition: Token i cannot be the dummy ROOT node and must not have been assigned a HEAD yet.
- **RIGHT-ARC:** For any dependency label l, add a dependency arc (i, l, j) to A, where i is the node on top of the stack σ and j is the first node in the buffer β; then, remove the first node j in the buffer β and push it to the top of the stack σ.
  Pre-condition: Token j must not have been assigned a HEAD yet.
- **REDUCE:** Pops the stack σ.

  Pre-condition: The top token of the stack σ must already have a HEAD.
- **SHIFT:** Removes the first node i in the buffer β and pushes it on top of the stack σ.

Some key differences between the arc-standard and arc-eager transition systems are (Nivre, 2013):

- Arc-standard system builds the parse tree in a bottom-up manner. It means, to add an arc between two nodes, the DEPENDENT node must have already found all its DEPENDENTs. It introduces a sort of non-determinism as it is often necessary to postpone the attachment of the *right* DEPENDENT. In the case of the arc-eager system, an arc is always added at the earliest possible opportunity, thus, building the tree in a top-down fashion.
- Termination of the arc-eager system does not depend on the condition of the stack, it terminates as soon as the buffer is empty. On the other hand, the arc-standard system terminates if and only if, the buffer is empty and the stack only has the dummy *ROOT* node left in it.
- Arc-eager system has one extra transition, *REDUCE,* which is not available in the arc-standard system.

2.1.2 Oracle

Oracle is an important part of transition-based parsers; given a gold tree for a sentence, an oracle is used for predicting an optimal sequence of transitions that will derive the gold tree. Oracles can be categorized into two classes- static oracles and dynamic oracles.

2.1.2.1 Static Oracle

Generally, oracles are designed as functions from trees to sequences, which map a single set of actions to a gold tree. In a static oracle, there are rules specified based on which a single static sequence of transitions is being produced. Thus, this type of oracles is known as static oracles (Goldberg and Nivre, 2012). One drawback of static oracles in greedy dependency parsing is that the parser often gets deviated from the gold sequence and reaches configurations which could not lead to the correct tree. It makes the parser's classifier to deal with configurations unknown to it and eventually moves toward a sequence of errors. To overcome this obstacle, Goldberg and Nivre (2012), introduced the concept of a dynamic oracle.

2.1.2.2 Dynamic Oracle

A dynamic oracle permits all valid transition sequences leading to the gold tree instead of forcing a single transition sequence in the case of static oracles. Another crucial characteristic of a dynamic oracle is, it is well-defined and correct for all configurations even if some of the configurations do not reach the gold tree. In such cases, the oracle permits all the transitions leading to a tree with minimum loss compared to the gold tree.

2.2 Parsing Systems

In this section, the dependency parsers used in this thesis, namely Stanford nndep, spaCy, SyntaxNet, MaltParser, and UDPipe, are briefly described. Explanation of the algorithms, feature models, transition systems related to the parsers is given.

2.2.1 MaltParser

MaltParser is a transition-based parser that implements several parsing algorithms, including the arc-standard and the arc-eager transition systems (Nivre, Hall, and Nilsson, 2006). There are two built-in learners in MaltParser since version 1.3, LIBSVM (Chang, Lin, 2011) and LIBLINEAR (Fan et al., 2008). LIBSVM is a library for Support Vector Machines which can perform support vector classification, regression, and distribution estimation. It

also supports multi-class classification[6]. LIBLINEAR is a machine learning package for linear classification. The default learning method is LIBSVM.

The feature model used in MaltParser consists of POS tags, dependency relations (DEPREL) and lexical features (LEX). It considers:

- Part-of-speech features of the first two tokens in the STACK and first three tokens in the BUFFER;
- Dependency features of the top token in the STACK, its leftmost and rightmost DEPENDENTs and the first token in the BUFFER;
- Lexical features of the top token in the STACK, its HEAD and first two tokens in the BUFFER.

A parsing model for Estonian using MaltParser has been developed before, which has been used in this thesis to evaluate and compare its performance with the other parsers.

2.2.2 Stanford Neural Network Dependency Parser

Stanford neural network dependency parser (Chen and Manning, 2014) implements the arc-standard transition system and employs greedy parsing technique. The parser uses a feed-forward neural network classifier and a dynamic oracle at each state to decide among the transitions. The classifier predicts the correct transition based on the features extracted from the configurations available to the parser at that particular state and chooses the highest scoring transition.

In a transition based model, dependency trees are constituted by following certain transition sequences. There could be several possible sequences that would lead to the same tree, which makes it necessary to find the highest-scoring sequence. In greedy transition-based parsing, the highest-scoring transition from the current configuration is being applied repeatedly until a terminating configuration is reached.

Stanford nndep parser uses word embeddings that represent each word as a d-dimensional vector $e_i^w \in \mathbb{R}^d$ and the full embedding matrix is $E^w \in \mathbb{R}^{d \times N_w}$ where N_w is the dictionary size. Chen and Manning (2014) also introduced dense feature embeddings by mapping POS tags and arc labels to a d-dimensional vector space, where $e_i^t, e_j^l \in \mathbb{R}^d$ are the representations of the i^{th} POS tag and the j^{th} arc label. Correspondingly, the POS and label embedding matrices are $E^t \in \mathbb{R}^{d \times N_t}$ and $E^l \in \mathbb{R}^{d \times N_l}$ where N_t and N_l are the numbers of distinct POS tags and arc labels.

A set of features is chosen based on the stack/buffer position for each type of information, namely word, POS and label, which are denoted as S^w, S^t, S^l respectively.

S^w contains n_w = 18 elements:

1. The top 3 words on the stack and buffer: $s_1, s_2, s_3, b_1, b_2, b_3$;
2. The first and second leftmost/rightmost children of the top two words on the stack:
$$lc_1(s_i), rc_1(s_i), lc_2(s_i), rc_2(s_i), i = 1, 2$$
3. The leftmost of leftmost/rightmost of rightmost children of the top two words on the stack:
$$lc_1\big(lc_1(s_i)\big), rc_1\big(rc_1(s_i)\big), i = 1, 2$$

[6] https://www.csie.ntu.edu.tw/~cjlin/libsvm/

S^t contains n_t = 18 elements, representing the corresponding POS tags, and S^l ($n_l = 12$) holds the corresponding arc labels of words excluding those 6 words on the stack/buffer.

As per the author's knowledge, no attempts have been made before in developing a Stanford nndep model to parse Estonian. This is the first ever experiment where such a model is developed and having its performance evaluated and compared with the models of some other parsers.

2.2.3 spaCy Parser

The spaCy parser (Honnibal, Goldberg, Johnson, 2013) implements the arc-eager transition system along with the use of dynamic oracle and greedy parsing technique described in the section 2.2.2.

An important aspect of the spaCy parser is the implementation of non-monotonic state transition. Monotonicity of the arc-eager transition system can be defined as the consistency of the actions with respect to the previous action. It ensures the single HEAD constraint of the arc-eager system where exactly one HEAD is assigned to each word. This naturally forms a relationship between the Right-arc and Reduce actions and, Shift and Left-arc actions, in a sense that, a word must be popped from the stack using Reduce which was pushed by Right-arc while a word pushed by Shift action must be popped out by the Left-arc action. The Right-arc and the Shift moves determine if the position of the HEAD would be to the left or to the right relative to the pushed token, respectively. It often happens that, the next move is being decided in a state where information about the continuation of the sentence is missing and leads to a wrong HEAD assignment. Honnibal et al. (2013) suggested a non-monotonic version of the arc-eager transition system that allows the parser to correct HEAD assignments done previously incorrectly. That means, it can overwrite an arc attachment made by an earlier move.

spaCy uses an averaged Perceptron learner (Collins, 2002) and the extended feature set described in (Zhang and Nivre, 2011). If the first token (top) in the STACK is denoted by S_0, its HEAD S_{0h}, leftmost/rightmost DEPENDENT by S_{0l}, S_{0r} respectively, the first three tokens in the BUFFER by N_0, N_1, N_2 and the leftmost DEPENDENT of N_0 by N_{0l}, then the baseline feature template consists of:

- POS tag of S_0, S_{0h}, S_{0l}, S_{0r}, N_0, N_1, N_2, N_{0l};
- Word form of S_0, N_0, N_1, and N_2.

The extended feature set additionally includes the following features:

- Distance between S_0 and N_0: The word form and the POS tag of S_0 and N_0 are combined and added to the feature set;
- Valency of S_0 and N_0: The numbers of left and right DEPENDENTs are calculated separately and then, combined with the word form and the POS tag of S_0 and N_0 to form a new feature;
- Unigram information S_{0h}, S_{0l}, S_{0r}, N_{0l}: A new feature is formed by combining the word form, POS tag and dependency label information of S_{0h}, S_{0l}, S_{0r}, N_{0l} with the same information of S_0 and N_0;
- Third-order feature of S_0 and N_0: This new feature includes the unigram word form, POS tag and dependency labels of S_{0h2}, S_{0l2}, S_{0r2} and N_{0l2} along with the POS tag combination of S_0 and N_0. Here, S_{0h2}, S_{0l2}, S_{0r2} and N_{0l2} refer to the HEAD of S_{0h}, the second leftmost and second rightmost DEPENDENT of S_0, and the second leftmost DEPENDENT of N_0, respectively.

- Set of dependency labels of S_0 and N_0: The set of unique dependency labels from the DEPENDENTs of S_0 and N_0 is created and combined with the word form and POS tag of S_0 and N_0.

As of June 2017, like Stanford nndep, this is the first attempt to parse Estonian text with spaCy.

2.2.4 SyntaxNet Parser

SyntaxNet dependency parser (Andor et al., 2016) is a neural network based dependency parser, first developed in Google for Tensorflow AI framework. It implements a transition-based, non-recurrent neural network using the Nivre's arc-standard transition system (Nivre, 2004) and the feature embedding introduced by Chen and Manning (2014). It maintains multiple hypotheses by performing beam search and introduces Global Normalisation with a Conditional Random Field (CRF) objective (Lafferty et al., 2001). CRF helps in overcoming the label bias problem that locally normalised models often suffer from.

Label bias problem describes the phenomenon when there is only one outgoing transition from a given state and the state must put all its probability mass to that transition even if it was never observed by that state during training. It means the state might be forced to ignore its observation, thus, leading to a wrong transition.

Beam search algorithm is a greedy graph-searching algorithm that uses breadth-first search to generate the search tree. At each level of the tree, a limited number of most promising successors of the states are stored. This number is known as the beam width. The wider the width is, the more states are kept for expanding in the next levels. Beam search sacrifices completeness to offer speed and memory optimization, as the goal state might be pruned during searching, thus, not finding the correct solution.

Since the parser uses beam inference, the partition function is being estimated from the summation of the elements in the beam and using early updates (Collins and Roark, 2004; Zhou et al., 2015). The gradients are computed based on this approximate partition function and the parser performs a full back-propagation training of all neural network parameters based on the CRF loss.

The SyntaxNet has trained model available for several languages including Estonian, which has been used in the experiments for this thesis.

2.2.5 UDPipe

UDPipe is a complete pipeline for natural language processing which consists of a tokenizer, lemmatizer, morphological analyser, POS tagger and dependency parser (Straka et al., 2016). The dependency parser used in UDPipe is called *Parsito* (Straka et al., 2015). The only addition that was made on top of Parsito is an optional beam search decoding similar to Zhang and Nivre (2011).

Parsito is a transition-based, non-projective dependency parser which can parse both projective and non-projective sentences. It employs an extended version of Nivre's arc-standard system for non-projective dependency parsing which has an extra transition called *swap* to reorder two nodes (Nivre, 2009).

It uses a neural network classifier inspired by Chen and Manning (2014) for predicting the correct transitions. The feature set used for training the classifier is the same as Stanford nndep parser that has been described in the section 2.2.2 previously. Distributed representations of the word form, POS tag and arc label are used to represent each node.

The classifier is trained using a search-based oracle. To determine the transition to follow for a given parser configuration, every applicable transition is performed in sequence and the classifier being trained is used to parse the rest of the tree. It means, in every step, the classifier follows the transition predicted by itself. Then, such transition is chosen from the original configuration which generates the dependency tree with the highest attachment score.

UDPipe has a trained parser model for Estonian that available for use but the model was trained on UD Estonian treebank 1.2, which made that model unusable for the experiments for this thesis work. A new model is trained on UD treebank version 1.3 so that its performance can be compared with the other parsers trained on the same dataset.

2.3 Previous Work on Dependency Parsing of Estonian Text

In parsing Estonian, there have been two efforts made in the past in developing a syntactic dependency parser, one is based on Constraint Grammar (CG) framework (Karlson et al., 1995) and the other one is statistical parser using MaltParser (Nivre, Hall, and Nilsson, 2006).

Following the publication of the Constraint Grammar framework by Karlsson et al. (1995) for disambiguating and parsing non-restricted text, development of Estonian Constraint Grammar (EstCG) was started. This CG based parser is capable of determining clause boundaries and dependency relations, surface syntactic analysis and morphological disambiguation by using separate sets of grammar rules (Müürisep, 2001). There is a module for identifying particle verbs and several valency lexicons are integrated into it. Dependency grammar consists of approximately 600 rules. The analysis obtained from the CG parser inspired the development of the first version of the Estonian Dependency Treebank (Muischnek et al., 2014b).

MaltParser was chosen to conduct the experiments on statistical analysis of Estonian text using a portion of this treebank (191,000 tokens) which had nearly 400,000 words (Muischnek et al., 2014a). A dataset in CoNLL-X format was created from the CG formatted text, which employed 22 fine-grained POS tags along with the 15 regular POS tags. All the 27 syntactic labels from EstCG annotation remained in the CoNLL-X data set while introducing a new label named ROOT to label the main verb of the main clause. MaltOptimizer optimization tool was used to find the most suitable parameters and training model, which recommended to the Covington's algorithm (Covington, 2001) in the non-projective mode with a specific feature model.

3 Datasets and Evaluation Measures

This chapter presents an overview of the dataset and data formats used in training or using the models. The training and test set of input data is of CoNLL-U format and has been taken from the UD Estonian treebank (version 1.3).

3.1 Universal Dependencies (UD) Treebank

Universal Dependencies treebank project (Nivre et al., 2016) started with the aim of developing treebank annotations, which will have consistency across languages and make it easier to develop multi-lingual parsers and perform cross-lingual learning.[7] The basic idea of the UD project is to facilitate consistent annotation scheme of similar constructions across languages by providing a universal inventory of guidelines and categories while keeping the possibility of creating language-specific extensions open if needed. UD annotation scheme is the result of evolution and combination of Stanford dependencies (de Marneffe and Manning, 2008; de Marneffe et al., 2014), Google Universal part-of-speech tags (Petrov et al., 2012), and the Interset interlingua for morphosyntactic tag sets (Zeman, 2008).

Stanford dependencies are the backend of the Stanford parser which was first developed in 2005 and has become the standard of dependency analysis of English.

Google Universal tag set is widely used as a standard to map diverse tag sets to a common standard. It was created by McDonald and Nivre (2007) based on ConLL-X shared task data for doing cross-linguistic error analysis. Then, Das and Petrov (2011) first used the tag set for unsupervised part-of-speech tagging. They created an extended set of POS tags in 2012 (Petrov et al., 2012) which is now known as Universal POS tags. There are 17 different tags in the set, of which the UD Estonian treebank uses 15, excluding PART and DET.

Interset is a tool for conversion among various morphological tag sets in natural language processing. A set of features encoded by different tag sets is defined in Interset. This set of features contains all relevant information to port from one tag set to another. It was first used in an experiment with cross-lingual de-lexicalised parser adaptation (Zeman and Resnik, 2008).

Universal Dependency Treebank (UDT) project in 2013 was the first attempt to bring together Stanford Dependencies and Google Universal tags to create a universal annotation scheme (McDonald et al., 2013). Treebanks for 6 languages were released in that year followed by 11 languages in 2014. HamelDT, a project to develop a common annotation scheme for treebanks of different languages, provided Stanford/Google annotation for 30 languages in its second version in 2014. The Universal Stanford Dependencies, Interset feature inventory, Google Universal tag set, and CoNLL-U were merged together to create the new Universal Dependencies.

3.1.1 UD Estonian Treebank

UD Estonian treebank was first released in version 1.2 in November 2015. In the experiments for this thesis, the treebank from version 1.3, released in May 2016, is used. The UD Estonian treebank has 34,628 tokens, 34,628 words and 3,172 sentences collected from the corpora of fiction, news, and science.

[7] http://universaldependencies.org/introduction.html

3.2 CoNLL-U Format

CoNLL-U format is a representation of dependency trees in text format. It is a plain text file with annotations encoded in the UTF-8 format with three types of lines:

- Single tab characters are used to separate the 10 fields of a word line;
- Sentence boundaries are marked by an empty line;
- A hash (#) symbol in the beginning of a line indicates comments.

Each word of a sentence is represented in a single line (called word line). A word line consists of the following fields:

1. ID: This is an integer indexing the words in their linear order, the first word of each sentence gets the ID 1. It can be a range for multi-word tokens.
2. FORM: Word form or punctuation symbol.
3. LEMMA: Lemma or stem of word form.
4. UPOSTAG: These are the Universal part-of-speech tags
5. XPOSTAG: This column represents the language-specific part-of-speech tag, if available. Otherwise, an underscore is used.
6. FEATS: FEATS is a list of morphological features defined in the Universal feature inventory or a list of an extended languages-specific feature set. Again, underscore is used if not available.
7. HEAD: ID of the HEAD word of the current word, zero (0) if the current word is the root.
8. DEPREL: Dependency relation between the HEAD word and the DEPENDENT word, root if and only if the HEAD is zero (0). The dependency relations can be from the Universal dependency relation set or from a subtype of the set specific to a language.
9. DEPS: Enhanced dependency graph in the form of a list of HEAD-DEPREL pairs.
10. MISC: Any other annotation.

3.3 Custom TAG set for Stanford nndep and spaCy

Morphological features have a substantial influence on parsing text of morphologically rich languages such as Estonian. As the Stanford nndep and spaCy parsers can only use the POS columns of the CoNLL-U format, we wanted to provide the morphological information to the parsers by constructing an extended tag set encoding both the POS tags and morphological features. Thus, we collected all the morphological features for each POS tag from the treebank and created two custom tag sets. One set of tags contained all the features those were available in the corpus and the other one only had features which were thought to be more important for parsing. Table-2 lists the features used for constructing the latter tag set. For example, the custom tags for the POS Noun from both tag sets are given below:

Tag set with all features:

NOUN:Gen:Plur:Past:Part:Act

Here, NOUN is the original POS tag and the present features are Case, Number, Tense, VerbForm, and Voice.

Table-2: The list of morphological features chosen for certain POS tags to construct the custom tag set with reduced features

Universal POS tag	Significant morphological features
NOUN	Number, Case, Voice
VERB	Number, Person, VerbForm, Voice
ADJ	VerbForm, Case, Degree, Number
PRON	Case, Number
PROPN	Case, Number
NUM	Case, Number

Tag set with reduced features:

NOUN:Gen:Plur:Act

In this case, only the Case, Number, and Voice features are included in the tag.

These morphological features were chosen upon consulting with a linguist who suggested that these features could affect parser decisions. For instance, it was assumed that the parse tree is not affected by whether a verb is in the present or simple past tense. However, some other morphological feature can affect the parse tree considerably.

Custom tags for all other POS tags contained only the original POS tag itself, like PUNCT, ADV, CONJ and so on.

3.4 Evaluation Measures

In this section, the evaluation measures used to evaluate the performance of the parsers are briefly described.

3.4.1 Labelled Attachment Score (LAS)

Labelled Attachment Score (LAS) determines how accurate the parsers are in attaching the correct label to the correct HEAD.

3.4.2 Unlabelled Attachment Score (UAS)

Unlabelled Attachment Score (UAS) represents the accuracy of attachment (i.e. finding the correct HEAD).

3.4.3 Label Accuracy (LA)

Label Accuracy (LA) shows the correctness in assigning the correct labels.

3.4.4 Precision and Recall

Precision shows the percentage of the labels those are actually correct out of the total number of identified labels. In other words, it measures the correctness of the parser.

Recall shows the ratio of correctly identified labels over the total number of correct DEPRELs in the input data. In a way, it measures the completeness, thus, the quality of the parser in finding correct dependency relation.

Precision and Recall can be expressed mathematically by the following formulas:

$$Precision = \frac{tp}{tp+fp} \qquad Recall = \frac{tp}{tp+fn}$$

true positives (tp): The number of labels detected CORRECTLY as belonging to that DEPREL class.

false positives (fp): The number of labels detected INCORRECTLY as belonging to that DEPREL class.

false negatives (fn): Number of labels not detected as belonging to that class of label but should have been.

4 Training Methods

This chapter explains the methods followed, training commands and all other necessary steps for training the models for Stanford nndep, spaCy, and UDPipe. Besides that, the procedure to use the already existing models for MaltParser and SyntaxNet is also described.

4.1 Training Stanford Neural Network Dependency Parser

This section details all the information regarding training of the Stanford nndep model[8]. It explains what a language pack is, the usage of word embeddings, the training command, and its different hyperparameters.

4.1.1 Language Pack

In the Stanford nndep parser, the language pack is a Java class which is needed to define the default character encoding, the list of punctuation POS tags and sentence final punctuation words, and to specify the tokenizer. It is also possible to train and test a model on CoNLL format files without providing a tokenizer. As the Estonian language uses the Latin alphabets, it was possible to use the default English *PennTreebankLanguagePack* without any complications.

4.1.2 Word Embeddings

Word embeddings are dense low-dimensional vector representations of the words in a corpus which can be trained using different neural network models. The word embeddings were trained using word2vec (Mikolov et al., 2013) on Estonian Reference Corpus[9] [10].

4.1.3 Training Command

The most basic form of the command that was used to train the models is given below with an explanation of the hyperparameters in Table-3.

java -cp "stanford-corenlp-3.6.0.jar:" edu.stanford.nlp.parser.nndep.DependencyParser -tlp edu.stanford.nlp.trees.PennTreebankLanguagePack -trainFile estonian/et-ud-train.conllu -devFile estonian/et-ud-dev.conllu -embedFile estonian/embeddings_W0.txt -embeddingSize 300 -model nndep.estonian.model.txt.gz*

[8] https://stanfordnlp.github.io/CoreNLP/
[9] http://www.cl.ut.ee/korpused/segakorpus
[10] "We thank Alexander Tkachenko for providing the trained embeddings."

Table-3: Explanation of the parameters passed to the command to train the Stanford nndep model

Parameter	Explanation
-cp "stanford-corenlp-3.6.0.jar:"*	Adding stanford-corenlp-3.6.0.jar along with all other jars available in the current directory to the classpath
edu.stanford.nlp.parser.nndep.DependencyParser	Main class
-tlp edu.stanford.nlp.trees.PennTreebankLanguagePack	TreebankLanguagePack
-trainFile estonian/et-ud-train.conllu	UD Estonian treebank training dataset
-devFile estonian/et-ud-dev.conllu	UD Estonian treebank validation dataset
-embedFile estonian/embeddings_W0.txt	Word embeddings for Estonian
-embeddingSize 300	300 is the dimension of the vectors in the embedding file
-model nndep.estonian.model.txt.gz	The model that is being created

The default number of iteration of 20000 was kept unchanged. After every iteration, the parser checks the UAS and the previous models are overwritten by the current one if the score exceeds the previous UAS score. This gives us the best model after the training is finished. It took on average 93 hours to train a model.

4.1.4 Hyperparameter Tuning

Several models were trained using different parameter settings to find the best-performing model. We describe the parameter tuning experiments in this section.

Model-1:

In the beginning, we trained the model using all the default training options except the embedding size, which in our case, was 300 in contrast to the default 50. This embedding size option was same in all the subsequent training experiments. This model had a UAS of 74.8 and LAS of 68.7. To explore different hyperparameters, the subsequent experiments were performed.

Table-4: UAS and LAS comparison among Stanford neural network dependency parser models trained with different options

Model no.	Training options			UAS		LAS
	Hidden size	cPOS	Custom feature set	Development	Test	Test
01	200	false	-	75.7	74.8	68.7
02	200	true	-	78.3	77.1	71.2
03	500	false	-	76.5	75.2	68.8
04	500	true	-	77.8	76.5	70.5
05	500	false	all	79.8	79.3	75.9
06	500	false	reduced	**80.5**	**79.6**	**76.2**
07	500	false	reduced	80.1	79.1	75.6

Model-2:

The first option we experimented with was hiddenSize. Hidden size is the dimensionality of the hidden layer of the neural network classifier. Increasing the hidden size increases the model capacity and thus, can potentially lead to the improvement of the overall accuracy of the created model. We trained a model with the hidden size set to 500 and obtained a UAS of 77.1 and LAS of 71.2. We can say by seeing the numbers that increasing the dimension of the hidden layer brings small improvement on the performance of the parser.

Model-3:

In our next experiment, we set cPOS to true, which tells the parser to use the Universal tags. By default, cPOS is false, meaning that the more fine-grained part-of-speech (i.e. the language specific custom part-of-speech) tags will be used to train the model. By training the model with coarse POS tags while keeping all other hyperparameters to the values same as in Model-1, we obtained small improvement over the Model-1.

Model-4:

After getting improved results from Model-2 and Model-3 experiments, we decided to train the Model-4 with hiddenSize of 500 and cPOS set to true. Model-4 got a UAS score of 76.5 compared to the score 74.8 of Model-1 and LAS score was 70.5 compared to 68.7.

Model-5 and Model-6:

Stanford neural network dependency parser's current implementation (version 3.6.0) does not use the morphological features listed in the 6th column of the CoNLL-U format treebank data set. This setting would work well with languages like English which do not have a broad morphological feature set and the Universal POS tag set covers most of the morphological aspects of the language. However, this does not suit to a morphologically rich language like Estonian. Thus, we experimented with the two different custom morphological tag sets described earlier in section 3.3. These two experiments were performed with setting the hiddenSize parameter to 500 and the cPOS parameter to false. Results show that model trained with the tag set with reduced features (UAS of 79.6 and LAS of 76.2) had a slightly better performance over the model with tag set that included all the features (UAS of 79.3 and LAS of 75.9).

Model-7:

We trained this model without including the default *PennTreebankLanguagePack* and the results were slightly lower than the same model with the language pack.

From Table-4 that shows the results of all the described experiments, we can see that the Model-6 has the best result. Thus, we decided to use this model for detailed evaluation in the next chapter.

4.2 Training spaCy Parser

This section describes the procedure of training the model for the spaCy parser. The spaCy parser needs various language specific information, such as stop list, tag map, and tokenizer exception file to train a model for that language.

4.2.1 Language Subclass Creation

The very first step of adding a new language into spaCy is to create a new language subclass as a subpackage of spaCy which should be named according to the languages' ISO code. So, to add Estonian, all the code and resources specific to Estonian were placed into a directory *spacy/et* which then can be imported as *spacy.et*. This new language class had to be registered in *spacy/__init__.py* to be able to load it later using *spacy.load()* method. Additionally, it was listed in the *setup.py* file.

4.2.2 Stop List Creation

Stop list is a list of common function and closed-class words. The *stop list* can contain any number of words and there is no universal list of stop words for any language. The stop words for Estonian were taken from internet sources[11, 12]. The full list of the stop words is given in *Appendix A*.

4.2.3 Tag Map Creation

A tag map is needed to map down the custom part-of-speech tags of any language to the Universal POS tag set. The data structure used for the tag map is a Python dictionary where the dictionary keys are strings containing tags from the custom tag sets and the values are also dictionaries. The value dictionary must have an entry called 'POS' whose value must be one of the Universal POS tags. Morphological features or token attributes can be added to the tag map as well. We used the custom tag sets described in section 3.3. A snippet of

[11] https://github.com/6/stopwords-json/blob/master/dist/et.json
[12] estnltk/estnltk/textclassifier/analyzer

Table-5: UAS, LAS and LA comparison between two spaCy models trained with different custom tag sets

Model	UAS	LAS	LA
All features	83.0	77.0	91.1
Reduced features	**83.1**	**77.2**	**91.9**

the tag map with all morphological features is given in *Appendix B* and the tag map with reduced features is given in *Appendix C*.

4.2.4 Tokenizer Exceptions Creation

Tokenizer exceptions are mainly special case rules defined to let the tokenizer perform freely without worrying about how these cases will interact with the rest of the tokenizer. A Python dictionary is used to store the exception list. The dictionary keys represent the exceptional words and the corresponding values, which are lists of dictionaries, map the original form or full form (in a case of abbreviations) of the words. Currently, the list contains the abbreviations of the months' names and can be extended in the future, if necessary. The list of the tokenizer exceptions can be found in *Appendix D*.

4.2.5 Training Brown Clusters

Brown clustering (Brown et al., 1992) is a variation of hierarchical clustering in the sector of natural language processing. The main idea is to cluster the words of a text corpus into classes based on the context those words occur in. In other words, the probability of a word belonging to a certain class is determined based on the clusters of the previous words in that sentence. Some words have similar meaning and syntactic function with other words. For instance, the probability distribution for words around January is similar to the words in the vicinity of March. An open source implementation of Brown Clustering (Liang, 2005) was used to train the brown cluster model[13]. The number of clusters was set to 1000 and the minimum frequency of occurrence of a word to be considered for clustering was 10.

4.2.6 Experimental Results

Table-5 presents the UAS, LAS, and LA of the two models on the development dataset of the UD Estonian treebank. One model was trained with all the morphological features included in the custom tag set and the other one had only reduced features present in the custom tag set.

4.3 Training UDPipe Parser

UDPipe is developed in a way that makes it easy to train a parsing model for any language. The following command was executed to train the model:

/udpipe/src/udpipe –train /estonian-ud-1.3-170605.udpipe /nlp_data/et_ud/et-ud-train.conllu

[13] https://github.com/percyliang/brown-cluster

/udpipe/src/udpipe – is the executable compiled and built from the source code;

-train – a flag indicating that the task is a training task (-parse is used for parsing);

/estonian-ud-1.3-170605.udpipe – is the model to be trained;

/nlp_data/et_ud/et-ud-train.conllu – UD training data in CoNLL-U format

4.4 SyntaxNet Parser

The pre-trained model for Estonian was acquired from the Tensorflow resource archive[14]. SyntaxNet was built and installed from the source code following the instruction for manual installation given in the SyntaxNet GitHub repository[15].

Assuming SyntaxNet was installed in the directory *$HOME/models/*, the test dataset was available in the directory *$HOME/nlp_data/et_ud/*, and the pretrained model for Estonian was downloaded and unzipped in *$HOME/syntaxnet_models/Estonian/*, following command was used to parse the test dataset:

cat $HOME/nlp_data/et_ud/et-ud-test.conllu | $HOME/models/syntaxnet/syntaxnet/models/parsey_universal/parse.sh --conll $HOME/syntaxnet_models/Estonian > $HOME/syntaxnet_parse_conll_output.conll

--*conll* flag was used to get the parse output in CoNLL format.

4.5 MaltParser

The pre-trained model for MaltParser was acquired from the Github repository of EstSyntax[16]. MaltParser 1.9.0[17] was used to run the model and parse the test data set from UD Estonian treebank. The command to run the model is given below:

java -jar MaltParser_download_directory\maltparser-1.9.0\maltparser-1.9.0.jar -c dets16kogu -i UD_Estonian_treebank_directory\et-ud-test.conllu -o output.conll -m parse

-*c* flag defines the model name, in this case, *dets16kogu*

-*i* flag indicates the location of the input data

-*o* flag defines the output file

-*m* indicates the action parser should perform (train or parse), in this case, *parse*

[14] https://github.com/tensorflow/models/blob/master/syntaxnet/g3doc/universal.md
[15] https://github.com/tensorflow/models/tree/master/syntaxnet
[16] https://github.com/EstSyntax/EstMalt/tree/master/EstUDModel
[17] http://www.maltparser.org/download.html

5 Evaluation and Analysis

In this section, an analysis of the performance of the five parsers is presented. An evaluation script written in Perl programming language is used in evaluating the performance of the parsers[18]. In addition to LAS and UAS, which are standard measures in dependency parsing, this script also computes more detailed results, such as HEAD and dependency relation accuracy per POS tag, precision, and recall of label accuracy etc. First, a comparative evaluation of UAS, LAS, and LA is given which is followed by an analysis of the accuracy of the parsers in finding HEADs and labels correctly. The chapter ends with an evaluation of recall and precision of label accuracy of the parsers.

Table-6: Comparison of LAS, UAS, and LA of the parsers on UD Estonian treebank 1.3 test data (scores obtained by comparing the parse outputs against the test data that contains gold POS tags) (highest scores in bold and lowest scores in italic)

	LAS	**UAS**	**LA**
Stanford	*76.3*	*80.4*	87.6
spaCy	76.6	82.2	*85.5*
SyntaxNet	78.3	83.4	87.1
MaltParser	**80.0**	**83.6**	89.2
UDPipe	79.1	82.5	**90.1**

5.1 Comparison of UAS, LAS, and LA

Table-6 presents the LAS, UAS and LA of all parsers on UD Estonian 1.3 test data. This test data was parsed by the parsers and the parser outputs were evaluated against the same test data, which is considered as gold standard input. The numbers in the Table-6 show the percentage of assigning the correct labels (LA), identifying the correct HEADs (UAS) and attaching the correct labels to the correct HEADs (LAS).

From Table-6, it is clear that MaltParser's model is the best in terms of attachment score (both labelled and unlabelled) and is second to UDPipe in label accuracy. UDPipe can be named as the second best in general even though SyntaxNet's UAS is slightly better than UDPipe's. Stanford obtained the lowest score for both LAS and UAS. On the other hand, spaCy performed poorly in finding the right labels.

[18] https://github.com/elikip/bist-parser/blob/master/bmstparser/src/utils/eval.pl

Table-7: Accuracy in finding the correct HEAD word per POS tag (highest scores in bold and lowest scores in italic)

	Words	**Stanford nndep**	**spaCy**	**SyntaxNet**	**MaltParser**	**UDPipe**
NOUN	6154	*78*	80	**81**	**81**	80
VERB	3409	*79*	**84**	**84**	83	**84**
ADV	2294	*77*	*77*	**79**	**79**	*76*
ADJ	1947	*86*	*86*	87	**89**	88
PRON	1570	*85*	87	**89**	86	87
PROPN	1388	*80*	82	**85**	84	83
CONJ	885	*71*	78	**80**	79	75
AUX	620	**96**	*93*	*93*	**96**	**96**
ADP	513	91	89	*87*	**95**	92
SCONJ	474	87	*85*	86	88	**93**
NUM	455	*80*	83	82	**87**	82
INTJ	35	**77**	**77**	**77**	*74*	*74*
X	14	50	**79**	*21*	36	29
SYM	9	*44*	**67**	56	56	56
TOTAL	19767	*80*	82	83	**84**	83

5.2 POS-based HEAD accuracy

This section evaluates the performance of the parsers in finding the correct HEAD word per gold POS tag. Here, the POS tags refer to the Universal POS tags which have been introduced in Chapter 3. In a dependency relationship between two words, HEAD is the word that gets modified by its DEPENDENT. In Table-7, *Words* column shows the total number

of words of the respective POS tag in the test dataset. The other columns represent the percentage of correctness in finding the HEAD word by the respective parsers.

One notable POS tag is *CONJ*, coordinating conjunctions which are words that express a semantic relationship between words or larger constituents by linking them together (e.g. ja (and), või (or), aga (but)). For *CONJ*, Stanford nndep performed comparatively low with 71% while spaCy, SyntaxNet and MaltParser achieved scores close to each other with SyntaxNet getting the highest of 80 percent.

Adpositions (*ADP*) are a collective set of prepositions and postpositions. Adpositions, together with nominals, form adpositional phrases that normally function as an adverbial in the sentence, but can act as an attribute also. Most of the times, case forms of nouns can perform the same functions. In Estonian, adpositions do not constitute a tightly closed class, i.e. it is difficult to determine the exact boundary between word classes (Muischnek et al., 2005). Sometimes, adpositions are identical to case forms of some nouns or non-finite forms of some verbs. Moreover, many of the adpositions can act as an adverb and form a particle verb together with a verb. Despite these characteristics of Estonian adpositions, all the parsers performed well in identifying correct HEADs for adpositions by scoring nearly or above 90%.

All parsers were extremely good in detecting correct auxiliary verb (*AUX*) HEADs where 96% is the highest and 93% being the lowest. Sometimes, lexical verbs (the main verb of the sentence) cannot express some grammatical properties like person, tense, mood, voice etc. Auxiliary verbs accompany a lexical verb to express such distinctions. Some examples of tense auxiliaries are, ***oleme*** + teinud (***have/has*** done, e.g. ***olen*** teinud (I ***have*** done)); while ***pidama*** + tegema (***must*** do, e.g. ***pean*** tegema (I ***must*** do)) are examples of modal auxiliaries.

Differences in accuracy among the parsers for POS tags *X* and *SYM* are large but these numbers are not reliable as the frequencies of these tags are very low. Overall, the MaltParser tops the list with 84% total correctness followed by the SyntaxNet and UDPipe both with 83%.

5.3 POS-based label accuracy

In this section, the accuracy of the parsers in correctly labelling the arcs is evaluated. These labels represent the dependency relation between words. Table-8 represents the percentage of correctness in finding the labels of the respective parsers. Similar to Table-7, *Words* column in Table-8 shows the total number of words of the respective POS tag in the test dataset. MaltParser outperforms all the other parsers in identifying the dependency relations correctly. Out of 14 POS tags, it obtains the highest accuracy for 12. On the other hand, spaCy has the lowest accuracy in 10 POS tags. The difference between parsers is the largest (10%) in the case of adjectives (*ADJ*). Adjectives are words that generally modify nouns by specifying their properties or attributes. UDPipe is able to detect 92% labels correctly and spaCy 82%, which is the lowest. In the case of auxiliary verbs (*AUX*), again MaltParser topped together with UDPipe with staggering 99% correctness while SyntaxNet bottomed with 92%. All parsers show significant performance for Adpositions (*ADP*) in here as well. SyntaxNet and spaCy score exactly the same with 91%, MaltParser's score is 99% while UDPipe gets 98% and Stanford nndep gets 97% correct. It is worth to note that MaltParser, Stanford nndep, and UDPipe's accuracies are very close to each other in most of the occasions. Combining the results of all the individual POS tags, UDPipe obtains the highest score of 90% while the lowest was 86% by spaCy.

Table-8: Accuracy in finding the correct dependency relation per POS tag (highest scores in bold and lowest scores in italic)

	Words	**Stanford nndep**	**spaCy**	**SyntaxNet**	**MaltParser**	**UDPipe**
NOUN	6154	86	*83*	84	**88**	**88**
VERB	3409	*80*	84	**85**	*80*	**85**
ADV	2294	93	*90*	91	**94**	93
ADJ	1947	89	*82*	85	91	**92**
PRON	1570	86	*85*	**88**	**88**	**88**
PROPN	1388	87	*85*	86	**90**	**90**
CONJ	885	**100**	*97*	98	**100**	**100**
AUX	620	98	93	*92*	**99**	**99**
ADP	513	97	*91*	*91*	**99**	98
SCONJ	474	*98*	*98*	*98*	*98*	**99**
NUM	455	*87*	**93**	*87*	**93**	**93**
INTJ	35	89	*71*	74	**97**	**83**
X	14	**64**	50	*14*	50	43
SYM	9	67	*56*	**78**	67	67
TOTAL	19767	88	*86*	87	89	**90**

Table-9: Accuracy in finding both the correct HEAD and dependency relation per POS tag (highest scores in bold and lowest scores in italic)

	Words	**Stanford nndep**	**spaCy**	**SyntaxNet**	**MaltParser**	**UDPipe**
NOUN	6154	74	*73*	75	**78**	76
VERB	3409	*73*	79	**80**	75	79
ADV	2294	73	*72*	74	**77**	*72*
ADJ	1947	83	*78*	80	**86**	85
PRON	1570	80	*79*	**83**	**83**	82
PROPN	1388	*77*	*77*	79	**81**	80
CONJ	885	*71*	76	**79**	**79**	75
AUX	620	95	91	*90*	**96**	**96**
ADP	513	90	87	*86*	**94**	92
SCONJ	474	86	*84*	85	88	**92**
NUM	455	*75*	*75*	77	**84**	79
INTJ	35	**74**	69	*66*	**74**	*66*
X	14	29	**50**	*7*	29	14
SYM	9	*44*	*44*	**56**	*44*	*44*
TOTAL	19767	*76*	77	78	**80**	79

5.4 POS-based HEAD and Label accuracy

An analysis of identifying both the HEAD words and labels correctly, meaning that the correct label is attached to the correct HEAD, is presented in this section. Table-9 lists the accuracy (in percentage) of the parsers. Based on the analysis of POS-based HEAD and POS-based label accuracy, MaltParser is expected to have the best performance in this case as well. MaltParser achieves the best (80%) overall correctness. The largest variance in score

Table-10: Collapsed dependency relations and the new dependency relations

Collapsed Dependency Relations	New Dependency Relation
acl, acl:relcl	acl
advmod and advmod:quant	advmod
cc and cc:preconj	cc
compound and compound:prt	compound
csubj and csubj:cop	csubj
nmod and nmod:poss	nmod
nsubj and nsubj:cop	nsubj

can be observed in the case of numerals (NUM) with 9% and adjectives (*ADJ*), conjunctions (CONJ), adpositions (ADP), and subordinating conjunctions (SCONJ) with 8% difference between the highest and lowest scores. All the parsers struggle in finding both the HEAD and labels correctly for nouns (*NOUN*) and adverbs (*ADV*) where most of the times the scores are below 75%. Once again, MaltParser and UDPipe scored very closely with 80% and 79%, respectively.

5.5 Label Precision and Recall

This section analyses the precision and recall of label accuracy of the parsers. Table-10 presents the pairs of dependency relationships those are truncated into single relations for the ease of analysis. Firstly, the precision and recall of label accuracy are evaluated, which is then followed by the analysis of precision and recall in attaching the correct label to the correct HEAD.

5.5.1 Precision Based on Label

In Table-11, the precision of accuracy per label is listed. The column 'Frequency (gold)' indicates the number of words with respective labels present in the gold input (test data set). This 'Frequency (gold)' column is same in the Tables 11-13. Although based on the HEAD and label accuracy results presented in the previous sections, MaltParser was expected to have the best precision in identifying labels correctly, it is outperformed by UDPipe in this case. However, the other parsers performed relatively well too. The precision of all parsers for most of the dependency relations was above 75 percent. These numbers were over 90% for some syntactic relations, such as *adjectival modifier (amod), auxiliary (aux), case marking (case), coordinating conjunction (cc), marker (mark), negation modifier (neg)*. One exception is *adverbial clause modifier (advcl).* Adverbial clauses modify a verb or a predicate (e.g. adjective) as a modifier, not as a core complement. All the parsers except UDPipe performed poorly in identifying *advcl*, especially MaltParser with a precision as low as 50%.

Table-11: Precision of Label Accuracy (highest scores in bold and lowest scores in italic)

Label	Frequency (gold)	Precision				
		Stanford nndep	spaCy	SyntaxNet	MaltParser	UDPipe
acl	507	74.7	79.2	82.8	*71.0*	**83.0**
advcl	310	67.0	73.1	72.9	*49.4*	**80.4**
advmod	1871	90.1	*81.0*	88.4	86.5	**91.0**
amod	1319	94.5	*89.8*	91.4	**97.5**	95.4
appos	146	**81.0**	70.9	67.4	*67.3*	79.2
aux	395	97.5	*90.3*	91.8	**100**	97.0
case	509	98.2	*90.9*	92.8	**99.0**	98.6
cc	910	92.2	91.8	*91.7*	95.1	**97.3**
ccomp	121	54.2	60.1	**63.8**	*47.6*	53.2
compound	302	90.1	*81.6*	82.6	**91.6**	85.4
conj	1119	*71.0*	76.5	**79.7**	76.4	78.7
cop	337	*80.1*	85.3	84.1	87.4	**88.0**
csubj	96	83.7	73.1	80.9	*64.7*	**86.2**
dep	207	*51.5*	59.6	**64.0**	NaN	63.4
det	322	82.0	*76.2*	81.3	**84.1**	82.7
discourse	26	95.8	*56.7*	78.3	**100**	95.5
dobj	1201	85.4	*80.0*	84.0	**87.8**	86.5
foreign	24	**85.7**	*84.2*	80.0	*50.0*	81.8

list	5	**100**	50.0	50.0	*0*	NaN
mark	562	98.4	*95.5*	97.2	98.4	**99.5**
name	218	89.9	*84.3*	88.5	**94.3**	90.5
neg	225	**100**	99.6	*99.1*	**100**	**100**
nmod	4205	87.9	85.9	*85.0*	**92.7**	92.0
nsubj	2048	*81.0*	81.2	81.6	84.0	**85.2**
nummod	384	**95.8**	*91.8*	92.0	95.4	94.7
parataxis	243	70.9	71.9	81.0	*60.3*	**81.8**
punct	0	0	0	0	-	-
root	1806	*87.0*	88.3	89.1	87.5	**90.3**
vocative	7	*50.0*	*50.0*	**100**	60.0	*50.0*
xcomp	342	78.4	*76.8*	**81.5**	81.1	77.3
Total	19767	83.2	78.5	82.9	76.2	82.2

5.5.2 Recall Based on Label

Table-12 shows the recall of label accuracy of the parsers. For some labels, the recall is quite good. For instance, the recall for *case*, *mark*, *name*, *neg* labels are over 90%. Among these, the recall of *mark* and *neg* are close to 100% for all the parsers, which is very impressive. Similar to the case of precision, the recall of *advcl* is also low, with 73.9% being the highest by UDPipe.

Table-12: Recall of Label Accuracy (highest scores in bold and lowest scores in italic)

Label	Frequency (gold)	Recall				
		Stanford nndep	spaCy	SyntaxNet	MaltParser	UDPipe
acl	507	76.4	*73.7*	**83.0**	82.9	80.8
advcl	310	*60.3*	69.4	71.0	65.2	**73.9**
advmod	1871	77.0	80.6	78.6	**81.0**	*73.2*
amod	1319	94.2	*84.3*	86.0	95.7	**96.4**
appos	146	*58.2*	68.5	59.6	**70.6**	65.1
aux	395	97.0	89.4	*88.4*	98.2	**99.2**
case	509	98.0	92.0	*91.8*	**99.8**	98.6
cc	910	*86.7*	89.9	87.9	**91.1**	89.0
ccomp	121	*63.6*	68.6	**74.4**	73.6	69.4
compound	302	88.4	89.8	88.6	**91.6**	*86.0*
conj	1119	*78.3*	80.6	**82.6**	**82.6**	81.0
cop	337	81.0	82.8	81.3	*78.0*	**89.3**
csubj	96	*52.7*	**80.5**	77.4	70.6	60.9
dep	207	49.8	51.2	**55.1**	0	*41.1*
det	322	84.8	*84.5*	87.6	**95.3**	93.5
discourse	26	88.5	*65.4*	69.2	**96.2**	80.8
dobj	1201	81.6	81.1	*80.7*	**87.1**	85.9
foreign	24	25.0	**66.7**	*16.7*	*16.7*	37.5
list	5	**20.0**	**20.0**	***20.0***	*0*	*0*

mark	562	**98.6**	98.4	*98.2*	*98.2*	**98.6**
name	218	94.0	90.8	*88.5*	**97.7**	95.9
neg	225	*98.7*	99.6	*98.7*	**100**	99.1
nmod	4205	89.5	*83.9*	89.2	**95.0**	94.2
nsubj	2048	78.6	*78.2*	80.8	**81.4**	81.4
nummod	384	88.5	*84.4*	90.1	96.1	**97.1**
parataxis	243	68.3	70.4	**75.3**	*44.4*	66.7
punct	0	NaN	NaN	NaN	-	-
root	1806	*87.0*	88.3	89.1	87.8	**89.7**
vocative	7	*14.3*	28.6	*14.3*	**42.9**	28.6
xcomp	342	*73.1*	82.5	78.4	**89.2**	83.6
Total	19767	74.2	76.7	75.3	76.2	77.1

5.5.3 Precision Based on Label and Attachment

Table-13 shows the precision for the parsers in correctly finding both the labels and the HEAD those labels are attached to. As both HEAD and relation are measured here, there is a decrease in both precision and recall. However, MaltParser still performs the best. All parsers perform well for *amod, aux, case, mark,* and *name* with over 80% precision and are extremely good in finding the correct HEAD for the label *neg* with almost 100% accuracy. As expected, parsers struggle to identify the label *advcl* and its attachments correctly where the best parser so far, MaltParser, obtained only 39.4%.

Table-13: Precision of Label and Attachment Accuracy (highest scores in bold and lowest scores in italic)

Label	Frequency (gold)	Precision				
		Stanford nndep	spaCy	SyntaxNet	MaltParser	UDPipe
acl	507	68.5	74.3	**78.2**	*67.0*	74.6
advcl	310	53.8	62.9	61.3	*39.4*	**66.7**
advmod	1871	78.3	*70.9*	75.7	74.4	**78.9**
amod	1319	88.4	86.5	*86.5*	**91.7**	88.4
appos	146	**74.3**	66.7	*58.9*	62.8	70.0
aux	395	93.9	*88.2*	88.4	**96.1**	92.6
case	509	91.3	*86.2*	87.1	**94.4**	92.7
cc	910	78.0	*76.9*	77.4	79.8	**84.8**
ccomp	121	52.8	59.4	**63.8**	*47.6*	51.9
compound	302	87.8	*80.2*	80.8	**90.3**	84.0
conj	1119	*50.5*	64.4	**66.7**	62.5	61.8
cop	337	*70.1*	**79.5**	78.5	79.1	79.0
csubj	96	83.7	73.1	79.3	*64.7*	**84.3**
dep	207	*39.5*	51.1	**57.9**	NaN	55.2
det	322	77.8	*73.1*	77.5	**80.3**	77.8
discourse	26	75.0	*53.3*	65.2	**76.0**	68.2
dobj	1201	80.2	*75.8*	78.1	**83.0**	78.9
foreign	24	57.1	68.4	**80.0**	*37.5*	72.7

list	5	0	0	0	0	NaN
mark	562	86.2	*81.7*	84.2	87.5	**90.7**
name	218	88.6	*82.6*	86.7	**92.9**	87.0
neg	225	99.6	98.7	*98.2*	**99.6**	99.1
nmod	4205	76.5	75.6	*74.9*	**82.8**	79.9
nsubj	2048	*78.0*	78.9	79.6	82.3	**83.1**
nummod	384	82.0	81.3	79.5	**85.3**	*79.2*
parataxis	243	56.0	56.7	**66.4**	*46.9*	60.6
punct	0	0	0	0	-	-
root	1806	*87.0*	88.3	89.1	87.4	**90.3**
vocative	7	50.0	*25.0*	**100**	40.0	50.0
xcomp	342	73.0	73.3	**78.7**	**78.7**	*71.6*
Total	19767	71.7	70.1	75.1	69.3	74.3

5.5.4 Recall Based on Label and Attachment

The numbers in the Table-14 show the recall of attaching correct labels to the correct HEADs by the parsers. It can be seen from the table that all parsers perform well for the labels *aux*, *case*, *compound*, *mark*, *name* and *neg*. As seen in the analysis in section 5.5.2, recall is nearly 100% in identifying the correct label *neg*, which stayed almost unchanged after considering attachment of correct HEAD in the calculation.

Table-14: Recall of Label and Attachment Accuracy (highest scores in bold and lowest scores in italic)

Label	Frequency (gold)	Recall				
		Stanford nndep	spaCy	SyntaxNet	MaltParser	UDPipe
acl	507	70.1	*69.1*	**78.4**	78.3	72.6
advcl	310	*48.4*	59.7	59.7	51.9	**61.3**
advmod	1871	65.0	**70.2**	66.5	69.1	*60.7*
amod	1319	88.2	*81.2*	81.4	**90.1**	89.2
appos	146	53.4	64.4	*52.1*	**65.8**	57.5
aux	395	93.4	87.3	*85.1*	94.4	**94.7**
case	509	91.2	87.2	*86.1*	**95.1**	92.7
cc	910	*72.5*	75.2	74.0	76.2	**76.5**
ccomp	121	*62.0*	67.8	**74.4**	73.6	67.8
compound	302	86.1	88.3	86.7	**90.3**	*84.7*
conj	1119	*55.7*	67.8	**69.1**	67.5	63.6
cop	337	70.9	77.2	76.0	*70.6*	**80.1**
csubj	96	*52.7*	**80.5**	75.9	70.6	59.4
dep	207	38.2	44.0	**49.8**	0	*35.8*
det	322	*80.4*	81.1	83.5	**91.0**	87.9
discourse	26	69.2	61.5	*57.7*	**73.1**	*57.7*
dobj	1201	76.6	76.9	*75.1*	**82.4**	78.4
foreign	24	16.7	**54.2**	16.7	*12.5*	33.3

list	5	0	0	0	0	0
mark	562	86.3	*84.2*	85.1	87.4	**89.9**
name	218	92.7	89.0	*86.7*	**96.3**	92.2
neg	225	98.2	98.7	*97.8*	**99.6**	98.2
nmod	4205	78.0	*73.8*	78.6	**85.14**	81.9
nsubj	2048	*75.9*	76.1	78.7	**79.8**	79.4
nummod	384	*65.5*	67.9	71.6	70.3	**81.3**
parataxis	243	53.9	55.6	**61.7**	*34.6*	49.4
punct	0	NaN	NaN	NaN	-	-
root	1806	*87.0*	88.3	89.1	87.7	**89.7**
vocative	7	*14.3*	*14.3*	*14.3*	**28.6**	**28.6**
xcomp	342	*68.1*	78.7	75.7	**86.6**	77.5
Total	19767	65.9	69.7	68.5	69.3	69.7

5.6 Parsing plain text

The results in the previous sections were obtained by supplying the parsers with gold-standard POS tags, as this enabled to perform POS-based analysis of the parsing component only. We were curious to see how the parsers perform when fed with plain text. But, due to time limitation, this goal could not be accomplished entirely. Only SyntaxNet was able to parse Estonian plain text cleanly and thus, it was possible to evaluate its labelled and unlabelled attachment scores, and label accuracy. In parsing plain text, SyntaxNet obtains a LAS of 70.1%, a UAS of 83.4% and LA of 78.9%. These numbers are 78.3%, 83.4%, and 87.1% respectively in parsing text with gold-standard POS tags. All other parsers could not produce an output that could be evaluated using the evaluation script. The evaluation script requires the output to have exactly same number of lines as in the gold input file (UD test data set). Stanford parser struggled in parsing sentences containing double-quotation punctuation mark ("). For spaCy, it fails in treating the hyphenated compound words as a single word. Instead, it creates three or more different tokens out of the compound word. For example, “red-green-blue” would be parsed as “red”, “-”, “green”, “-” and “blue”, these five separate

tokens. spaCy also fails in parsing abbreviations and numeric dates those include a punctuation mark (usually a period mark). In the case of UDPipe, no single pattern in errors could be observed. In some cases, it parsed one single line into multiple lines while vice-versa was also observed. UDPipe parsed incorrectly some sentences with the double-quotation mark as well. For MaltParser, the plain text could not be parsed as it requires the input text to be tagged already before it could be parsed by MaltParser (i.e. first six columns of the CoNLL format need to be present in the input data). None of the other four parsers could produce an appropriate tagged output of the input data that can be fed into MaltParser.

5.7 Discussion and Recommendation for Future Work

Based on the analysis in this chapter, it is evident that MaltParser's performance was the best among the five parsers. It achieved high results in both identifying the HEAD words and the dependency labels correctly. One reason behind this could be the optimisation of the parser for parsing Estonian using the MaltOptimizer (Muischnek et al., 2016). Based on the optimisation suggestion, Covington's algorithm (Covington, 2001) in non-projective mode was employed. Non-projective dependency parsing is particularly advantageous for languages like Estonian where long distance dependencies and free word order are common characteristics. The UDPipe is also capable of parsing non-projective sentences. Moreover, it implements a search-based oracle, a variation of dynamic oracle that can be applied to any transition system. These features might have contributed in achieving the second-best performance by the parser. For SyntaxNet, not enough information could be gathered whether some sort of optimisation was performed before training the model for Estonian. However, it is noteworthy that, SyntaxNet implements Global Normalisation with a Conditional Random Field (CRF) that makes the parser capable of revising a wrongly made decision in the later stages during the training. This feature might have influenced the parser's satisfactory performance. On the other hand, the Stanford nndep and the spaCy parsers, initially developed focusing on English, were trained without making any language specific modification in the code, training parameters or algorithms. It was done so to fulfil the purpose of testing the efficiency of these parsers in parsing Estonian when used off-the-shelf. Looking at the parsing results of these two parsers, especially the labelled attachment scores (LAS), it can be said that, there is room for improvement before these can be of great efficiency in parsing Estonian text. The SyntaxNet parser performed relatively well than the Stanford nndep and spaCy. However, it also needs further development to produce better parsing result. In our opinion, either the best performing MaltParser or the second best, UDPipe, should be the choice while selecting a dependency parser to parse Estonian.

5.7.1 Recommendation for Future Work

Due to the limited amount of time, some questions had to be left unanswered. For instance, it would be interesting to know the significance of the morphological features in parsing accuracy. An effort was made in this thesis to test their implication by creating custom POS tag sets by embedding morphological features with the tags. These tag sets were used in the training data for the Stanford nndep and the spaCy parsers as these two parsers do not use the feature column of CoNLL-U data format when training their models. The purpose of using these tag sets was to observe the effect of morphological features in parsing accuracy. However, more research needs to be done to come to a rational decision.

Another necessary work could be to find solutions to the issues in parsing plain text. For example, Stanford nndep and UDPipe struggled with double quotation punctuation mark; spaCy was unable to identify hyphenated compound words as a single word. Solving these problems would help to better understand the potential of these parsers for Estonian.

6 Conclusions

In this thesis, an evaluation of five different dependency parsers, namely MaltParser, spaCy, Stanford nndep, SyntaxNet, and UDPipe, is presented with the aim of finding these parsers' efficiency in parsing Estonian text while being used off-the-shelf without any major language specific adaptation.

There were pre-trained models available for SyntaxNet and MaltParser which were used in this thesis. Several new models were trained with Stanford parser with different values for certain hyperparameters to get the best possible model. Two custom POS tag sets with morphological features embedded were created to observe the effect in the performance. One tag set includes all the features available in the corpus while the other uses a reduced feature set, which are thought to affect more in parsing decisions. Using these two POS tag sets did improve both the UAS and LAS scores by 3-4% from the models trained with Universal POS tags for Stanford parser. The model trained with custom POS tags with reduced feature set achieved slightly better results compared to the one using full feature set. The two models of spaCy also behaved similarly, the model with full feature set obtained lower score than the model with reduced feature set.

In our opinion, all the parsers performed well considering the complexity, morphological richness and free word order characteristics of Estonian grammar. MaltParser achieved the best numbers in most of the evaluation factors followed by the UDPipe. UDPipe scored better than the MaltParser in quite a few cases while the other numbers were close to the MaltParser's numbers. However, one drawback of MaltParser compared to UDPipe is that the MaltParser does not include a tokenizer and parts-of-speech tagger. The input text needs to be tokenized and tagged by a POS tagger before it could be parsed with the MaltParser. On the other hand, UDPipe has its own tokenizer and POS tagger along with a lemmatizer, and morphological analyser. Also, the UDPipe parser requires less work to setup the environment and train a model compared to the other parsers. Among the five parsers evaluated in this thesis, we would recommend the UDPipe as the most preferable for parsing Estonian.

References

Agichtein, E., & Gravano, L. (2000). Snowball: Extracting Relations from Large Plain-text Collections. *Proceedings of the Fifth ACM Conference on Digital Libraries*, (pp. 85-94).

Andor, D., Alberti, C., Weiss, D., Severyn, A., Presta, A., Ganchev, K., . . . Collins, M. (2016). Globally Normalized Transition-Based Neural Networks. *CoRR*.

Bahl, L., Jelinek, F., & Mercer, R. (1983). A Maximum Likelihood Approach to Continuous Speech Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 179-190.

Brown, P. F., deSouza, P. V., Mercer, R. L., Pietra, V. J., & Lai, J. C. (1992). Class-based n-gram models of natural language. *Computational Linguistics*, 467-479.

Chang, C.-C., & Lin, C.-J. (2011). LIBSVM: A Library for Support Vector Machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 27:1-27:27.

Charniak, E. (1997). Statistical Techniques for Natural Language Parsing. *AI Magazine*, 33-44.

Chen, D., & Manning, C. (2014). A Fast and Accurate Dependency Parser using Neural Networks. *Empirical Methods in Natural Language Processing (EMNLP)*.

Collins, M. (2002). Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - Volume 10*, (pp. 1-8).

Collins, M., & Roark, B. (2004). Incremental Parsing with the Perceptron Algorithm. *Proceedings of the 42Nd Annual Meeting on Association for Computational Linguistics.*

Covington, M. (2001). A fundamental algorithm for dependency parsing. *Proceedings of the 39th Annual ACM Southeast Conference*, (pp. 95-102).

Das, D., & Petrov, S. (2011). Unsupervised part-of-speech tagging with bilingual graph-based projections. *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, (pp. 600-609).

de Marneffe, M.-C., & Manning, C. (2008). The Stanford Typed Dependencies Representation. *Coling 2008: Proceedings of the Workshop on Cross-Framework and Cross-Domain Parser Evaluation*, (pp. 1-8).

de Marneffe, M.-C., Dozat, T., Silveira, N., Haverinen, K., Ginter, F., Nivre, J., & Manning, C. (2014). Universal Stanford dependencies: A cross-linguistic typology. *LREC*, 4585-4592.

Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., & Lin, C.-J. (2008). LIBLINEAR: A Library for Large Linear Classification. *The Journal of Machine Learning Research*, 1871-1874.

Goldberg, Y., & Nivre, J. (2012). A Dynamic Oracle for Arc-Eager Dependency Parsing. *Proceedings of COLING 2012: Technical Papers*, (pp. 959-976).

Honnibal, M., Goldberg, Y., & Johnson, M. (2013). A Non-Monotonic Arc-Eager Transition System for Dependency Parsing. *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, (pp. 163-172).

Hutchins, W., & Somers, H. (1992). *An Introduction to Machine Translation.*

Karlsson, F., Voutilainen, A., Heikkila, J., & Anttila, A. (1995). *Constraint Grammar: A Language-Independent System for Parsing Unrestricted Text.* Walter de Gruyter & Co. Hawthorne, NJ, USA.

Kübler, S., McDonald, R., Nivre, J., & Hirst, G. (2009). *Dependency Parsing.* Morgan and Claypool Publishers.

Lafferty, J., McCallum, A., & Pereira, F. (2001). Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. *Proceedings of the Eighteenth International Conference on Machine Learning*, (pp. 282-289).

Liang, P. (2005, May). Semi-Supervised Learning for Natural Language. *Master's Thesis*.

McDonald, R., & Nivre, J. (2007). Characterizing the Errors of Data-Driven Dependency Parsing Models. *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, (pp. 122-131).

McDonald, R., & Satta, G. (2007). On the Complexity of Non-projective Data-driven Dependency Parsing. *Proceedings of the 10th International Conference on Parsing Technologies*, (pp. 121-132).

McDonald, R., Nivre, J., Quirmbach-Brundage, Y., Goldberg, Y., Das, D., Ganchev, K., . . . Lee, J. (2013). Universal Dependency Annotation for Multilingual Parsing. *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, (pp. 92-97).

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., & Dean, J. (2013). Distributed Representations of Words and Phrases and Their Compositionality. *Proceedings of the 26th International Conference on Neural Information Processing Systems*, (pp. 3111-3119).

Muischnek, K., Müürisep, K., & Puolakainen, T. (2005). Adpositions in Estonian computational syntax. *Proceedings of the Second ACL-SIGSEM Workshop on the Linguistic Dimensions of Prepositions and their Use in Computational Linguistics Formalisms and Applications*, (pp. 2-10).

Muischnek, K., Müürisep, K., & Puolakainen, T. (2014a). Dependency Parsing of Estonian: Statistical and Rule-based Approaches. *Frontiers in Artificial Intelligence and Applications*, 111-118.

Muischnek, K., Müürisep, K., Puolakainen, T., & Liin, K. (2016). Parsing Estonian: Tools and Resources. *Second International Workshop on Computational Linguistics for Uralic Languages.*

Muischnek, K., Müürisep, K., Puolakainen, T., Aedmaa, E., Kirt, R., & Särg, D. (2014b). Estonian Dependency Treebank and its annotation scheme. *Proceedings of 13th Workshop on Treebanks and Linguistic Theories (TLT13)*, (pp. 285-291).

Müürisep, K. (2001). Parsing Estonian with Constraint Grammar. *Proceedings of the 13th Nordic Conference of Computational Linguistics (NODALIDA 2001).*

Nadeau, D., & Sekine, S. (2007). A survey of named entity recognition and classification. *Linguisticae Investigationes*, 3-26.

Nivre, J. (2003). An Efficient Algorithm for Projective Dependency Parsing. *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, (pp. 149-160).

Nivre, J. (2004). Incrementality in Deterministic Dependency Parsing. *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together* (pp. 50-57). Association for Computational Linguistics.

Nivre, J. (2008). Algorithms for Deterministic Incremental Dependency Parsing. *Computational Linguistics*, 513-553.

Nivre, J. (2009). Non-projective Dependency Parsing in Expected Linear Time. *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1 - Volume 1*, (pp. 351-359).

Nivre, J. (2013). *Transition-based Parsing.* Retrieved from http://stp.lingfil.uu.se/~nivre/master/transition.pdf

Nivre, J., Hall, J., & Nilsson, J. (2006). MaltParser: A Data-Driven Parser-Generator for Dependency Parsing. *Proceedings of the fifth international conference on Language Resources and Evaluation*, (pp. 2216-2219).

Nivre, J., Marneffe, M.-C. d., & Filip Ginter, Y. G. (2016). Universal Dependencies v1: A Multilingual Treebank Collection. *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*, (pp. 1659-1666).

Petrov, S., Das, D., & McDonald, R. (2012). A Universal Part-of-Speech Tagset. *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, (pp. 2089-2096).

Straka, M., Hajič, J., & Straková, J. (2016). UDPipe: Trainable Pipeline for Processing CoNLL-U Files Performing Tokenization, Morphological Analysis, POS Tagging and Parsing. *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*, (pp. 4290-4297).

Straka, M., Hajič, J., Straková, J., & Hajič jr., J. (2015). Parsing Universal Dependency Treebanks using Neural Networks and Search-Based Oracle. *Proceedings of Fourteenth International Workshop on Treebanks and Linguistic Theories (TLT 14).*

Turing, A. (1950). Computing Machinery and Intelligence. *Mind*, 433-460.

Zeman, D. (2008). Reusable Tagset Conversion Using Tagset Drivers. *Proceedings of the International Conference on Language Resources and Evaluation, LREC 2008*, (pp. 213-218).

Zeman, D., & Resnik, P. (2008). Cross-Language Parser Adaptation between Related Languages. *Proceedings of IJCNLP 2008 Workshop on NLP for Less Privileged Languages*, (pp. 35-42).

Zhang, Y., & Nivre, J. (2011). Transition-based Dependency Parsing with Rich Non-local Features. *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: Short Papers - Volume 2*, (pp. 188-193).

Zhou, H., Zhang, Y., Huang, S., & Chen, J. (2015). A Neural Probabilistic Structured-Prediction Model for Transition-Based. *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, (pp. 1213-1222).

Appendix

Appendix A – Stop Words of Estonian for spaCy

aga

ega ehk ei et

ja jaa jah

ka kas kes kui kõik

ma me mida midagi mind minu mis mu mul mulle

nad nii ning

ole oled oleme olen oli oma on

pole

sa seda see selle sest siin siis

ta te

või

ära

Appendix B – A snippet of the TAG_MAP with *all* morphological features for spaCy

TAG_MAP = {

"NOUN:Par:Sing:Past:Part:Act":{"POS":"NOUN", "Case": "Par", "Number": "Sing", "Tense": "Past", "VerbForm": "Part", "Voice": "Act"},

...

"PUNCT:Yes":{"POS":"PUNCT", "Connegative": "Yes"},

…

"VERB:Ind:Sing:3:Past:Fin:Act":{"POS":"VERB", "Mood": "Ind", "Number": "Sing", "Person": "3", "Tense": "Past", "VerbForm": "Fin", "Voice": "Act"},

…

"ADV:Part:Act":{"POS":"ADV", "VerbForm": "Part", "Voice": "Act"},

...

"ADJ:Pos:Past:Part:Pass":{"POS":"ADJ", "Degree": "Pos", "Tense": "Past", "VerbForm": "Part", "Voice": "Pass"},

…

"PRON:Nom:Sing:Int,Rel":{"POS":"PRON", "Case": "Nom", "Number": "Sing", "PronType": "Int,Rel"},

…

"PROPN:Yes":{"POS":"PROPN", "Connegative": "Yes"},

…

"CONJ:Neg":{"POS":"CONJ", "Negative": "Neg"},

…

"AUX:Ind:Plur:3:Pres:Fin:Act":{"POS":"AUX", "Mood": "Ind", "Number": "Plur", "Person": "3", "Tense": "Pres", "VerbForm": "Fin", "Voice": "Act"},

…

"ADP:Prep":{"POS":"ADP", "AdpType": "Prep"},

...

"SCONJ":{"POS":"SCONJ"},

"NUM:Gen:Sing:Letter:Card":{"POS":"NUM", "Case": "Gen", "Number": "Sing", "NumForm": "Letter", "NumType": "Card"},

...

"INTJ":{"POS":"INTJ"},

"SYM:Nom:Sing:Digit:Card":{"POS":"SYM", "Case": "Nom", "Number": "Sing", "NumForm": "Digit", "NumType": "Card"},

…

"X:Yes:Gen:Sing":{"POS":"X", "Connegative": "Yes", "Case": "Gen", "Number": "Sing"

...

}

Appendix C – A snippet of the TAG_MAP with *reduced* morphological features for spaCy

```
TAG_MAP = {
"NOUN:Nom:Sing":{"POS":"NOUN", "Case": "Nom", "Number": "Sing"},
...
"PUNCT":{"POS":"PUNCT"},
…
"VERB:Sing:3:Fin:Act":{"POS":"VERB", "Number": "Sing", "Person": "3", "VerbForm": "Fin", "Voice": "Act"},
…
"ADV":{"POS":"ADV"},
...
"ADJ:Nom:Cmp:Sing:Part":{"POS":"ADJ", "Case": "Nom", "Degree": "Cmp", "Number": "Sing", "VerbForm": "Part"},
…
"PRON:Nom:Sing":{"POS":"PRON", "Case": "Nom", "Number": "Sing"},
…
"PROPN:Nom:Sing":{"POS":"PROPN", "Case": "Nom", "Number": "Sing"},
…
"CONJ":{"POS":"CONJ"},
…
"AUX":{"POS":"AUX"},
…
"ADP":{"POS":"ADP"},
...
"SCONJ":{"POS":"SCONJ"},
"NUM:Nom:Sing":{"POS":"NUM", "Case": "Nom", "Number": "Sing"},
…
"INTJ":{"POS":"INTJ"},
"SYM":{"POS":"SYM"},
"X":{"POS":"X"}
}
```

Appendix D – TOKENIZER_EXCEPTIONS for spaCy

Word	Original Form (ORTH)	Lemma
jaan.	jaan.	jaanuar
veebr.	veebr.	veebruar
apr.	apr.	aprill
aug.	aug.	august
sept.	sept.	september
okt.	okt.	oktoober
nov.	nov.	november
dets.	dets.	detsember
jaan	jaan	jaanuar
veebr	veebr	veebruar
apr	apr	aprill
aug	aug	august
sept	sept	september
okt	okt	oktoober
nov	nov	november
dets	dets	detsember

Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Nusaeb Nur Alam**,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:

    1.1. reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until the expiry of the term of validity of the copyright, and

    1.2. make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until the expiry of the term of validity of the copyright,

of my thesis

The Comparative Evaluation of Dependency Parsers in Parsing Estonian,

supervised by Kairit Sirts,

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, **18.08.2017**