

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Akshay Aggarwal

Consistency of Linguistic Annotation

Institute of Formal and Applied Linguistics

Supervisor of the master thesis:	RNDr. Daniel Zeman, PhD Koldo Gojenola, PhD
Study programme:	Computer Science
Study branch:	Computational Linguistics

Prague 2020

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ..... signature of the author

I would like to express my gratitude to my supervisors, Daniel Zeman from Charles University for his dedicated guidance, his valuable comments as well as his recommendation of the thesis topic; and Koldo Gojenola from UPV/EHU for the much needed push to allow me to keep things structured.

I am very thankful to Prof. Markéta Lopatková and Prof. Vladislav Kuboň from Charles University, Prague and also Dr. Bobbye Pernice from Universität des Saarlandes for their immense help in handling all the difficulties encountered during the entire period of study in LCT programme.

A very special note of thanks is extended to Chiara Alzetta of ItaliaNLP Lab for her constant help in an experiment. From the discussion about the algorithm, to actually running it, the experiment on LISCA tools would not have been possible without her valuable inputs.

A very sincere vote of thanks is also extended to Charles University in Prague for the university's provision of computing and storage resources. A very heartfelt thanks is also due, to the brilliant researchers and teachers therein. The thesis would not have been possible without their brilliant shaping of the foundations of the subject within me.

Computational resources were supplied by the Ministry of Education, Youth and Sports of the Czech Republic under the Projects CESNET (Project No. LM2015042) and CERIT-Scientific Cloud (Project No. LM2015085) provided within the program Projects of Large Research, Development and Innovations Infrastructures.

Finally, I would like to thank all the people who supported me during the entire period, my family and friends. Special thanks are due to my friends in Prague for helping me keep my sanity in place.

Title: Consistency of Linguistic Annotation

Author: Akshay Aggarwal

Institute: Institute of Formal and Applied Linguistics

Supervisors: RNDr. Daniel Zeman, PhD, Institute of Formal and Applied Linguistics; Koldo Gojenola, PhD, Computer Languages and Systems, University of Basque Country (UPV-EHU), Spain

Abstract: This thesis attempts at correction of some errors and inconsistencies in different treebanks. The inconsistencies can be related to linguistic constructions, failure of the guidelines of annotation, failure to understand the guidelines on annotator's part, or random errors caused by annotators, among others. We propose a metric to attest the POS annotation consistency of different treebanks in the same language, when the annotation guidelines remain the same. We offer solutions to some previously identified inconsistencies in the scope of the Universal Dependencies Project, and check the viability of a proposed inconsistency detection tool in a low-resource setting. The solutions discussed in the thesis are language-neutral, intended to work with multiple languages with efficiency.

Keywords: Annotation Consistency; Annotation Inconsistency; Error Mining; Language Independent; Universal Dependencies; UD Project; Syntax; Morphology

Contents

List of Abbreviations **4**

1 Introduction **5**
1.1 Inter-conversion of Treebanks 5
1.2 Universal Dependencies (UD) Project 6
1.3 Motivation for the Problem 7
1.4 Formal Problem Statement 8
1.5 Data Source 8
1.6 Organizational Layout of the Document 9
1.7 A Brief Overview of Conventions Used 9

2 Problems Identified in UD Treebanks **11**
2.1 Annotation Consistency in Different Treebanks 11
2.2 Problems Caused by Change of Guidelines in UDv2 12
2.2.1 Conversion Errors in Conjunctions 12
2.2.2 `AUX` and `VERB` Distinctions 12
2.3 Non-projective Structures 13
2.3.1 Related Terms and Formal Definition 14
2.3.2 Punctuation Induced Non-Projectivity 15

3 Related Work on Solutions to Identified Problems **17**
3.1 Annotation Consistency across Treebanks 17
3.1.1 Consistency in POS Annotation 18
3.1.2 Consistency in Dependency Annotation 18
3.1.3 LISCA 19
3.2 Error Mining Methods 21
3.2.1 Automatic Error Mining Based on n-gram Approach . . . 21

4 Estimating POS Annotation Consistency of Different Treebanks in a Language (Experiment 1) **22**
4.1 KL_{cpos^3} and Metric Definition 22
4.2 Dataset 24
4.3 θ_{pos} Scores for UDv2.5 25
4.4 Dataset Size and θ_{pos} 26
4.5 Genre Distribution and θ_{pos} 30
4.5.1 Relevant Literature on Textual Genres and Their Similarity 30
4.5.2 Inter-Genre Similarity 32
4.5.3 Combination of Genres 35
4.5.4 Adulterant Genres in Dataset 38
4.6 Framing Overall Θ_{pos} Limit 40
4.7 θ_{pos} Scores for UDv2.5, Annotated To Mark Consistent And Inconsistent Treebanks 41
4.8 Discussion And Conclusion 44
4.8.1 Out of Vocabulary Words 44
4.8.2 Using θ_{pos} Scores To Localise Inconsistency 44

4.8.3 Split Into Constituent Genres As Requirement 45
4.8.4 Conclusion 45

5 `conj_head`: Head Identification Error in Coordinating Conjunctions (Experiment 2) 46
5.1 Observations About Problem Statement 46
5.1.1 Direction of Dependency 47
5.1.2 Identifying Correct Conjunct for Misdirected Dependencies 51
5.1.3 Conjunction Sandwich 55
5.2 Dataset 57
5.3 Experimental Setup 57
5.4 Algorithm 58
5.5 Evaluation and Results 68
5.5.1 Originally Non-Projective Attachments 69
5.5.2 Processing Pipeline Independent of Projectivity of Attachment 73
5.6 Discussion and Conclusion 78

6 Mining Errors in Low-Resource Languages by Combining LISCA And Cross-Validation (Experiment 3) 81
6.1 Dataset 82
6.2 Experimental Setup 83
6.3 Arcs in Focus 84
6.4 Statistics 84
6.4.1 Baseline Run 84
6.4.2 Experimental Runs 85
6.5 Analysis 85
6.5.1 Baseline vs Cross Validation: Who did it better? 86
6.5.2 Comparing Different Experimental Runs 87
6.6 Error Typologies 89
6.6.1 Identification Error of Case-Marker: `Case Error` 89
6.6.2 Annotation Error in Multi-Word Expression (MWE): `MWE Error` 90
6.6.3 Annotation Error in Construction With Reported Speech: `Reported Speech` 91
6.6.4 Head Identification Error: `Wrong Head` 92
6.6.5 Annotation Error in Proper Nouns: `Naming Error` 93
6.6.6 Dependency Head Located in Subtree: `Tree Error` 94
6.7 Results and Discussion 96
6.7.1 0-scored Arcs as Search Criteria 96
6.7.2 Cross Validation as Strategy 96
6.7.3 Error Typologies and Annotation 97
6.7.4 Conclusion 97

7 `AUX` vs. `VERB`: Attempt at Separation of Verbs and Auxiliary Verbs (Experiment 4) 98
7.1 Observations About Problem Statement 98
7.2 Dataset Definition 99
7.3 Experiment 99

7.4 Results . . . 102
7.5 Discussion of the Results . . . 103

8 Future Work Recommendations 105
8.1 Enhanced Dependencies . . . 105
8.2 Ellipsis . . . 105
8.3 `FalseNonProjective`: Introduction of False Non-Projectivity into the Annotation . . . 106
8.4 Function Words and Associated deprels . . . 106
8.5 `auxHead`: Auxiliary as Head of Dependency . . . 107
8.6 `nmod4obl`: Confusion of `nmod` and `obl` Relations . . . 108
8.7 Punctuation . . . 109
8.8 Unspecified Dependencies - `dep` deprel . . . 110

Conclusion 111

Bibliography 113

List of Figures 127

List of Tables 128

A Appendix 130
A.1 Terminology Pertaining to UD . . . 130
A.1.1 CoNLL-U Format . . . 130
A.1.2 UD Annotation . . . 131
A.2 List of Language Codes . . . 132
A.3 Multiple Treebanks in Languages (UDv2.5) . . . 135
A.4 PUD Treebanks . . . 136
A.5 Relaxations to Non-Projectivity . . . 137
A.5.1 Statistics of Non-Projectivities in UDv2.5 . . . 137

List of Abbreviations

- **CLAS**- Content-based Labelled Attachment Score
- **GS**- Gold Standard
- **LTR**- Left To Right written order language
- **MWE**- Multi-Word Entity
- **NER**- Named Entity Recognition
- **PCA**- Principal Component Analysis
- **POS**- Part Of Speech
- **RTL**- Right To Left written order language
- **sd**- Standard Deviation
- **SOTA**- State Of The Art
- **TAME**- Time, Aspect, Modality, Evidentiality
- **UD**- Universal Dependencies

1. Introduction

According to Wikipedia definition of the word[1], a treebank is a parsed text corpus, which annotates syntactic or semantic structure. Built usually (but not always) on top of a POS-annotated corpus, a treebank might seek to include phrase structure (Example- PennTreebank [Marcus et al., 1994]), dependency structure (Example- Prague Dependency Treebank [Böhmová et al., 2003]) or semantic information (Example- FrameNet [Baker et al., 1998]).

A treebank can be constructed manually, by linguists spending a considerable time developing the treebank; or semi-automatically, wherein the data is automatically annotated, and then checked for consistency. Regardless of the method used for its creation, a treebank is an essential element in the field of computational linguistics. A treebank can be used to study linguistic structures, find out features associated with a language, or to understand the constructional peculiarities within a language, among others.

In this work, our main focus is on syntactic treebanks and especially dependency treebanks, rather than semantic ones. Therefore, the term 'treebank' shall be used to refer to a syntactic (dependency) treebank henceforth, unless specified otherwise.

1.1 Inter-conversion of Treebanks

There exist a multitude of treebanks for different languages as they can be seen on Wikipedia[2], for example. As noted by Kakkonen [2006], there exist a variety of formats and annotation schemes even for the treebanks for the same language. A well known example to this is the case of distinctive POS tagging schemes for PennTreebank[3] and for British National Corpus[4], both of which are meant for annotation of English language. Kakkonen, in his work also notices that there exist tools which are meant to work for a particular tagset, or for a particular annotation scheme. Given enough similarities in annotation schemes, a (automatic) conversion process can be drafted from one annotation schema to another.

This conversion process of a treebank from one annotation scheme to another can be either 1:1 (one-to-one mapping) or n:m (many-to-many mapping). As in Machine Translation, the approach can also be pivot-based, i.e. conversion to an intermediate set, and then from the intermediate set to the desired set. For example, Interset [Zeman, 2008] uses the pivot-based approach, implemented in form of a Perl library. More often than not, the conversion between different schemes can be applied deterministically, making use of rule-based approach whenever needed.

It is important to note that not all the conversions are deterministic. If we consider an example of a dependency treebank where the dependency structure is changed from function-word head to content-word head structure, the entire dependency structures need to be modified. An attempt to approach such in-

[1] `https://en.wikipedia.org/wiki/Treebank`

[2] `https://en.wikipedia.org/wiki/Treebank#Syntactic_treebanks`

[3] `https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html`

[4] `http://www.natcorp.ox.ac.uk/docs/c5spec.html`

deterministic conversion in a deterministic manner would introduce problems in the resulting annotation. Such problems can be characterized by loss of information, loss of language-specific patterns, induced inconsistencies in the data, among others.

Knowing the downside of fully-automatic conversion techniques, one can argue that we could do the task of treebank conversion manually, rather than automatically. This is not an ideal proposition because of multiple reasons as listed:

1. The treebanks differ in size, ranging from thousands to millions of tokens. An example would be WikiText-2 dataset [Merity et al., 2017], which contains around 2 million words, extracted from Wikipedia articles. The manual annotation on data as large as this requires time, money and significant human effort.

2. In case of multiple annotators for a given data, the different annotators may not always agree on the annotation principles for the same amount of data. This is especially the case when the guidelines are not specific enough, or in cases where the local grammatical tradition differs from the guidelines.

3. In case of low-resource languages, it might be difficult to find a knowledgeable annotator for the language, thus rendering the process to be painfully slow, and in some cases inaccurate.

To combat this problem, an approach of semi-automatic conversion is preferred over manual or fully-automatic conversion. The semi-automated conversion procedure can be done by converting the data from one annotation style to another automatically, followed by a human annotator verifying the results and correcting them if needed. A trade-off between the fully-automatic and manual techniques, the semi-automatic approach is considerably faster than its manual counterpart, and allows the conversion process to be controlled for a higher degree of quality-check as compared to a fully-automatic approach (cf. Fort and Sagot [2010]). In practice, there can be a significant number of iterations (or revisions) of the treebanks that might be needed before the converted data is again available at par with or better than the data quality in the original scheme. Since the research breakthroughs and improvements don't wait for the data to be perfect, the task of checking for consistency, and/or quality of the treebanks has gained momentum in recent years as a research problem.

1.2 Universal Dependencies (UD) Project

A rather more detailed history of UD Project can be accessed online[5]. This section deals with a shorter version thereof.

As elaborated in the previous section, there are multiple and (possibly) conflicting annotation styles, even for the treebanks for the same language. Like any other measurement criteria where the standardized unit (in form of SI unit, or ISO standards) was needed to be defined, the different annotation styles required a similar form of standardization.

[5] `https://universaldependencies.org/introduction.html#history`

Although there already existed annotation schemes that were used as de facto standards, with the example of The Stanford dependencies [de Marneffe et al., 2013], Google universal tagset [Petrov et al., 2012], HamleDT [Zeman et al., 2014a], among others. However, there was still the problem of which annotation style to go for. McDonald et al. [2013] in their Universal Dependency Treebank (UDT) Project tried to provide with a universal annotation language, covering 6 languages in 2013, and expanding to 11 languages the following year.

With the modifications resulting in development of HamleDT 2.0 [Zeman et al., 2014b], and Universal Stanford Dependencies (USD) [de Marneffe et al., 2014], the Universal Dependencies (UD) Project was thus born in 2014 as a means of unifying all the novel features of different annotation formats as a universal annotation scheme consistent among different languages.

The version 1.0 of UD (also referred to as UDv1.0) [Nivre et al., 2015] was launched in January 2015, and covered 10 treebanks in 10 different languages. With the iterative methodology, the project evolved to contain 146 treebanks in 83 languages in UDv2.4 [Nivre et al., 2019], and 157 treebanks in 90 languages in UDv2.5 [Zeman et al., 2019]. It is worth noting that not all the treebanks were manually annotated directly in the UD style. Rather, most treebanks are semi-automatically converted from the original source to the UD format according to a set of guidelines[6].

1.3 Motivation for the Problem

Since the introduction of UD, it has fast become a standard reference to compare scores relating to parser performance (Che et al. [2018], Alonso et al. [2017]), study of language-specific features [Alzetta et al., 2018], and for dependency parsing shared tasks on UD [Zeman et al., 2018]. Given how different UD treebanks are being considered as benchmarks for comparison of different scores, it only makes sense to be considered them as Gold Standard (GS) data.

We discussed earlier how many of the UD treebanks are generated through a semi-automatic process, and thus are liable to contain a significant amount of errors. Such errors are detrimental in a GS, because of multiple reasons. Some of the reasons are listed as follows:

1. In the case of parser evaluation, the parser learns errors from the data as well, replicating them when used on test data. While this affects parser evaluation scores, it also means that the parser does not learn the features of the language correctly, thus causing increasingly more errors on unseen data.

2. Since semi-automatic conversion is also likely to introduce more errors, this can result in inflating already known errors, and/or deflating known features. These patterns can become a nuisance on the treebank-level or might disappear altogether. Consider the case of a language-feature F which is a rare phenomenon in language L, with the relative occurrence of $x_0\%$ in the original data. Due to conversion process, it is possible that the relative occurrence might change to $x_1\%$, where $x_1 \neq x_0$.

[6] `https://universaldependencies.org/guidelines.html`

- In case of the inflation of error ($x_1 > x_0$), the data which otherwise did not exhibit F suddenly starts displaying the pattern, thus affecting the quality of the data.
- In case of the deflation of pattern ($x_1 < x_0$), the data might not exhibit F at all, increasing its rarity. Considering the case of parser evaluation as above, the parser might decide to overlook this feature in entirety, thus losing out on essential data.

3. With respect to identification of language-specific features, it is very possible that a lot of features might start getting wrongly associated with a language (the case of inflation as above) or they might be deemed a rare status (the case of deflation as above). Such instances, while seemingly harmless for high or medium resource languages, can pose serious problems with respect to low-resource languages, impacting the way the given language is studied.

The problems as mentioned above are but a subset of multiple problems associated with an erroneous GS, and how they affect UD and the research around it. As such, these problems need to be minimized as much as possible, with attempts at their elimination in an ideal case. However, doing the task (of correcting the GS) manually is again a difficult one and the automatic methods are not always 100% reliable and/or effective. While the methods often work well for individual languages or a language family, they often fail to generalize in a language-neutral sense. This is because of the different properties of languages, different language families, among others.

1.4 Formal Problem Statement

Having learned about the UD project, problems concerning semi-automatic conversions, and possible effects of these problems within the scope of UD, we can now formulate our problem statement for the scope of the thesis as follows:

Given the different treebanks in UD, the thesis aims to identify errors and inconsistencies in treebanks, and provide corresponding automated correction tools for them. The inconsistencies might be related to linguistic annotation, improper adherence to guidelines, lack of guidelines related to an observed phenomenon, annotator caused error, among others. The proposed methods and tools should ideally not require a human annotator for verification, and should be as language-neutral as possible. The tools can be adapted with language specific methods but that is out of the scope of this thesis.

1.5 Data Source

When the work on the present thesis document was started in February 2019, UDv2.3 [Nivre et al., 2018] was the latest release. Most of the experiments contained within this document were first developed for UDv2.3. However, with the release of UDv2.4 and subsequently UDv2.5, the experiments were carried forward to the newer dataset. The results throughout the length of the document are reported over UDv2.4 and UDv2.5 data.

There are some experiments that work well for UDv2.4 and UDv2.5 throughout, and there are some that work better for only one of the releases, mainly owing to the error instance being fixed in iterative format, and/or continuously evolving guidelines. To facilitate the understanding of individual experiments better, each experiment shall contain a note specifying the dataset (which also mentions the release version of UD) on which the experiment was conducted.

1.6 Organizational Layout of the Document

We first continue the preface of the document by very quickly noting a few conventions that are used throughout this document. In Chapter 2, we take a look at the different categorisation of errors, and then the typology of different problems identified in UD treebanks. We continue the document with Chapter 3, containing the background on the research pertaining to the problems from Chapter 2. In the subsequent chapters (Chapters 4 - 7), we layout the individual problems, and elaborate on the method/approach undertaken to solve the problem(s). In Chapter 8, we discuss on some of the open problems as identified by other authors, which were not undertaken in the current work. We officially conclude the document with a chapter on Conclusions.

Attached to the document are also a series of Appendices. The appendices contain the data meant to help the reader understand some of the terms used through out the document, with an example being a list of ISO language codes used throughout.

1.7 A Brief Overview of Conventions Used

This section is an overview on some of the important conventions used throughout the length of the document.

1. The following conventions hold with respect to the UD terminology. A short introduction to different terms associated with UD can be accessed in Appendix A.1.

    - Unless otherwise mentioned, part-of-speech (POS) refers to 'UPOS' field in the CoNLL-U format (Appendix A.1.1). The two terms are used interchangeably, unless mentioned otherwise.
    - Syntactic Relations in UD can be referred to by either of 'relation(s)', 'deprel', or 'dependency relation'. Unless otherwise mentioned, the instances refer to the 'DEPREL' field in CoNLL-U format (Appendix A.1.1).

2. The POS tags, as well as dependency relations are formatted in the same formatting style, with one essential difference. Both the categories are marked typographically using a separate tag in LaTeX. We refer to this formatting style as 'Tag' category, shown in the example below.

    Example 1. `VERB` is a POS tag, while `nmod` is a deprel.

3. The POS tags are always capitalized (written in upper-case), while the deprels are always non-capitalized (written in lower-case).

4. The use of 'Tag' category is also reserved for nomenclature of problems. Thus, a problem identified as `ProblemX` will act as the unique identifier for the problem across the length of the document.

5. The languages are referred to by their language-identification codes whenever possible.

   - A complete list of languages in UDv2.5, with their identification codes can be seen in Appendix A.2.
   - The language codes are also formatted using 'Tag' category as defined above. Given the nature of the dependency relations, it should be easily possible to disambiguate the language code from the former.
   - In case of an unclear distinction, the language name corresponding to the language-code shall follow in parentheses.

6. The name of the different treebanks are written in the format of `LanguageCode`-treebank_name. The truecasing in the name of the treebank is optional.
   For example, the SynTagRus treebank for `ru` can be referred to by either of `ru`-syntagrus or `ru`-SynTagRus.

7. The tokens taken from a language other than `en` follow a pattern when mentioned inline:

   - For the tokens with Latin based orthography, the token is marked in bold, followed by a literal translation in parenthesis. Consider the following example from `nl`, written inline in text with `en`.

     Example 2. Lorem ipsum text **hier** (here).

   - For the tokens with non-Latin based orthography, the token is again marked in bold, followed by the transliteration of the token in italics, and the literal translation of the token in regular case, separated by a semi-colon. The transliteration, and the translation are mentioned in the parenthesis following the token. Consider the following example from `ru`, written inline in text with `en`.

     Example 3. **да** (*da*; yes), this is Lorem ipsum text here.

   - For the case where LTR (Left-To-Right) languages are mentioned inline with RTL (Right-To-Left) languages, the transliteration and translation are written for the tokens in the order of utterance. Consider the following example, assuming **A, B and C** is written in RTL as **C and B ,A**. The example would be written inline with `en` in the following way:

     Example 4. This is the Lorem Ipsum for RTL language- **C and B ,A** (*A, B and C*; A, B and C)

2. Problems Identified in UD Treebanks

Ever since the UD project was introduced in 2014, and since the revision of guidelines in UDv2, there have been multiple publications that highlight the problems in UD treebanks. Some of the problems highlighted in these publications have been found to be global in nature (i.e. they occur in almost all treebanks, regardless of the language), while the others are related to a specific group of languages. Before we start discussing the problems, we shall specify the general kind of errors.

Agrawal et al. [2013] define different kinds of errors that can be found in a treebank. The first kind are the random errors, characterised by the inconsistencies introduced by the annotators owing to the distractions while undertaking the annotation procedure. The systematic and recurrent errors are introduced not in isolated scenarios as random errors, but can be found across the treebank in a consistent manner. These errors are usually related to the guidelines of the treebank, in either of two ways. The guidelines could be misunderstood by the annotator(s), and/or the guidelines might themselves be unclear (or not appropriate to handle some cases), leaving the annotator(s) in a jeopardy. Alzetta et al. [2017] extend the definition of systemic and recurrent errors to also include the cases of conversion errors, caused by improper mapping of original annotation scheme to a new scheme. Throughout the length of this document, we focus on the errors of the second kind (systemic and recurrent errors), and propose corrective measures.

It is worth pointing out why the experiments listed in this thesis were chosen to work on, and not others. As we will see, apart from the first problem listed in next few sections, almost all of the error typologies were pointed out from a common source [Alzetta et al., 2017]. The authors of the paper note that the mined patterns were found to be common across different sections of the `it` treebank, and across different languages as well. We therefore work on the set of error types as identified by Alzetta et al. [2017], and work on them, given that the error types are common across different treebanks.

2.1 Annotation Consistency in Different Treebanks

UDv2.5 [Zeman et al., 2019], as mentioned earlier, contains 157 treebanks in 90 languages. As such, there are multiple languages with more than one treebank, with some containing up to 6 treebanks. A list of languages in UDv2.5 such that they contain more than one treebank is listed in Appendix A.3. Regardless of the differences in genre or the teams involved for building the treebank, the different treebanks for a language should be consistent with respect to the annotation guideline(s), both intra and inter treebanks. However, this is often not the case, primarily because of the different sources of origin of the individual treebanks.

The problem of determining the degree to which the different treebanks differ

from each other has been studied in some detail over multiple years, but is not yet entirely solved. We discuss the different solutions proposed over time regarding this problem in Sections 3.1.1 and 3.1.2.

We return to this problem in Chapter 4, when we try to devise a metric to compare the different treebanks on basis of their POS annotation.

2.2 Problems Caused by Change of Guidelines in UDv2

A summarized version of changed guidelines from UDv1 to UDv2 can be accessed online[1]. Most of the changes in guidelines could be processed in an automatic manner. For example the renaming of particular POS tags or dependency relations could be implemented across the different treebanks in a deterministic manner. However, there were some changes that could not be applied deterministically, and those form the majority of the problems in this section.

It is important to note that the changes had to be applied to 64 treebanks in 47 languages as they moved from UDv1.4 to UDv2.0, and so the analysis might be limited to these 64 treebanks only in this case. However, it is worth scouting for these patterns in the newer treebanks, given how some (if not all) of them might be a cause of concern therein.

The dependency tree structures shown throughout the length of this document are generated as per Parsito format [Straka et al., 2015].

2.2.1 Conversion Errors in Conjunctions

In the changed guidelines, there were two changes with respect to conjunction tags `CCONJ` and `SCONJ`; and the dependency relations, `cc` and `conj`. The changes are listed as follows:

1. The POS tag `CONJ` in UDv1 was changed to `CCONJ` in UDv2, to make it more parallel to `SCONJ`.

2. The conjunctions are attached to the immediately succeeding conjunct in UDv2, as opposed to UDv1 where they were attached to the first conjunct.

Of the two changes in guidelines, the first one (renaming of tag) can be applied deterministically, and automatically throughout the treebank(s). The second change, however, can be classified as head identification error. The pattern in question (referred to as `conj_head` henceforth) was also identified by Alzetta et al. [2017] in their paper, where they note that it contributes to 24.65 % of total discovered error instances and is major error category. Keeping this in consideration, we take a look at this error type in Chapter 5 in detail.

2.2.2 `AUX` and `VERB` Distinctions

The following is a list of changes for the category of auxiliaries from UDv1.4 to UDv2:

[1] `https://universaldependencies.org/v2/summary.html`

1. The definition of AUX was extended to include copula verbs, and non-verbal TAME (Time, Aspect, Mood, Evidentiality. Might/might not include Voice and Polarity) particles.

2. The aux relation was also expanded to include non-verbal TAME particles, as in the case of AUX.

3. The relation auxpass was removed from UDv2.0, making it as a subcategory of the larger aux relation, in the form of aux:pass. Essentially speaking, auxpass was demoted to a sub-category of aux relation.

Considering the changes that the auxiliaries went under with the change in guidelines, the line of distinction between the POS VERB and AUX became fuzzier. At the time of writing this document, the distinction between the two is not always explicit. For a given language, this is also governed by the definition of the terms in UD, and how those definitions agree with the traditional language-grammar. This is noted in part in the guidelines for UDv2 as well, where the following point is noted, with reference to the definition[2] of AUX:

```
[AUX] is often a verb (which may have non-auxiliary uses as
well) but many languages have nonverbal TAME markers and
these should also be tagged AUX.
```

One of the proposed change in guidelines was to get rid of AUX altogether[3]. However, as per findings of de Lhoneux and Nivre [2016], a parser is not able to learn the distinction between the two categories, when they are merged together. The authors observe a decrease in parsing scores when the two categories are not explicitly separated. This was the principal motivation behind keeping the two separate. However, there still exist problems with respect to the differentiation between the two categories, as can be seen in the list of open issues on the subject[4].

In UDv2.4, it was proposed to limit the AUX of each language by a list. The list would essentially identify all auxiliaries by a common definition, and thus would be able to create a better distinction between the two conflicting categories of AUX and VERB. This could be realized just in part though, principally because of the conflicts between traditional grammar-based definitions of the two categories, and the definitions as per UD.

With respect to this particular error type, we tried to segregate the classes of AUX and VERB in our experiments in Chapter 7, without using the aforementioned list.

2.3 Non-projective Structures

While non-projectivity is not tackled as an issue in the scope of the current research, it is nonetheless an important linguistic phenomenon that warrants attention. In this section, we cover the concept as a primer, such that the reader

[2] https://universaldependencies.org/u/pos/all.html#aux-auxiliary

[3] https://github.com/UniversalDependencies/docs/issues/275

[4] https://github.com/universaldependencies/docs/issues?utf8=%E2%9C%93&q=is%3Aopen+aux

is not lost about the topic when it is referred to in future chapters. We discuss an open problem about non-projective structures later in Section 8.3.

Let us understand non-projectivity through the following example from LinES treebank in `en` data, and the tree structure as shown in Figure 2.1.

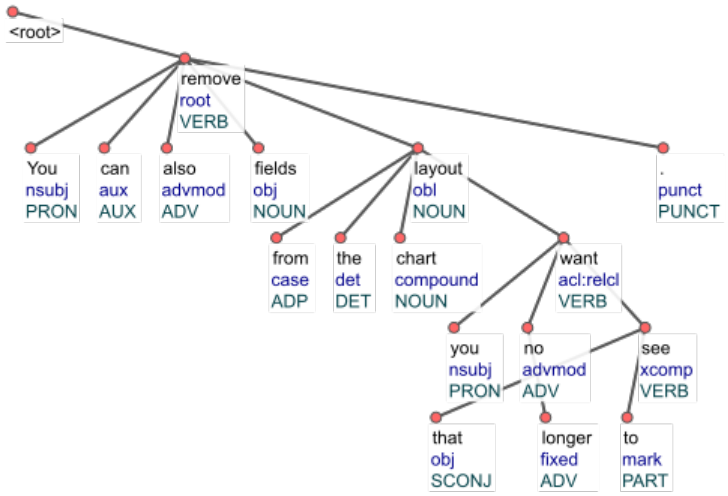


Figure 2.1: Sample Non-projective Tree
Note: *that* should be tagged as `PRON`, and not as `SCONJ`

In the graph, notice the edge going from *see* to *that*. We can see that the edge crosses over another edge in order to link the two tokens. Informally, presence of such crossing edges in a tree makes it non-projective in nature.

2.3.1 Related Terms and Formal Definition

To define the concept of non-projective structures in a formal manner, we need to define a few notations. We use the same notations as used by Mambrini and Passarotti [2013].

If a node j depends on a node i, we call node j as a child node of i (also, i is parent node of j), represented as $i \rightarrow j$. We use $i < j$ to denote the node i precedes node j in the word-order in tree T. A node v lying in between the nodes i and j in the tree can be represented as $v \in (i, j)$. Also, we use the notation $v \in Subtree_i$ if node v is part of the subtree rooted at node i.

From Havelka [2007], we can define the condition of projectivity of a tree as follows:

Definition 1. A given tree T is projective in nature iff

$$i \rightarrow j \;\; \& \;\; v \in (i, j) \implies v \in Subtree_i \qquad \forall i, j, v \in T \tag{2.1}$$

If a given tree does not satisfy the above condition, it is said to be non-projective in nature. Furthermore, in case of non-projectivity, node v is said to be in gap, represented as $v \in Gap_{i \leftrightarrow j}$. The double headed arrow signifies the nodes being considered irrelevant of their order of occurrence in the tree.

Mambrini and Passarotti [2013], in their work on `grc`, highlight that the distribution of non-projective structures might be affected by genre distribution. In

particular, poetic style is liable to contain more non-projective structures than prose. The claim about genre distribution affecting projectivity is also supported by Yadav et al. [2017], where they look at different genres (news and conversations) using different parameters to account for lack of non-projective structures in the conversational genre, than in news genre.

2.3.2 Punctuation Induced Non-Projectivity

A punctuation node can induce non-projectivity in either of the two ways as mentioned below:

1. Non-projective attachment of a punctuation node.
2. Non-projectivity caused by only punctuation node(s) in gap.

According to UD guidelines, a punctuation node should be attached to the surrounding dependent unit. However, it is not always possible to identify the correct dependent where the node should be attached. Consider the following example from `en`-lines UD v2.5 treebank, and the associated dependency tree in Figure 2.2, with specific reference to the punctuation mark immediately following the token marked in bold. While the punctuation token could have been correctly marked to either of *right* or *said*, it is attached to *'s* causing non-projectivity.

Example 5. That's **right**, said Quinn.

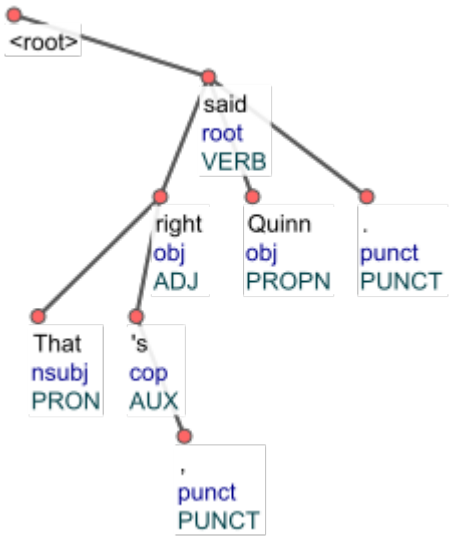


Figure 2.2: Punctuation Node Attached Non-Projectively

Similarly, the punctuation node(s) can induce non-projectivity, by attaching itself to the wrong node. Consider the following example from `en`-EWT UDv2.5 treebank, and the associated dependency tree in Figure 2.3, with specific reference to the punctuation mark immediately following the token in bold. A faulty association of this punctuation induces non-projectivity in another node.

Example 6. Analyst Team 1 : **Coach** : Lisa Gilette

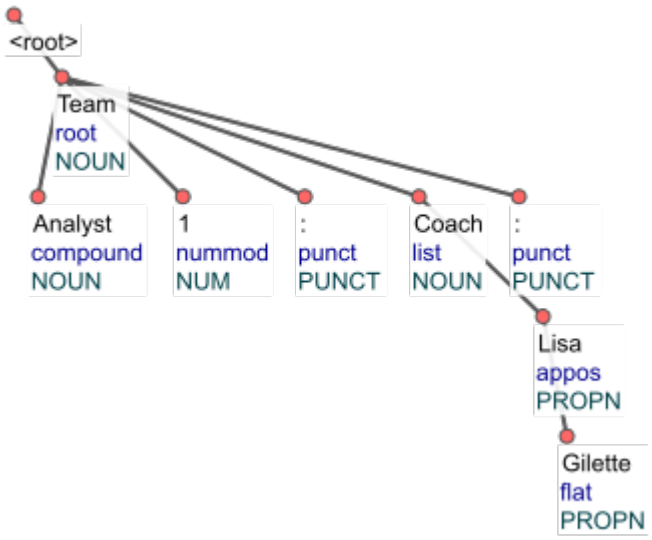


Figure 2.3: Punctuation Node Causing Non-Projectivity

To summarise, we can say that a non-projective edge $i \rightarrow j$ is a case of Punctuation Induced non-projectivty if any of the following conditions are met:

1. Either of head (node i), or dependent (node j) is a punctuation node.
2. The nodes in $Gap(i, j)$ consist of **only** punctuation node(s).

Projectivity in itself is a strict constraint for a multitude of natural languages. Therefore, there have been multiple relaxations that have been suggested over time on the strict constraint of projectivity. Appendix A.5 discusses some of these relaxations, and then lists in tabular form the statistics related to non-projectivity in different treebanks in UDv2.5 data.

3. Related Work on Solutions to Identified Problems

In this chapter, we discuss some of the solutions that have been proposed or used by the different researchers. The solutions discussed here are limited in the scope of the problems identified in the last chapter. It is important to note that there have been numerous papers studying the different treebanks in UD, and the set of problems encountered while changing the annotation from the guidelines for UDv1 to UDv2. While such research is helpful in pointing out cases where the annotating teams had difficulties during the conversion procedure, we do not discuss those references here, unless needed.

Before proceeding further, it is imperative to understand a subtle difference between error detection and inconsistency detection. If the errors are consistent in their distribution across the data, an inconsistency detection tool would fail in the discovery of such errors. In such case, the non erroneous part of the annotation would be the inconsistency and might be flagged as a false negative, provided the tool is biased towards the erroneous annotation. Any tool that tries to discover inconsistencies need not find such consistent error patterns. This is the major difference between error detection and inconsistency detection. Error mining methods are primarily based on detecting deviations from a standard clean reference (usually gold or platinum standard), and should be able to provide an analysis of the error patterns regardless of whether or not the error is present consistently. In this chapter, we use error mining and error detection interchangeably.

The rest of the chapter is organised as follows. We first discuss existing literature on inconsistency detection in Section 3.1, and the relevance of the literature to the problem identified in Section 2.1 on Annotation Consistency in Different Treebanks. We then focus on the literature relevant to error detection in Section 3.2.

3.1 Annotation Consistency across Treebanks

Owing to different annotation schemes for the different treebanks of a given language, there is no standard evaluation metric to compare the consistency of treebanks' annotation to each other to the best of our knowledge.

One of the most commonly used approaches to find the inconsistencies in the annotation is to train a high quality parser or a tagger model on a given training data, and evaluating the cases where the prediction from the trained model differs from the annotation of the test data. This approach can also be extended by bootstrapping different trained models, with the majority consensus being compared against the available annotation. While this approach can point to individual inconsistencies, it does not say anything about the errors in the treebank. Furthermore, the different treebanks of the same language can have different annotation inconsistencies with the errors being consistent in their presence throughout. Additionally, the consistent errors in the different treebanks can be vastly different from each other as well.

To ascertain annotation quality of one or more treebanks, both inconsistency detection and error detection should be used. In case of individual treebanks, UD website[1] shows against each treebank a metric score that is an approximation of the quality of the treebank. The metric is calculated heuristically[2], depending on multiple factors like the number of genres present in the treebank, the score as provided by official UD validator[3], among others. When it comes to comparing annotation quality among multiple treebanks, there exist no metrics or tools to the best of our knowledge. However, some techniques have been used more often than others for such comparisons.

3.1.1 Consistency in POS Annotation

Dickinson and Meurers [2003a,b] are the most well known pieces of work in detecting inconsistency in POS annotation, essentially forming the base of majority of inconsistency detection. The work focuses on finding a n-gram of tokens in the corpus that occurs in the same context (referred to as a variation nucleus) such that the different occurrences of the variation nucleus are annotated differently. Originally coined for continuous annotation[4], the method was eventually adapted to look for inconsistencies in discontinuous annotation as well [Dickinson and Meurers, 2005].

Chun et al. [2018] compare the POS annotation consistency for different treebanks in `ko` by using the relative frequency of the individual POS tags. The authors also briefly mention the cause of the variation in distribution of the individual POS tags. While such analysis is slightly helpful in terms of drawing a comparison, it does not consider the interaction of different POS tags with each other. To illustrate such interactions, a n-gram based approach might be utilised. Even so, absence of `SCONJ` tag in one treebank prevents the analysis with respect to other treebanks.

3.1.2 Consistency in Dependency Annotation

The original method of using variation nuclei for continuous annotation as proposed by Dickinson and Meurers [2003a,b] was extended for discontinuous annotation in Dickinson and Meurers [2005], as mentioned earlier. By extending the method to discontinuous annotations, Dickinson and Meurers were able to look at more patterns in TIGER corpus. Moreover, this meant that instead of looking at plain POS tags and identifying the variations therein, the words could now be looked at in order to generalize the context.

Alonso and Zeman [2016] compared the treebanks for `es` in UDv1.3 [Nivre et al., 2016]. They assess the similarity of the different treebanks using depen-

[1] `universaldependencies.org`

[2] For more details on the associated heuristics, refer to `https://github.com/UniversalDependencies/tools/blob/master/evaluate_treebank.pl`

[3] refer to `https://github.com/UniversalDependencies/tools/blob/master/validate.py`

[4] The annotation of the current token is based on the annotation of a contiguous token in word order. Discontinuous annotation implies the annotation of current token is dependent on another token that might not be contiguous in the word order, as in the case of dependency parsing.

dency parsing. A high-efficiency parser was trained on one of the treebanks, and then tested on another. The idea was to notice the drop in LAS scores, and if the difference in scores was more than what was intuitive, the treebanks were marked as not similar enough. The same technique of evaluating the different treebanks for `ru` against each other was also used in Droganova et al. [2018]. In their work spanning the different `ru` treebanks in UD, Droganova et al. also point out problems with individual treebanks. The problems pointed out therein can be used as a starting point to scout for patterns that are present across the different treebanks for the language.

Nivre and Fang [2017] proposed an evaluation metric called CLAS (Content-based LAS) score that disregard the punctuation and other functional nodes, evaluating LAS based on content words only. The change of evaluation metric from LAS to CLAS was meant as a way to give equal treatment to the languages with weak morphology and languages with strong morphology. For example, a single inconsistency in `fi` will affect the parsing score more than a single inconsistency in `en` owing to the differences in the extent of morphology used by the languages. The metric was evaluated as a secondary measure in CoNLL 2017 Shared Task [Zeman et al., 2017]. The primary metric for the Shared Task was macro-averaged LAS score for the different languages. It was reported that there is no significant performance difference in parser performances when the evaluation metric was changed from macro-averaged LAS score to CLAS score.

An important point to note here is that the metrics LAS and CLAS are associated with the performance of parsers. The metric scores would be lower in case even when the parser is able to parse the data better than the manual annotation. The two metrics (and also unlabelled attached score or UAS) therefore cannot be relied upon for detection of the inconsistencies.

Chun et al. [2018] compare the dependency annotation consistency among different treebanks in `ko` by again focusing on the relative frequency of the dependency labels, offering reasons for the variation in distribution of the individual label. A dependency label is determined by the choice of the parent label as well, and thus the method of Chun et al. is of little help in flagging any inconsistencies.

3.1.3 LISCA

Dell'Orletta et al. [2013] used an unsupervised algorithm which attempts to find the inconsistencies in dependency annotation by building a statistical model on the data from a given reference corpus (ideally, a gold standard). This algorithm, called LISCA, creates a language model for the given dependency arcs, learning for each arc its probability of occurrence based on a subset of local and global attributes associated with the arc. The eventually created language model can then be used to rank the dependency arcs in another parsed corpus by their probability of occurrence. Figure 3.1 shows graphically some of the features used by LISCA to calculate score for an arc.

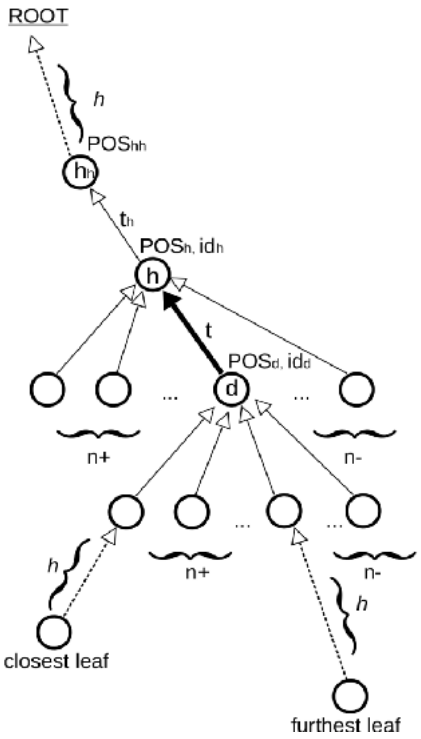


Figure 3.1: Features Used by LISCA to Calculate Plausibility Score for an Arc (marked in bold). Figure borrowed from Alzetta et al. [2017]

Local Feature: Distance in terms of tokens between d and h
Local Feature: Associative strength linking grammatical categories POS_d and POS_h
Local Feature: POS of the head governor and type of syntactic dependency connecting it to h
Global Feature: Distance of d from the root of the tree
Global Feature: Distance of d from the closest or the most distant leaf node
Global Feature: Number of siblings to the right of node d in the linear order of the sentence
Global Feature: Number of children to the left of node d in the linear order of the sentence

LISCA was used to identify the errors in newspaper section of Italian UD Treebank in Alzetta et al. [2017]. In their work, they narrow the search space for the errors by binning the arcs according to the scores into 10 bins of equal size and an extra bin to include the extra cases. The bins were then manually inspected for errors, while concentrating on the last two (and the extra) bins containing the arcs with lowest scores. Analysing the data, 36% of the arcs in the low ranking bins consisted of random errors, while the remaining ones were found to be systemic and recurrent errors (even in treebanks of different languages).

While the algorithm mentioned above successfully points out the arcs that are inconsistent in their annotation in the different datasets, it is sensitive to the genre of the data. The authors note that the data should ideally belong to the same register or genre for the algorithm to function at its best. While this is problematic because in some treebanks it is not possible to separate the data from different genres, there is also a possibility of unavailability of enough data

in a particular genre (i.e. a single genre contributing in a very small manner to the size of the treebank).

Added to these difficulties is the difficulty of training the algorithm. The algorithm essentially needs to be trained on a gold standard data, from which it builds a statistical model that is used to generate the probability scores of a dependency arc. In case of languages with no high-quality parsers available or for low-resource languages, this poses a cold-start problem where we do not have the data to train the algorithm, and so the algorithm cannot be used at all.

We tried solving this problem of cold-start by using the method of k-fold cross validation (with varying values of k) in training the algorithm. We discuss the experiment in more detail in Chapter 6.

3.2 Error Mining Methods

Error mining in treebanks can be done in multiple ways. There is a possibility of using hand-written rules, and scouting for the patterns in the relevant treebank. This manual approach works well for finding error typologies that are known beforehand. The other approach is to combine the statistical approach, with the manually defined rules [Ambati et al., 2011]. This method is referred to as heuristics-based search since it identifies a lot of patterns, which can then be used to look for errors in the data (in some cases, this can be done automatically). The last approach is automatic scouting for error patterns within the scope of the treebank, also known as automatic error mining.

3.2.1 Automatic Error Mining Based on n-gram Approach

Boyd et al. [2008] first introduced the idea of error mining methods in dependency treebanks using variation nuclei, expanding on the idea of using n-grams based variation nuclei for discontinuous annotations from Dickinson and Meurers [2005]. This is often referred to as the first automatic error mining method in dependency treebanks.

de Marneffe et al. [2017] extended and evaluated the method proposed by Boyd et al., in context of UD Treebanks for three languages (`en`, `fi`, `fr`). The authors further extended the method to use word lemmas instead of simply using word forms, and also evaluate on the automatically annotated treebanks to identify more inconsistencies. The first extension of using lemmas works well for languages that are not too morphologically-rich (`en`, `fr`), but fails otherwise. The second extension is done at the cost of a drop in precision, but without a significant gain in recall.

The method proposed by Boyd et al. has an inherent problem instance of data sparseness. de Kok et al. [2009] implemented an algorithm based on n-grams and suspicion sharing across the n-grams by extending the methods of Sagot and de la Clergerie [2006] and van Noord [2004]. Their approach however, relies on classifying each sentence within the results of a parsed corpus as a parsable or unparsable sentence. This classification of individual sentence needs to be done manually, and is therefore not optimal for large treebanks.

4. Estimating POS Annotation Consistency of Different Treebanks in a Language (Experiment 1)

We introduced the problem of inter treebank POS annotation quality in Section 2.1 earlier, followed by a discussion of the literature relevant to the problem in Section 3.1.1.

In this chapter, we propose a metric to estimate the POS annotation consistency of treebanks. The metric is based on KL_{cpos^3} metric [Rosa and Žabokrtský, 2015], which in turn is based upon Kullback-Liebler Divergence (KL Divergence).

We start by a short introduction to KL_{cpos^3} metric and a definition of the proposed metric in Section 4.1. We define our dataset for the experiments in this chapter in Section 4.2, followed by the metric values being listed for different treebanks in UDv2.5 [Zeman et al., 2019] in Section 4.3. The experiments are detailed in Section 4.4 and Section 4.5, with their results summarised in Section 4.6. We mark the treebanks as consistent or inconsistent in their POS annotation in Section 4.7. The chapter concludes with a discussion on the metric in Section 4.8.

4.1 KL_{cpos^3} and Metric Definition

In a delexicalised cross-language model transfer scenario, Rosa and Žabokrtský [2015] show that KL-Divergence score of POS trigrams can be effectively used for source selection for POS Tagging . In their approach, they are able to select effectively not just a single source, but are also able to rank multiple sources by specifying weights to individual source in a multi-source transfer scenario. Computing the KL-Divergence on POS trigrams, they call the measure KL_{cpos^3}, defined as follows:

Definition 2.

$$KL_{cpos^3}(tgt, src) = \sum_{\forall cpos^3 \in tgt} f_{tgt}(cpos^3) \log \frac{f_{tgt}(cpos^3)}{f_{src}(cpos^3)} \tag{4.1}$$

where $cpos^3$ is a coarse POS tag trigram, and

$$\begin{aligned} f(cpos^3) &= f(cpos_{i-1}, cpos_i, cpos_{i+1}) \\ &= \frac{count(cpos_{i-1}, cpos_i, cpos_{i+1})}{\sum_{\forall cpos_{a,b,c}} count(cpos_a, cpos_b, cpos_c)} \end{aligned} \tag{4.2}$$

with $count_{src}(cpos^3) = 1$ for each unseen trigram.

Intuitively, treebanks of the same language (despite the differences in the genres covered) should be better fit for single-source transfer than a treebank from another language. This is the primary motivation for using KL_{cpos^3} (as defined for a single-source transfer scenario) to assess the annotation consistency among the treebanks of a language. However, KL_{cpos^3} is a variant of KL-Divergence, and thus asymmetric, making it unfit in its original form for assessing annotation consistency symmetrically. We refer to the symmetric variant of the metric as θ_{pos} defined for the treebanks A and B as follows:

$$\boxed{\theta_{pos}(A, B) = KL_{cpos^3}(A, B) + KL_{cpos^3}(B, A)} \tag{4.3}$$

where $KL_{cpos^3}(P, Q)$ indicates KL_{cpos^3} score of Q as an estimator for P.

Since KL_{cpos^3} is a non-negative divergence metric, so is θ_{pos}. While either metric is numeric in nature, the KL_{cpos^3} scores can be used as an estimator of quality in presence of an absolute gold standard. However, in absence of an absolute gold standard, the scores for the metric in different treebanks can not be compared directly. In such case (of lack of absolute gold standard), there should be an upper bound that needs to be placed on the θ_{pos} scores. As long as the θ_{pos} scores are lower than this upper bound, the considered pair of treebanks can be considered as harmonious in terms of their POS annotation. We call this upper bound as Θ_{pos}. The metrics θ_{pos} and Θ_{pos} are linked together in the following definition.

Definition 3. Given two treebanks A and B, we say the treebanks are in harmony with (or, are harmonious to) each other in terms of POS annotation, if the symmetric measure of their mutual divergence (given by θ_{pos}) is less than or equal to a threshold (given by Θ_{pos}).
Formally, it can be represented as:

$$\begin{aligned} \boxed{\theta_{pos}(A, B) = KL_{cpos^3}(A, B) + KL_{cpos^3}(B, A)} \\ \leq \Theta_{pos}(A, B) \end{aligned} \tag{4.4}$$

where $KL_{cpos^3}(P, Q)$ indicates KL_{cpos^3} score of Q as an estimator for P.

Even though Θ_{pos} is a bound on the θ_{pos} metric, the former is essentially a property of the latter. For a given set of guidelines, and a given set of data, the upper bound value would need to be estimated often, albeit using the same technique. In the remaining chapter, we try to estimate the upper bound in a language-independent manner by looking at the influence of size of data, and the POS distribution in individual genres on θ_{pos} metric. While the methods that we shall discuss shortly can be applied for estimations across different guidelines and different set of data, care must be taken while estimating the upper bound for a new guideline (or even on different iterations of UD data). If the estimated value of Θ_{pos} is too large, we run the risk of saying the treebanks are harmonious even when they might not be. Also, if the value is too small, we could be overlooking at the effect of domain change and dataset size, to mistakenly announce the pair of treebanks as being non-harmonious to each other.

4.2 Dataset

UDv2.5 [Zeman et al., 2019] contains 157 treebanks in 90 languages. There are multiple languages with more than one treebank, with some containing up to 6 treebanks. A list of all such languages, with the associated treebanks can be seen in Appendix A.3. We list θ_{pos} scores of the different treebanks in different languages in the next section. In the listing of scores, small treebanks where the total number of sentences is 1000 or less are not included.

As mentioned earlier, the treebanks in UD are assigned a score based on a variety of factors, including the errors identified by the official UD validator, among others. The score rating of a treebank can be loosely understood as an evaluation of how well the treebank adheres to the UD guidelines. While it is possible to have a high score without the treebank being internally consistent, it is logical to assume that a treebank that adheres better to the guidelines will contain fewer inconsistency errors.

We want to estimate the Θ_{pos} scores to the best of our ability, and so, working with a pair of low quality treebanks would be the worst approach that can be undertaken. To that effect, we estimate the bounding score on treebanks with the ratings of at least 3.5 stars (out of 5 stars). The treebanks selected in this manner can be considered to be of high quality. The selection of treebanks in this manner also enforces an important assumption, that there is a considerably lower number of annotation inconsistencies within the data in a treebank. The assumption would also imply that in a pair of considered treebanks, while the treebanks might not be annotated consistently with respect to each other, the individual treebanks are assumed to be internally consistent with respect to their annotation.

The assumption as mentioned above is a strict constraint, and might not always hold. An alternative assumption can be used in cases where the stricter version is not expected to hold. The relaxed version of the assumption assumes that the data belonging to one particular genre in a treebank would be annotated consistently throughout. This is a relaxation in the sense that given multiple genres in a treebank, the entire treebank might not be annotated consistently. However, the data in individual genres is annotated consistently. The experiments listed in this chapter work within the bound of these assumptions.

4.3 θ_{pos} Scores for UDv2.5

Languages with 2 Treebanks

Treebank1	Treebank2	θ_{pos}
ar-NYUAD	ar-PADT	2.497
es-AnCora	es-GSD	0.352
et-EDT	et-EWT	0.413
fi-FTB	fi-TDT	1.195
gl-CTG	gl-TreeGal	0.714
grc-Perseus	grc-PROIEL	4.641
ja-GSD	ja-BCCWJ	0.951
ko-GSD	ko-Kaist	2.56
nl-Alpino	nl-LassySmall	0.664
pl-LFG	pl-PDB	0.623
pt-Bosque	pt-GSD	0.678
ro-Nonstandard	ro-RRT	1.233
sl-SSJ	sl-SST	2.405
sv-LinES	sv-Talbanken	0.443
tr-GB	tr-IMST	1.477
zh-GSD	zh-HK	1.958

Languages with 3 Treebanks

de	GSD	HDT
HDT	0.49	-
LIT	1.383	1.1

la	ITTB	Perseus
Perseus	1.106	-
PROIEL	3.763	3.901

no	Bokmaal	Nynorsk
Nynorsk	0.095	-
NynorskLIA	2.291	2.375

Languages with 3+ Treebanks

ru	GSD	SynTagRus
SynTagRus	0.567	-
Taiga	1.027	0.631

cs	CAC	CLTT	FicTree
CLTT	1.453	-	-
FicTree	1.138	2.657	-
PDT	0.373	1.935	1.006

en	EWT	GUM	LinES	ParTUT
GUM	0.26	-	-	-
LinES	0.407	0.455	-	-
ParTUT	0.62	0.432	0.581	-
ESL	0.592	0.799	0.564	0.823

fr	FQB	GSD	ParTUT	Sequoia	Spoken
GSD	1.582	-	-	-	-
ParTUT	1.942	0.683	-	-	-
Sequoia	1.693	0.248	0.524	-	-
Spoken	3.644	3.089	2.599	2.732	-
FTB	2.226	0.379	0.7	0.272	3.507

it	ISDT	ParTUT	VIT	PoSTWITA
ParTUT	0.133	-	-	-
VIT	0.121	0.194	-	-
PoSTWITA	1.67	1.478	1.764	-
TWITTIRO	1.501	1.376	1.594	0.347

4.4 Dataset Size and θ_{pos}

$KL_{cpos^3}(tgt, src)$ is defined on distributions of trigrams found in *tgt* and *src*. The calculated metric scores should therefore be affected by the presence or absence of the POS trigrams. The presence or absence of POS trigrams can similarly affect the calculations of θ_{pos} metric scores. In this part of the experiment, we use k-fold cross validation to check the effect of presence or absence of POS trigrams in the data. We use k-fold cross validation here as it allows us to check how the calculated scores are affected based on the size of the data alone, and also to frame an association of the scores with the presence or absence of POS trigrams, if any.

The presence or absence of data from different genres can affect the calculation of θ_{pos} scores. In order to discount such effect, the entire data used for the analysis should belong to the same genre. For this experiment, we used `cs`-PDT (rated 4.5/5 stars) and `et`-EDT (rated 4/5 stars) treebanks. The motivation behind the selection of languages is primarily the difference in their language families. Additionally, the two treebanks contain a large number of sentences belonging to the *news* genre, making it easier for the data to be studied across multiple k-fold runs with different k-values. Table 4.1 lists the sentences counts associated to the considered genres in either treebank.

Language	**Genre**	**Sentences**
`cs`	News	53 075
`et`	News	13 557

Table 4.1: Sentence Counts in `cs`-PDT and `et`-EDT Treebanks

To check the effect of data size on θ_{pos} metric, we ran k-fold cross validation on the data from the aforementioned treebank in the following manner:

1. Concatenate the different splits of the treebank together before downsampling the concatenated data to a fixed number of instances.

2. For different predetermined k-values, the downsampled data is split into k folds. In each fold, the θ_{pos} scores are calculated between the fold's splits.

3. In each fold, we try to estimate the projection of trigram distribution from the test set for the fold, onto the training set for the fold. Considering that the larger training set will contain more trigrams, we estimate the projection from the smaller test set. Essentially, the training set in a fold corresponds to *src*, while the test set corresponds to *tgt*. We calculate coverage of different POS trigrams in each fold. The coverage is calculated by counting the number of trigrams common to both *src* and *tgt*, expressed as a percentage of the total number of trigrams in *tgt*.

The methodology as stated above is listed for a single repetition over a single treebank. To get a better estimation of the values, the method was repeated 100 times each for both the treebanks. In each repetition, the seed values were uniquely selected so as to get different downsamples every time. Table 4.2 lists the number of instances the treebank was downsampled to, and the considered k

values for the downsampled data. The table also lists the θ_{pos} scores and coverage scores for each fold. The scores are averaged over the 100 repetitions for each k-value, with the standard deviation (sd) also mentioned therein.

Language	**Downsample**	**k value**	θ_{pos} **Score**	**Coverage (in %)**
cs	50000	5	0.021 ± 0.001	83.904 ± 0.563
		10	0.037 ± 0.001	75.457 ± 0.602
		20	0.069 ± 0.002	66.138 ± 0.656
		50	0.161 ± 0.005	52.754 ± 0.832
		100	0.304 ± 0.011	42.368 ± 0.843
		250	0.663 ± 0.03	29.353 ± 0.864
		500	1.092 ± 0.063	20.802 ± 1.021
et	12000	4	0.064 ± 0.002	76.15 ± 0.807
		6	0.087 ± 0.003	69.739 ± 0.957
		8	0.109 ± 0.004	65.237 ± 0.83
		12	0.155 ± 0.006	58.667 ± 1.032
		16	0.2 ± 0.007	54.124 ± 1.029
		24	0.286 ± 0.012	47.77 ± 1.046
		48	0.52 ± 0.02	37.096 ± 0.947
		120	1.038 ± 0.053	24.485 ± 1.151

Table 4.2: θ_{pos} and Coverage of POS Trigram Scores (± sd) Averaged over 100 Different Runs to Highlight the Effect of Size Disparity. The values in the θ_{pos} and Coverage columns are the representative scores for the k-value, selected from the scores of individual runs such that the score is statistically equal to scores of more than 50% of the runs in the fold. The statistical value is calculated at 95% confidence using One Sampled t-test.

Looking at the scores for the two languages, there is a clear negative correlation between coverage and θ_{pos} score. Coverage of different POS trigrams is, however, dependent upon the size of the datasets being compared. In case of a really small dataset, the number of different POS trigrams or even the total number of POS trigrams is not comparable.

Figures 4.1 and 4.2 consist of two graphs each. The graphs show how the number of (i) distinct POS trigrams, and (ii) total number of POS trigrams is affected by a change in the dataset size. While the first graph in each figure shows the variability across the entire downsampled data (50000 sentences in `cs` in Figure 4.1, and 12000 sentences in `et` in Figure 4.2); the second graph zooms in on the progression over 2000 sentences.

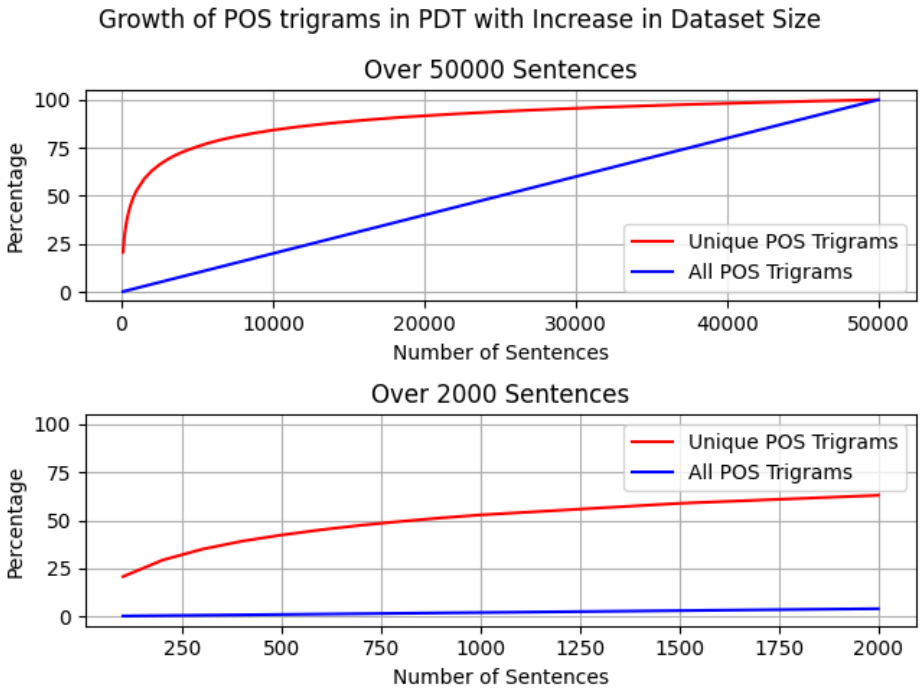


Figure 4.1: Growth of POS Trigrams in PDT with Increase in Dataset Size

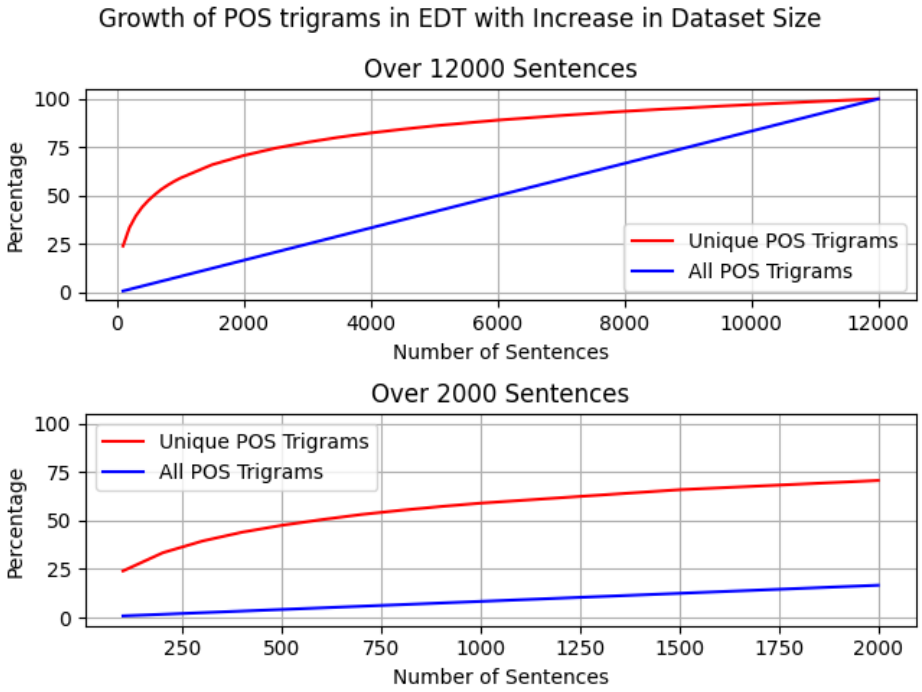


Figure 4.2: Growth of POS Trigrams in EDT with Increase in Dataset Size

As can be seen from the figures, the growth pattern is similar across both the languages. We can see that in case of a considerably small dataset size, the POS trigrams can not be considered as representative of those present in the entire dataset. We claim that for a proper estimation of the annotation consistency in two datasets belonging to the same genre, either dataset requires at least 400 sentences ($\approx$ 40% of unique POS trigrams) for the estimation to be reliable. The

minimum limitation on the size of the datasets ensures that the distribution of POS trigrams in either dataset is not skewed because of a small size.

Claim 1. *Data across two datasets A, B can be compared iff*

$$\boxed{size(A) \geq 400 \;\; \& \;\; size(B) \geq 400}$$

where $size(X)$ refers to the size of dataset X in terms of the number of sentences

Table 4.3 shows the average sentence length of sentences in different treebanks for `ar`. If we consider equal number of sentences from either of `ar`-NYUAD or `ar`-PADT treebanks and compare the POS annotation consistency with `ar`-PUD treebanks, the total number of syntactic words differ by a factor of almost 2.

Counts	`ar`-NYUAD	`ar`-PADT	`ar`-PUD
Syntactic Words	738889	282384	20751
Sentences	19738	7664	1000
Average	37.434	36.845	20.751

Table 4.3: Average Sentence Lengths in `ar` Treebanks. In `es`, the token **vámonos** (Let's go) is split into 2 syntactic words **vamos** (go-*1P-Pl.*) and **nos** (*1P.-Pl.*) for annotation.

When calculating the θ_{pos} scores for a set of treebanks, the average sentence length in either treebank should also be taken into account. It makes sense to limit the size of the datasets in consideration not in absolute terms, but also in reference to each other. Keeping this in mind, we update Claim 1 to account for the average sentence length in Claim 2.

Claim 2. *Data across two datasets A, B can be compared iff*

$$\boxed{size(A) \geq 400 \;\; \& \;\; Avg(A) \geq Avg(B) \implies size(B) \cdot \frac{Avg(B)}{Avg(A)} \geq 400} \tag{4.5}$$

where

1. $Avg(X) = \frac{TotalSyntacticWords(X)}{size(X)}$ is the average sentence length in dataset X
2. $size(X)$ refers to the number of sentences in dataset X

From the results of the data in Table 4.2, when the test split is composed of 500 instances ($k = 100$ for `cs`; $k = 24$ for `et`), the θ_{pos} metric is ≈ 0.3. Considering that the larger k-values in either dataset do not satisfy the condition in Equation 4.5, we use the values as per the aforementioned k-values to estimate the maximal value for θ_{pos} when there is a size variance in the datasets.

As mentioned earlier, the treebanks in the consideration are ranked high in their quality check. Considering that some treebanks might not have such high quality of annotation, we allow some room for the change in θ_{pos} metric.

If the datasets A, B contain data from the same genre, and the size of the datasets is comparable (as per Equation 4.5), the upper limit on the θ_{pos} score can be specified as per Equation 4.6.

$$\boxed{\theta_{pos}(A, B) \leq \Theta_{pos}(A, B) = 0.5} \tag{4.6}$$

4.5 Genre Distribution and θ_{pos}

There can be significant difference(s) between genres in terms of syntactic annotations that are typical of the genre. While this difference is best exhibited across treebanks containing data from different genres, it can also be exhibited within a given treebank. The problem of music genre classification in speech data has been studied in detail, with different audio similarity metrics being proposed as well (cf. Kalapatap et al. [2017], Pampalk et al. [2005], among others). In the written data, while there has been some research on the study of inter-genre variations for language acquisition [Casañ-Pitarch, 2017], the classification of genres in textual corpus is identified mainly by the source of data.

4.5.1 Relevant Literature on Textual Genres and Their Similarity

In Biber [1989], a line of distinction is drawn between text type and genre as the basis of classification of texts. While the former is 'defined and distinguished on the basis of systematic nonlinguistic criteria', the latter is 'defined on the basis of strictly linguistic criteria (similarities in the use of cooccuring linguistic features)' [Biber, 1989, p. 39]. In Biber [1991], the different genres in `en` are studied in different dimensions, focusing on one dimension at a time. The dimensions are a group of factors that associate the different features of a discourse, and are as listed in Table 4.4. In the same work, the author notes that a given genre can contain multiple sub-genres which may or may not be internally coherent to each other [Biber, 1991, p. 170], and that no dimension in itself can attribute to the similarity or dissimilarity of the genres. In a later study that seeked to understand the variations of the genres based on these identified dimensions across 4 languages, the author notes that 'even when defined at a high level of generality, parallel registers are more similar cross-linguistically than are disparate registers within a single language' [Biber, 1995, p. 279].

S.No.	Dimension Name	Characteristic of Dimension
1.	**Involved vs Informational Production**	interactional, affective, involved purposes, associated with strict real-time production and comprehension constraints
2.	Narrative vs Non-Narrative Concerns	primary narrative purpose
3.	**Explicit vs Situation-Dependent Reference**	identifies referents fully and explicitly through relativization
4.	Overt Expression of Persuasion	speaker's expression of own point of view or with argumentative styles to persuade the addressee
5.	Abstract vs Non-Abstract Information	highly abstract and technical informational focus
6.	**On-Line Information Elaboration**	production under highly constrained conditions where information is presented in relatively loose, fragmented manner

Table 4.4: Identified Dimensions for Comparison of Genres. The characteristic of individual dimensions is as found in [Biber, 1991, p. 115]. Dimension 5 on 'Abstract vs Non-Abstract Information' is noted to be not universal across all languages [Biber, 1995, p. 278]

The dimensions marked in bold in Table 4.4 can be summarised under the notion of *deep formality*, as coined in Heylighen and Dewaele [1999]. Heylighen and Dewaele are able to classify linguistic constructions into different genres according to the measurement of their formality, based on a numerical measure of formality. The numerical measure, however doesn't account for all the dimensions marked in bold, but mainly to the first dimension on 'Involved vs Informational Production'. The formality of a construction was numerically calculated in terms of F-measure (formality measure), as defined in Equation 4.7. Mosquera and Pozo [2011] discovered that a numerical I-measure (informality measure, needed for working with Web2.0 data, given in Equation 4.8) combined with F-measure worked better in identification of formality levels in data than when either of the measure was used on its own.

$$\text{F-measure} = \frac{f_{noun}+f_{adjective}+f_{preposition}+f_{article}-f_{pronoun}-f_{verb}-f_{adverb}-f_{interjection}+100}{2} \tag{4.7}$$

$$\text{I-measure} = (f_{mistyped} + f_{interjection} + f_{emoticon}) * 100 \tag{4.8}$$

where f_A represents frequency of A.

In our experiment, we tried to experiment with a combination of F-measure and I-measure, as well as with the measures by themselves. Considering that the absolute frequency would be dependent on the size of the database, the measure scores were computed in terms of relative frequencies. However, we found no correlation between θ_{pos} scores between two genres, with their F-measure or I-measure scores or a combination of the two.

4.5.2 Inter-Genre Similarity

pl-LFG treebank in UDv2.5 (rated 4 stars on a scale of 5 stars) contains data from 8 different genres[1]. The sentence counts of different genres are shown in Table 4.5. We club together the different kind of data in *spoken* genre, as one. We remove *academic*, *blog* and *legal* data from our consideration owing to a considerably low number of sentences. Table 4.6 shows the genre distribution in UDv2.5 fi-TDT data. In this case, the data with source as *europarl* and university articles (*uni_articles*) is kept separate from other categories. The genres we work with are marked in bold in the table.

Genre	Sentence Count	Avg()
fiction	7 252	7.124
news	6 744	8.401
nonfiction	1 273	7.719
social	526	6.977
spoken	1253	6.047
academic	51	8.118
blog	136	7.772
legal	11	9.273

Table 4.5: Genre Distribution in UDv2.5 pl-LFG treebank

Genre	Sentence Count	Avg()
fiction	2739	11.981
wiki	2269	14.049
grammar	2002	8.48
blog	1781	12.533
legal	1141	20.968
news	3064	13.026
europarl	1082	18.441
uni_articles	1058	13.261

Table 4.6: Genre Distribution in UDv2.5 fi-TDT treebank

In order to establish that the different genres are annotated consistently within themselves, we downsample the dataset for each genre in fi-TDT treebank to 900 sentences. On this downsampled data, we perform 2-fold cross validation split, and calculate the θ_{pos} score for the splits. We repeat this calculation 100 times, such that the data is downsampled differently each time, as per a different seed value. Table 4.7 shows the calculated θ_{pos} scores averaged over 100 different runs.

[1]For understanding of what genre category involves exactly what kind of data, refer to the github page of the treebank at https://github.com/UniversalDependencies/UD_Polish-LFG

Genres	θ_{pos} ($\pm$ sd)	Θ_{pos}
fiction	0.316 $\pm$ 0.015	0.5
wiki	0.3 $\pm$ 0.017	0.5
grammar	0.427 $\pm$ 0.021	0.5
blog	0.332 $\pm$ 0.017	0.5
legal	0.216 $\pm$ 0.035	0.5
news	0.286 $\pm$ 0.015	0.5
europarl	0.233 $\pm$ 0.017	0.5
uni_articles	0.3 $\pm$ 0.014	0.5

Table 4.7: θ_{pos} ($\pm$ sd) Scores Averaged Over 100 Different Runs for Different Genres in UDv2.5 `fi`-TDT Treebank To Show Intra-Genre Annotation Consistency

As can be seen from Table 4.7, the different genres in the treebank are internally consistent in their annotation, as per the constraint in Equation 4.5.

We start the inter-genre analysis by downsampling the datasets for different genres in the dataset. Table 4.8 shows the count of sentences in the downsampled data from each genre. Each genre is downsampled to the number of instances such that the condition as specified in Equation 4.5 is satisfied.

Language	**Genre (X)**	**Downsampled To**	$size(X) \cdot \frac{Avg(X)}{Avg(A)}$
`pl`	fiction	500	424
	news	500	500
	nonfiction	500	459
	social	500	415
	spoken	600	432
`fi`	fiction	1000	571
	wiki	1000	670
	grammar	1000	404
	blog	1000	598
	legal	1000	1000
	news	1000	621

Table 4.8: Counts of Sentences for Different Genres in Downsampled Data from UDv2.5 `fi`-TDT and `pl`-LFG Treebanks. A in $Avg(A)$ in the third column refers to the genre with the highest number of average words per sentence in each language, marked in bold.

For downsampled data from each genre, we compute the θ_{pos} scores. We present the scores for `pl` data in Table 4.9 and for `fi` data in Table 4.10. It is worth noting that for most genres, the Θ_{pos} constraint as employed in Equation 4.5 isn't enough, as θ_{pos} frequently surpasses the imposed limit of 0.5.

Genres	news	nonfiction	social	spoken
fiction	0.754 ± 0.047	0.556 ± 0.028	0.726 ± 0.032	1.059 ± 0.047
news	-	0.55 ± 0.032	0.906 ± 0.044	1.53 ± 0.071
nonfiction	-	-	0.624 ± 0.027	1.285 ± 0.046
social	-	-	-	1.178 ± 0.033

Table 4.9: θ_{pos} Scores (± sd) Averaged over 100 runs for Inter-Genre Analysis in Downsampled UDv2.5 `pl`-LFG Data

Genres	blog	grammar	wiki	legal	news
fiction	0.356 ± 0.014	0.47 ± 0.019	1.552 ± 0.041	1.559 ± 0.04	1.323 ± 0.044
blog	-	0.504 ± 0.018	1.307 ± 0.042	1.328 ± 0.026	1.113 ± 0.043
grammar	-	-	1.166 ± 0.041	1.554 ± 0.036	0.888 ± 0.035
wiki	-	-	-	1.229 ± 0.032	0.473 ± 0.021
legal	-	-	-	-	1.078 ± 0.026

Table 4.10: θ_{pos} Scores (± sd) Averaged over 100 runs for Inter-Genre Analysis in Downsampled UDv2.5 `fi`-TDT Data

We attempted to associate the θ_{pos} scores across two genres based on if the genre belonged to spoken discourse, or from textual medium. However, as can be seen from the tables above, the scores can not be estimated on the basis of such distinction.

Looking at the data in the tables above, the maximal θ_{pos} score of 1.559 is computed between *legal* and *fiction* categories. We hypothesise that a combination of F-score metric with a metric on 'Narrative vs Non-Narrative Concerns' can be used to explain such high score. However, there exists no numeric metric to compute a genre's score on its 'Narrative vs Non-Narrative Concerns' to the best of our knowledge. We therefore, are unable to associate the upper limit on θ_{pos} scores with respect to individual genres.

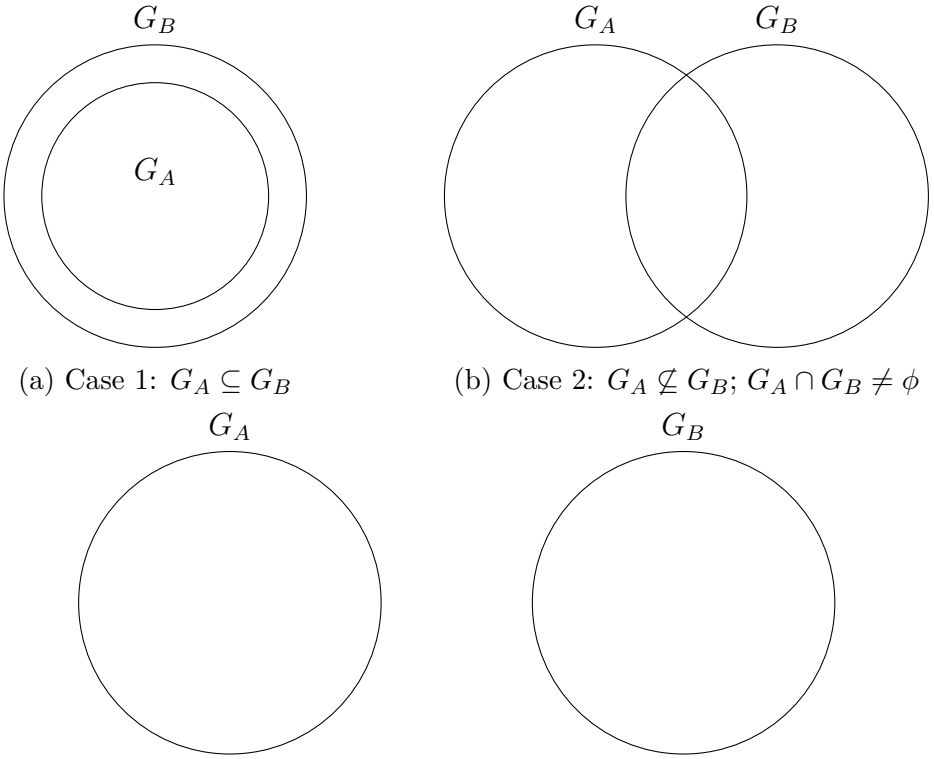
However, we can estimate a general upper bound. We allow some room for change in θ_{pos} score owing to high quality of annotation as while accounting for variability of dataset size change. With that in mind, we frame the general upper bound on θ_{pos} scores between genre x in dataset A (written as A_x) and genre y in dataset B (written as B_y) as in Equation 4.9, given below:

$$\boxed{\theta_{pos}(A_x, B_y) \leq \Theta_{pos}(A_x, B_y) = 2.0} \tag{4.9}$$

4.5.3 Combination of Genres

In the previous section, we looked at how the θ_{pos} score changes when data from one genre is compared against another. In this subsection, we study how the different genres in combination with each other affect the θ_{pos} scores.

We denoted the set of genres in treebank X as G_X. Given two treebanks A and B with at least one different genre, the different genres in the two treebanks G_A and G_B can be either of the three cases as shown in Figure 4.3.



(a) Case 1: $G_A \subseteq G_B$

(b) Case 2: $G_A \not\subseteq G_B$; $G_A \cap G_B \neq \phi$

(c) Case 3: $G_A \not\subseteq G_B$; $G_A \cap G_B = \phi$

Figure 4.3: Interaction of Genres in Treebanks A and B, such that $|G_A| \leq |G_B|$

To see how the θ_{pos} scores are affected in either of the cases, we perform the following experiment on UDv2.5 `pl`-LFG data.

1. Downsample the number of sentences in *fiction* and *news* genres to 2000 sentences each. Using 2-fold cross-validation, split the downsampled into 2 halves. We refer to one half as *base* set for the genre, and the other as the *test* set for the genre, each containing 1000 sentences.

2. Downsample the number of sentences in *spoken* genre to 1000 sentences.

3. Concatenate the downsampled *spoken* data and the *test* set from the other genres. Refer to this dataset as *all_genres*.

$$all_genres = spoken + fiction_test + news_test$$

4. Combine the *test* sets to result in *news_fiction_test* set.

$$news_fiction_test = news_test + fiction_test$$

5. Combine the *base* sets to result in *news_fiction_base* set.

$$news_fiction_base = news_base + fiction_base$$

6. Combine the downsampled *spoken* data with either *test* set to result in *spoken_genre_test* data, where *genre* is a placeholder for either of *fiction* or *news*.

$$spoken_news_test = spoken + news_test$$
$$spoken_fiction_test = spoken + fiction_test$$

7. For Case 1, we study the change in θ_{pos} score when the data as mentioned in Table 4.11 are compared against each other.

G_A	G_B
$G_{news_base} = \{news\}$	$G_{news_fiction_test} = \{news, fiction\}$
$G_{news_base} = \{news\}$	$G_{spoken_news_test} = \{spoken, news\}$
$G_{news_base} = \{news\}$	$G_{all_genres} = \{news, fiction, spoken\}$
$G_{fiction_base} = \{fiction\}$	$G_{news_fiction_test} = \{news, fiction\}$
$G_{fiction_base} = \{fiction\}$	$G_{spoken_fiction_test} = \{spoken, fiction\}$
$G_{fiction_base} = \{fiction\}$	$G_{all_genres} = \{news, fiction, spoken\}$
$G_{news_fiction_base} = \{news, fiction\}$	$G_{all_genres} = \{news, fiction, spoken\}$

Table 4.11: Datasets Compared when $G_A \subset G_B$ and $|G_A| < |G_B|$

8. For Case 2, we study the change in θ_{pos} score when the data as mentioned in Table 4.12 are compared against each other.

G_A	G_B
$G_{news_fiction_base} = \{news, fiction\}$	$G_{spoken_news_test} = \{spoken, news\}$
$G_{news_fiction_base} = \{news, fiction\}$	$G_{spoken_fiction_test} = \{spoken, fiction\}$

Table 4.12: Datasets Compared when $G_A \not\subseteq G_B$; $G_A \cap G_B \neq \phi$ and $|G_A| \leq |G_B|$

9. For Case 3, we study the combinations as listed in Table 4.13.

G_A	G_B
$G_{news_base} = \{news\}$	$G_{spoken_fiction_test} = \{spoken, fiction\}$
$G_{fiction_base} = \{fiction\}$	$G_{spoken_news_test} = \{spoken, news\}$
$G_{spoken} = \{spoken\}$	$G_{news_fiction_test} = \{news, fiction\}$

Table 4.13: Datasets Compared when $G_A \not\subseteq G_B$; $G_A \cap G_B = \phi$ and $|G_A| \leq |G_B|$

10. We also calculate θ_{pos} scores for each *base* and *test* sets with each other, and with *spoken* data, to better know how the scores are being impacted.

11. We repeat all the above steps 100 times, each with a different seed value to result in differently downsampled data. We present the calculated scores averaged over 100 runs for different cases in Tables 4.14 - 4.16. In the tables, the 'Average' column contains the average of means from the preceding columns.

	news_test	*fiction_test*	Average	*news_fiction_test*
news_base	0.257 ± 0.01	0.646 ± 0.034	0.452	0.304 ± 0.016
fiction_base	0.64 ± 0.034	0.278 ± 0.013	0.46	0.348 ± 0.019

	news_test	*spoken*	Average	*spoken_news_test*
news_base	0.257 ± 0.01	1.503 ± 0.049	0.88	0.489 ± 0.022

	fiction_test	*spoken*	Average	*spoken_fiction_test*
fiction_base	0.278 ± 0.013	0.99 ± 0.036	0.63	0.41 ± 0.018

	news_test	*fiction_test*	*spoken*	Average	*all_genres*
news_base	0.257 ± 0.01	0.646 ± 0.034	1.503 ± 0.049	0.802	0.463 ± 0.022
fiction_base	0.64 ± 0.034	0.278 ± 0.013	0.99 ± 0.036	0.64	0.351 ± 0.014
news_fiction_base	0.3 ± 0.015	0.351 ± 0.021	1.144 ± 0.035	0.6	0.247 ± 0.011

Table 4.14: θ_{pos} (± sd) Scores Averaged over 100 Runs, Reported for Case When $G_A \subset G_B$ and $|G_A| < |G_B|$

	spoken	*news_test*	Average	*spoken_news_test*
news_base	1.503 ± 0.049	0.257 ± 0.01	0.88	0.489 ± 0.022
fiction_base	0.99 ± 0.036	0.64 ± 0.034	0.81	0.499 ± 0.02
news_fiction_base	1.144 ± 0.035	0.3 ± 0.015	0.7	0.498 ± 0.023

	spoken	*fiction_test*	Average	*spoken_fiction_test*
news_base	1.503 ± 0.049	0.646 ± 0.034	1.075	0.854 ± 0.036
fiction_base	0.99 ± 0.036	0.278 ± 0.013	0.63	0.41 ± 0.018
news_fiction_base	1.144 ± 0.035	0.351 ± 0.021	0.747	0.498 ± 0.023

Table 4.15: θ_{pos} (± sd) Scores Averaged over 100 Runs, Reported for Case When $G_A \not\subseteq G_B$; $G_A \cap G_B \neq \phi$ and $|G_A| \leq |G_B|$

	spoken	*fiction_test*	Average	*spoken_fiction_test*
news_base	1.503 ± 0.049	0.646 ± 0.034	1.075	0.854 ± 0.036

	spoken	*news_test*	Average	*spoken_news_test*
fiction_base	0.99 ± 0.036	0.64 ± 0.034	0.81	0.499 ± 0.02

	news_test	*fiction_test*	Average	*news_fiction_test*
spoken	1.493 ± 0.048	0.987 ± 0.034	1.24	1.138 ± 0.03

Table 4.16: θ_{pos} (± sd) Scores Averaged over 100 Runs, Reported for Case When $G_A \not\subseteq G_B$; $G_A \cap G_B = \phi$ and $|G_A| \leq |G_B|$

From the tables, it can be observed that the decomposition of a treebank into its constituent genres forms the first basis for the combination of the different genres. Once the individual genres have been identified and checked for the inter-generic θ_{pos} scores, the overall metric score is less than the average of the metric scores calculated for individual pair of genres in the treebank(s). Upon a closer inspection, it was discovered that when there are multiple genres present in the treebank, the θ_{pos} metric score is dominated by the POS trigrams that are typical of the language, and the genre-specific POS trigrams become more and more obscure.

Assuming treebanks A and B can be split into their constituent genres such that $G_A = \{A_1, A_2, ..., A_i\}$ and $G_B = \{B_1, B_2, ..., B_j\}$, the overall limit on the $\theta_{pos}(A, B)$ score can be specified as in Equation 4.10.

$$\boxed{\theta_{pos}(A,B) \leq \Theta_{pos}(A,B) = Average(\theta_{pos}(A_x, B_y))} \quad \forall[A_x \in G_A; B_y \in G_B] \tag{4.10}$$

4.5.4 Adulterant Genres in Dataset

In our analysis so far, we have restricted ourselves to instances when the data in the different genres could be reliably compared as per Equation 4.5 reproduced below:

$$\boxed{size(A) \geq 400 \;\;\&\;\; Avg(A) \geq Avg(B) \implies size(B) \cdot \frac{Avg(B)}{Avg(A)} \geq 400} \tag{4.5}$$

where

1. $Avg(X) = \frac{TotalSyntacticWords(X)}{size(X)}$ is the average sentence length in dataset X
2. $size(X)$ refers to the number of sentences in dataset X

We define a genre as an adulterant genre in the dataset if the number of instances in the genre does not satisfy Equation 4.5. In this subsection, we take a look at how the presence of adulterant genres affect the θ_{pos} scores.

From Table 4.9, the maximal θ_{pos} score of 1.53 was computed between *news* and *spoken* genres in data from `pl`-LFG treebank. We calculate the effect of the adulterant genres in the following manner:

1. Downsample data from *fiction*, *news* and *spoken* genres in `pl`-LFG treebank to 500, 500 and 600 sentences respectively.
2. Concatenate data from each of *academic*, *blog* and *legal* genres with the downsampled *fiction* dataset to result in *fiction-academic*, *fiction-blog*, and *fiction-legal* datasets.
3. Repeat Step2 above after replacing downsampled *fiction* dataset with the downsampled *news* dataset to result in *news-academic*, *news-blog*, and *news-legal* datasets.

4. Concatenate the downsampled datasets from *fiction* and *news* genre to result in *fiction_news* dataset.

5. Concatenate data from each of *academic*, *blog*, *legal* genres with the *fiction_news* dataset to result in *fiction_news-academic*, *fiction_news-blog* and *fiction_news-legal* datasets.

6. Concatenate *academic*, *blog* and *legal* datasets to result in a *others* dataset.

7. Concatenate downsampled *fiction* dataset with the *others* dataset to result in *fiction-others* dataset.

8. Repeat Step7 above after replacing downsampled *fiction* dataset with downsampled *news* dataset to result in *news-others* dataset.

9. Concatenate downsampled *fiction* and *news* datasets with the *others* dataset to result in a *all-genres* dataset.

10. Calculate θ_{pos} score of downsampled *spoken* dataset with each of the created datasets.

	spoken
fiction	1.059 ± 0.047
fiction-academic	1.072 ± 0.046
fiction-blog	1.09 ± 0.044
fiction-legal	1.065 ± 0.047
fiction-others	2.413 ± 0.384

	spoken
news	1.53 ± 0.071
news-academic	1.552 ± 0.069
news-blog	1.54 ± 0.065
news-legal	1.547 ± 0.071
news-others	2.63 ± 0.334

	spoken
fiction	1.059 ± 0.047
news	1.53 ± 0.071
fiction_news	1.196 ± 0.048
fiction_news-academic	1.215 ± 0.048
fiction_news-blog	1.223 ± 0.046
fiction_news-legal	1.206 ± 0.048
all-genres	2.309 ± 0.358

Table 4.17: θ_{pos} Scores (± sd) Averaged over 100 Different Runs With Adulterant Genres are Present in `pl`-LFG Data

We observe that a lower number of adulterant genres in the data don't affect the θ_{pos} scores heavily. However, the presence of multiple adulterant genres pushes the θ_{pos} scores by almost 1.5 as compared to when there are no adulterants present. Taking into account also the standard deviation score, and the high annotation quality of the treebank, we can add a headroom of ±2 if adulterant genres are present.

Assuming treebanks A and B can be split into their constituent genres such that $G_A = \{A_1, A_2, ..., A_{n1}\}$ and $G_B = \{B_1, B_2, ..., B_{n2}\}$. Of these constituent genres, at least k genres in $G_A \cup G_B$ are adulterant genres, represented by set of

adulterant genres $G_{adulterant}$. The overall limit on the $\theta_{pos}(A, B)$ score, as specified in Equation 4.10, can be updated as in Equation 4.11.

$$\theta_{pos}(A,B) \leq \Theta_{pos}(A,B) = \begin{cases} Average(\theta_{pos}(A_x, B_y)) + 2.0 & \text{if } G_{adulterant} \neq \phi \\ \\ Average(\theta_{pos}(A_x, B_y)) & \text{if } G_{adulterant} = \phi \end{cases} \tag{4.11}$$
$$\forall [A_x, B_y \in (G_A \cup G_B) - G_{adulterant}]$$

4.6 Framing Overall Θ_{pos} Limit

We studied the effects of size and genre variation in treebanks in the previous sections. It was stated earlier that in order for two datasets to be compared, they should satisfy the condition as mentioned in Equation 4.5 (restated below).

$$size(A) \geq 400 \;\; \& \;\; Avg(A) \geq Avg(B) \implies size(B) \cdot \frac{Avg(B)}{Avg(A)} \geq 400 \tag{4.5}$$

where

1. $Avg(X) = \frac{TotalSyntacticWords(X)}{size(X)}$ is the average sentence length in dataset X
2. $size(X)$ refers to the number of sentences in dataset X

For given datasets of the same genre such that the datasets satisfy the condition in Equation 4.5 above, the upper limit on the θ_{pos} metric score for the datasets to be deemed as consistent in their annotation is specified in Equation 4.6 (restated below).

$$\theta_{pos}(A,B) \leq \Theta_{pos}(A,B) = 0.5 \tag{4.6}$$

For the different genres that are present in two treebanks, if the number of instances in the genre do not satisfy Equation 4.5, we call it as an adulterant genre. In case of multiple genres being present in either treebank under consideration, the upper limit on θ_{pos} scores is determined on basis of whether or not there is an adulterant genre present as per Equation 4.11 reproduced below:

$$\theta_{pos}(A,B) \leq \Theta_{pos}(A,B) = \begin{cases} Average(\theta_{pos}(A_x, B_y)) + 2.0 & \text{if } G_{adulterant} \neq \phi \\ \\ Average(\theta_{pos}(A_x, B_y)) & \text{if } G_{adulterant} = \phi \end{cases} \tag{4.11}$$
$$\forall [A_x, B_y \in (G_A \cup G_B) - G_{adulterant}]$$

where $\theta_{pos}(A_x, B_y)$ refers to the calculated θ_{pos} score calculated between genre x present in treebank A and genre y present in treebank B. The upper limit on

individual θ_{pos} value between two genres A_x and B_y for the genres to be declared consistent with each other is given by Equation 4.9 as stated below:

$$\boxed{\theta_{pos}(A_x, B_y) \leq \Theta_{pos}(A_x, B_y) = 2.0} \tag{4.9}$$

Regardless of the genre composition of the treebanks under consideration, the treebanks with θ_{pos} score ≤ 0.5 are termed as consistent with respect to their POS annotation. Similarly, the treebanks with θ_{pos} score ≥ 4.0 are termed as inconsistent with respect to their POS annotation. If both the treebanks under consideration contain a singular genre each (i.e. $|G_A| = |G_B| = 1$), they would be termed as inconsistent in their POS annotation if their $\theta_{pos} \geq 2$. For all other treebanks, a crude estimate on whether the POS annotation of a treebank is consistent with other treebank(s) or not can be made if just the percentage composition of different genres in the treebanks is known, regardless of whether it is possible to split the treebank into the constituent genres. However, for a fine-tuned estimation, it is imperative to be able to split the treebank into its constituent genres.

For estimating the annotation consistency of a given pair of treebanks, we proceed as follows:

1. If the θ_{pos} score of the pair of the treebanks is less than or equal to 0.5, the treebanks are pronounced as consistent in their POS annotation with respect to each other.

2. If the θ_{pos} score of the pair of the treebanks is greater than or equal to 4.0, the treebanks are pronounced as inconsistent in their POS annotation with respect to each other.

3. If possible, split the treebanks into the constituent genres.

4. Isolate the adulterant genres, if any, and calculate θ_{pos} scores taking one genre from the remaining genres in either treebank and average the resultant scores. The Θ_{pos} value is calculated as per Equation 4.11.

5. In case the split into constituent genres is not possible but the percentage composition of the constituent genres is known, estimate the upper limit of the pair of genres from either treebank as per Equations 4.6 and 4.9. The Θ_{pos} limit is the calculated on the average of these estimated values.

6. In case the percentage composition of different genres is not known either, we cannot estimate the Θ_{pos} limit.

7. Based on the calculated Θ_{pos} limit and the θ_{pos} value of the pair of treebanks, the treebanks can be pronounced as consistent or inconsistent, as the case may be.

4.7 θ_{pos} Scores for UDv2.5, Annotated To Mark Consistent And Inconsistent Treebanks

This section lists the θ_{pos} scores in UDv2.5 data [Zeman et al., 2019], as listed earlier in Section 4.3. Instead of just listing the scores, the scores are color coded

as per Table 4.18 below.

Color	Significance
Red	Inconsistent in POS Annotation
Green	Consistent in POS Annotation
Gray	Could Not Be Estimated

Table 4.18: Color Codes Used to Mark Consistent or Inconsistent Treebanks based on θ_{pos} Scores in Table 4.19

Treebank1	Treebank2	θ_{pos}
ar-NYUAD	ar-PADT	2.497
es-AnCora	es-GSD	0.352
et-EDT	et-EWT	0.413
fi-FTB	fi-TDT	1.195
gl-CTG	gl-TreeGal	0.714
grc-Perseus	grc-PROIEL	4.641
ja-BCCWJ	ja-GSD*	0.951
ko-GSD*	ko-Kaist	2.56
nl-Alpino	nl-LassySmall	0.664
pl-LFG	pl-PDB*	0.623
pt-Bosque	pt-GSD*	0.678
ro-Nonstandard	ro-RRT	1.233
sl-SSJ+	sl-SST	2.405
sv-LinES	sv-Talbanken	0.443
tr-GB	tr-IMST	1.477
zh-GSD	zh-HK	1.958

de	GSD*	HDT*
HDT*	0.49	-
LIT	1.383	1.1

la	ITTB	Perseus+
Perseus+	1.106	-
PROIEL	3.763	3.901

no	Bokmaal	Nynorsk
Nynorsk	0.095	-
NynorskLIA	2.291	2.375

ru	GSD*	Taiga+
Taiga+	1.027	-
SynTagRus	0.567	0.631

cs	CAC	CLTT	FicTree
CLTT	1.453	-	-
FicTree	1.138	2.657	-
PDT	0.373	1.935	1.006

en	EWT	GUM	LinES	ParTUT
GUM	0.26	-	-	-
LinES	0.407	0.455	-	-
ParTUT	0.62	0.432	0.581	-
ESL	0.592	0.799	0.564	0.823

fr	FQB+	GSD*	ParTUT+	Sequoia	Spoken
GSD*	1.582	-	-	-	-
ParTUT+	1.942	0.683	-	-	-
Sequoia	1.693	0.248	0.524	-	-
Spoken	3.644	3.089	2.599	2.732	-
FTB	2.226	0.379	0.7	0.272	3.507

it	ISDT	ParTUT	VIT*	PoSTWITA
ParTUT	0.133	-	-	-
VIT*	0.121	0.194	-	-
PoSTWITA	1.67	1.478	1.764	-
TWITTIRO	1.501	1.376	1.594	0.347

Table 4.19: θ_{pos} Scores in UDv2.5 Marked for Consistency or Inconsistency in POS Annotation. Note that treebanks with at least one adulterant genres are marked with superscript plus sign (+). Additionally, if a treebank cannot be split into constituent genres, it is marked with superscript asterisk sign (∗) in the table.

Table 4.20 marks the Θ_{pos} limit for treebanks that were marked as inconsistent in the table above. We omit the Θ_{pos} limit for `grc` treebanks, since the reported θ_{pos} score for the treebanks in the language exceed the hard limit of 4.0.

Treebank Pair	θ_{pos}	Θ_{pos}	**Comments**
`ar`-NYUAD & `ar`-PADT	2.497	0.5	Same Genre Violation of Equation 4.6
`cs`-CAC & `cs`-CLTT	1.453	1.388	No Adulterant Genre Violation of Equations 4.6, 4.11
`cs`-CLTT & `cs`-FicTree	2.657	2.0	One Genre Each Violation of Equation 4.9
`cs`-CLTT & *cs*-PDT	1.935	1.688	No Adulterant Genre Violation of Equation 4.11
`fi`-FTB & `fi`-TDT	1.195	1.187	No Adulterant Genre Violation of Equations 4.6, 4.11
`fr`-FTB & `fr`-Spoken	3.507	2.0	One Genre Each Violation of Equation 4.9
`fr`-Sequoia & `fr`-Spoken	2.732	2.0	No Adulterant Genre Violation of Equations 4.9, 4.11
`la`-ITTB & `la`-PROIEL	3.763	2.0	No Adulterant Genre Violation of Equations 4.6, 4.9, 4.11

Table 4.20: Comparison of θ_{pos} Score and Θ_{pos} Limit for Pairs of Treebanks Marked as Inconsistent in Table 4.19

There are two important points that need to be specified here:

1. The affiliation of individual sentences in any given treebank is optional and not standardized. If the `README.md` file associated with a treebank in question does not specify how to split the treebank into the constituent genres, the information can be queried through the data providers of the treebank in question. The following treebanks could not be assessed for the annotation consistency with other treebanks in the language as the information on their genre split could not be fetched through either of the treebank's `README.md` file or through the treebank's data providers:

   - `gl`-CTG
   - `no`-Bokmaal

- no-Nynorsk
- tr-IMST

2. For treebanks with adulterant genres, the higher Θ_{pos} limit on the θ_{pos} scores can be problematic. Even though the θ_{pos} scores for the treebanks with adulterant genre(s) have also been marked as consistent in Table 4.19, it is recommended to check the annotation consistency of the treebank without the adulterant genre(s) as well.

4.8 Discussion And Conclusion

4.8.1 Out of Vocabulary Words

The metric θ_{pos} uses POS trigrams to compute the divergence of the annotation. Since the metric is delexicalised, the concept of out-of-vocabulary (OOV) words does not make sense in the calculation of the metric score. In case of either treebank being annotated (semi-)automatically by a POS tagger, the improper annotation of OOV words can affect the scores negatively.

In UD tagset, X tag is reserved for words such that they can not be categorised under any of the other POS. While it is recommended to be used in a restricted manner[2], the tag can exhibit itself abundantly depending on the origin source of the data, with the genres containing Web2.0 data being especially susceptible.

For most treebanks, the influence of OOV words should be minimal. Nonetheless, care must be taken when the X tag is present in the trigrams of either of the treebanks.

4.8.2 Using θ_{pos} Scores To Localise Inconsistency

While the θ_{pos} metric is primarily meant to identify if the given treebanks under consideration are consistent in their POS annotation, the metric can also be employed to localise points of inconsistency, if required.

Consider the example of UDv2.5 fi-FTB and fi-TDT treebanks in UDv2.5. While the data in fi-FTB is composed of *grammar-examples* genres, the data in fi-TDT treebank is composed of multiple genres, including *grammar-examples*. While calculating the θ_{pos} scores to estimate annotation-consistency for different genres across the two treebanks, it was noted that

$$\theta_{pos}(\texttt{fi-TDT}_{grammar-examples}, \texttt{fi-FTB}_{grammar-examples}) = 0.707 > 0.5$$

which is a clear violation of the condition as specified in Equation 4.6. We believe that the inconsistency in the annotation can be localised to the genre in fi-TDT treebank. Consequently, concentrating simply on the instances from *grammar-examples* genre in fi-TDT treebank should be enough to bring the overall θ_{pos} score between the two treebanks under the Θ_{pos} limit for the treebanks to be marked as inconsistent.

[2] https://universaldependencies.org/u/pos/X.html

4.8.3 Split Into Constituent Genres As Requirement

The estimation of the upper limit on θ_{pos} scores, viz. Θ_{pos}, is primarily based on the requirement that the composition of different genres in the treebanks is known. While the limit is best estimated when the instances from different genres can be split into individual datasets and the adulterant genres identified, it is possible to get a crude estimate of the limit. For example, one can estimate all the common genres with θ_{pos} scores of 0.5, and the different genres have a θ_{pos} score of 2.0. An average of these estimates should give a crude estimate on the Θ_{pos} limit without accounting for an adulterant genre.

In practice, however, it might not always be possible to split a treebank into constituent genres or even identify the percentage composition of each genre in the data. The Θ_{pos}, in this case, can not be estimated reliably. It is therefore recommended to use the metric on the treebanks which can be split into their constituent genres, to attain best results.

4.8.4 Conclusion

In this experiment, we proposed a numeric measure based off KL_{cpos^3} measure [Rosa and Žabokrtský, 2015] to identify if two treebanks are consistent in their POS annotation. The upper limit on the measure was also estimated using treebanks with a high annotation quality, belonging to different language families.

In addition to knowing if the treebanks are annotated consistently with respect to each other, the measure can also be used intra-treebank as well as to localize the genre(s) that cause an inconsistency among the treebank pair. We also evaluated different treebanks in UDv2.5 data [Zeman et al., 2019] and identified the consistent and inconsistent treebank pairs based on the proposed measure.

5. conj_head: Head Identification Error in Coordinating Conjunctions (Experiment 2)

As discussed in Section 2.2.1, `conj_head` refers to the head identification error for a given coordinating conjunction. This error is characterized by the coordinating conjunction being linked to the previous conjunct (in UDv1), rather than by the next conjunct (in UDv2). We define the problem statement as identification of correct head for a given coordinating conjunction. In our treatment of the problem in this section, we start with a glance through some of the observations on the problem in Section 5.1, allowing us to define our effective dataset in Section 5.2. We elaborate on our proposed solution to the problem, and the explanation of the algorithm used in the experiment in Section 5.3 and 5.4 respectively. We finally evaluate the experiment in Section 5.5.

5.1 Observations About Problem Statement

Identification of coordinating conjunctions, and separating them from subordinating conjunctions is a problem in itself and warrants a separate discussion of its own. Combined with the possible association of multiple deprels to a particular POS tag (cf. Section 8.4), it is necessary to explicitly put a constraint on the instances to be considered during the scope of this experiment. To that effect, we identify coordinating conjunctions with their POS tag marked as `CCONJ`, and the deprel as `cc`. We disregard other deprels associated with the POS `CCONJ` in the current context, effectively limiting the number of instances to be considered. In other words, we assume that any token that is POS tagged as `CCONJ` and with deprel as `cc` is a coordinating conjunction and that every coordinating conjunction is tagged in this manner, without exceptions. While the assumption is not fool proof and is not guaranteed to always hold, a deviation from this assumption would be an error in labeling rather than in dependency structure, i.e., an error type that is outside of the scope of the present experiment.

In the following subsections, we take a look at some of the quirks associated with the problem. Through these quirks, we seek to (i) discover triggers that can help us in identification of problematic instances; and (ii) identify possible problems that we can run into while handling the aforementioned problematic instances. Throughout the rest of the experiment, we shall employ the following terminology:

1. The terms “coordinating conjunction”, and “conjunction” are used interchangeably.

2. The term “coordination” refers to the entire construction that consists of conjuncts, and (typically one) conjunction.

3. In the following example, the coordination (*Jack and Jill*) is marked in bold, while the conjunction (*and*) is marked in italic.

Example 7. **Jack *and* Jill** went up the hill to fetch a pail of water.

5.1.1 Direction of Dependency

Owing to the change in associated dependency from right-headed to left-headed[1], the intuitive approach to the problem at hand is to first look for the direction of dependency for a given coordinating conjunction token, identifying the instances where the attachment is right-headed. However, the identification of the correct direction can be non-trivial if worked in a language-independent manner. Consider the case of `sa`, and how it differs from `en`, as in Example 8. In `en`, the coordinating conjunction occurs in between the different conjuncts. In the given example for `sa`, the conjunction is linked with the last conjunct in a form that is typical of the language. The word of interest in the example is marked explicitly in bold. Referred to as monosyndentic postposing by Stassen [2000], he observes that the phenomenon is relatively common in languages around the world. In the same article, the author also observes that the case of syndentic preposition (as opposed to postposition in the given example) is unattested for the the first conjunct. A brief typology of monosyndetic coordinations is listed in Table 5.1. We do not discuss other types of coordination like polysyndetic, asyndetic, or coordination by juxtaposition in the table. While polysyndetic coordination would essentially require the same treatment as monosyndetic coordination, the others are not relevant to the problem owing to the lack of a defined conjunction.

Example 8.
Text (`sa`): तस्य त्रयः पुत्राः परमदुर्मेधसः वसुशक्तिः उग्रशक्तिः **अनन्तशक्तिश्च** इति बभूवुः।
Translit: *tasya trayaḥ putrah paramadurmedhasah vasushakti ugrashaktih* ***anantashaktishca*** *iti babhuvuh .*
Lit.: His three sons extremely-stupid Vasushakti Ugrashakti **Anantashakti-and** known-by-these-names there-were .
Translated: There were his three extremely stupid sons, called Vasushakti, Ugrashakti, **and** Anantashakti.

Syndetion Type	**Structure Variations**	**Rarity**
Coordination as a token	*A co B*	Common
Postposing on Conjunct 1	*A-co B*	Common
Postposing on Conjunct 2	*A B-co*	Common
Postposing on Conjuncts	*A-co B-co*	Common
Preposing on Conjunct 1	*co-A B*	Unattested
Preposing on Conjunct 2	*A co-B*	Common
Preposing on Conjuncts	*co-A co-B*	Rare

Table 5.1: Possible Syndetion Typologies across Languages
A, *B*- conjuncts
co- conjunction
Z-co, *co-Z*- conjunction attached to *Z*

[1] `https://universaldependencies.org/v2/coordination.html#left--vs-right-headed-coordination`

In the example, note that the coordinating conjunction च (*ca*; and) appears postposed on the last conjunct, unlike in English. It is also worth pointing out that a given language can exhibit multiple kind of syndetion typologies as listed in Table 5.1, without restricting itself to just one. For example, a conjunction can also appear as a separate token in `sa`. Similarly in `he`, the conjunctions can occur in preposed form on the second conjunct, or as a separate token on its own. However, given the possibility of inflectional affixes, a singular word can have multiple prefixes which may or may not imply a case of coordination.

The above cases exhibit two problematic instances. Nonetheless, they can easily be handled similarly as follows:

1. A conjunction may be conventionally written as an affix of the neighboring word (as in case of `he` and `sa` as above). We rely on the word segmentation of the CoNLL-U file[2] (assuming it is correct), so we only work with conjunctions that are either written separately or have been separated during word segmentation.

2. The directions "left" and "right" in left-headedness (right-headedness) are to be understood logically, disregarding the right-to-left writing systems of languages like `he`. Therefore, the head is said to be to the left of the dependent, if its numeric position (ID in the CoNLL-U file) is lower than the position of the dependent.

3. For a language that showcases only left-headed conjunctions[3], a right-headed conjunction is an erroneous annotation, and vice-versa for languages showing only right-headed conjunctions. This reversed direction of dependency (or reversed headedness of the conjunction head) forms the basis for mining of the problematic instance.

Table 5.3 shows the total count of instances of coordinating conjunctions (tokens with `CCONJ` as POS tag, and `cc` as deprel), along with the number of instances that have reverse direction of dependency in different treebanks of UDv2.4 [Nivre et al., 2019]. The different PUD treebanks[4] contain the same sentences, translated into the corresponding language from `en`. Keeping this in mind, PUD treebanks are analysed separately in Table 5.2. The tables also show the number of instances where a case of misdirected dependency of conjunction head causes a non-projectivity in the sentence.

It must be stressed here that the problem at hand is not about detecting the cases of misdirected dependencies, but rather the selection of a more relevant head for the dependency. However, the identification of misdirected dependencies can be the first step towards detection of such cases, as discussed earlier. Notice that the notion of misdirected dependency can be defined only for languages such that the conjunctions in the language are either of left-headed or right-headed, but not both (as in the case of `sa` from before). In our work, we focus on languages where the conjunctions are only right-headed, i.e. the correct head should be located

[2]For details on CoNLL-U format, refer Appendix A.1.1

[3]The language's characteristic of whether it showcases conjunctions as left-headed, or right-headed, or both should be included in the language-specific documentation.

[4]Appendix A.4 mentions in detail about how the different PUD treebanks were created, and how they are recommended to be used.

towards the logical right of the conjunction. Tables 5.2 and 5.3 mark languages that show either of left-headed conjunctions or a mix of left and right-headed conjunctions with an asterisk superscript. However, the marking should not be considered exhaustive.

Treebank	**Total**	**Misdirected (% Total)**	**Non-Proj**
ar-pud	651	5 (0.768)	2
cs-pud	626	5 (0.799)	-
de-pud	733	8 (1.091)	-
en-pud	575	-	-
es-pud	553	-	-
fi-pud	596	1 (0.168)	-
fr-pud	537	-	-
hi-pud	789	25 (3.169)	-
id-pud	545	3 (0.550)	-
it-pud	576	1 (0.174)	-
ja-pud	-	-	-
ko-pud	79	-	-
pl-pud	571	5 (0.876)	-
pt-pud	533	2 (0.375)	-
ru-pud	588	-	-
sv-pud	593	6 (1.012)	-
th-pud	588	3 (0.510)	-
tr-pud	490	2 (0.408)	-
zh-pud	283	3 (1.060)	-

Table 5.2: Misdirected Coordinating Conjunctions in UDv2.4 PUD Treebanks

The PUD treebanks contain the same set of sentences, therefore allowing for a parallel comparison. From Table 5.2, notice that while the en-PUD treebank contains 575 instances of coordinating conjunctions, PUD treebanks for ja, and ko have less than 100 instances each. Similarly, there are other treebanks with 700+, as well as those with less than 300 instances of coordinating conjunctions. It is also interesting to note that the number of misdirected dependencies expressed as a percentage of total number of coordinating conjunctions also ranges widely from 0% (en, es, fr, ko, ru) to 3.169% (hi).

Treebank	Total	Misdirected	Non-Proj
af-afribooms	**1832**	**1829**	**130**
aii-as	**25**	**8**	**-**
akk-pisandub	100	-	-
am-att	**80**	**73**	**1**
ar-nyuad	48768	1532	-
ar-padt	**13855**	**1411**	**80**
be-hse	590	1	-
bg-btb	4794	6	-
bm-crb	64	-	-
br-keb	204	6	-
bxr-bdt	70	61	13
ca-ancora	14067	19	-
cop-scriptorium	675	7	-
cs-cac	21798	509	8
cs-cltt	1805	20	-
cs-fictree	7410	108	3
cs-pdt	49302	1372	9
cu-proiel	**4865**	**3263**	**124**
cy-ccg	305	-	-
da-ddt	3097	177	61
de-gsd	8675	169	4
de-hdt	68917	1	-
de-lit	1686	17	-
el-gdt	2017	24	-
en-esl	3189	2	-
en-ewt	8197	8	-
en-gum	3212	26	4
en-lines	2510	10	-
en-partut	1647	6	1
es-ancora	14233	36	-
es-gsd	12784	226	9
et-edt	15957	37	-
et-ewt	1120	1	-
eu-bdt	4620	318	56
fa-seraji	7653	101	5
fi-ftb	4726	32	-
fi-tdt	8284	12	-
fo-oft	296	1	-
fr-fqb	97	-	-
fr-ftb	11605	45	5
fr-gsd	10068	2	-
fr-partut	853	4	-
fr-sequoia	1621	-	-
fr-spoken	1042	-	-
fro-srcmf	10075	13	-
ga-idt	640	1	-
gl-ctg	**4261**	**4127**	**-**
gl-treegal	700	-	-
***got-proiel**	**5017**	**3084**	**125**
grc-perseus	**5316**	**5098**	**780**
grc-proiel	**13980**	**10704**	**771**
gun-dooley	**24**	**3**	**-**
gun-thomas	26	1	-
he-htb	4724	21	2
hi-hdtb	6426	9	3
hr-set	7236	78	-
hsb-ufal	419	5	-
hu-szeged	**1809**	**390**	**2**
hy-armtdp	1561	8	-
id-gsd	3549	215	18
it-isdt	8131	2	-
it-partut	1680	6	-
it-postwita	2801	14	-
it-vit	8120	284	2

Treebank	Total	Misdirected	Non-Proj
ja-bccwj	**16120**	**11574**	**-**
ja-gsd	-	-	-
ja-modern	**479**	**271**	**-**
kk-ktb	180	6	-
kmr-mg	**355**	**41**	**1**
ko-gsd	**223**	**52**	**4**
ko-kaist	5136	2	-
kpv-ikdp	48	1	-
kpv-lattice	56	2	-
krl-kkpp	156	-	-
*la-ittb	16789	673	6
***la-perseus**	**1255**	**960**	**65**
***la-proiel**	**14575**	**10311**	**638**
lt-alksnis	1648	40	-
lt-hse	287	-	-
lv-lvtb	8043	9	-
lzh-kyoto	923	-	-
mr-ufal	62	1	-
mt-mudt	1514	4	-
myv-jr	314	3	-
nl-alpino	3853	7	-
nl-lassysmall	2501	6	-
no-bokmaal	10709	-	-
no-nynorsk	10847	4	-
no-nynorsklia	2421	188	1
orv-rnc	1460	-	-
orv-torot	**13640**	**6683**	**453**
pcm-nsc	135	-	-
pl-lfg	3227	-	-
pl-pdb	10670	138	-
pt-bosque	5153	57	9
pt-gsd	7717	82	-
qhe-hiencs	493	3	-
ro-nonstandard	14953	12	-
ro-rrt	6275	155	8
ru-gsd	2952	44	2
ru-syntagrus	38914	86	1
ru-taiga	1712	-	-
***sa-ufal**	**32**	**17**	**-**
sk-snk	3162	29	-
sl-ssj	4665	-	-
sl-sst	1082	26	1
sme-giella	**984**	**891**	**60**
sr-set	2900	18	-
sv-lines	2941	11	1
sv-talbanken	3510	112	-
swl-sslc	5	-	-
*ta-ttb	46	1	-
te-mtg	**11**	**4**	**-**
tl-trg	-	-	-
tr-gb	160	6	-
tr-imst	825	69	10
ug-udt	462	2	-
uk-iu	4753	-	-
ur-udtb	3248	10	1
vi-vtb	**1177**	**340**	**-**
wbp-ufal	-	-	-
wo-wtb	1365	1	-
yo-ytb	148	-	-
yue-hk	76	3	-
zh-cfl	92	-	-
zh-gsd	1739	21	1
zh-hk	71	2	-

Table 5.3: Misdirected Coordinating Conjunctions in UDv2.4 Treebanks Values in bold indicate treebanks with misdirected dependencies forming 10%+ of the total coordinating conjunctions

5.1.2 Identifying Correct Conjunct for Misdirected Dependencies

For a given token in a tree, we define the level of the token as the minimum number of dependency edges from the root of the tree to the token itself. For example, the node connected directly to the root of the tree is at level 1, since there is a single dependency link. Any token directly connected to this node would therefore be at level 2, and is referred to at a lower level than the node.

At this point, we shall also overtly state some of the assumptions that we work with during the course of the experiment. First, we assume that the conjuncts are attached correctly but only the conjunction is wrongly attached. For the experiment, we do not attempt to correct the cases where the wrong token is marked as a conjunct or where the conjuncts are wrongly attached to their corresponding heads. It is very likely that if there is an error in the annotation of conjuncts, the correction of the associated conjunction would not be the intended one. Second, we assume that the current head of a misdirected conjunction may/may not be a conjunct. Even when the conjunction is attached to a conjunct, the trivial solution of finding the next conjunct towards the logical right might not work as intended. This is best exhibited in cases where there are multiple coordination structures within a single sentence. Even if the conjunction is marked to a conjunct, it is possible for the conjunct to belong to a different coordination and therefore the conjunction in question would be falsely attached in the wrong coordination (as mentioned later in this section). In the event that the conjunction is attached not to the conjunct, but to a random node, the search for the correct conjunct that should be the head of the conjunction becomes more complex.

For conjunctions with misdirected dependencies, we distinguish between two kinds of attachments. Depending on whether or not the attachment (to the wrong conjunct) is projective in nature, we use different strategies for the identification of the correct head for the conjunction.

Conjunction Attached Projectively

For the misdirect conjunctions such that they are attached projectively, we limit our search for the more relevant head to a maximum of one level from the current head. If the difference in levels of the wrong head and the more relevant head differs by more than 1, we hypothesize that the annotation for the sentence is erroneous and therefore it cannot be corrected automatically. To limit the level change by 1, we try to find the correct conjunct from within the current head's siblings, parent or the children nodes. We do not look for a candidate node in the current head's extended family to accommodate for multiple coordination within a sentence wherein a search on similar level across the tree could have disastrous consequences. To that effect, we limit our search for a candidate head such that it is on the same level as the current head (Figure 5.1), or is within the subtree of this head, implying a search at a lower level (Figure 5.2a). This works only for the cases where the current head is a conjunct itself. In cases where the current head is located within the subtree of the conjunct, we need to first climb to a higher level to locate the intended conjunct, and then locate a relevant head to the logical right (Figure 5.2b).

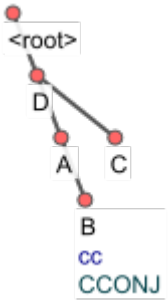
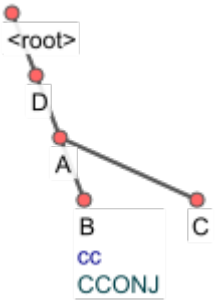
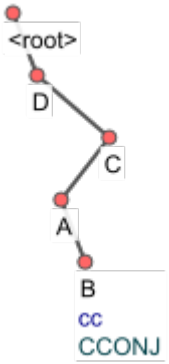


Figure 5.1: Possible Wrong Attachments of a Coordinating Conjunction: Correct Head as Wrong Head's Sibling

Note: D is the common ancestor of nodes A, B and C
Note: C is the more relevant head that conjunction B should be attached to, instead of the current misdirected attachment with node A
Note: A, C and D might be the head of their own subtrees



(a) Correct Head as Conjunction's Sibling



(b) Correct Head as Conjunction's Grandparent

Figure 5.2: Possible Wrong Attachments of a Coordinating Conjunction
Note: D is the common ancestor of nodes A, B and C
Note: C is the more relevant head that conjunction B should be attached to, instead of the current misdirected attachment with node A
Note: A, C and D might be the head of their own subtrees

Notice that while the 3 cases as mentioned in Figures 5.1 and 5.2 are separate, there is no deterministic way of knowing what case an identified problematic instance might refer to. As such, we handle the 3 cases in decreasing order of priority, i.e. we try to handle the case as in Figure 5.1 first. In case the attempt fails, owing to multitude of reasons as explained later in Section 5.4 (no siblings to attach to, lack of a candidate head in the siblings, for example), we try to solve it with respect to the case as in Figure 5.2a, and in case of a failure therein as well, eventually as in Figure 5.2b. If a particular instance is still not corrected after the consideration of the last case, we leave it unchanged.

Conjunction Attached Non-Projectively

In case of misdirected conjunctions that are attached non-projectively, the previous approach of limiting the level change with respect to the wrong head does not function well. The approach fails mainly because if the attachment is non-projective in nature, it is very likely that the conjunction is attached to a head in a different coordination. Consider the part of a sentence taken from UDv2.4 `hi`-hdtb treebank in Example 9 and the tree for the corresponding example in Figure 5.3. The token in bold is attached non-projectively because its current head is not a part of the same coordination structure as the conjunction itself.

Example 9.
Text (`hi`): वे सांसद **या** विधायक बनने के बाद लाभ के पद पर आसीन हैं या उससे पहले से हैं।
Translit: *ve saamsad* ***yaa*** *vidhaayaka banane ke baada laabha ke pada para aasiin hain yaa usase pahale se hain .*
Lit.: they senator or legislator become-*Inf.* *Acc.* after power *Poss.* position on situated are or therefrom before *Dat.* are .
Translated: They have been in a position of power from before they became senator or legislator, or after.

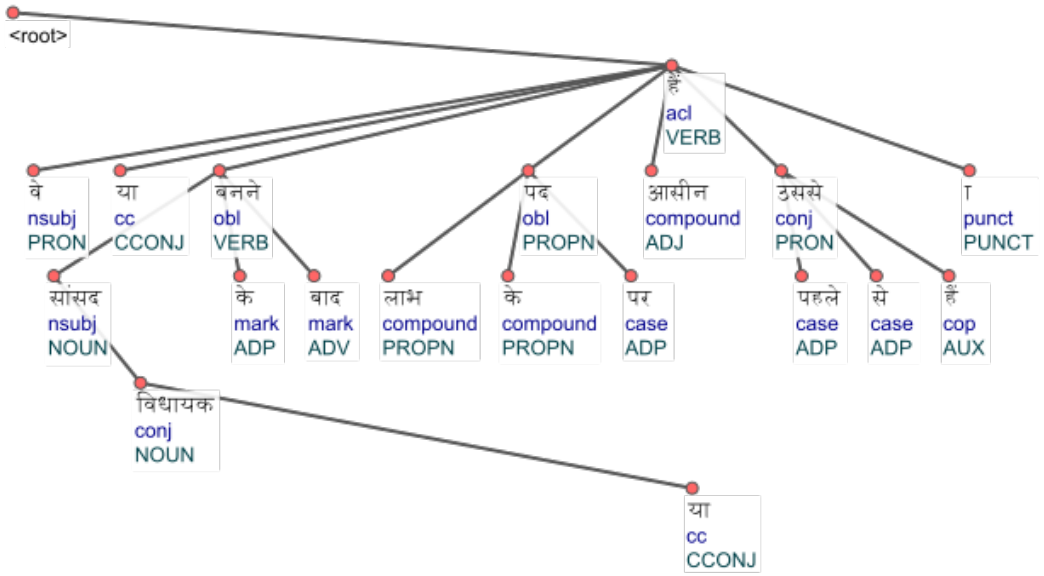


Figure 5.3: Original Annotation for Example 9
Note: First **या** (*yaa*, or) should be attached to **विधायक** (*vidhaayaka*; legislator), and not to **हैं** (*hain*; are)
Note: Second **या** (*yaa*, or) should be attached to **उससे** (*usase*; therefrom), and not to **विधायक** (*vidhaayaka*; legislator)

Since the conjunction is associated to a conjunct in the different coordination structure, the trivial approach to the problem is to look at the next available conjunct in the tree such that it satisfies the right-headedness criteria, and associate the conjunction to the said conjunct. In the previous example, the problem can simply be solved by associating the conjunction to the next available conjunct, marked by the deprel `conj`. However, this might not be always possible if

the next conjunct is not explicitly marked by the deprel. Consider the following example from UDv2.4 `en`-EWT treebank and the associated dependency tree in Figure 5.4. The token of interest is marked in bold.

Example 10.
But other people do like the way they think he will vote, and the ones who favor him seem to outnumber the ones who oppose him .

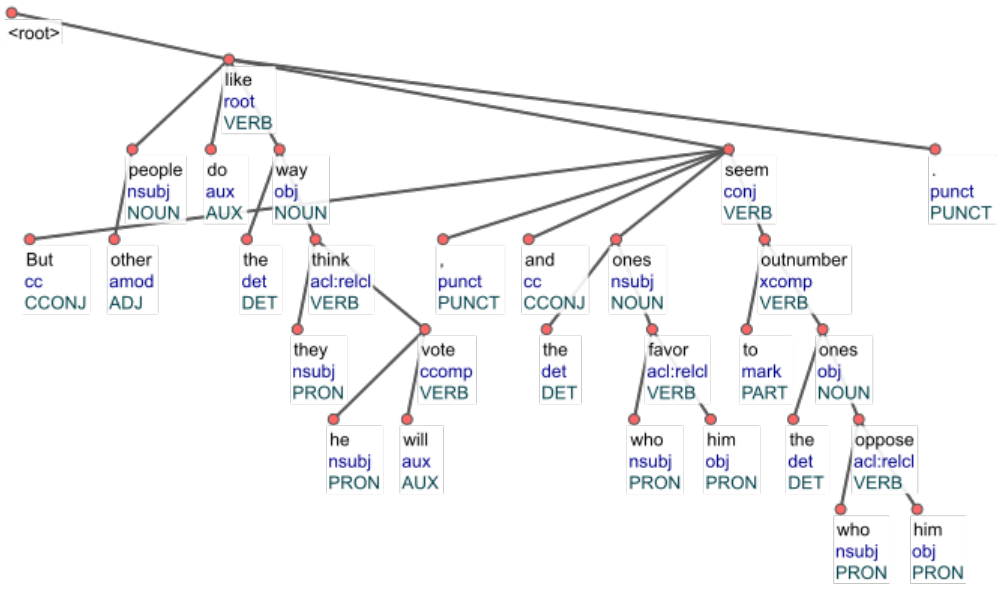


Figure 5.4: Dependency Tree for Example 10
Note: *But* should be connected to *like*, and not to *seem*

In the example, the correct conjunct is not marked explicitly by the deprel `conj`. Likewise in certain cases, the correct conjunct for the conjunction can not be found by the search for the deprel alone. However, based on the position of the content word token(s) that precede the conjunction in the word order, the correct conjunct can be determined to some extent. In the UDv2 guidelines, the dependencies that have a functional word as the head should be avoided (content-head dependency vs. function-head dependencies). Nonetheless, in some cases, the function words do form head of dependencies (when an auxiliary verb forms the root of a tree, for example). We treat the cases where the function word forms the head of a dependency as exceptional cases. As such, the correct conjunct position can be determined on the basis of the preceding content word token(s), including pronouns. The addition of pronouns is attributed to the fact that different pronouns can act as conjuncts in a sentence.

As can be seen in the last column in Tables 5.2 and 5.3, the misdirected dependencies such that they introduce non-projectivities are relatively uncommon. In our treatment of instances of the kind, we attempt to look for the next conjunct (if marked explicitly by the deprel) in the word order, such that the candidate conjunct follows the conjunction. In case the conjunct is not explicitly marked, we attempt to associate the conjunction to the immediately preceding content word in the word order. This ensures that the misdirection is not resolved, but the conjunction is now closer to the actual conjuncts and thus can be found in a process similar to the level-based analysis as done in previous subsection.

5.1.3 Conjunction Sandwich

We have so far discussed only the cases where the problem can be identified by the wrong direction of dependency. However, when the direction of dependency is correct, mining for the problematic instances becomes troublesome. In Figure 5.3 reproduced below, notice that the first conjunction token **या** (*yaa*; or) is linked in the correct direction, but to the wrong head. This instance of the correct direction of attachment, albeit to wrong head, can be present in the original annotation, or might be introduced after the tree has been corrected for the misdirected dependency. We refer to such cases as a Conjunction Sandwich, since the conjunction is sandwich-ed in between the conjuncts, with a wrong choice of head but with the direction of attachment being as expected (right-headed in our case). The problem of not being able to identify the correct conjunct as elaborated earlier in Example 10 can manifest itself in such cases as well, making this problem significantly harder to detect.

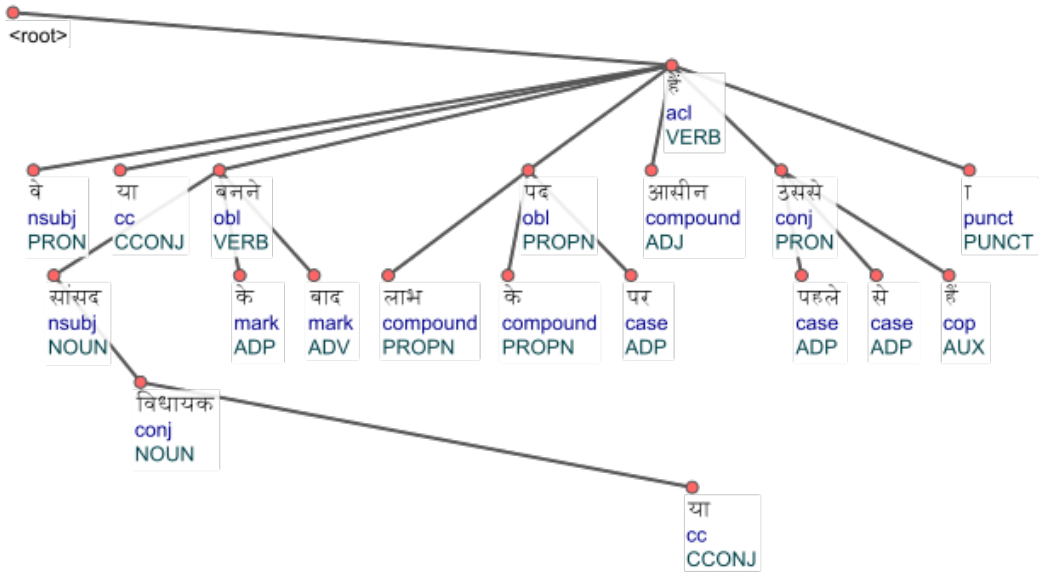


Figure 5.3: Original Annotation for Example 9
Note: First **या** (*yaa*; or) should be attached to **विधायक** (*vidhaayaka*; legislator), and not to **हैं** (*hain*; are)
Note: Second **या** (*yaa*; or) should be attached to **उससे** (*usase*; therefrom), and not to **विधायक** (*vidhaayaka*; legislator)

In a given dependency tree, we can express node A being followed by node B in top-down ordering of tokens as $A < B$. To establish node A is linked to node B, such that A is the head of the relation, and B is the dependent, we can write $A \rightarrow B$. In case where the direction of the relation is not important, we can express it by using double headed arrows as $A \leftrightarrow B$.

In a dependency tree, given two undirected edges $i_1 \leftrightarrow j_1$ and $i_2 \leftrightarrow j_2$, the edges are said to be overlapping if $i_1 < i_2 < j_1 < j_2$ or $i_1 > i_2 > j_1 > j_2$. In the example figure above, we can see that one of the ways in which a case of conjunction sandwich manifests itself is in the form of overlapping edges. However, this might not always be the case. The edges can overlap also because of the

faulty annotation of other tokens in the tree, and that renders this check unreliable. In the example figure above, the edge containing the first conjunction also overlaps with the edge containing second conjunction, attached non-projectively. The constraint (of overlapping edges) was also tightened to look for a conjunction being the sole node in the gap of a non-projective attachment, but the number of cases that were flagged in the process remained very low (mostly less than 1% of total number of conjunctions across different languages, depending on the language as some languages allow less number of non-projective structures than others). Of the total number of cases that were flagged by the tighter constraint, the majority were false positives.

Consider the following example from UDv2.4 `en`-lines treebank, and the associated dependency tree in Figure 5.5. In this case, the edges do not overlap, but the conjunction is still attached to the wrong head.

Example 11.
That was also mentioned by Mrs Oomen-Ruitjen **and** Mrs Glase.

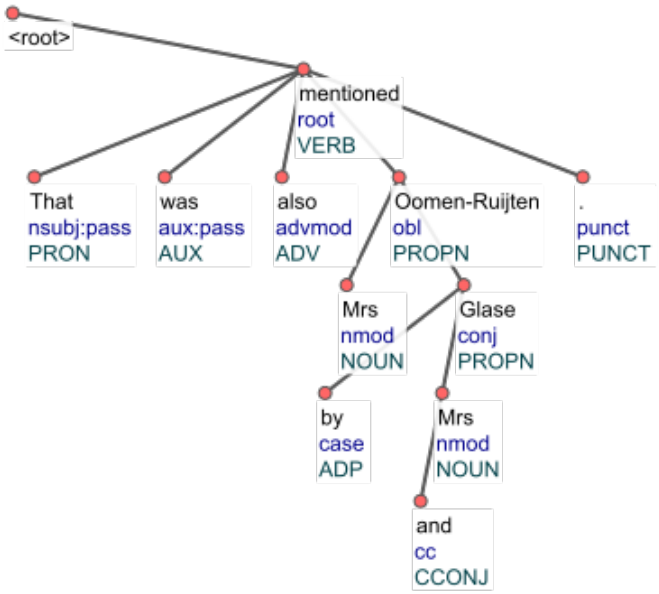


Figure 5.5: Dependency Tree for Example 11
Note: *by* should be attached to *Oomen-Ruitjen*, and not to *Glase*
Note: *and* should be attached to *Glase*, and not to *Mrs*

The trivial approach in the case of a Conjunction Sandwich would be to look for a conjunct explicitly marked by the `conj` deprel, such that the said conjunct follows the conjunction in question. For example, in Figure 5.3, the attachment for the first conjunction can be corrected by looking for the first explicitly marked conjunct that follows the current parent. This is an unreliable approach nonetheless, because (i) the conjunct needs to be explicitly marked by the deprel, which is not always the case; and (ii) the approach cannot work in the case of nested coordination, often picking up on a conjunct from another coordination structure.

Since none of the cases discussed in this section could be reliably scouted for, we do not deal with identification and/or correction of Conjunction Sandwich in the current research.

5.2 Dataset

The experiment was initially started on UDv2.3 [Nivre et al., 2018], but owing to the release of UDv2.4 [Nivre et al., 2019] in May 2019, the experiment was transported entirely to UDv2.4. It is worth noting that there were far more cases of this problem being identified in UDv2.3, rather than in UDv2.4. Nonetheless, there exist significant cases of the problem (attachment of a conjunction to an incorrect head, and in wrong direction) in UDv2.4 as well.

We limit our treatment of the problem to `af`, and `ar`. As can be seen from Table 5.3, the languages contain treebanks such that (i) they do not display any postposed variant of conjunctions as denoted by asterisk superscript in the table; (ii) the number of misdirected dependencies in the treebank is more than 10% of the total number of conjunctions; and (iii) the languages do not have non-projectivity as a major characteristic, and yet the number of misdirected non-projective attachments is high in the treebank (unlike `grc` and `la` which have non-projectivity as a characteristic feature). Additionally, the languages belong to different language families viz. Germanic Indo-European and Semitic Afro-Asiatic, thus ensuring that the results of the experiment are not specific to a limited set of languages.

The number of instances of misdirected dependencies in different treebanks of UDv2.4 was highlighted in Table 5.2 and 5.3. The count of instances for `af`-afribooms and `ar`-padt are highlighted again in Table 5.4 for reference.

Treebank	**Total**	**Misdirected**	**% Total**	**Non-Proj**
`af`-afribooms	1 832	1 829	99.836	130
`ar`-padt	13 855	1 411	10.184	80

Table 5.4: Misdirected Coordinating Conjunctions in UDv2.4 Treebanks for `af` and `ar`

5.3 Experimental Setup

At the end of Section 5.1.3, we mentioned how we would not deal with the cases where the direction of attachment is already correct. Thus, in the experiments, our treatment is limited to the instances with misdirected dependencies. To that resort, we start by identification of conjunctions such that they are associated in the wrong direction. Upon identification of such tokens, if they are attached non-projectively, we associate the token (still in the wrong direction) to the nearest content word that precedes the said token. In case the new attachment is now projective, we can terminate dealing with this case here. In the case of the new attachment being non-projective again, we try finding a content word (including pronouns) that is closest to the conjunction in word-order, and try attachment with this found word. If the new attachment is projective, we have dealt with the problem of non-projectivity for now, and the node in question can be associated to a more relevant head as other nodes that were originally projectively attached.

The problem with this approach (of reducing a case of non-projective attachment artificially to that of a projective attachment) is twofold. Primarily, the algorithm, as mentioned in Section 5.1.2, looks for the candidate conjunct at a

level that is determined by the attachment to the wrong conjunct. In principle, the choice of a wrong parent while solving non-projectivity could eventually lead to the corrected attachment with the wrong parent, resulting in a conjunction sandwich. Secondly, the approach does not take into account the cases when the attachment needs to be made to a function word, rather than a content word. The same issue can be raised for even the way the projective attachments are handled in general.

5.4 Algorithm

We start with defining some wrapper functions in Algorithm 1 and 2. While the first one checks for the coordinating conjunctions that are attached in wrong direction, the second one tries to change the parent of the given node x to a new parent z. In case the new attachment would be non-projective, the function rolls back to the previous parent. If projectivity is preserved, the function returns a **true** value, which allows us to terminate the function whenever the function call is made inside another function. The function also checks against making the node attached directly to the root of the tree, thereby making sure there is just one root node at any instance.

Algorithm 1 misdirectedDependency()

Input: Node x

if $x.upos$ == "CCONJ" **and** $x.udeprel$ == "cc" **and** $x.parent.id < x.id$ **then**
    return true
end if
return false

Algorithm 2 setParent()

Input: Node x, Original Parent y, New Parent Candidate z

$x.parent \leftarrow z$
if $isnonprojective(x)$ == **true or** $z.id == 0$ **then**
    $x.parent \leftarrow y$
    return false
else
    return true
end if

Having defined our wrapper functions, we start by trying to projectivize the conjunctions attached non-projectively in the wrong direction. We start by first looking for the next explicitly marked conjunct, and try to attach the conjunction to the said conjunct. We define this procedure in Algorithm 3.

Algorithm 3 nextConjHead()

Input: Node x such that $misdirectedDependency(x)$ == **true** and $isnonprojective(x)$ == **true**, Original Parent y

List $L \leftarrow$ containing nodes arranged in the increasing order of their id
{i.e. $i < j \implies L[i].id < L[j].id \quad \forall L[i], L[j] \in L$}
for node z in L **do**
  {Process nodes in increasing order of their id}
  if $z.id > x.id$ **then**
    if $z.udeprel ==$ "$conj$" **then**
      return $setParent(x, y, z)$
    end if
  end if
end for
return false

In case of scenarios like in Example 9 (figure reproduced again below) with respect to the second conjunction, the non-projective attachment is made projective, and rectified with respect to the correct head automatically. However, there are cases when this approach may fail, owing to the next marked conjunct being located far away (and thus new attachment being non-projective again) or the next conjunct not being marked explicitly. In such cases, we move to the next step, and try to associate the current conjunction to the content word or pronoun that immediately precedes the given token. To look for the immediately preceding content word or pronoun, we look for the following POS tags- `ADJ`, `ADV`, `NOUN`, `PROPN`, `VERB`, and `PRON`. As with previous approach, we rollback the changes in case the attachment to new candidate head is non-projective in nature, going back to the original parent. The procedure is elaborated in Algorithm 4.

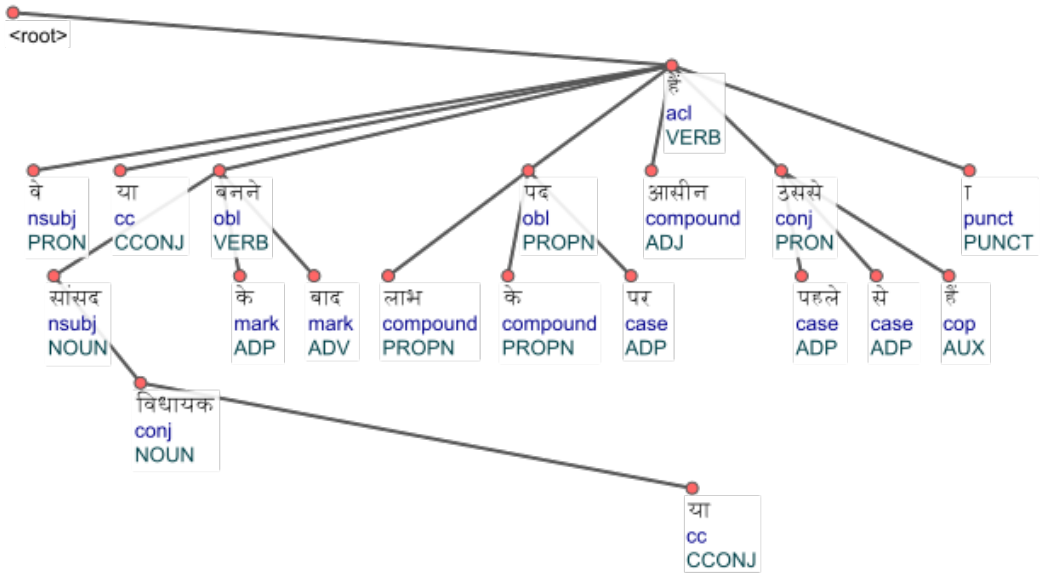


Figure 5.3: Original Annotation for Example 9
Note: First **या** (*yaa*; or) should be attached to **विधायक** (*vidhaayaka*; legislator), and not to **हैं** (*hain*; are)
Note: Second **या** (*yaa*; or) should be attached to **उससे** (*usase*; therefrom), and not to **विधायक** (*vidhaayaka*; legislator)

Algorithm 4 projTempFix()

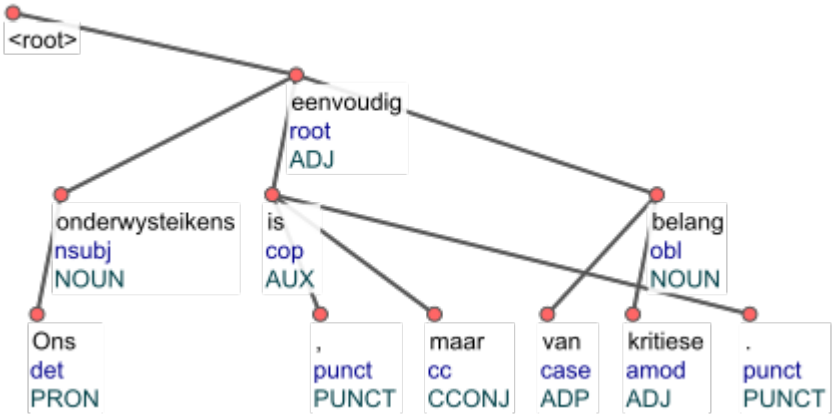
Input: Node x such that $misdirectedDependency(x)$ == **true** and $isnonprojective(x)$ == **true**, Original Parent y

$candidates = []$
{Empty List}
for all z such that $z.id < x.id$ **do**
  if $z.upos$ in [“ADJ”, “ADV”, “$NOUN$”, “$PROPN$”, “$VERB$”, “$PRON$”] **then**
    $candidates.append(z)$
    {Add z to $candidates$ list}
  end if
end for
{The content nodes are organised in the list, in word-order. We need to work with only the last candidate.}
if $candidates$ == [] **then**
  return false
else
  $candidate = candidates[-1]$
  {Pick up the last element from $candidates$ list, and try changing it to head}
  return $setParent(x, y, candidate)$
end if

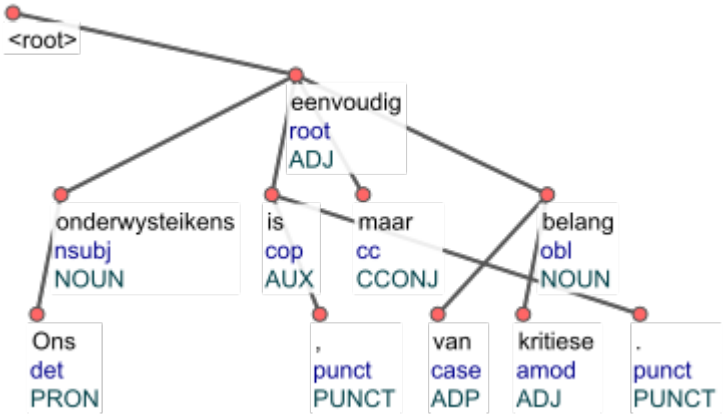
Using the above algorithm, we are able to find better candidates for the originally misdirected non-projective dependency. Consider the following sentence,

as taken from UDv2.4 `af` treebank in Example 12, and the corresponding original and modified annotations in Figure 5.6, with the token of interest marked in bold. We can see that the original annotation contains the conjunction attached non-projectively. However, following the correction, the non-projectivity is solved and the new attachment is closer to the correct annotation. It is worth noting that the non-projectivity related to the punctuation marks can be solved easily (cf. Section 2.3.2).

Example 12.
Text (af): Ons onderwysteikens is eenvoudig , **maar** van kritiese belang.
Lit: Our educational-target-*Pl.* is simple , **but** of critical significance .
Translated: Our educational targets are simple, but of critical significance.



(a) Original Annotation



(b) Modified Annotation

Figure 5.6: Change in Annotation for Example 12
Note: **maar** (but) should be attached to **belang** (significance)

At this point, we have exhausted our treatment of non-projective misdirected dependencies. A misdirected non-projective attachment of conjunction is either projective after this step, or is unaffected. We discuss the second case in Section 5.5 when we discuss the results of the experiment in more detail. For the first case of non-projective attachments, we have now removed the non-projectivity from the attachment, making sure they can be handled in the same manner as the other originally projective attachments, as elaborated earlier.

For the common treatment of the attachments independent of their projectivity status, we look for the conjunctions such that they are attached in wrong direction. As a first step in search for a candidate conjunct, we look for the content words at the same level as the current node. We start by checking if there is a single remaining sibling that does not have a POS tag of `X`, `PUNCT`, or `SYM` since we want to avoid the linking of the conjunction to these POS tags. In case of the condition being satisfied, and thus the availability of a single sibling as a candidate head, the token of interest is tried to be attached to this candidate head, and a **true** value is returned, indicating the success in search for the candidate. The effectiveness of case when a single sibling is present is demonstrated in Figure 5.7 containing an extension of the Example 12, after the modification from Figure 5.6.

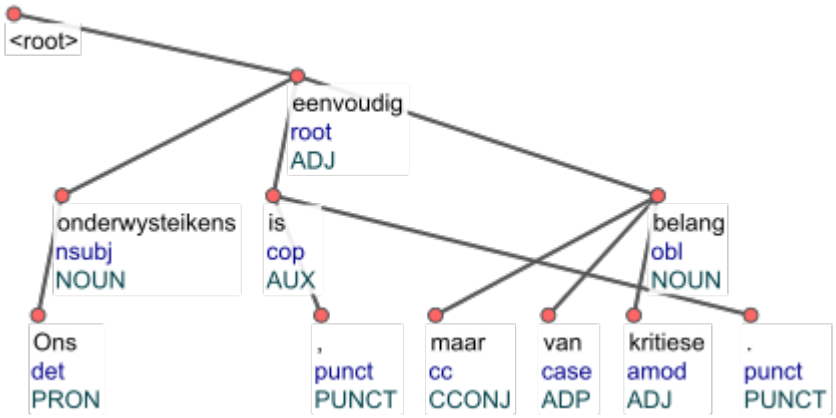


Figure 5.7: *attachToSibling()*: Single Sibling Available

In case there are multiple siblings, we try to find the nearest sibling that has the deprel as `conj`, and try attaching the conjunction to this marked conjunct. Essentially, this check would ensure that there is no need to search for another candidate, as the nearest sibling is the one that should be the head. In case the attachment to the marked conjunct will be non-projective in nature, the candidate would be located further away, and is not fit to being the head. However, this approach might fail owing to the conjunct not being explicitly marked. In the final search for the candidate conjunct in the siblings, we try to find the candidate by restricting the deprels to `obl`, `xcomp`, `nmod`, and `nsubj` amongst the siblings, attaching therein if such a case is found.

The choice of the deprels is not arbitrary, but is based on an elimination procedure whereby we discarded most deprels. For selection of the candidate deprels, we restricted ourself to the core arguments, non-core dependents and nominal dependents that correspond to nominal and clausal structural categories[5]. Of these relations, `dislocated`, `expl`, `nummod`, `vocative` can be outright discarded from the consideration. For `appos`, the documentations marks explicitly the case where the deprel is chained in presence of a coordination[6], marking the subsequent tokens as `conj`, rather than as `appos`. The deprels `acl` and `advcl` function

[5]The first two columns, and the entries against the structural categories as indicated in `https://universaldependencies.org/u/dep/all.html`

[6]`https://universaldependencies.org/u/dep/appos.html`

as clausal modifiers in form of an adjective[7], or an adverb[8] respectively. Since the dependents are explicitly clausal, we can very well discard them from consideration of conjuncts, alongwith `appos`.

The documentations for the deprels `obj`[9] and `iobj`[10] states that in presence of more than one proto-patients, the primary is to be labelled as `obj`, and the rest as `iobj`. However, even in presence of multiple objects (tri-transitive verbs are rare, but nonetheless present in Caucasian languages like Georgian, and Svan for example), the objects to the verb are often associated in form of causatives (cf. [Chirikba, 2003, p. 39], [Boeder, 2005, p. 43]). This can be further extrapolated into a lack of conjunction between such objects of the tri-transitive verbs, thereby ensuring that we can safely discard the deprels `obj` and `iobj` from our consideration as well.

The documentation[11] for deprel `csubj` states that the deprel is used when the subject itself is a clause. The guideline would ensure that there are very few cases when the deprels might be chained together by coordination, and even more so while they are at the same level in the tree. We therefore remove the deprel from consideration.

Comparing the documentations of `ccomp`[12] and `xcomp`[13], there is no way to say if either deprel is a better fit for the candidate head. In our experiments, the experiment performance went down when `ccomp` was included in the final list of head deprels. Keeping that in mind, we only include `xcomp` and discard `ccomp` from our consideration of candidate head deprels.

We could not find a strong reason for discarding the remaining deprels, viz. `obl`, `nmod`, and `nsubj`, and thus included them with `xcomp` in the list of deprels that can be searched for, while looking for a candidate conjunct.

The restriction with respect to deprels is necessary to make sure we don't over-generate and rehang the conjunction to a wrong head. As demonstrated earlier, a non-first conjunct may or may not be labelled by the `conj` deprel. The deprels associated with the core arguments, non-core dependents and nominal dependents are governed (non-deterministically) by the POS tag of a token and the head of this token, whereas the `conj` deprel is not limited by the POS tag of either the token or its head. Therefore, we can rely upon the other deprels to be annotated better than the `conj` relation. However, if the deprel is not restricted, the token of interest might associate itself to the wrong sibling, but in the correct direction, making it as a case of a conjunction sandwich (which as we mentioned earlier, is significantly harder to detect).

We formally define the constraints and the processing in Algorithm 5. Notice how we decide on whether or not the algorithm terminates by continuously checking the condition of projectivity, and returning a value from the function only if the condition of projectivity with respect to the new parent is maintained. It is also important to note that we always limit our search for a suitable candidate to cases where the candidate occurs later than the conjunction we are trying to

[7] `https://universaldependencies.org/u/dep/acl.html`
[8] `https://universaldependencies.org/u/dep/advcl.html`
[9] `https://universaldependencies.org/u/dep/obj.html`
[10] `https://universaldependencies.org/u/dep/iobj.html`
[11] `https://universaldependencies.org/u/dep/csubj.html`
[12] `https://universaldependencies.org/u/dep/ccomp.html`
[13] `https://universaldependencies.org/u/dep/xcomp.html`

rehang.

Algorithm 5 attachToSibling()

Input: *node* such that $misdirectedDependency(node) ==$ **true**

{Try to attach to a sibling node}
$count \leftarrow 0$
$origParent \leftarrow node.parent$
for all *siblings* of *node* **do**
  if *siblings.upos* not in [“X”, “$PUNCT$”, “SYM”] **and** *siblings.id* > *node.id* **then**
    $TargetSibling \leftarrow siblings$
    $count \leftarrow count + 1$
  end if
end for
if $count == 1$ **then**
  {Just one sibling, attach to this sibling}
  if $setParent(node, origParent, TargetSibling)$ **then**
    return true
  end if
end if
{More than one siblings, narrow search by deprels}
for *sibling* of *node* **do**
  if *sibling.udeprel* == “*conj*” **and** *sibling.id* > *node.id* **then**
    if $setParent(node, origParent, sibling)$ **then**
      return true
    end if
  end if
end for
for *sibling* of *node* **do**
  if *sibling.udeprel* in [“*obl*”, “*xcomp*”, “*nmod*”, “*nsubj*”] **and** *node.id* < *sibling.id* **then**
    if $setParent(node, origParent, sibling)$ **then**
      return true
    end if
  end if
end for
return false

If there is no suitable candidate in the same level as the current level of the conjunction, we try to ascend one level and try to attach the node to the next aunt (parent's sibling) in Algorithm 6. The condition may arise owing to not finding a suitable candidate in siblings, or in case where there are no siblings to search for. We do not set any checks with respect to deprels, but still keep a check on the condition of projectivity and the node order. Consider the following part of sentence from `af` treebank in Example 13 with the corresponding annotations in Figure 5.8, with the token of interest marked in bold. Figure 5.8b shows the part where the algorithm connects the conjunction to the parent's sibling, after having failed trying to find an attachment amongst the siblings.

Algorithm 6 attachToAunt()

Input: *node* such that *misdirectedDependency*(*node*) == **true**

{Try to attach to the first relevant aunt node}
origParent ← *node.parent*
aunts = []
for *sibling* of *origParent* **do**
  if *sibling.id* > *node.id* **then**
    aunts.append(*sibling*)
  end if
end for
{The candidate aunts would be arranged in word-order in the *aunts* list}
if *aunts* is not empty **then**
  setParent(*node*, *origParent*, *aunts*[0])
end if
return false

Example 13.
Text (af): Indien die laaste dag vir betaling op 'n openbare vakansiedag **of** oor die naweek val, ...
Lit: In-the-event-that the last day for pay on a public holiday **or** over the weekend fall, ...
Translated: If the last day for payment falls on a public holiday or over the weekend, ...

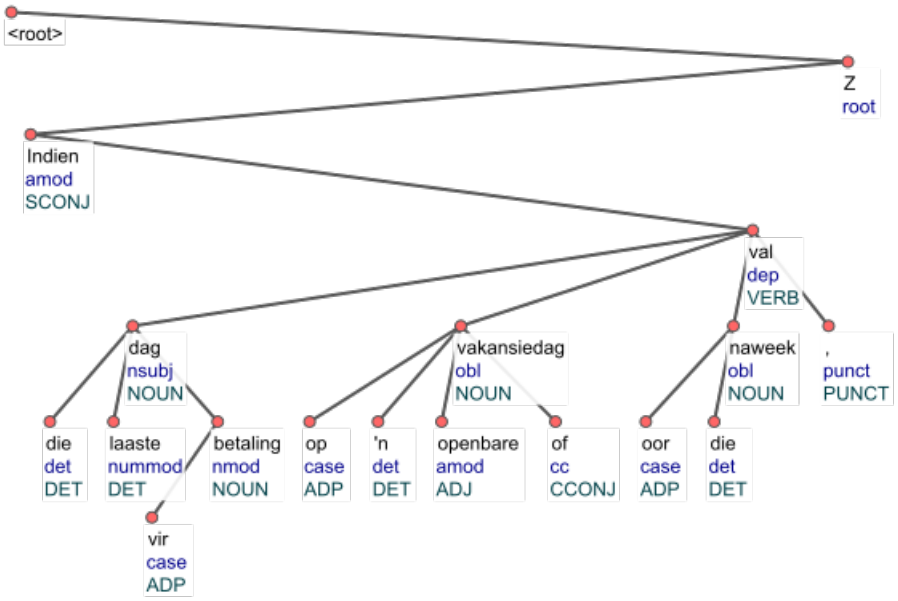
In the event that a suitable candidate is not found, a **false** value is returned. This implies that our search for a suitable candidate has failed even after trying to ascend one level. As last resort, we try to attach the conjunction to the grandparent, while preserving projectivity in Algorithm 7. An example of case where this function is needed is elaborated in Example 14 containing the sentence from `af` treebank, with the corresponding annotations in Figure 5.9.

Algorithm 7 attachToGrandparent()

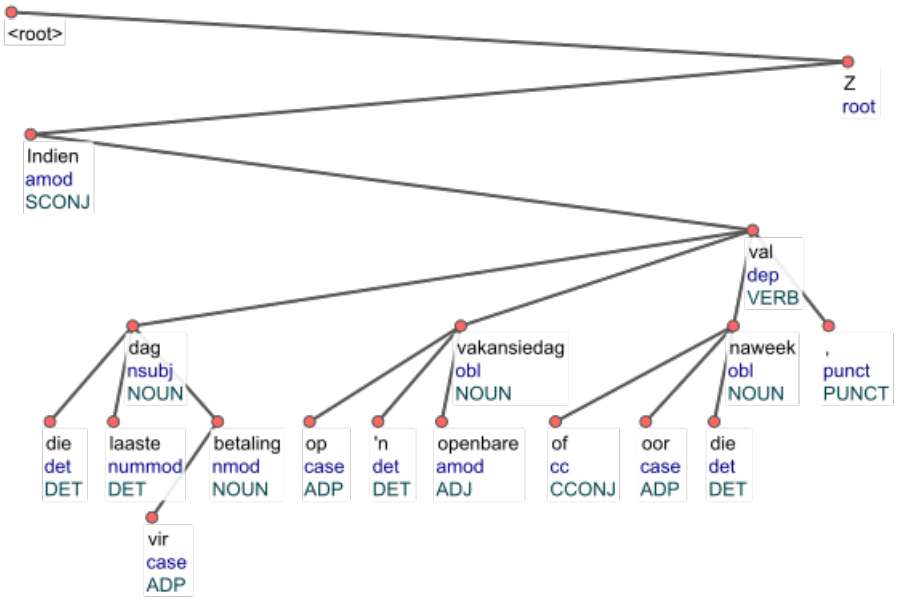
Input: *node* such that *misdirectedDependency*(*node*) == **true**

{Try to attach to the grandparent node}
origParent ← *node.parent*
grandparent ← *origParent.parent*
if *setParent*(*node*, *origParent*, *grandparent*) **then**
  return true
end if
return false

Example 14.
Text (af): Ons onderwys- **en** vaardigheidsprogramme sal ons produktiwiteit en mededingendheid verhoog .
Lit: Our education- **and** skills-program-*Pl.* shall our productivity and competitiveness increase .
Translated: Our education and skills programs will increase our productivity and competitiveness.

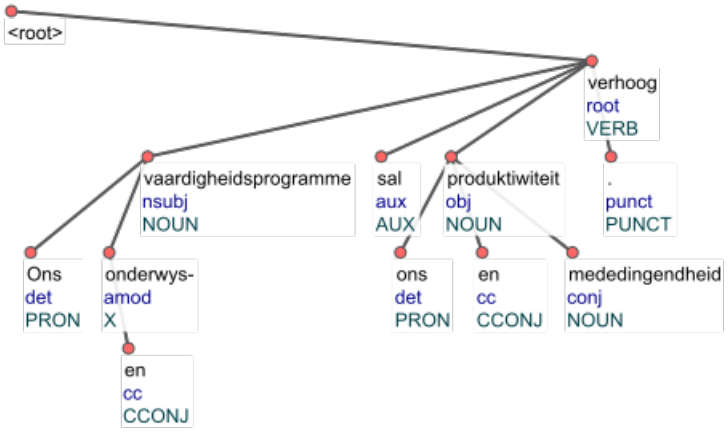


(a) Original Annotation

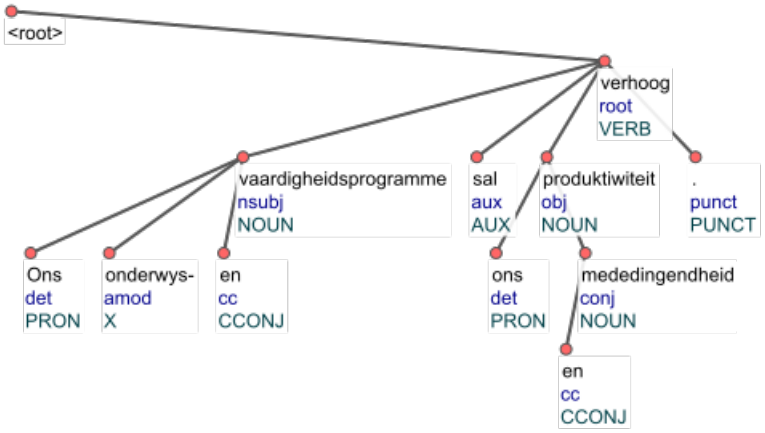


(b) Modified Annotation after attachToAunt()

Figure 5.8: Change in Annotation for Example 13
Note: *Z* is used to denote the position of original root of the sentence
Note: **of** (or) should be attached to **naweek** (weekend)



(a) Original Annotation



(b) Modified Annotation after attachToGrandparent()

Figure 5.9: Change in Annotation for Example 14
Note: **en** (and) should be attached to **vaardigheidsprogramme** (skills-program)

Having established all the possible cases, we can wrap them all in a nice function that takes care of all the cases, in priority order. Algorithm 8 shows the complete algorithm, in order of execution of the functions defined throughout the section.

Algorithm 8 fixconjhead()

Input: *node* such that $misdirectedDependency(node) ==$ **true**

```
if isnonprojective(node) == true then
    if not nextConjHead(node, node.parent) then
        if not projTempFix(node, node.parent) then
            Do Nothing
        end if
    end if
end if
{Made non-projective attachments projective}
if attachToSibling(node) then
    return
else if attachToAunt(node) then
    return
else if attachToGrandparent(node) then
    return
else
    return
end if
```

5.5 Evaluation and Results

We implement the algorithm in form of a Udapi-python [Popel et al., 2017] block[14]. The runtime of the block for the data is as mentioned in Table 5.5, as run on Ubuntu 18.04 (64-bit) on a 4-core Intel i5-6300 HQ processor.

Language	Time (in ms)
af	81.33 ± 7.094
ar	317.05 ± 23.996

Table 5.5: Average Runtime ($\pm$ sd) for Udapy Python Block Implementation
Note: Does Not include time taken to read the original CoNLL-U file

In this section, we would first evaluate the treatment of originally nonprojective attachments with respect to the individual segments of the algorithm, followed by a discussion of the instances not handled by the algorithm dealing specifically with nonprojective attachments. The instances were manually annotated for the direction of dependency, as well as for the choice of the correct head. Next, we would look at the part of the algorithm that is common to all tokens, irrespective of their projectivity status. Our focus would be on the nodes that were affected at major steps, and the nodes that were unaffected by the end of the algorithm. We then look at the overall evaluation of the algorithm, as manually annotated for the correct attachment to the parent node, on limited subsamples.

[14]Code alongwith manually annotated data available at `https://github.com/Akshayanti/Masters-Thesis-CUNI-2020/tree/master/conj_head`

5.5.1 Originally Non-Projective Attachments

The number of originally nonprojective nodes affected by the first part of the algorithm, where they were associated with either the next marked conjunct, or where their attachment was temporarily made projective is as listed in Table 5.6. We discuss on the effect of individual functions in the following subsections.

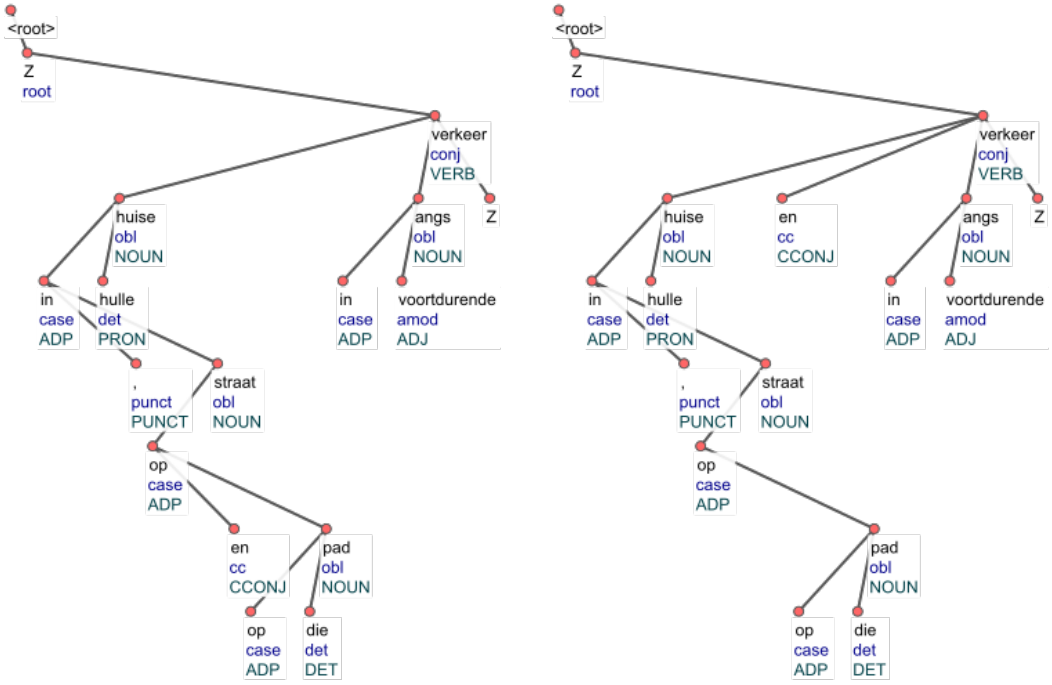
Lang.	**Total**	*nextConjHead()*	*projTempFix()*	Unaffected
`af`	130	20	106	5
`ar`	80	-	44	36

Table 5.6: Nodes Affected: Non Projective Attachment

Effect of *nextConjHead()*

Of all the nodes affected by *nextConjHead()* algorithm (cf. Algorithm 3), 85% of the nodes were associated to the right parent. Of the remaining 15%, we found annotation errors which resulted in a failure in identification of the more relevant head. The annotation errors in these case were primarily associated with the wrong token being marked as a conjunct, or an explicitly marked conjunct of another coordination structure being selected as candidate (the conjunct in current coordination structure was not explicitly marked). While the modified annotation in these cases corrected the direction of dependency, there was a failure in determination of the correct head of attachment. This is a case of conjunction sandwich, as discussed earlier, and introduced in this context owing to a faulty correction. An example of such case is shown in Example 15, with the associated annotations in Figure 5.10. As before, the token of interest is marked in bold.

Example 15.
Text (`af`): ... in hulle huise, op straat **en** op die pad in voortdurende angs verkeer ...
Lit: ... in *3Pl-Poss.* house, on street **and** on the path-*Sg.* in continuous anxiety find-*Pres.* ...
Translated: ... in their house, on street and on the road in continuous anxiety ...



(a) Original Annotation (b) Final Annotation

Figure 5.10: Annotation Error in Example 15

Note: Z is used as a placeholder for omitted text
Note: **,** should be attached to **huise** (house) or to the immediately succeeding **op** (on)
Note: **straat** (street) and its children should be attached to **verkeer** (find-*Pres.*)
Note: **en** (and) should be attached to **pad** (path-*Sg.*)
Note: **pad** (path-*Sg.*) and its children should be attached to **verkeer** (find-*Pres.*)
Note: **verkeer** (find-*Pres.*) is wrongly marked as `conj`

Effect of *projTempFix()*

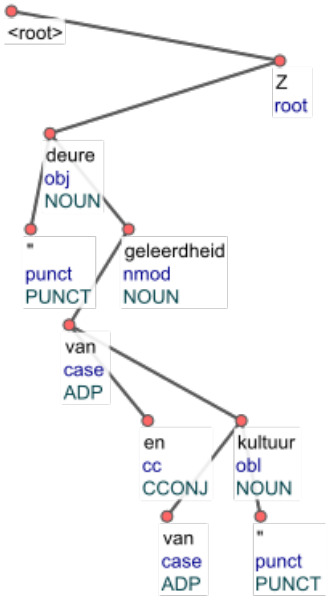
Table 5.7 shows the number of instances that were forced into a projective attachment when *projTempFix()* algorithm (cf. Algorithm 4) was used on them. The total count of such instances is listed in the second column. The values in third column onward refer to the results of the manual verification, verified with respect to the direction of new attachment and the relevancy of the choice of head for the new attachment. The manual verification was done after the projectivised token was subjected to the overall algorithm. The third column refers specifically to the cases where the direction was corrected, but the choice of head of attachment was not correct, thereby resulting in a conjunction sandwich. The value in the fourth column refers to the count of tokens that had no change whatsoever in their attachment, before and after the algorithm. The value in the last column represents the count of tokens such that the attachment to new parent was correct in both the aspects.

Lang.	**Total**	Conj. Sand.	Unfixed	Correct
af	106	12	91	3
ar	44	-	42	2

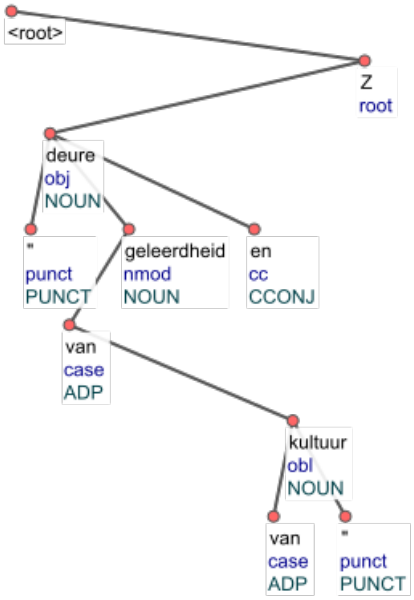
Table 5.7: Evaluation: *projTempFix()*

While the results for the forced projectivisation are primarily negative, there seems to be a pattern to the results. In the analysis of the instances marked as unfixed for either language, it was found that the correct attachment was not possible because the relevant part of the dependency tree had the original annotation wrong and non-projective while the correct annotation would be projective. Consider one such instance in Example 16, and the associated dependency trees in Figure 5.11, with the token of interest marked in bold. Notice the adposition **van** (of) being shared wrongly with **kultuur** (culture) token in Figures 5.11a, 5.11b. The overall corrected annotation is reflected in Figure 5.11c.

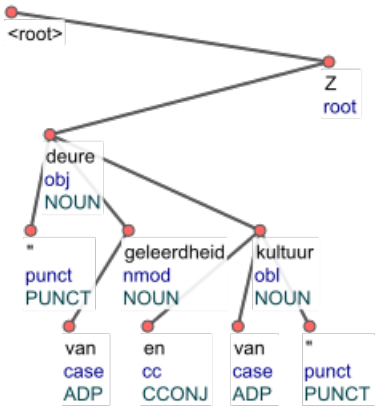
Example 16.
Text (af): "deure van geleerdheid **en** van kultuur"
Lit: " doors of learning **and** of culture "
Translated: "doors of learning and of culture"



(a) Original Annotation



(b) Final Annotation

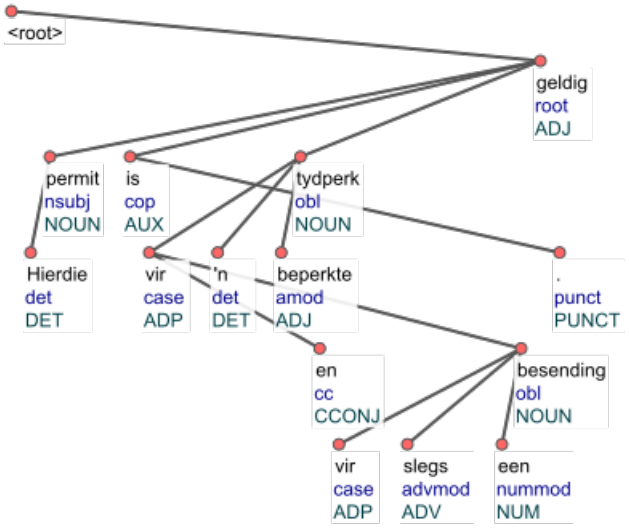


(c) Overall Corrected Annotation

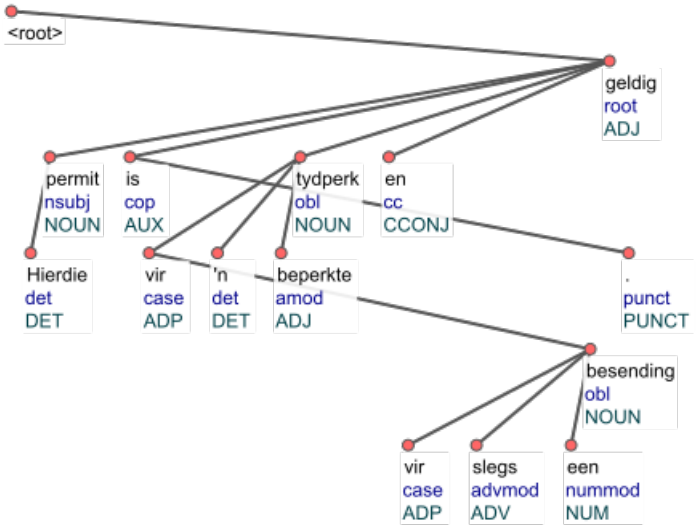
Figure 5.11: Dependency Trees for Example 16
Note: *Z* is used to denote the node where the subtree is attached in the original sentence

The problem of the false annotations with respect to non-projectivity is an open problem that is not handled in the current work. The problem is discussed in brief in Section 8.3. In the current context, the cases of conjunction sandwich are also attributed to such wrong annotations. Consider the sentence from `af` treebank in Example 17 and the associated dependency trees in Figure 5.12, with the token of interest marked in bold. Note that the adposition **vir** (for) is shared wrongly by **besending** (consignment), bringing false non-projectivity into the sentence structure.

Example 17.
Text (`af`): Hierdie permit is vir 'n beperkte tydperk **en** vir slegs een besending geldig .
Lit: this permit is for a limited period **and** for only one consignment valid .
Translated: This permit is valid for a limited period and for only one consignment.



(a) Original Annotation



(b) Final Annotation

Figure 5.12: Dependency Trees for Example 17
Note: **besending** (consignment) should be linked to **geldig** (valid)
Note: **en** (and) should be linked to **besending** (consignment)
Note: **.** should be linked to **geldig** (valid)

Non-Projective Attachments Not Handled

All the cases that were unprocessed after the attempt at projectivisation of non-projective attachments were processed by the overall algorithm. We do not discuss here the statistics on the correction procedure of such cases, but leave it for the next section when we evaluate the overall algorithm.

5.5.2 Processing Pipeline Independent of Projectivity of Attachment

As can be seen from Table 5.8, the manual evaluation if done on a randomly chosen sample of the affected nodes would be very heavily biased on the results

from *attachToSibling()* algorithm. To counter this effect, we decided to separately evaluate the algorithms, and so a random sample of 100 affected nodes was chosen containing the nodes affected by *attachToSibling()* algorithm only. To measure the efficiency of *attachToAunt()* and *attachToGrandparent()* algorithms, another sample containing 100 randomly sampled instances was chosen. All the sampled instances were then manually annotated for the correctness in their attachment to the correct conjunct, as well as the direction of the attachment.

Lang.	**Total**	*attachToSibling()*	*attachToAunt()*	*attachToGrandparent()*	Unaffected
af	1809	1665	8	124	12
ar	1411	952	58	178	223

Table 5.8: Nodes Affected: Overall

Overall Evaluation

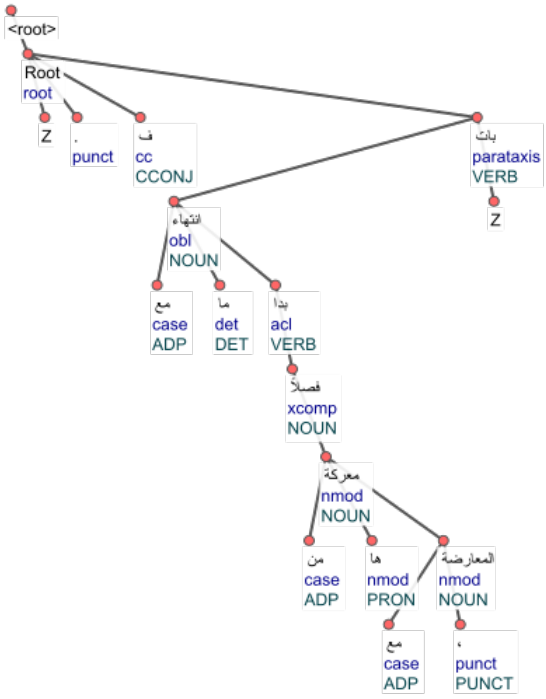
We estimated the effect of *projTempFix()* earlier, and so to estimate the effects of the different algorithms in an overall manner, such tokens were not included in either sample. Furthermore, the difference between the total count of instances and the listed instances identified as either of correct or as a case of conjunction sandwich marks the number of instances that were still misdirected in their attachment. Table 5.9 lists the results of the manual evaluation.

Algo. →	*attachToSibling()*			Others		
Lang. ↓	**Total**	Conj. Sand.	Correct	**Total**	Conj. Sand.	Correct
af	100	1	99	32	1	22
ar	100	2	98	100	4	28

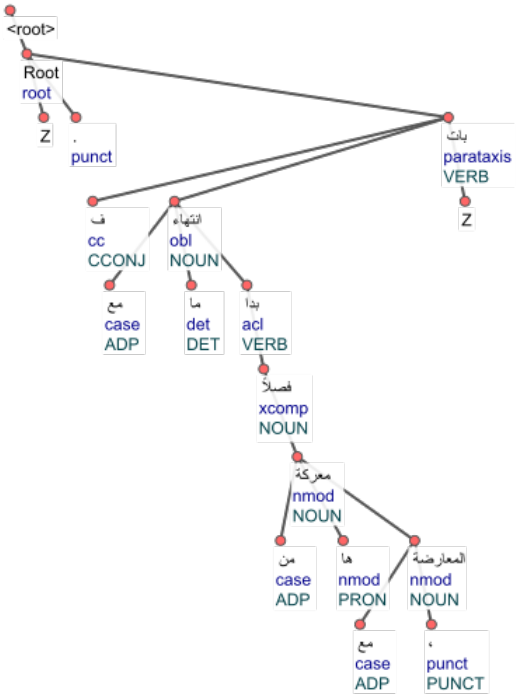
Table 5.9: Overall Evaluation of Affected Nodes on Randomly Sampled Instances

We based *attachToSibling()* algorithm based on the assumption that we would not need to descend the tree level in the search for the correct conjunct and that we need to only ascend the level in the tree. In the analysis of instances with an introduced conjunction sandwich in ar, the correct conjunct could have been found by descending the tree level. Example 18 shows the relevant part of one such example, with the corresponding dependency trees before and after the correction procedure in Figures 5.13a and 5.13b respectively. In the example, *Z* is used to denote the omitted part of the tree, while *Root* is used to donate the root of the tree. The token of interest is marked in bold.

Example 18.
Text (ar in RTL): ... Z بات ، المعارضة مع معركتها من فصلاً ما بدا ما انتهاء **ف**مع . Z ...
Translit (Top-down): Z . **f**-mae aintiha' ma bada fslaan min maerakat-ha mae almuearadat , bat Z
Lit. (Top-down): ... . **And**-with finishing what appear-*Perf.-3P.* chapter from battle-it-*3P.-Sing.* with opposition , become-*Perf.-3P.* Z
Translated: ... . With the end of what appeared to be a chapter in the battle against the opposition, it became ...



(a) Original Annotation



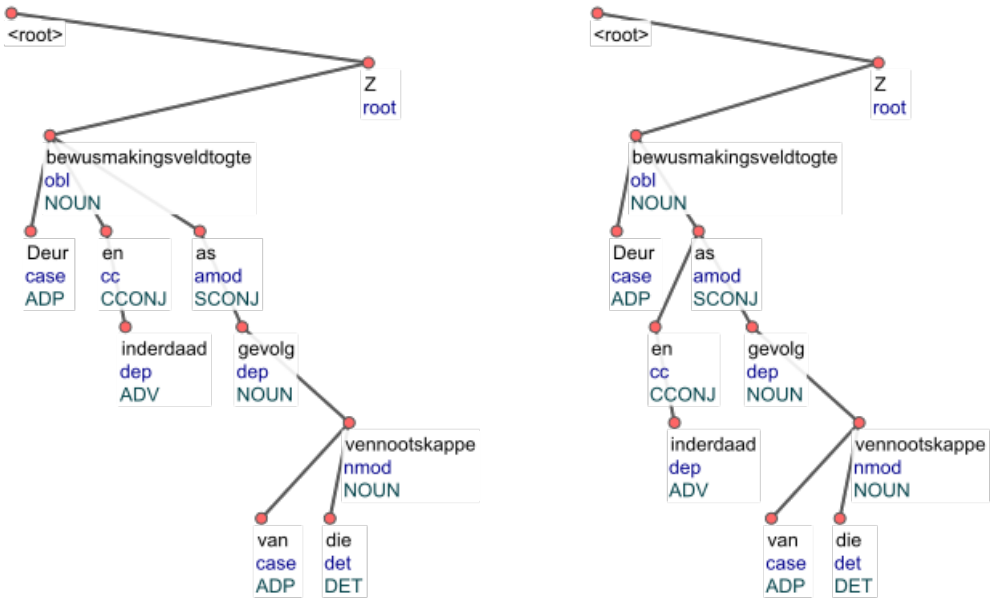
(b) Final Annotation

Figure 5.13: Introduced Conjunction Sandwich in `ar`
Note: Z is used as a placeholder for omitted text
Note: Root is used as a placeholder for root of the tree
Note: ف (*f*; And) should be attached to انتهاء (*aintiha'*; finishing) and not to بات (*bat*; become-*Perf.-3P*)

In case of `af`, the actual conjunct could not be discovered because of the improper annotation of the subtree. The modification as done by *attachToSi-*

bling() algorithm attached the conjunction to where the conjunct should have been. Attachment to the right conjunct in this case was not possible because (i) the change of levels in present annotation would bypass the enforced limit of one level; and (ii) the new attachment would have been non-projective in nature, and was therefore not allowed. Example 19 and the associated dependency trees in Figure 5.14 demonstrate this with a part of the actual sentence. As in previous example, *Z* is used to denote the omitted part of the tree, while also showcasing the relative position of the root of the tree. The token of interest is marked in bold.

Example 19.
Text (af): Deur bewusmakingsveldtogte **en** inderdaad as gevolg van die vennootskappe Z
Lit: Through awareness-campaigns **and** indeed as consequence of the partnerships ...
Translated: Through awareness campaigns **and** indeed because of the partnerships ...



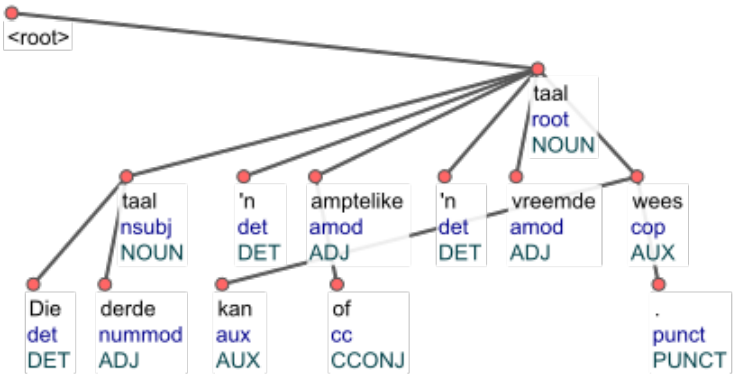
(a) Original Annotation (b) Final Annotation

Figure 5.14: Introduced Conjunction Sandwich in `af`
Note: Z is used as a placeholder for omitted text, and also to mark the position of the root of the tree
Note: **gevolg** (consequence) should be the head of the subtree, with **as** (as) attached to it
Note: **inderdaad** (indeed) should be attached to **gevolg** (consequence) after the change of subtree head
Note: **en** (and) should be attached to **gevolg** (consequence) after the change of subtree head

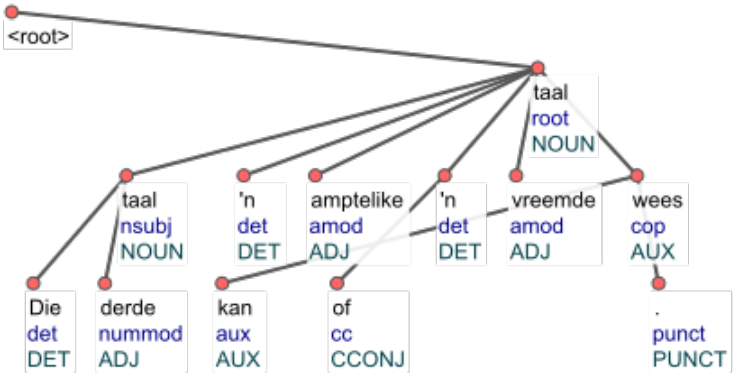
The instances of misdirected dependency in conjunctions that escaped processing by *attachToSibling()* algorithm were then processed by algorithms *attachToAunt()* and *attachToGrandparent()* in that order. We found that the majority

of these cases were still misdirected even after being processed by the overall algorithm. We discuss such cases in the next section where we discuss some insights into the processing of the algorithm step by step. Of the instances that led to a case of conjunction sandwich, the majority of the cases were caused by an annotation error, caused due to improper selection of the head of the relevant subtree. The example from `af` treebank in Example 20, and the associated dependency tree in Figure 5.15 demonstrates this. In the example, the conjunction sandwich is caused because of the improper annotation in the tree. The conjunction of interest is marked in bold.

Example 20.
Text (`af`): Die derde taal kan 'n amptelike **of** 'n vreemde taal wees.
Lit: The third language can a official **or** a alien language be .
Translated: The third language may be an official or a foreign language .



(a) Original Annotation



(b) Modified Annotation

Figure 5.15: Change in Annotation for Example 20
Note: **of** (and) should be attached to **vreemde** (alien) and not to **'n** (a)

Unaffected Nodes

By unaffected nodes, we refer to the instances of misdirected dependency which were not at all touched by the entire algorithm. We hypothesized earlier that if the rehanging of the node requires a change in more than one level (of the level of wrong conjunct), it is likely to be an annotation error that needs manual correction. We found that to be true for more than 50% of the cases in either

treebank with respect to all the unaffected cases. For the remaining cases, the major reason why the node could not be rehung was associated with the limit of deprels in *attachToSibling()* algorithm. Since that was also the case for a majority of cases where the misdirected dependency persisted, we discuss the unaffected nodes with them in the final discussion.

5.6 Discussion and Conclusion

We started by identifying the cases of conjunctions that were attached non-projectively to the parent, and employed algorithms *nextConjHead()* and *projTempFix()* to find a better candidate. We got mixed results in all analyzed cases in both treebanks. From our understanding of the patterns exhibited in the two languages, the first algorithm works only if there exists an explicitly marked conjunct. This was true in case of `af` where the conjuncts are explicitly marked with `conj` deprel, but when the conjuncts are not explicitly marked (as in case of `ar`), the algorithm doesn't work as intended.

The force projectivisation in *projTempFix()* algorithm did not have the desired effect. In the analysis of the instances, we found that this was mainly due to falsely annotated non-projectivities. In general, if the conjunction was in gap of another non-projective attachment to the same parent, the algorithm didn't work. The inefficiency of the algorithm in such cases could be exhibited in the form of node not being affected at all, or the new attachment eventually leading to a case of conjunction sandwich. If the conjunction (and any punctuation nodes attached to this token) is the only non-projective attachment to the parent, the algorithm would be able to make the correction effectively, and without an error.

Given the aforementioned concerns about the algorithms, it would be recommended to not use the algorithms in case of a language that displays high amount of non-projectivity in sentence structures (for example, `grc`) and/or on a treebank has not been checked for the annotation consistency of the non-projective structures (as in the scope of the current experiment). Furthermore, in a case where the algorithms are used, it would be advised to have an annotator look at the corrections for higher reliability.

The common part of the pipeline started with *attachToSibling()* algorithm that seeks to associate a misdirected conjunction to a sibling token, attached to the same parent. The number of cases that were found to introduce a case of conjunction sandwich could have been caused due to multiple reasons. We limited the search of a candidate head by the candidate's UPOS (more specifically, blacklisting a few UPOS) in case of a single available candidate, as mentioned earlier in the definition of the algorithm. In the event of the candidate being marked by the blacklisted UPOS, no matter the choice of deprels (except `conj`), the candidate was discarded from consideration. In the algorithm, we looked for the candidate sibling within the same subtree, and not at the same level in the next subtree. This was the reason why some of the conjuncts that were located in the following subtree were discarded by the algorithm, and rather their parent (the conjunction's aunt node) was selected as the new candidate, thus introducing conjunction sandwich. The third and the final cause of conjunction sandwiches was rooted in our assumption. In the search for a candidate head, the choice was limited by the current level of attachment. We looked for a candidate at the same

or at a higher level of the current attachment, thereby missing a few cases when the candidate was located at a lower level.

While the aforementioned reasons did bring about the cases of a conjunction sandwich, a relaxation of the choice of UPOS, deprels would have catastrophic effects whereby the conjunction would be rehanged to any available node. The search for a candidate at the same level, but in the following subtree is a promising approach, but it does warrant caution in the case of the suitable candidate being the aunt, and not the new candidate thus discovered. This selection of the candidate head would be non-deterministic in nature, and also depends on the annotation consistency of the given tree. Since the number of cases that were ignored, or generated conjunction sandwich into the annotation were significantly low, the problem of descending down a level to search for a candidate node can be safely ignored. In experiments where the approach was tried, the selection of the candidate node became non-deterministic, and generated a lot of false positives and introduced plethora of conjunction sandwiches.

In the evaluation, *attachToSibling()* algorithms performs very well, even after accounting for sampling error. For the instances that are not processed by the algorithm, *attachToAunt()* and *attachToGrandparent()* algorithms don't perform as well. Upon analysis of instances that are passed to the latter algorithms, we observe a pattern. In general, if a conjunction occurs at a position such that it can change the level at which it is associated with, it will further be processed by the algorithms *attachToAunt()* and *attachToGrandparent()*, or would remain a case of misdirected dependency. The position of an instance in a dependency tree can be more often than not given by Equation 5.1, where *co* is the conjunction of interest in dependency tree T, attached to the node u.

$$\boxed{co : u \rightarrow co \;\; \& \;\; \nexists(x)[u \rightarrow x \;\; \& \;\; co \neq x] \quad co, u \in T} \tag{5.1}$$

A conjunction that satisfies above property can move around the tree, and can be associated to an aunt, to a grandparent, or the root of the tree, as relevant. In case the token does not satisfy the above property, and also is not affected by *attachToSibling()* algorithm, it will continue being a misdirected dependency. Table 5.10 shows the total number of instances with misdirected dependency, before and after the pipeline.

Lang.	**Total**	Before (in %)	After (in %)
af	1832	1829 (99.84)	106 (5.79)
ar	13855	1411 (10.18)	398 (2.87)

Table 5.10: Misdirected Dependencies: Before and After
Note: % is calculated against the total number of conjunctions, in the second column

The algorithm *attachToGrandparent()* processes more instances than *attachToAunt()* algorithm, as evident from Table 5.8. however, this processing is without any observable effect. A major reason for this is that the algorithm does not seek to find a conjunct in the grandparent's sibling, and thus just changes the level of attachment without changing the direction explicitly. This is helpful only in very limited number of cases as shown in Example 14 earlier.

The number of false positives and true positives is quite low for the joint evaluation of *attachToAunt()* and *attachToGrandparent()* algorithms. This prevents discussion of the efficiency of algorithms, as most of the instances that were passed on to these algorithms had no choice of candidates to attach themselves to.

In conclusion, even though the individual algorithms of the entire pipeline vary in their results and efficiency, the approach is promising. We analysed the cases where the automation can go wrong, and the factors that would prevent automation in certain cases, and in certain language typologies.

6. Mining Errors in Low-Resource Languages by Combining LISCA And Cross-Validation (Experiment 3)

While discussing the available tools for detecting annotation consistency, we discussed about LISCA [Dell'Orletta et al., 2013] in Section 3.1.3. To briefly summarise the contents of the aforementioned section, LISCA (**LI**nguistically-driven **S**election of **C**orrect **A**rcs) takes as input a reference corpus, and assigns to each arc a plausibility score based on the occurrence of similar arcs. For calculation of the plausibility scores, the algorithm relies on global, as well as local features of each arc. Figure 3.1 reproduced below, shows the features as used by LISCA to model the given training data.

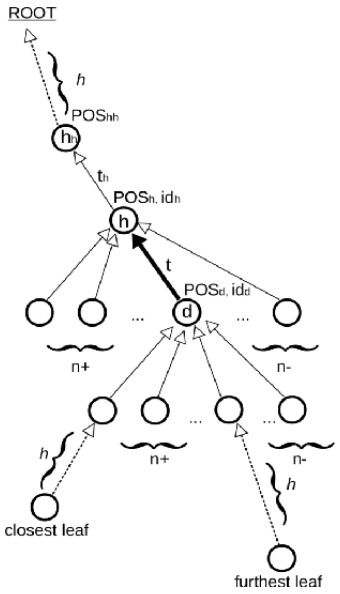


Figure 3.1: Features Used by LISCA to Calculate Plausibility Score for an Arc (marked in bold). Figure borrowed from Alzetta et al. [2017].

Local Feature: Distance in terms of tokens between d and h
Local Feature: Associative strength linking grammatical categories POS_d and POS_h
Local Feature: POS of the head governor and type of syntactic dependency connecting it to h
Global Feature: Distance of d from the root of the tree
Global Feature: Distance of d from the closest or the most distant leaf node
Global Feature: Number of siblings to the right of node d in the linear order of the sentence
Global Feature: Number of children to the left of node d in the linear order of the sentence

There are two goals of the experiment in this chapter. For the low-resource languages, when there is no reference corpus, LISCA cannot be used directly. A common approach used in the case of low-resource languages, k-fold cross-validation is explored in this experiment. However, just using cross-validation is not enough, as the choice of the number of folds can affect the results significantly. In this experiment, we therefore (i) evaluate if k-fold cross-validation is an optimal strategy against the approach of keeping the test and train data separated, and (ii) try to map the behaviour of the algorithm to the choice of the number of folds in k-fold cross-validation approach.

In the following subsections, we shall elaborate on the experiment with LISCA. We start with the specification of the dataset to be used for the experiment in section 6.1, followed by the elaboration on experimental setup in section 6.2. Section 6.3 specifies the manner of investigation for the analysis of arcs. We report the preliminary statistics on different runs in section 6.4, before diving into an analysis of the results in section 6.5. Section 6.6 deals with the typologies of errors discovered over the complete experiment. The chapter concludes with a discussion on the findings of the experiment in section 6.7.

6.1 Dataset

The experiment was conducted entirely on data from `hi`-HDTB treebank from UDv2.4[1] [Nivre et al., 2019]. The motivation behind the limiting of the dataset to a particular language is threefold. Firstly, the treebank in question is limited to news genre. The lack of variability in the genre in the treebank can be used to frame a better statistical model than when there would be different genres present. Secondly, the treebank is medium sized (16,000+ sentences containing around 350k tokens) as can be seen in Table 6.1. The medium sized treebank is optimal in the manner that a variety of values (of the number of folds in k-fold cross validation procedure) can be experimented with. Furthermore, the different values of the parameter can be used to ascertain the performance of the algorithm in both large-sized and small-sized treebanks. Lastly, the author has `hi` as their native language, making it easier for them to analyse the given data, thus reducing the source of ambiguity during the process of manual annotation and verification of results from the results of the algorithm.

Split	Sentences	Tokens
dev	1 659	35 217
test	1 684	35 430
train	13 304	281 057
Total	**16 647**	**351 704**

Table 6.1: Size of `hi`-HDTB treebank

[1]Code alongwith manually annotated data is available at `https://github.com/Akshayanti/Masters-Thesis-CUNI-2020/tree/master/lisca_cv`

6.2 Experimental Setup

For the remainder of the experiment, we adapt the usage of *iteration* and *run* as follows. The results of one *run* would be analysed together. For a given k value in k-fold cross validation, the experimental data is split into k different folds, running k *iterations* for one run.

For the different runs of the current experiment, the total number of sentences poses a problem in the terms of how many folds the data can be split into[2]. To combat this problem, we first concatenate the different splits of the treebank into one. The concatenated split is then downsampled to 16,000 sentences. This downsampled data becomes our functional dataset for the experiment. The downsampling is needed to allow for the different values of k to work. While the data if downsampled to 16,640 instances would have also worked, we chose to set the count to 16,000 sentences for empirical reasons. The number of sentences from the original splits that feature in the downsampled version are as listed in Table 6.2.

Split Name	**Sentences**		**Tokens**	
	Before	**After**	**Before**	**After**
dev	1 659	1 601	35 217	33 964
test	1 684	1 614	35 430	33 981
train	13 304	12 785	281 057	270 249
Total	16 647	16 000	351 704	338 194

Table 6.2: Counts of Sentences and Tokens from Individual Splits, Before and After Downsampling

Setup for Baseline Run

We call an arc as belonging to downsampled train data if (i) the arc was part of the train set in the original data, and (ii) the arc is present in the downsampled data as well. The arcs belonging to downsampled dev data and downsampled test data are also defined similarly.

For establishing a baseline, we train the algorithm on downsampled dev and downsampled train sets, concatenated together. The trained algorithm is then run against the downsampled test data to get the plausibility score of the individual arcs present therein.

Setup for Experimental Runs

The experiments were conducted on 3 different values of k. The chosen values were $k = \{2, 4, 8\}$. When the values of $k \geq 10$ were considered, the resulting data folds became smaller enough to not yield satisfactory results.

For each value of k, the cross-validation procedure was applied to get the plausibility scores for the arcs in the entire downsampled dataset. The LISCA algorithm for each iteration was run by Alzetta et al. separately. Algorithm 9 summarises the procedure involved so far.

[2]16,647 can be factorised as 3 x 31 x 179, which allows limited manipulation in the number of folds that can be worked with for equal distribution of instances.

Algorithm 9 Experimental Setup for k-fold Cross Validation

Input: Downsampled `hi`-HDTB Treebank T

for all k in {2, 4, 8} **do**
  $T.folds \leftarrow \{T.1, ..., T.k\}$ subject to conditions:
    $T = \bigcup\{T.1, ..., T.k\}$ {Condition 1}
    $sentences(T.i) = sentences(T)/k \ \forall T.i \in T$ {Condition 2}
    $\bigcap\{T.x1, T.x2\} = \phi \ \forall\{T.x1, T.x2\} \in T.folds$ {Condition 3}
  for *iteration* in 1, ..., k **do**
    $fold.test \leftarrow T.iteration$
    $fold.training \leftarrow T - T.iteration$
    $lisca.iteration \leftarrow$ trained LISCA model on $fold.training$
    $lisca.iteration$ is used to assign plausibility score to arcs in $fold.test$
  end for
end for

6.3 Arcs in Focus

The evaluation of a trained LISCA model on a given test data generates several types of statistics. In addition, the individual arcs in the results of the LISCA algorithm are split into 10 equal bins in descending order of their plausibility scores, with an additional bin for the remnants. The statistics are presented on a per-bin basis and include POS distribution, deprel distribution, POS and deprel distribution, syntactic link length distribution, among others. While the per-bin statistics are a useful feature, the cross validation process in the context of current experiment does not need such per-bin statistics. Instead we focus on individual arcs and their plausibility scores in the current experiment.

Henceforth, we call a particular arc as flagged in a particular run if its plausibility score in the run is designated as 0, i.e. the arc is deemed as improbable by the run. While Alzetta et al. [2017] looked at all the instances in the last two bins (and the extra remnant bin), the current setup narrows down the search scope. The last two (and the extra remnant) bins in question are the only ones containing arcs with 0-score or with scores that are very close to 0. As we would show later (in section 6.4.2), the scores for non-zero scored arcs would fluctuate with different datasets of the same language, or even based on the number of folds in cross validation. This can be extrapolated to state that the non-zero scored arcs in the bins in question can also vary in their scores, making the bin-specific treatment incomparable across different runs. In contrast, looking at zero-scored arcs gives us a uniform base for analysis throughout, considering that the arc was marked as improbable, and not probable with a low score.

6.4 Statistics

6.4.1 Baseline Run

The baseline run tried to find the low-probability arcs in the downsampled test data. Table 6.3 shows the basic statistics of the run.

Statistic	Count / Value
Min Score	0.00
Max Score	1.82 E-07
Flagged Arcs (in %)	221 (0.7 %)
Total Arcs	33 739

Table 6.3: Statistics for Arc Scores in Baseline Run. The percentage score of Flagged Arcs is calculated against the Total Arcs count.

Once the plausibility scores are assigned for the arcs, the flagged arcs were manually checked to see if they are erroneous or not. Of the 221 flagged arcs in the run, 110 arcs were found to be erroneous. The complete typology of the errors is reserved for later. However, Table 6.4 shows the classification of errors from the run into Random or Systemic Errors.

Statistic	Count (% Total Arcs)
No Error	111 (49.3 %)
Systemic Errors	96 (43.4 %)
Random Errors	14 (6.3 %)
Total Flagged Arcs	221

Table 6.4: Classification of Errors in Baseline Run

6.4.2 Experimental Runs

Table 6.5 shows the number of arcs that were flagged across different experimental runs. As mentioned earlier, the maximum plausibility score of arcs in a given run fluctuates with the different k-values across different runs, even when the overall experimental data remains the same.

k-value	Min Score	Max Score	0-score arcs	Total arcs
2	0.00	1.96 E-07	3 487	336 079
4	0.00	1.93 E-07	2 620	336 079
8	0.00	1.91 E-07	2 319	336 079

Table 6.5: Statistics for Arc Scores in Experimental Runs

The number of 0-scored arcs went down with an increasing k-value. In addition, all the arcs flagged in a particular run were also present in a run with a lower k-value, i.e. the arcs flagged in run with $k = 4$ were also present in $k = 2$. Similarly, the arcs flagged in run with $k = 8$ were present in the run with $k = 4$ as well as one with $k = 2$. We compare the performance of the different experimental runs against each other in section 6.5.2.

6.5 Analysis

In this section, we analyse the experiment in two parts. In the first part of the analysis (Section 6.5.1), we check the usefulness of k-fold cross validation

against the arcs from only the downsampled test data, comparing them at the same time. The primary motive of this analysis is to understand how the cross validation technique performs in relation to the baseline approach at identifying erroneous instances in a low-resource setting.

In the second part of the analysis (Section 6.5.2), we look at all the arcs that are flagged in different cross validation runs, regardless of them belonging to the downsampled test, dev or train data. The motive of this analysis is to understand how the difference in number of folds during cross validation affects the flagged instances.

6.5.1 Baseline vs Cross Validation: Who did it better?

Table 6.6 shows the number of test arcs that were flagged across different cross validation runs. The values in the last column represent the count of instances that were flagged by the experimental run as well as the baseline run.

k-value	# Flagged	# Also Flagged by Baseline (% # Flagged)
2	333	211 (63.36%)
4	254	205 (80.71%)
8	226	205 (90.71%)

Table 6.6: Commonly Flagged Instances from Downsampled Test Data in Baseline and Experimental Runs

Table 6.7 shows the counts of arcs in downsampled test data that were flagged across different runs, and the count of flagged arcs that were erroneous.

Run	# Flagged (TP+FP)	# Errors TP	Error Precision (in %) TP*100/(TP+FP)
Baseline	221	109	49.32 %
Experimental ($k = 2$)	333	160	48.05 %
Experimental ($k = 4$)	254	127	50.00 %
Experimental ($k = 8$)	226	114	50.44 %

Table 6.7: Error Counts in Downsampled Test Data across Different Runs
Note: TP = True Positives
Note: FP = False Positives

Table 6.7 can be analysed in two different ways. The first analysis would focus on the error precision for each run. We notice that an increase in k-value in cross-validation approach results in an increasing precision. While the experimental run with $k = 2$ had a precision lower than the precision of the baseline run ($\Delta = -1.27\%$), the other experimental runs had a higher precision than the baseline run ($\Delta = 0.68\%$ for $k = 4$ and $\Delta = 1.12\%$ for $k = 8$). In this aspect, cross-validation technique still outperforms the trivial technique used in the baseline task. However, the choice of k-value in this case needs to be monitored for a higher precision.

The second analysis of data in Table 6.7 would essentially focus on the number of identified erroneous arcs in the individual run. Considering that we are interested in a higher number of error arcs, either of the experimental runs outperform the baseline task in that aspect as well.

We therefore are able to establish that the cross-validation technique is a better choice than the trivial approach. Table 6.8 shows the typology of different errors as identified in the different runs. We discuss the most relevant error typologies in Section 6.6.

Error Typology	**Baseline**	$k = 2$	$k = 4$	$k = 8$
`advcl4advmod`	2 (0.9%)	2 (0.6%)	2 (0.8%)	2 (0.9%)
`advcl4det`	-	2 (0.6%)	-	-
`amod4acl`	2 (0.9%)	3 (0.9%)	2 (0.8%)	2 (0.8%)
`amod4xcomp`	2 (0.9%)	3 (0.9%)	3 (1.2%)	2 (0.9%)
`compound4det`	-	2 (0.6%)	1 (0.4%)	1 (0.4%)
`compound4obj`	1 (0.5%)	2 (0.6%)	2 (0.8%)	2 (0.9%)
`nmod4obl`	-	4 (1.2%)	2 (0.8%)	1 (0.4%)
`obl4advcl|acl`	1 (0.5%)	2 (0.6%)	2 (0.8%)	1 (0.4%)
`obl4discourse|mark`	-	3 (0.9%)	1 (0.4%)	-
`Case Error`	5 (2.3%)	6 (1.8%)	5 (2.0%)	5 (2.2%)
`MWE Error`	5 (2.3%)	5 (1.5%)	5 (2.0%)	5 (2.2%)
`Naming Error`	9 (4.1%)	11 (3.3%)	9 (3.5%)	8 (3.5%)
`POS Error`	5 (2.3%)	5 (1.5%)	3 (1.2%)	3 (1.3%)
`Reported Speech`	4 (1.8%)	2 (0.6%)	2 (0.8%)	2 (0.9%)
`Tree Error`	20 (9.0%)	29 (8.7%)	25 (9.8%)	22 (9.7%)
`Wrong Head`	38 (17.2%)	54 (16.2%)	42 (16.5%)	40 (17.7%)
Random Errors	15 (6.8%)	25 (7.5%)	21 (8.3%)	18 (8.0%)
No Error	112 (50.7%)	173 (50.0%)	127 (50.0%)	112 (49.6%)
Total Flagged Arcs	**221**	**333**	**254**	**226**

Table 6.8: Typology of Errors in Downsampled Test Data across Different Runs. Percentages are calculated against the Total number of Flagged Arcs in the Run. Error Typologies marked in bold have been previously pointed out by Alzetta et al. [2017]

6.5.2 Comparing Different Experimental Runs

For the analysis of the different cross-validation runs, we noticed that the count of flagged instances decreased with the increase in the number of folds. We hypothesise that as we increase the number of folds, the detection of rare errors improves while the detection of frequent errors deteriorates. having noted this, we analysed the effect of each k-value in the following manner. For the 0-scored arcs that were common to all the runs, 200 randomly chosen arcs (out of 2319) were evaluated manually. Out of the arcs common only to the runs corresponding to $k = \{2, 4\}$, 100 were randomly chosen for manual evaluation. Finally, 100 of the arcs that are local only to the run corresponding to $k = 2$ were chosen randomly for manual evaluation. The manual evaluation on a flagged instance was meant to classify if the flagged instance is indeed an error, and if so, of what kind.

The manual annotation on limited subsamples as above does not offer a comparative viewpoint of the performance of the different runs. To combat this, we

estimate the normalized frequency of each error type over 1000 instances in Table 6.9. The values are calculated as per the equation given below. The equation normalizes the frequency of an error over 1000 flagged arcs, based on the distribution of the error in the annotated samples.

$$f_{error} = \begin{cases} k_{2,4,8} \cdot 5 & \text{for } k = 8 \\ \left[\frac{k_{2,4,8} \cdot 2319}{200} + \frac{k_{2,4} \cdot 301}{100}\right] \cdot \frac{1000}{2620} & \text{for } k = 4 \\ \left[\frac{k_{2,4,8} \cdot 2319}{200} + \frac{k_{2,4} \cdot 301}{100} + \frac{k_2 \cdot 867}{100}\right] \cdot \frac{1000}{3487} & \text{for } k = 2 \end{cases}$$

where

- f_{error} represents the normalised frequency of *error*
- $k_{2,4,8}$ represents the counts of *error* in the annotated sample of arcs commonly flagged by all the experimental runs
- $k_{2,4}$ represents the counts of *error* in the annotated sample of arcs commonly flagged by runs with $k = 2$ or $k = 4$, but not flagged by run with $k = 8$
- k_2 represents the counts of *error* in the annotated sample of arcs flagged only by the run with $k = 2$, but not flagged by runs with $k = 4$ or $k = 8$

Error Typology	$k = 2$	$k = 4$	$k = 8$
`advmod4amod`	7	9	10
`dep4det`	6	8	5
`dep4discourse|mark`	7	6	5
`nsubj4obj`	8	7	5
`obl4advcl|acl`	6	8	5
`Case Error`	16	8	5
`MWE Error`	15	13	15
`Naming Error`	43	51	50
`POS Error`	10	13	15
`Reported Speech`	12	13	15
`Tree Error`	48	61	60
`Wrong Head`	163	167	180
Random Errors	141	116	115
No Error	518	520	515
Total Errors	**482**	**480**	**485**

Table 6.9: Error Frequencies for Experimental Runs, Normalized Over 1000 Flagged Arcs. Error Typologies marked in bold have been previously pointed out by Alzetta et al. [2017]

Perhaps the most striking result from Table 6.9 is how the different experimental runs are almost similar in their performance. Notice that in Table 6.7, an increase in number of folds was accompanied by an increase in the calculated

error precision. The analysis in the two cases is different. While in Table 6.7 we checked if using the cross-validation to train the algorithm has any significant performance gain; the results in Table 6.9 analyzes if the number of folds has any correlation with error-detection rate when there is no reference corpus, and the algorithm is trained and tested on the same data.

Since the number of folds in cases when the algorithm is trained and tested using cross-validation has little to no effect in performance gain, the only point of differentiation between different runs is with respect to the number of flagged arcs. While it is lucrative to use less number of folds (or a lower k value), the approach would be bottle-necked by the size of the dataset.

An error type is considered significant if its normalized frequency is more than 1% (frequency > 10) in Table 6.9. We discuss the significant error types in the next section.

6.6 Error Typologies

In this section, we elaborate on the error typologies discovered throughout the scope of the experiment such that the discovered error type is present in more than 1% of the arcs flagged in any run, baseline or experimental. The focused errors in this section include: `Case Error`, `MWE Error`, `Naming Error`, `Reported Speech`, `Tree Error` and `Wrong Head`. Since 'Random Errors' are not systemic in nature, we do not elaborate on them. Additionally, `POS Error` corresponds to an error in the POS annotation label, and is not elaborated upon any further in this section.

6.6.1 `Case Error`: Identification Error of Case-Marker

In `hi`, the different grammatical cases are more often than not marked by case-marker tokens. This error corresponds to such case-markers being marked by deprels other than `case`. Additionally, the deprel is the preferred choice for constructions that involve possessions as well. In the event that the used deprel is other than `case`, we call it as `Case Error`.

Part of sentence from UDv2.4 `hi`-HDTB treebank in Example 21, and the associated dependency tree in Figure 6.1 highlights the error type. The token of interest is marked in bold.

Example 21.
Text (`hi`): मार्क्रसवादी कम्युनिस्ट पार्टी (माकपा) **के** दस सांसदों
Translit: *Marx-vaadi Communist Party (MaCPa) **ke** das saansadon*
Lit.: Marxist Communist Party (MaCPa) *Poss.* ten senator-*Acc.-Pl.*
Translated: Ten senators of Marxist Communist Party (MaCPa)

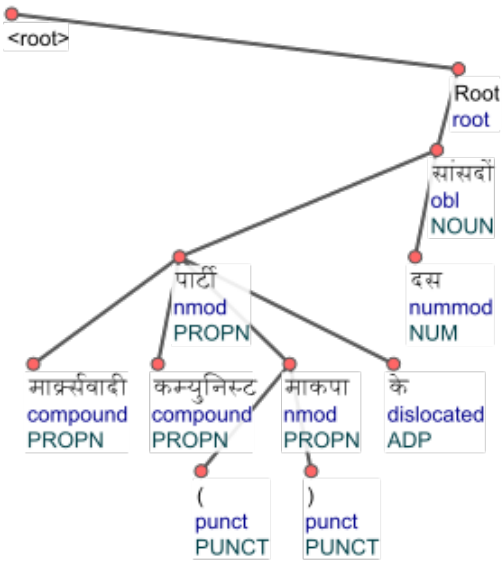


Figure 6.1: `Case Error` in Example 21
Note: Root is used as a placeholder for root of the tree
Note: Typo in मार्क्र्सवादी (*Marx-vaadi*; Marxist). Corrected token should be मार्क्सवादी (*Marx-vaadi*; Marxist)
Note: मार्क्र्सवादी (*Marx-vaadi*; Marxist) should be attached to पार्टी (*Party*; Party) using deprel `flat`, and not with `compound`
Note: कम्युनिस्ट (*Communist*; Communist) should be attached to पार्टी (*Party*; Party) using deprel `flat`, and not with `compound`
Note: माकपा (*MaCPa*; abbreviation of Marxist Communist Party) should be attached to पार्टी (*Party*; Party) using deprel `appos`, and not with `nmod`
Note: के (*ke*; *Poss.*) should be attached to पार्टी (*Party*; Party) using deprel `case`, and not with `dislocated`

6.6.2 `MWE Error`: Annotation Error in Multi-Word Expression (MWE)

The different tokens in a Multi-Word Expression (MWE) are combined by either of the deprels in UDv2: `fixed`, `compound` or `flat`. Of these, `fixed` is used for completely fixed grammaticized (function word-like) MWEs (like 'in spite of'), and `compound` applies to endocentric (headed) MWEs (like 'apple pie').

The usage of `fixed` deprel is covered separately in `Naming Error`. For instances when a MWE should be annotated as either of `compound` or `fixed` deprels, but is annotated otherwise, we refer to the error as `MWE Error`. Example 22 shows the error type in a sentence from UDv2.4 `hi`-HDTB treebank, with the associated dependency tree in Figure 6.2. The MWE is marked in bold.

Example 22.
Text (hi): इसकी सबसे बड़ी विशेषता यह है कि सामान्य कार्य चलता रहेगा और किसी चीज की प्रोसेसिंग भी **अपने आप** होती रहेगी।
Translit: *iski sabse badi visheshta yeh hai ki saamaanya kaarya chaltaa rahegaa aur kisi cheej ki processing bhi **apne aap** hoti rahegi.*
Lit.: *3P.Poss. Superlative* big feature this is that normal work run-*Imp. Fin.* and some thing *Poss.* processing also **by-itself** happen will .
Translated: Its biggest quality is that the processing can take place by itself while the normal task is being taken care of.

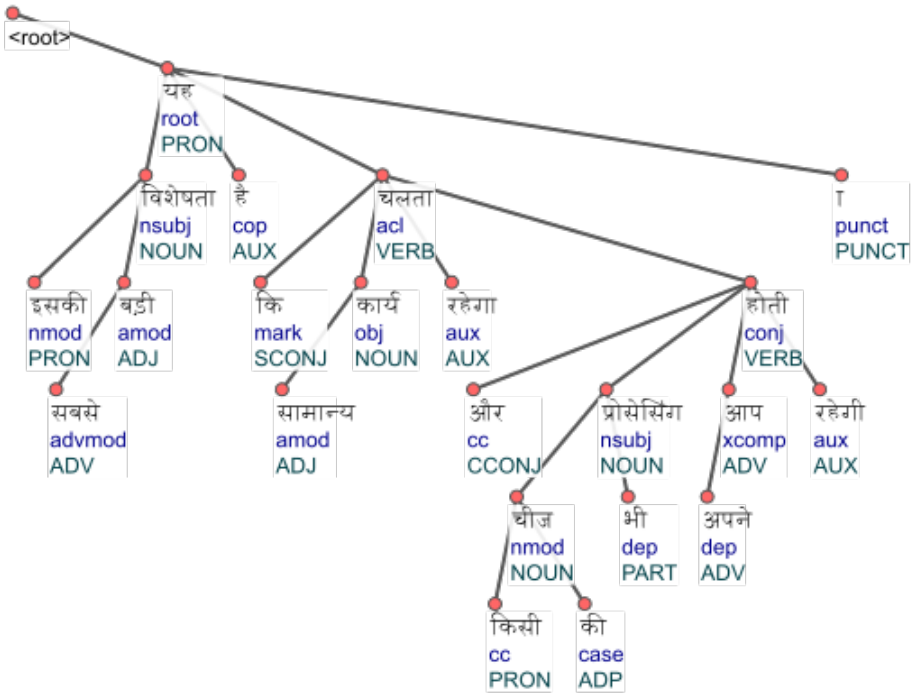


Figure 6.2: `MWE Error` in Example 22
Note: अपने (*apne*; -Refl.) should be connected to आप (*aap*; *2P-Formal*) using deprel `fixed`, and not `dep`

6.6.3 `Reported Speech`: Annotation Error in Construction With Reported Speech

According to UDv2 guidelines for treatment of reported speech[3], the reported speech is connected to the main clause by using either of the deprel `ccomp` or `parataxis`.

The error `Reported Speech` corresponds to case when the reported speech and main clause are not connected by proper deprels, as in the example from UDv2.4 `hi`-HDTB treebank.

Example 23.
Text (hi): समिति ने कहा था कि सभी संस्थान मौजूदा आईआईटी के स्तर की तुलना में काफी पीछे हैं।

[3] `https://universaldependencies.org/u/dep/parataxis.html#treatment-of-reported-speech`

Translit: *samiti ne kahaa thaa ki sabhi sansthaan maujoodaa IIT ke star ki tulnaa mei kaafi peeche hain.*
Lit.: Committee *Erg.* say be-*Perf.* that all institutes present IIT *Poss.* level in-comparison-with quite behind is-*Pl.* .
Translated: Committee had said that all institutes are far behind the level of the current IITs.

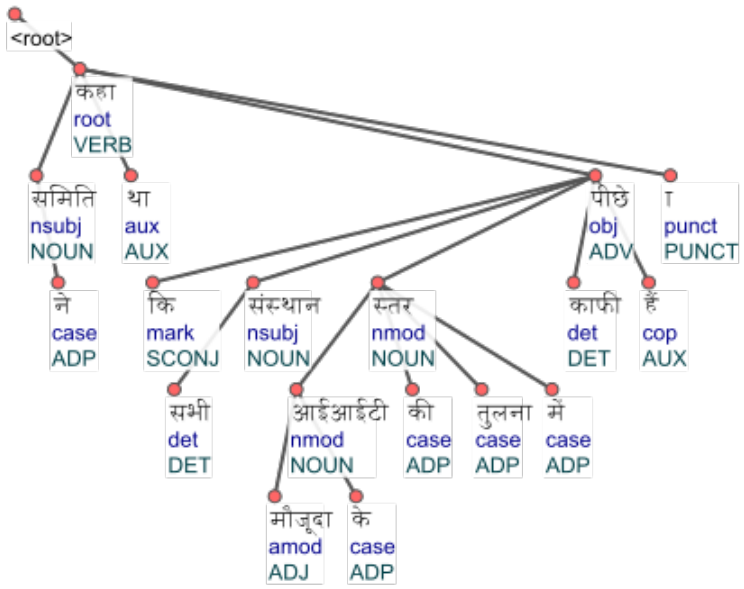


Figure 6.3: `Reported Speech Error` in Example 23
Note: पीछे (*peeche*; behind) should be attached to कहा (*kaha*; say) using the deprel `ccomp`

6.6.4 `Wrong Head`: Head Identification Error

The error refers to the cases when the dependent in the flagged arc is attached to a wrong head. This is the umbrella error type for all the cases of head identification error that cannot be categorised more specifically into other error types.

While Alzetta et al. [2017] mention head labelling error as a sub-type of the error patterns discussed therein, we identify this error in a category on its own. We separate this error type because multiple parsers/taggers determine the deprel of a dependent in an arc based on the head of the said dependent. Keeping this in mind, `Wrong Head` is very likely to result in a faulty deprel annotation as well. However, attachment to the correct head in this case should essentially result in a correction of the annotated deprel as well.

Consider Example 24 and the associated dependency tree in Figure 6.4. The example is part of a sentence taken from the UDv2.4 `hi`-HDTB treebank, and shows the token of interest (marked in bold) attached to a wrong head.

Example 24.
Text (`hi`): जिनकी **मदद** से **वह आवाज** को पहचान व समझ सकता है
Translit: *jinki* ***madad*** *se* ***vah aavaaz*** *ko pehchaan va samajh saktaa hai*
Lit.: whose help with it sound *Acc.* recognise and understand can is
Translated: With help of which, it can recognise and understand sound.

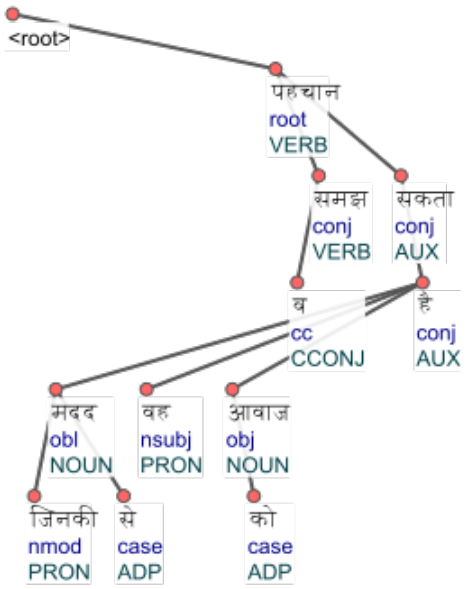


Figure 6.4: Head Identification Error in Example 24
Note: मदद (*madad*; help) should be attached to पहचान (*pehchaan*; recognise) and not to है (*hai*; is)
Note: वह (*vah*; it) should be attached to पहचान (*pehchaan*; recognise) and not to है (*hai*; is)
Note: आवाज (*aavaz*; sound) should be attached to पहचान (*pehchaan*; recognise) and not to है (*hai*; is)
Note: सकता (*sakta*; can) should be labelled as `aux` and not as `conj`
Note: है (*hai*; is) should be labelled as `aux` and not as `conj`
Note: है (*hai*; is) should be attached to पहचान (*pehchaan*; recognise) and not to सकता (*sakta*; can)

6.6.5 `Naming Error`: Annotation Error in Proper Nouns

`Naming Error` is often accompanied with a head-identification error. Annotation Errors in Proper Nouns can be of three kinds, and all of these are commonly grouped under `Naming Error`. The following are the possible cases of error in annotation:

1. **Proper Noun as Appositional Modifier** (`4appos`): The deprel `appos`[4] is used when the proper noun defines, modifies, names or describes a preceding nominal. It also includes parenthesized examples, and the abbreviations. This error is characterized by an attempt to connect the two nominals by relations such as `nmod`, when the actual deprel should be `appos`.

2. **Names/Dates without Syntactic Structure** (`4flat`): The different parts of a single name, or of a date should be attached to the head with the deprel `flat`[5]. The deprel is also used in cases of a honorofic or a title. This error type is characterized by usage of other deprels when `flat` should be the deprel of choice.

[4] `https://universaldependencies.org/u/dep/appos.html`
[5] `https://universaldependencies.org/u/dep/flat.html`

3. **Names with Syntactic Structure**: Names that follow a syntactic structure (like 'A Tale of Two Cities') should not be annotated with `flat` deprel, but with regular syntactic relations. In this case, the error is characterized by a name with syntactic structure being analysed in the same way as a name without syntactic structure.

Consider the part of a sentence from UDv2.4 `hi`-HDTB treebank showcasing all the above cases in Example 25 and the associated dependency tree in Figure 6.5

Example 25.
Text (`hi`): आरसी मिश्रा की पुस्तक 'मानवाधिकार संरक्षण विशेष संदर्भ, अपराधियों का निरोध एवं उपचार'
Translit: *Aarsi Mishra ki pustak ' Maanavadhikar sanrakshan vishesh sandarbh , apraadhiyon ka nirodh evam upchaar '*
Lit.: Aarsi Mishra *Poss.* book ' Human-Rights Protection Special Reference , criminal-*Pl. Acc.* prevention and cure '
Translated: Aarsi Mishra's book, 'Maanavadhikar sanrakshan vishesh sandarbh , apraadhiyon ka nirodh evam upchaar'

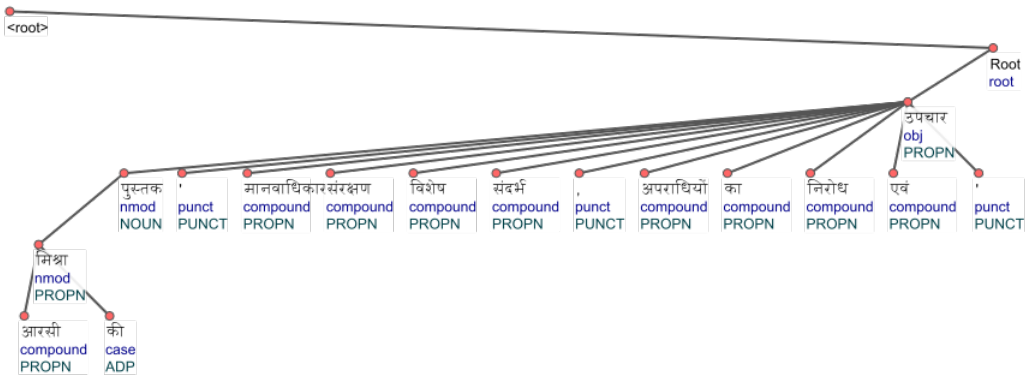


Figure 6.5: `Naming Error` in Example 25
Note: Root is used as a placeholder for root of the tree
Note: आरसी (*Aarsi*; Aarsi) should be attached to मिश्रा (*Mishra*; Mishra) using deprel `flat`, and not with `compound`
Note: The title of the book (limited by quotes) should be attached to पुस्तक (*pustak*; book) with the deprel `appos`
Note: The title of the book (limited by quotes) should be annotated with regular syntactic relations

6.6.6 `Tree Error`: Dependency Head Located in Subtree

A special case of `Wrong Head` error, this error type is used for the cases when the actual head of a dependency is located inside the subtree rooted at the dependent. In order to correct the dependency, it should be essentially inverted. Essentially speaking, a tree marked with this error type requires re-annotation before any analysis can be performed on it.

Example 26 shows an instance of this error in UDv2.4 `hi`-HDTB treebank, with the associated dependency tree in Figure 6.6. The dependent of interest is marked in bold, and the corrected instance is as shown in Figure 6.7.

Example 26.

Text (`hi`): आतंकियों द्वारा किसी विमान के अपहरण या आत्मघाती हमले को अंजाम **देने** की कोशिश किए जाने की खुफिया जानकारी

Translit: *aatankiyon dwara kisi vimaan ke apharan ya aatmghaati hamle ko anjaam* ***dene*** *ki koshish kiye jaane ki khufiya jaankaari*

Lit.: Terrorists by some plane *Poss.* kidnap or self-harm attack *Dat.* fruition give *Dat.* attempt do-*Pass.* to-be confidential information

Translated: The confidential information of attempt at some plane hijacking or suicide bombing by terrorists ...

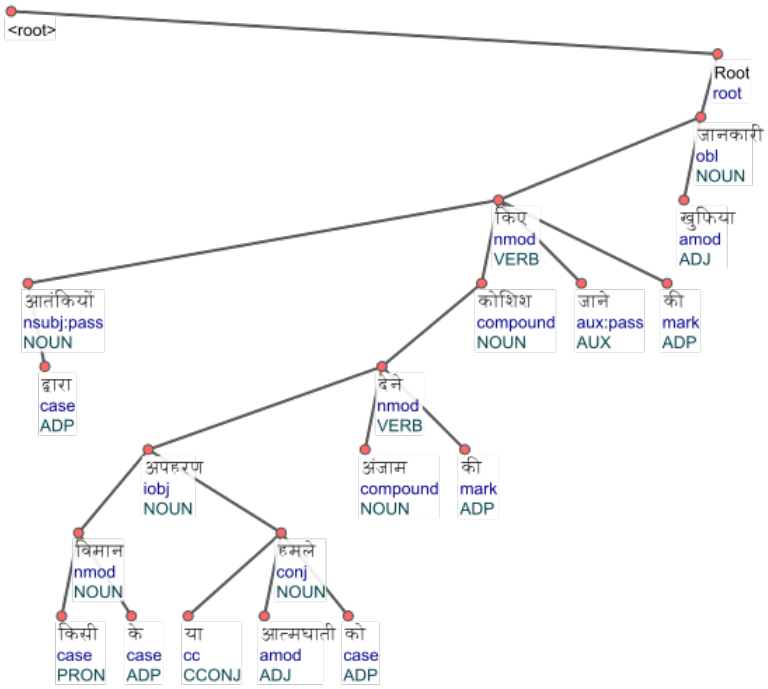


Figure 6.6: `Subtree Error` in Example 26
Note: Root is used as a placeholder for root of the tree

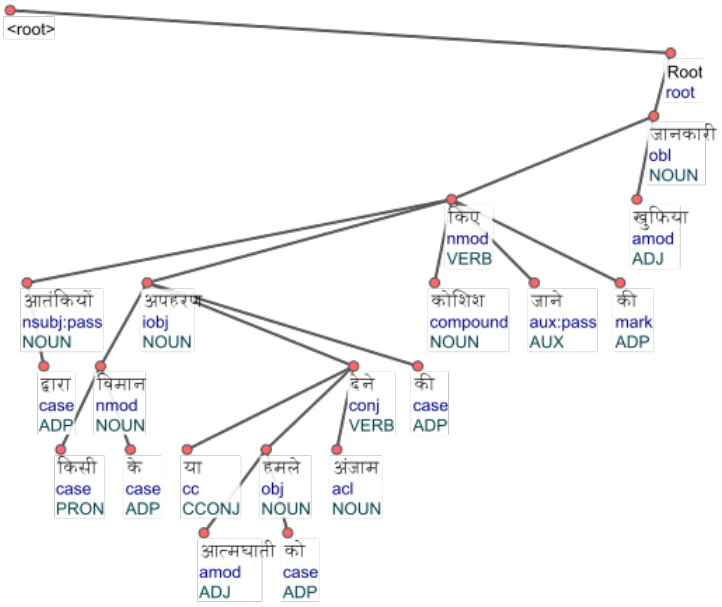


Figure 6.7: Correction of `Subtree Error` in Example 26
Note: Root is used as a placeholder for root of the tree

6.7 Results and Discussion

6.7.1 0-scored Arcs as Search Criteria

In their work, Alzetta et al. [2017] focused on a total of 39.7k arcs in their annotation process and were finally able to manually revised 789 arcs, giving an estimated error detection rate of 2% from the flagged instances. In our baseline run, a focus on 221 0-scored arcs led to an estimated error detection of 109 instances (49.32 %). We must stress here that the results across the two experiments are NOT directly comparable since the treebanks used in the cited authors' experiment was of far superior quality than the one used in the current experiment. A lower quality treebank would imply a higher distribution of errors, and that could be the sole reason why the focus on a smaller subset gave a satisfactory error detection rate. Additionally, the size difference in the cited authors' work and the baseline task is another reason why the two approaches cannot be compared. We must also stress here that in our baseline approach, the search scope was lowered significantly (as compared to the experimental runs). To establish any significant difference between either approach, more experiments should be conducted with the same treebank (ensuring the quality of experimental data is a controlled variable) to establish the probability distribution of errors in 0-scored arcs and in the approach as utilised by Alzetta et al..

6.7.2 Cross Validation as Strategy

Considering that the different runs perform almost similarly (Table 6.9), we argue that the size of dataset used is the determining factor in selection of the number of folds in k-fold cross validation.

For less number of folds (or a lower k value), the number of flagged arcs is

high, which eventually results in more errors detected. However, in case of a small dataset, the algorithm might be trained poorly if the number of folds is small. Thus, a higher number of folds (or a higher k value) is closer to optimum when the dataset is small in size. As the reference dataset size grows, lesser number of folds can be tried given that the algorithm can be trained well.

6.7.3 Error Typologies and Annotation

The annotations throughout the experiment were done by a single annotator. Even though inter-annotation inconsistency is a constant problem, the annotations done by a single author are even more prone to errors. While the annotations were checked multiple times, the possibility of annotation inconsistencies in manual annotation for error labelling cannot be discounted.

It is very likely for a single dependency arc to have an error that is defined separately under different labels. In such an event, the primary source of error was labelled as the error type. For example, if a dependency arc has *Case Error* as well as *Wrong Head* error, the former is very likely being caused by the latter. Therefore, the manual annotation for this instance would list it as a case of *Wrong Head*.

Under the different head identification errors, the annotation was in the following order of priority, arranged in descending order:

1. `MWE Error` or `Naming Error`
2. `Tree Error`
3. `Wrong Head`

In essence, if the head identification error could not be localised to a specific error type, it was labelled under the umbrella error label of `Wrong Head`.

6.7.4 Conclusion

In the experiment, we narrowed the search scope from the bins as used by Alzetta et al. [2017] to focus on the arcs that were considered as improbable by the algorithm. Additionally, we found that using cross-validation to train the algorithm has no significant performance gain.

For low-resource languages with little to no reference corpus data, we tried cross-validation approach for finding the errors. We discovered that the choice of folds in cross-validation strategy is determined by the size of the reference corpus; and in case of unavailability of one, the strategy can be used on the data itself without a significant loss in the error-detection rate.

7. AUX vs. VERB: Attempt at Separation of Verbs and Auxiliary Verbs (Experiment 4)

We earlier mentioned in Section 2.2.2 how the line of distinction between verbs (POS tag VERB) and auxiliary verbs (POS tag AUX) is not well-defined. We shall treat this problem in this section, with a glance through some of the observations on the problem in Section 7.1, followed by the definition of the working dataset in Section 7.2. We elaborate on the proposed solution to the problem, and the results of the experiment in Section 7.3 and 7.4 respectively. We finally conclude this section with a discussion of the results in 7.5.

7.1 Observations About Problem Statement

According to the definition in UD[1], AUX is used as a common POS tag for verbal auxiliaries, as well as non-verbal TAME markers. The class of copulas are also included in this list.

This definition of auxiliaries is a bit different from Shopen [2007] which separates the two classes of auxiliaries and copulas in different categories. The work also points out the correlation between the position of an inflected auxiliary in relation to the verb, and other word properties of the language, as first pointed by Greenberg [1963]. In his work, Greenberg notes that the position of an inflected auxiliary in relation to the verb is generally the same as the position of verb in relation to an object. It is important to note that this generalization only holds for the inflected auxiliaries, and thus languages where the auxiliaries are not inflected are automatically ruled out from the consideration. Shopen points out the well-known exception to this generalization in case of verb-second languages like those of de.

While the generalization made by Greenberg is a very good marker for possible identification of inflected auxiliaries, the requirement of identification of auxiliaries in noninflected form still remains as a problem. This problem can however, be mitigated in part by the usage of the list of tokens identified as auxiliary in a given language, as was started in UDv2.4 [Nivre et al., 2019] with the help of a validator (cf. Level 5 checks in validate.py[2] file). It must also be pointed out that since Greenberg did not extend this generalization to SVO languages, the generalization only holds for languages with VSO and SOV dominant word-order. Combining that with verb-second languages, the generalization can not be used globally across all the languages.

When the copulas are included in the definition of AUX, the already difficult problem of separating AUX and VERB becomes even harder. In many languages, auxiliaries are a subset of verbs, with respect to specific usages. In other words, the same token can act as a verb or an auxiliary, depending upon the usage. The

[1] https://universaldependencies.org/kpv/pos/AUX_.html
[2] https://github.com/UniversalDependencies/tools

list of copula in many languages is also a subset of verbs, called as copulative verbs. However, as Shopen notes, there are cases of languages where the copula are not verbal in nature. The function of a copula can be realized by other means as well. The most common of these, viz. juxtaposition (example language- Ilocano), and use of predicators (example language- Bambara) are listed in the work, where they may be combined with existing copulative verbs in the grammar of the language.

In essence, while the class `AUX` in UD includes the copulative verbs, predicators, and other non-verbal TAME markers, the class `VERB` is composed of open class categories of verbs.

7.2 Dataset Definition

This experiment uses `hi`-HDTB treebank from UDv2.4 [Nivre et al., 2019].

There are a few reasons for the choice of the language for the experiment. In `hi`, we can more often than not draw a clear line of distinction between auxiliary as defined by UD, and the verbs. While the auxiliaries undergo inflection, and also include predicators and other TAME markers, they are restricted to a few tokens which rarely, if at all, are used as independent verbs. The factors as listed above, combined with the author's native fluency in the language makes it an ideal candidate for this experiment.

7.3 Experiment

We approach the problem at hand as a classification problem. In the experiment, we create a classifier that tags the data in categories of whether a particular token is an instance of `AUX`, `VERB` or neither of the two. Since the training data needs to contain the information on what instances to mark in either category and also differentiate tokens not marked as either UPOS tag, we label the data using Named-Entity Recognition (NER) task tag format. The classifier we described above is available as off-the-shelf tool for NER task, and that is the reason the data was labelled using NER task tags. As the classifier predicts the output label for each token, it also outputs a confidence score associated with each predicted label. By analysing the confidence score of each prediction and comparing it with the already annotated data, we should be able to point out the anomalies.

If we consider the gold-standard (GS) as erroneous as in present case, we need some data in a higher quality of annotation. A platinum standard is considered as a super-refined gold standard from which even the GS can be evaluated and verified. However, given a lack of such platinum standard, we restrict to a manual evaluation of the output of the classifier, using k-fold cross validation technique to test and train the classifier on the same data. We first split the data into 10 folds, and then proceed to label the data using NER format.

Between the two tagsets available for NER labelling, we choose IOBES format for the classification of the data in the following manner: All the instances marked as `AUX` are labelled as "S-aux", and all the instances marked as `VERB` are labelled as "S-verb". The rest of the tokens are labelled with 'O' tag. We do not consider contiguous tokens as a continuous chain, and thus not use either of 'I', 'B' or 'E'

tags at all. This is also done so as to have better control over each token that the model learns to tag, thereby increasing the granularity of the data.

For the task of POS Tagging, Flair embeddings [Akbik et al., 2018] were the state-of-the-art (SOTA) at the time of performing this experiment. The embeddings were shown to be outperform several models available at the time, across multiple NLP tasks, and therefore were the natural choice for this experiment. However, there are several hyper-parameters that can be tuned with respect to the models. We decided to tune the hyper-parameters with their corresponding choices as listed in Table 7.1. The best choice for the hyper-parameter are also listed in the same table.

Hyper-Parameter	**Choices**	**Tuned Value**
Embeddings	**Stack1:** Forward and Backward Flair Embedding trained on `hi`-newswire **Stack2:** Word Embedding for `hi`, Forward and Backward Flair Embedding trained on `hi`-newswire	Stack2
Use CRF?	True, False	True
Use RNN?	True, False	True
RNN Layers	1, 2, 4	2
Size of Hidden Layer	32, 64, 128, 256	256
Dropout	Uniform Distribution in [0.0, 0.5]	0.25
Learning Rate	0.05, 0.1, 0.15, 0.2, 0.25	0.1

Table 7.1: Hyper-Parameters for Neural Network

With the optimized parameters, we train the model on each fold of the data, and test the trained model on the fold's test data. As mentioned earlier, the predicted output labels are accompanied with an associated confidence score that demonstrates the model's confidence in the predicted label. We here identify 6 categories of error patterns, based on the predicted label and the original label for the data, as listed in Table 7.2.

Category	**Original**	**Prediction**
aux_TP	S-aux	S-aux
O_TP	O	O
verb_TP	S-verb	S-verb
aux-O	O	S-aux
	S-aux	O
aux-verb	S-aux	S-verb
	S-verb	S-aux
verb-O	O	S-verb
	S-verb	O

Table 7.2: Categories of Error Patterns

The associated confidence scores for each prediction can be used to detect

the anomaly from what is labelled as per the original annotation, and what the classifier thinks should be the annotation label. For the cases where the original annotation is same as the classifier's prediction, we focus on the subset of the predictions where the confidence score is lower than 0.67. The idea is that since there are 3 categories, a prediction with the associated confidence lower than $\frac{2}{3}$ might be erroneous. For the instances where there is a mismatch between the predicted label and the originally annotated label, we focus on instances with the confidence in prediction higher than 0.995. The idea in this case is that if the model is really sure about the prediction, the original annotation might be erroneous, and is worth looking into. Figure 7.1 shows the distribution of confidence scores for instances where the predicted label matches the original label, with the associated confidence value lower than 0.80.

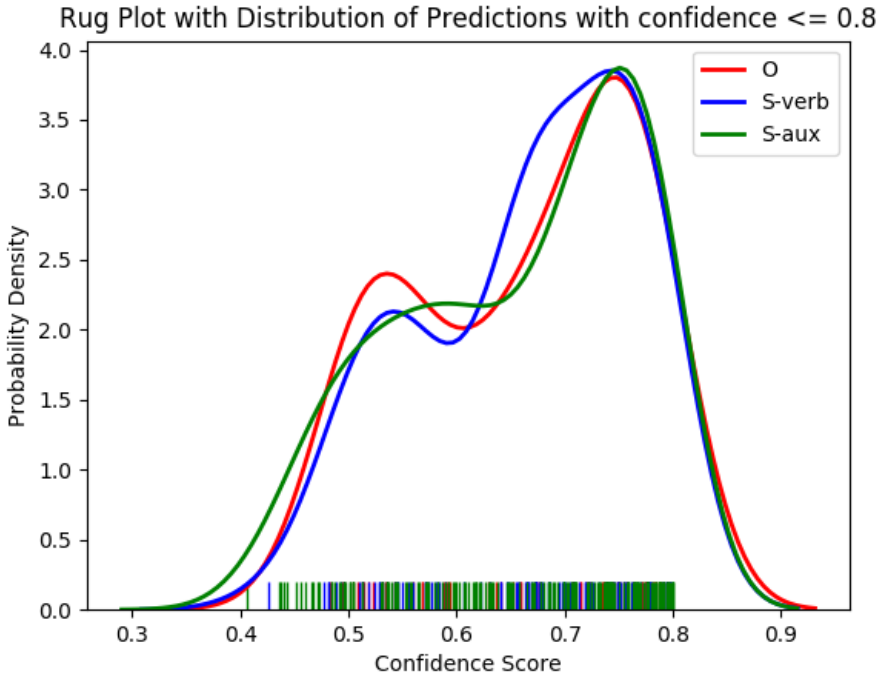


Figure 7.1: Rug plot with Distribution of Predictions with low confidence score

Having identified instances within each category that have confidence scores within the relevant bound, these instances were manually annotated to see which one of the original annotation or the predicted annotation is correct. We can summarize the entire experiment in the form of algorithm as defined in Algorithm 10.

Algorithm 10 Experiment to Identify Mislabelled AUX and VERB tags

Input: *data* ← UDv2.4 treebank

1: Convert *data.train*, *data.test* and *data.dev* to IOBES format
2: $model.config$ ← SOTA Classifier Configuration
3: *data.complete* ← *data.train* + *data.dev* + *data.test*
4: {The different splits of the data concatenated together}
5: *iter.id* ← fold of *data.complete*, numbered as *id*
6: {Performed 10-fold cross-validation to split *data.complete*}
7: *model* ← Classifier with configuration as per $model.config$
8: **for** *id* in {1, ... , 10} **do**
9: *model.id* ← *model* trained on *iter.id.train* data
10: *model.id.test* ← Prediction of *model.id* on *iter.id.test* data
11: **end for**
12: $identified.pure$ ← Original Annotation matches Prediction such that Confidence score ≤ 0.6700
13: $identified.cross$ ← Original Annotation differs from Prediction such that Confidence score ≥ 0.9950
14: Manual Annotation of $identified.pure$ and $identified.cross$

7.4 Results

Given that the experiment is a case of a multi-class classification, the model performance is expressed in form of confusion metrics for each class AUX, VERB along with the metrics like Precision, Recall, Accuracy, F1 Score.

The metrics corresponding to the best performing model on the entire treebank are listed in Table 7.3. The best performing model was trained on training set of UDv2.4 hi-HDTB data, and tuned over the dev set, and tested over the test set. The evaluation presented in the table corresponds to the automatic evaluation over test set.

Label	Precision	Recall	Accuracy	F1 Score
AUX	98.89	99.50	98.40	99.19
VERB	99.32	98.87	98.20	99.09

Table 7.3: Metrics of Best Model trained over UDv2.4 hi-HDTB Treebank. The metrics are reported for automatic evaluation over test data in the treebank.

When the models were trained on each of the folds, keeping the architecture of the best model, there was no loss in performance (metric considered- micro averaged F1 score). This essentially means that the instances corresponding to AUX and VERB are annotated consistently within the treebank. As mentioned in previous section, we focused on the instances of the tagged data with confidence scores in particular bounds, and manually annotated them[3]. Table 7.4 lists the number of instances that were focused on in each category (as defined in Table 7.2). The table also lists the number of instances that were identified as mislabelled, following the manual annotation procedure.

[3]Associated Code and annotated data can be accessed at https://github.com/Akshayanti/Masters-Thesis-CUNI-2020/tree/master/AUX-vs-VERB-UDv2.4

Category	Focused	Mislabelled	Percentage
aux_TP	83	3	3.61
O_TP	25	5	20.00
verb_TP	45	10	22.22
aux-O	10	9	90.00
aux-verb	42	23	54.76
verb-O	20	11	55.00
Overall	225	61	27.11

Table 7.4: Results of Manual Annotation
Note: *_TP is the identifier for instances for which the prediction matches the label. The values in the 'Focused' column refer to count of instances with Confidence Score ≤ 0.6700
Note: X-Y is the identifier for instances which were labelled as X but predicted as Y, or labelled as Y but predicted as X. The values in the 'Focused' column refer to count of instances with Confidence Score ≥ 0.9950

7.5 Discussion of the Results

Metric	Count
Sentences	16 647
Words	351 704
Tagged `AUX`	26 030
Tagged `VERB`	33 753

Table 7.5: Statistics for `hi`-HDTB Treebank

Table 7.5 lists the counts of sentences and the number of `AUX` and `VERB` tags in the entire `hi`-HDTB treebank. Of the total number of tags listed in either category, we are able to focus on just 225 instances where we might be able to identify the problems. Even out of those 225 identified instances, just a bit over 25% are actually erroneous.

While hypertuning the best configuration of the classifier, the parameters correspond to the F1 score on how well it fits to the original data. Essentially, the best performing model is biased in the way that it would always try to find a prediction that matches the original annotation. While there is no other way on how to hypertune the model, the experimental results are therefore liable to find comparatively less instances where the confidence score is within the bounds as considered in the experiment.

Further, the lack of a definable baseline for the attempted solution of the given problem makes it difficult for the current approach to be compared against a benchmark. Considering the lack of benchmark, we can crudely estimate the performance of the experiment by the ratio of the number of errors that were found in the focused cases to the number of instances that were focused on.

While certain patterns are more reliable than others (the case where predicted output doesn't match the original annotation), the overall performance for the experiment is low as can be attributed to different factors mentioned above. Given

the low scout-ability of the error cases in the experiment, the approach used in the experiment is not reliable enough for the process to be automated.

8. Future Work Recommendations

This chapter discusses in brief the other problems that have been recognised within the scope of UD. None of these works mentioned in this chapter have been discussed in the present version of the document. For future researchers interested in tackling more problems with respect to UD, this chapter could be a good point of reference.

8.1 Enhanced Dependencies

Enhanced Dependencies can be understood as an additional layer of annotation of dependencies in UD, which essentially marks added dependencies. Considering some of the restrictions imposed by the regular annotation scheme like a singular head constraint where each node can have only one head, the Enhanced Dependencies aim to cover aspects which can be missed by the regular annotation scheme. However, not all of the languages, or their treebanks have been annotated with the Enhanced Dependencies so far. While the enhanced dependencies have been deemed to be useful in multiple cases (like that of ellipsis, cf. Section 8.2), their full potential might not have been realized so far.

In our experiment on `conj_head` (cf. Chapter 5), we did not work with the problem of conjunction sandwiches. It is very likely that such problems which are difficult to be recognized by the regular dependencies can be searched for rather easily with the Enhanced Dependencies. For example, if Enhanced Dependencies mark all the conjuncts by the `conj` deprel, regardless of whether they are labelled by the deprel in the regular annotation or not, it would allow searching for the available conjuncts rather easily.

We leave it as an open problem for future research to identify cases which are more difficult to handle with regular dependencies, while trying to use Enhanced Dependencies. As an add-on to the task, it can also be tested if some algorithms mentioned in the research can be improved upon or discarded when Enhanced Dependencies are used.

8.2 Ellipsis

The problem with annotation of Elliptical Structures is big enough to warrant a discussion of its own in UD Annotation Guidelines[1,2].

Droganova and Zeman [2017] analyzed the elliptical constructions in UDv2.0 treebanks [Nivre et al., 2017] by principally using `orphan` relations[3] as a way to identify the cases of non-promoted dependents with promoted dependents. While this helps in identifying only a certain number of cases, it fails to identify the cases where the dependents are promoted.

[1] `https://universaldependencies.org/u/overview/enhanced-syntax.html#ellipsis`
[2] `https://universaldependencies.org/u/overview/specific-syntax.html#ellipsis`
[3] `https://universaldependencies.org/u/dep/orphan.html`

In Enhanced Dependencies, `orphan` is replaced by placing a null node to indicate the elided token. However, as discussed earlier, Enhanced Dependencies are not available for all languages or even all treebanks in the same language. Thus, the identification and correction of erroneous elliptical constructions remains a problem that needs to be solved within the scope of basic dependency graphs in UD.

8.3 `FalseNonProjective`: Introduction of False Non-Projectivity into the Annotation

While non-projectivity is a characteristic of some languages, and especially more so of certain genres (poetry, for example); the increasing count of non-projective trees has been shown to affect dependency parsing in a negative way. Owing to semi-automatic conversion scheme, a lot of non-projectivities might also be introduced artificially. Thus, it becomes important to not only identify such cases of false non-projectivities (i.e. the cases which should have been marked as projective, but were annotated as non-projective), but also to remove them as it affects the treebank quality in general.

Note that projectivisation or the act of making a non-projective tree as projective is a different research problem. While projectivisation is primarily aimed at trying to create parsers that can parse non-projective trees efficiently (cf. [Nivre and Nilsson, 2005], [Gómez-Rodríguez et al., 2009], [Hall and Novák, 2005], [Nivre, 2007], among others) and is therefore a parsing problem; `FalseNonProjective` is an erroneous non-projectivity introduced in the annotation where the tree is projective, and has no non-projective variants possible.

8.4 Function Words and Associated deprels

Conjunctions are identified by two POS tags, viz. `SCONJ`, `CCONJ`. The associated dependency relations for the two POS tags are `mark`, and `cc` respectively. While these are the usually associated dependency relations, the boundary between the two is fuzzy. In the sense, it is possible for a token to be marked by `SCONJ`, and have a `cc` dependency relation (similarly for `CCONJ` and `mark`). Added to this are the cases where the tokens marked by another POS tag can act as conjunctions. Consider the following example from `en`-ParTUT (UDv2.3) and the associated tree in Figure 8.1, where `PART` (*to*) acts as a conjunction, and thus the `mark` deprel associated to it.

Example 27. Ukraine's constitutional structure is for Ukraine's citizens alone **to** decide.

Furthermore, both the POS tags in question (`SCONJ`, `CCONJ`) can have other dependency relations attached to them as well. As such, it is difficult (and nonsensical) to limit the deprels for a particular POS tag to occur with a particular deprel (especially in this case). However, there might still be some processes we can observe (and correct). For example, if a particular token occurs more with the `mark` deprel, but is consistently labelled as `CCONJ`, the annotation should be taken a closer look at, and a possible disparity identified.

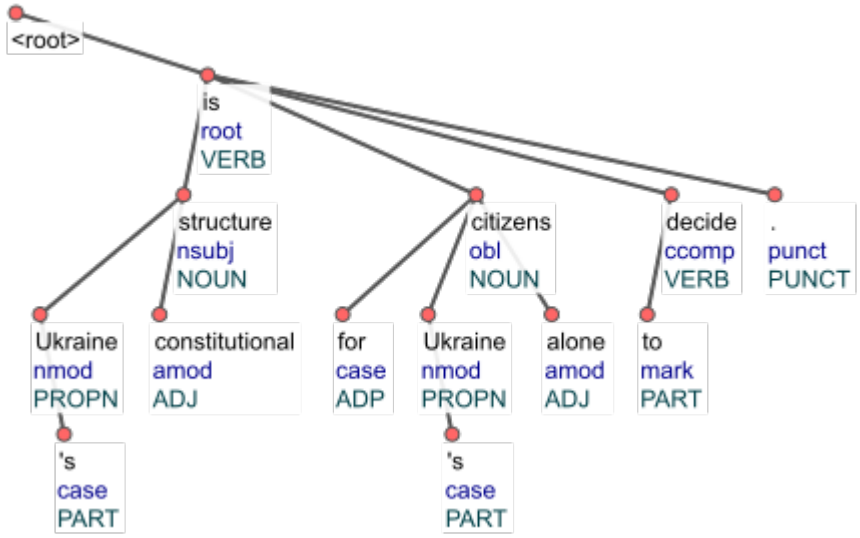
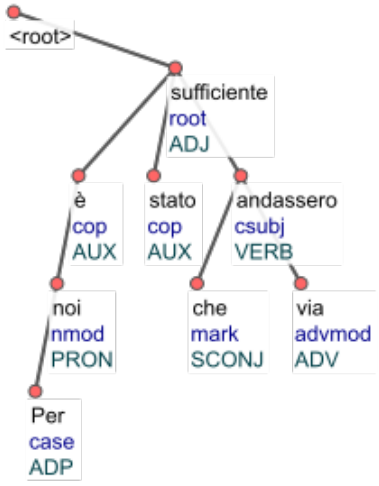


Figure 8.1: Dependency Tree for Example 27 showcasing association of `PART` with `mark` deprel

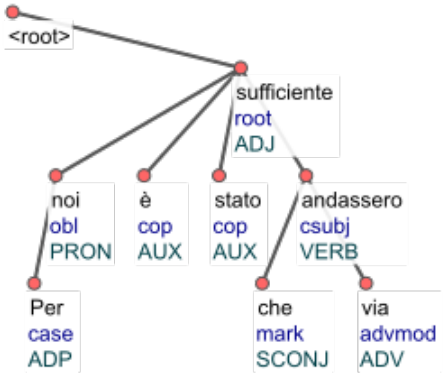
8.5 `auxHead`: Auxiliary as Head of Dependency

In the discussion of this problem, we refer to the case when an auxiliary (marked by either of `AUX` or `aux`) is treated as the head of a dependency relation. Although allowed in certain cases, the auxiliary should not be marked as the dependency head in general sense. Consider the following example in Figure 8.2, taken directly from Alzetta et al. [2017]. The token of interest is marked in bold.

Example 28.
Text (`it`): Per noi **è** stato sufficiente che andassero via
Lit.: For us it-has been enough that they-went away



(a) Original (Incorrect) Annotation



(b) Corrected Annotation

Figure 8.2: Example tree from Alzetta et al. [2017] showcasing `auxHead` error type. In the original example, **noi** (us) was annotated as a dependent of both **Per** (for), and **è** (has-been). Under UD representation, there can not be more than one head for any given node in regular annotation. As such, we believe it was a typo in the publication and not in their data. In this figure, we show the corrected dependencies.

In the figure, notice how the originally incorrect annotation has *è* (has-been) with POS `AUX` serving as a dependency head. Alzetta et al. notice that this particular error, classified as a head identification error, contributes to around 13 % of the total discovered erroneous instances. Since it is difficult to separate and identify the instances marked correctly as `AUX` (cf. Chapter 7 for the experiment on attempt at differentiation between `AUX` and `VERB` tags), the attempt at the solution for this problem was not worked at.

8.6 `nmod4obl`: Confusion of `nmod` and `obl` Relations

In UDv1, `nmod` relation was used for nominals modifying either predicates or other nominals. Following a change in guidelines in v2, the deprel was restricted to modifying nominals. Furthermore, a new relation `obl` (oblique) was introduced for oblique dependents of predicates.

To put it simply, this conversion implied the following in an equation format, where $\texttt{x}_{vi}$ refers to the dependency relation `x` as used in version i of UD treebanks:

$$\boxed{\texttt{nmod}_{v1} = \texttt{nmod}_{v2} \cup \texttt{obl}}$$

Depending on the parent node, the relations were modified as follows:

1. If the parent node was a verb, the deprel was changed from `nmod` to `obl`.
2. If the parent was a nominal predicate, the deprel could be either of `nmod` or `obl`, depending on if only the nominal was being modified, or the whole clause was being modified.

3. If the parent was a nominal, but not a nominal predicate, there was no change in the deprel.

4. If the parent was an adjective or an adverb, the deprel would be changed to `obl`, based on additional conditions.

5. In case none of the above conditions held true, the instance would deserve individual treatment.

The change in definition from UDv1 to UDv2 was the primary cause of the error, as identified in Alzetta et al. [2017]. In the same work, the authors note that this error contributes to around 7 % of total discovered errors in the newspaper section of the Italian UD Treebank (IUDT). In the work, the authors attribute this error pattern to annotation inconsistency internal to the treebank. Although a significantly important error, this is not covered in the scope of the current research. Nonetheless, this is an important error that should be taken care of in future.

8.7 Punctuation

The UD Annotation guidelines on punctuation are simple and straightforward[4]. There are discrepancies when it comes to implementation of the guidelines. Some of them are listed as below:

1. It is difficult to identify the next conjunct in case of missing `CCONJ` and `SCONJ` tags as in case of asyndetic coordination. In such cases, the information about the next conjunct should be deduced semantically in most cases. We saw a similar case in Section 5.1.3 (Example 10 and Figure 5.4) where the next conjunct is not clear, owing to other (more suitable) deprel(s) being used in place of `conj` deprel.

2. Re-attachment of a punctuation node is a problem that goes with the previous instance since it's not always clear at what level the punctuation must attach to.

3. For paired and nested punctuation, different languages use different sets of nested punctuation pairs, specifically with respect to quotation marks. As such, the treatment of paired punctuation pairs needs to be handled in a language-specific manner.

The `fixpunct.py` block in Udapi-python [Popel et al., 2017] tries to take care of significant number of edge cases in different UD treebanks. However, a more concrete solution is needed for the problems aforementioned.

[4] `https://universaldependencies.org/u/overview/specific-syntax.html#punctuation`

8.8 Unspecified Dependencies - `dep` deprel

According to the UD definition of `dep` deprel[5], the deprel is reserved for cases when a more precise relation cannot be found. This can be either owing to the sentence splitting in treebanks of some languages, or owing to the limitation in parsing software. Nonetheless, the relation should be avoided as much as possible.

Noticing that some treebanks follow sentence splits where the parts of sentences might be labelled as different sentences (as in the example of a list), the deprel in question is more liable to be used in such instances. However, looking at the data in UDv2.4, some languages have more than 1% of the tokens marked with such relation (Examples being `ko`, `ur`, `ja`-BCCWJ, `it`-PoSTWITA, `hi`-HDTB, `gl`-CTG, `cs`-PDT, among others). While these might be all true positives in other languages, a significantly higher count of `dep` is more troublesome and is less likely to be all true positives in such cases.

An experiment can be performed on such instances where the data without any `dep` deprel is used as a training set to parse the instances with the deprel in question and then the results verified. Nonetheless, the cases of tokens marked with deprel in question need to be reduced in some languages. As such, we leave it as a problem for future researchers to tackle.

[5] `https://universaldependencies.org/u/dep/dep.html`

Conclusion

Although the official title of the research seeks to deal with inconsistencies, this work covers aspects from both error detection and correction (Chapter 5 on `conj_head`), and inconsistency detection and correction (Chapters 4, 6 and 7).

In Chapter 4, we proposed a metric to attest the POS annotation consistency across treebanks that allegedly follow the same guidelines, for the same language. Through the use of the metric, we sought to answer how the different treebanks of a language, with variable size and genre distributions but following the same annotation guidelines, can be compared against each other. We also defined a reliable threshold on the proposed metric that would inform the annotators if the treebanks being compared are consistent in their annotation, or not. The metric was employed in the scope of UDv2.5 [Zeman et al., 2019] data to compare the different treebanks of different languages, and highlighting the ones that are inconsistent in their annotation. To the best of our knowledge, this is the first such metric that compares the treebanks directly, without an added variable of tagger or parser performance.

In Chapter 5, we revisited the error type identified by Alzetta et al. [2017] regarding the identification and correction of the head of a coordinating conjunction, referred to as `conj_head`. We identified the different facets that would come in the way of solving the problem, and proposed solutions for them. The effectiveness of the proposed solutions was demonstrated on languages belonging to different language families, followed by an identification of the cases where the proposed solutions do not work as expected. While the experiments were done primarily on right-headed languages, the approach is extensible to left-headed languages as well, but not to languages with a mix of left and right-headedness[6].

Chapter 6 focused on the LISCA algorithm, proposed by Dell'Orletta et al. [2013]. The algorithm needs a reference corpus to identify the inconsistencies in a treebank, based on the model framed off reference corpus. We checked the viability of the algorithm in a low-resource setting when there might not be a reference corpus to train the algorithm. We also investigated if the search space for the inconsistent arcs could be narrowed without a significant decrease in performance of the algorithm. Marking cross-validation technique as a viable option for the low-resource setting, we further examined the effect of the number of folds in k-fold cross-validation on the error mining process employing LISCA. A typology of different errors as identified in the experiment were also listed.

The experiment in Chapter 7 sought to address the issue of drawing a line of distinction between `AUX` and `VERB` categories in an automatic manner. We employed an automatic classification technique to separate the individual tokens as belonging to either of `AUX` or `VERB`, or neither. While there was a small subset of instances that could be identified, the lower success rate of distinction between the two categories highlighted the challenging nature of the task and that the problem presents much room for improvement.

As the cost of storage falls lower, the size of the treebanks will increase. Essentially, at one point it might be impossible for human annotators to be part of the error-identification and error-correction process for the entire treebank.

[6]For details on right-headed and left-headed languages, refer to Section 5.1.1

The current work is primarily aimed at finding the methods that don't need human annotators in the pipeline, and can be relied upon to fix the errors across different languages in a reliable manner. The research has been in some aspect successful at that front.

One major advantage of an iterative process, with respect to UD treebanks, is how individual error types can be focused on in each iteration. With the UD validator (cf. Level 5 checks in `validate.py`[7] file) identifying and notifying the development teams of the individual errors, the process no longer suffers from a cold start problem. There is a high chance that with upcoming iterations, more and more of the experiments discussed in the document would be rendered obsolete for new treebanks, but they are still necessary to fix the issues in the present treebanks.

It is important to note here that the different problems listed in this thesis document rarely occur in isolation. More often than not, many of the problems are intertwined with each other, resulting in error propagation. Having said that, the corrections are also propagated in a similar fashion, whereby finding and correcting the right error solves multiple intertwined issues at the same time. Consider the example of experiment on `conj_head` (in Chapter 5). Correction of this error instance in the specific case of `eu` also corrected the case of falsely annotated non-projectivities in the trees.

Of the problems mentioned in the chapter titled 'Future Work Recommendations' (Chapter 8), there are some that were not worked on at all in the current research and are left for future researchers (For example, `nmod4obl` in Section 8.6). Additionally, some other problems are still being worked on, and thus do not fall into the scope of the current thesis document (For example, `FalseNonProjective` and `auxHead` in Sections 8.3 and 8.5 respectively).

The author hopes that future researchers will be able to tackle the problems listed in this thesis in a greater capacity, and improve upon the methods already discussed in this research wherever possible.

[7] `https://github.com/UniversalDependencies/tools`

Bibliography

Bhasha Agrawal, Rahul Agarwal, Samar Husain, and Dipti M. Sharma. An Automatic Approach to Treebank Error Detection Using a Dependency Parser. In *International Conference on Intelligent Text Processing and Computational Linguistics*, volume 7816, pages 294–303, 03 2013. doi: 10.1007/978-3-642-37247-6_24. URL `https://doi.org/10.1007/978-3-642-37247-6_24`.

Alan Akbik, Duncan Blythe, and Roland Vollgraf. Contextual String Embeddings for Sequence Labeling. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1638–1649, Santa Fe, New Mexico, USA, August 2018. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/C18-1139`.

Héctor Martínez Alonso and Daniel Zeman. Universal Dependencies for the AnCora treebanks. *Procesamiento del Lenguaje Natural*, (57):91–98, 2016. ISSN 1135-5948.

Héctor Martínez Alonso, Željko Agić, Barbara Plank, and Anders Søgaard. Parsing Universal Dependencies without training. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 230–240, Valencia, Spain, April 2017. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/E17-1022`.

Chiara Alzetta, Felice Dell'Orletta, Simonetta Montemagni, and Giulia Venturi. Dangerous Relations in Dependency Treebanks. In *Proceedings of the 16th International Workshop on Treebanks and Linguistic Theories*, pages 201–210, Prague, Czech Republic, 2017. URL `https://www.aclweb.org/anthology/W17-7624`.

Chiara Alzetta, Felice Dell'Orletta, Simonetta Montemagni, and Giulia Venturi. Universal Dependencies and Quantitative Typological Trends. A Case Study on Word Order. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan, May 2018. European Language Resources Association (ELRA). URL `https://www.aclweb.org/anthology/L18-1719`.

Bharat Ram Ambati, Rahul Agarwal, Mridul Gupta, Samar Husain, and Dipti Misra Sharma. Error Detection for Treebank Validation. In *Proceedings of the 9th Workshop on Asian Language Resources*, pages 23–30, Chiang Mai, Thailand, November 2011. Asian Federation of Natural Language Processing. URL `https://www.aclweb.org/anthology/W11-3405`.

Collin F. Baker, Charles J. Fillmore, and John B. Lowe. The Berkeley FrameNet Project. In *36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics, Volume 1*, pages 86–90, Montreal, Quebec, Canada, August 1998. Association for Computational Linguistics. doi: 10.3115/980845.980860. URL `https://www.aclweb.org/anthology/P98-1013`.

Douglas Biber. A typology of English texts. *Linguistics*, 27(1):3–44, 1989. ISSN 0024-3949. doi: 10.1515/ling.1989.27.1.3.

Douglas Biber. *Variation across speech and writing*. Cambridge University Press, 1991.

Douglas Biber. *Dimensions of Register Variation: A Cross-Linguistic Comparison*. Cambridge University Press, 1995.

Winfried Boeder. The South Caucasian languages. *Lingua*, 115(1): 5–89, 2005. ISSN 0024-3841. doi: https://doi.org/10.1016/j.lingua.2003.06.002. URL http://www.sciencedirect.com/science/article/pii/S0024384103001244.

Alena Böhmová, Jan Hajič, Eva Hajičová, and Barbora Hladká. *The Prague Dependency Treebank*, pages 103–127. Springer Netherlands, Dordrecht, 2003. ISBN 978-94-010-0201-1. doi: 10.1007/978-94-010-0201-1_7. URL https://doi.org/10.1007/978-94-010-0201-1_7.

Adriane Boyd, Markus Dickinson, and W. Detmar Meurers. On Detecting Errors in Dependency Treebanks. *Research on Language and Computation*, 6(2):113–137, 2008. doi: 10.1007/s11168-008-9051-9. URL https://doi.org/10.1007/s11168-008-9051-9.

Sabine Buchholz and Erwin Marsi. CoNLL-X Shared Task on Multilingual Dependency Parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 149–164, New York City, June 2006. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/W06-2920.

Ricardo Casañ-Pitarch. A Proposal for Genre Analysis: The AMS model. In Chelo Vargas-Sierra, editor, *Professional and Academic Discourse: an Interdisciplinary Perspective*, volume 2 of *EPiC Series in Language and Linguistics*, pages 235–246. EasyChair, 2017. doi: 10.29007/hbg9. URL https://easychair.org/publications/paper/b6rp.

Wanxiang Che, Yijia Liu, Yuxuan Wang, Bo Zheng, and Ting Liu. Towards Better UD Parsing: Deep Contextualized Word Embeddings, Ensemble, and Treebank Concatenation. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 55–64, Brussels, Belgium, October 2018. Association for Computational Linguistics. doi: 10.18653/v1/K18-2005. URL https://www.aclweb.org/anthology/K18-2005.

Viacheslav Chirikba. Evidential category and evidential strategy in Abkhaz. *Typological Studies in Language*, 54:243–272, 2003. URL https://benjamins.com/catalog/tsl.54.14chi.

Jayeol Chun, Na-Rae Han, Jena D. Hwang, and Jinho D. Choi. Building Universal Dependency Treebanks in Korean. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, pages 2194–2202, Miyazaki, Japan, May 2018. European Language Resources Association (ELRA). URL https://www.aclweb.org/anthology/L18-1347.

Daniël de Kok, Jianqiang Ma, and Gertjan van Noord. A generalized method for iterative error mining in parsing results. In *Proceedings of the 2009 Workshop on Grammar Engineering Across Frameworks (GEAF 2009)*, pages 71–79, Suntec, Singapore, August 2009. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/W09-2609.

Miryam de Lhoneux and Joakim Nivre. Should Have, Would Have, Could Have. Investigating Verb Group Representations for Parsing with Universal Dependencies. In *Proceedings of the Workshop on Multilingual and Cross-lingual Methods in NLP*, pages 10–19, San Diego, California, June 2016. Association for Computational Linguistics. doi: 10.18653/v1/W16-1202. URL https://www.aclweb.org/anthology/W16-1202.

Marie-Catherine de Marneffe, Miriam Connor, Natalia Silveira, Samuel R. Bowman, Timothy Dozat, and Christopher D. Manning. More Constructions, More Genres: Extending Stanford Dependencies. In *Proceedings of the Second International Conference on Dependency Linguistics (DepLing 2013)*, pages 187–196, Prague, Czech Republic, August 2013. Charles University in Prague, Matfyzpress, Prague, Czech Republic. URL https://www.aclweb.org/anthology/W13-3721.

Marie-Catherine de Marneffe, Timothy Dozat, Natalia Silveira, Katri Haverinen, Filip Ginter, Joakim Nivre, and Christopher D. Manning. Universal Stanford dependencies: A cross-linguistic typology. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC-2014)*, pages 4585–4592, Reykjavik, Iceland, May 2014. European Languages Resources Association (ELRA). URL http://www.lrec-conf.org/proceedings/lrec2014/pdf/1062_Paper.pdf.

Marie-Catherine de Marneffe, Matias Grioni, Jenna Kanerva, and Filip Ginter. Assessing the Annotation Consistency of the Universal Dependencies Corpora. In *Proceedings of the Fourth International Conference on Dependency Linguistics (Depling 2017)*, pages 108–115, Pisa, Italy, September 2017. Linköping University Electronic Press. URL https://www.aclweb.org/anthology/W17-6514.

Felice Dell'Orletta, Giulia Venturi, and Simonetta Montemagni. Linguistically-driven Selection of Correct Arcs for Dependency Parsing. *Computación y Sistemas*, 17(2):125–136, 2013.

Markus Dickinson and W. Detmar Meurers. Detecting Inconsistencies in Treebanks. *IEEE Transactions on Learning Technologies - TLT*, 01 2003a.

Markus Dickinson and W. Detmar Meurers. Detecting Errors in Part-of-speech Annotation. In *Proceedings of the Tenth Conference on European Chapter of the Association for Computational Linguistics - Volume 1*, EACL '03, pages 107–114, Stroudsburg, PA, USA, 2003b. Association for Computational Linguistics. ISBN 1-333-56789-0. doi: 10.3115/1067807.1067823. URL https://doi.org/10.3115/1067807.1067823.

Markus Dickinson and W. Detmar Meurers. Detecting Errors in Discontinuous Structural Annotation. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pages 322–329, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics. doi: 10.3115/1219840.1219880. URL https://doi.org/10.3115/1219840.1219880.

Kira Droganova and Daniel Zeman. Elliptic Constructions: Spotting Patterns in UD Treebanks. In *Proceedings of the NoDaLiDa 2017 Workshop on Universal Dependencies (UDW 2017)*, pages 48–57, Gothenburg, Sweden, May 2017. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/W17-0406.

Kira Droganova, Olga Lyashevskaya, and Daniel Zeman. Data Conversion and Consistency of Monolingual Corpora: Russian UD Treebanks. In *Proceedings of the 17th International Workshop on Treebanks and Linguistic Theories (TLT 2018)*, number 155, pages 53–66, Linköping, Sweden, 2018. Linköping University Electronic Press. ISBN 978-91-7685-137-1.

Karën Fort and Benoît Sagot. Influence of Pre-Annotation on POS-Tagged Corpus Development. In *Proceedings of the Fourth Linguistic Annotation Workshop*, page 56–63, Uppsala, Sweden, July 2010. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/W10-1807.

Carlos Gómez-Rodríguez, David Weir, and John Carroll. Parsing Mildly Non-Projective Dpendency Structures. In *Proceedings of the 12th Conference of the European Chapter of the ACL (EACL 2009)*, pages 291–299, Athens, Greece, March 2009. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/E09-1034.

Joseph H Greenberg. Some Universals of Grammar with Particular Reference to the Order of Meaningful Elements. *Universals of Language*, 2:73–113, 1963. URL http://hdl.handle.net/11707/78.

Keith Hall and Václav Novák. Corrective Modeling for Non-Projective Dependency Parsing. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 42–52, Vancouver, British Columbia, October 2005. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/W05-1505.

Jiří Havelka. Beyond Projectivity: Multilingual Evaluation of Constraints and Measures on Non-Projective Structures. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 608–615, Prague, Czech Republic, June 2007. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/P07-1077.

Francis Heylighen and Jean-Marc Dewaele. Formality of Language: definition, measurement and behavioral determinants. *Interner Bericht, Center "Leo Apostel", Vrije Universiteit Brüssel*, 4, 1999.

Tuomo Kakkonen. Dependency treebanks: methods, annotation schemes and tools. In *Proceedings of the 15th Nordic Conference of Computational Linguistics (NODALIDA 2005)*, pages 94–104, Joensuu, Finland, May 2006. University of Joensuu, Finland. URL https://www.aclweb.org/anthology/W05-1714.

Prafulla Kalapatap, N. N. Tejas, Siddharth Dalmia, Prakhar Gupta, Bhaswant Inguva, and Aruna Malapati. A Novel Similarity Measure: Voronoi Audio Similarity for Genre Classification. *International Journal of Intelligent Systems Technologies and Applications*, 16(4):309–318, January 2017. ISSN 1740-8865. doi: 10.1504/IJISTA.2017.088054. URL https://doi.org/10.1504/IJISTA.2017.088054.

Francesco Mambrini and Marco Passarotti. Non-Projectivity in the Ancient Greek Dependency Treebank. In *Proceedings of the Second International Conference on Dependency Linguistics (DepLing 2013)*, pages 177–186, Prague, Czech Republic, August 2013. Charles University in Prague, Matfyzpress, Prague, Czech Republic. URL https://www.aclweb.org/anthology/W13-3720.

Mitchell Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. The Penn Treebank: Annotating Predicate Argument Structure. In *Proceedings of the Workshop on Human Language Technology*, HLT '94, page 114–119, USA, 1994. Association for Computational Linguistics. ISBN 1558603573. doi: 10.3115/1075812.1075835. URL https://doi.org/10.3115/1075812.1075835.

Ryan McDonald, Joakim Nivre, Yvonne Quirmbach-Brundage, Yoav Goldberg, Dipanjan Das, Kuzman Ganchev, Keith Hall, Slav Petrov, Hao Zhang, Oscar Täckström, Claudia Bedini, Núria Bertomeu Castelló, and Jungmee Lee. Universal Dependency Annotation for Multilingual Parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 92–97, Sofia, Bulgaria, August 2013. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/P13-2017.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer Sentinel Mixture Models. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=Byj72udxe.

Alejandro Mosquera and Paloma Moreda Pozo. The Use of Metrics for Measuring Informality Levels in Web 2.0 Texts. In *Proceedings of the 8th Brazilian Symposium in Information and Human Language Technology*, 2011. URL https://www.aclweb.org/anthology/W11-4523.

Joakim Nivre. Incremental Non-Projective Dependency Parsing. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*, pages 396–403, Rochester, New York, April 2007. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/N07-1050.

Joakim Nivre and Chiao-Ting Fang. Universal Dependency Evaluation. In *Proceedings of the NoDaLiDa 2017 Workshop on Universal Dependencies (UDW 2017)*, pages 86–95, Gothenburg, Sweden, May 2017. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/W17-0411`.

Joakim Nivre and Jens Nilsson. Pseudo-Projective Dependency Parsing. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 99–106, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics. doi: 10.3115/1219840.1219853. URL `https://www.aclweb.org/anthology/P05-1013`.

Joakim Nivre, Cristina Bosco, Jinho Choi, Marie-Catherine de Marneffe, Timothy Dozat, Richárd Farkas, Jennifer Foster, Filip Ginter, Yoav Goldberg, Jan Hajič, Jenna Kanerva, Veronika Laippala, Alessandro Lenci, Teresa Lynn, Christopher Manning, Ryan McDonald, Anna Missilä, Simonetta Montemagni, Slav Petrov, Sampo Pyysalo, Natalia Silveira, Maria Simi, Aaron Smith, Reut Tsarfaty, Veronika Vincze, and Daniel Zeman. Universal Dependencies 1.0, 2015. URL `http://hdl.handle.net/11234/1-1464`. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Joakim Nivre, Željko Agić, Lars Ahrenberg, Maria Jesus Aranzabe, Masayuki Asahara, Aitziber Atutxa, Miguel Ballesteros, John Bauer, Kepa Bengoetxea, Yevgeni Berzak, Riyaz Ahmad Bhat, Cristina Bosco, Gosse Bouma, Sam Bowman, Gülşen Cebiroğlu Eryiğit, Giuseppe G. A. Celano, Çağrı Çöltekin, Miriam Connor, Marie-Catherine de Marneffe, Arantza Diaz de Ilarraza, Kaja Dobrovoljc, Timothy Dozat, Kira Droganova, Tomaž Erjavec, Richárd Farkas, Jennifer Foster, Daniel Galbraith, Sebastian Garza, Filip Ginter, Iakes Goenaga, Koldo Gojenola, Memduh Gokirmak, Yoav Goldberg, Xavier Gómez Guinovart, Berta Gonzáles Saavedra, Normunds Grūzītis, Bruno Guillaume, Jan Hajič, Dag Haug, Barbora Hladká, Radu Ion, Elena Irimia, Anders Johannsen, Hüner Kaşıkara, Hiroshi Kanayama, Jenna Kanerva, Boris Katz, Jessica Kenney, Simon Krek, Veronika Laippala, Lucia Lam, Alessandro Lenci, Nikola Ljubešić, Olga Lyashevskaya, Teresa Lynn, Aibek Makazhanov, Christopher Manning, Cătălina Mărănduc, David Mareček, Héctor Martínez Alonso, Jan Mašek, Yuji Matsumoto, Ryan McDonald, Anna Missilä, Verginica Mititelu, Yusuke Miyao, Simonetta Montemagni, Keiko Sophie Mori, Shunsuke Mori, Kadri Muischnek, Nina Mustafina, Kaili Müürisep, Vitaly Nikolaev, Hanna Nurmi, Petya Osenova, Lilja Øvrelid, Elena Pascual, Marco Passarotti, Cenel-Augusto Perez, Slav Petrov, Jussi Piitulainen, Barbara Plank, Martin Popel, Lauma Pretkalniņa, Prokopis Prokopidis, Tiina Puolakainen, Sampo Pyysalo, Loganathan Ramasamy, Laura Rituma, Rudolf Rosa, Shadi Saleh, Baiba Saulīte, Sebastian Schuster, Wolfgang Seeker, Mojgan Seraji, Lena Shakurova, Mo Shen, Natalia Silveira, Maria Simi, Radu Simionescu, Katalin Simkó, Kiril Simov, Aaron Smith, Carolyn Spadine, Alane Suhr, Umut Sulubacak, Zsolt Szántó, Takaaki Tanaka, Reut Tsarfaty, Francis Tyers, Sumire Uematsu, Larraitz Uria, Gertjan van Noord, Viktor Varga, Veronika Vincze, Jing Xian Wang, Jonathan North Washington, Zdeněk Žabokrtský, Daniel Zeman, and Hanzhi Zhu. Universal Dependencies 1.3, 2016. URL `http:`

//hdl.handle.net/11234/1-1699. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Joakim Nivre, Željko Agić, Lars Ahrenberg, Maria Jesus Aranzabe, Masayuki Asahara, Aitziber Atutxa, Miguel Ballesteros, John Bauer, Kepa Bengoetxea, Riyaz Ahmad Bhat, Eckhard Bick, Cristina Bosco, Gosse Bouma, Sam Bowman, Marie Candito, Gülşen Cebiroğlu Eryiğit, Giuseppe G. A. Celano, Fabricio Chalub, Jinho Choi, Çağrı Çöltekin, Miriam Connor, Elizabeth Davidson, Marie-Catherine de Marneffe, Valeria de Paiva, Arantza Diaz de Ilarraza, Kaja Dobrovoljc, Timothy Dozat, Kira Droganova, Puneet Dwivedi, Marhaba Eli, Tomaž Erjavec, Richárd Farkas, Jennifer Foster, Cláudia Freitas, Katarína Gajdošová, Daniel Galbraith, Marcos Garcia, Filip Ginter, Iakes Goenaga, Koldo Gojenola, Memduh Gökırmak, Yoav Goldberg, Xavier Gómez Guinovart, Berta Gonzáles Saavedra, Matias Grioni, Normunds Grūzītis, Bruno Guillaume, Nizar Habash, Jan Hajič, Linh Hà Mỹ, Dag Haug, Barbora Hladká, Petter Hohle, Radu Ion, Elena Irimia, Anders Johannsen, Fredrik Jørgensen, Hüner Kaşıkara, Hiroshi Kanayama, Jenna Kanerva, Natalia Kotsyba, Simon Krek, Veronika Laippala, Phương Lê Hồng, Alessandro Lenci, Nikola Ljubešić, Olga Lyashevskaya, Teresa Lynn, Aibek Makazhanov, Christopher Manning, Cătălina Mărănduc, David Mareček, Héctor Martínez Alonso, André Martins, Jan Mašek, Yuji Matsumoto, Ryan McDonald, Anna Missilä, Verginica Mititelu, Yusuke Miyao, Simonetta Montemagni, Amir More, Shunsuke Mori, Bohdan Moskalevskyi, Kadri Muischnek, Nina Mustafina, Kaili Müürisep, Lương Nguyễn Thị, Huyền Nguyễn Thị Minh, Vitaly Nikolaev, Hanna Nurmi, Stina Ojala, Petya Osenova, Lilja Øvrelid, Elena Pascual, Marco Passarotti, Cenel-Augusto Perez, Guy Perrier, Slav Petrov, Jussi Piitulainen, Barbara Plank, Martin Popel, Lauma Pretkalniņa, Prokopis Prokopidis, Tiina Puolakainen, Sampo Pyysalo, Alexandre Rademaker, Loganathan Ramasamy, Livy Real, Laura Rituma, Rudolf Rosa, Shadi Saleh, Manuela Sanguinetti, Baiba Saulīte, Sebastian Schuster, Djamé Seddah, Wolfgang Seeker, Mojgan Seraji, Lena Shakurova, Mo Shen, Dmitry Sichinava, Natalia Silveira, Maria Simi, Radu Simionescu, Katalin Simkó, Mária Šimková, Kiril Simov, Aaron Smith, Alane Suhr, Umut Sulubacak, Zsolt Szántó, Dima Taji, Takaaki Tanaka, Reut Tsarfaty, Francis Tyers, Sumire Uematsu, Larraitz Uria, Gertjan van Noord, Viktor Varga, Veronika Vincze, Jonathan North Washington, Zdeněk Žabokrtský, Amir Zeldes, Daniel Zeman, and Hanzhi Zhu. Universal Dependencies 2.0, 2017. URL http://hdl.handle.net/11234/1-1983. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Joakim Nivre, Mitchell Abrams, Željko Agić, Lars Ahrenberg, Lene Antonsen, Katya Aplonova, Maria Jesus Aranzabe, Gashaw Arutie, Masayuki Asahara, Luma Ateyah, Mohammed Attia, Aitziber Atutxa, Liesbeth Augustinus, Elena Badmaeva, Miguel Ballesteros, Esha Banerjee, Sebastian Bank, Verginica Barbu Mititelu, Victoria Basmov, John Bauer, Sandra Bellato, Kepa Bengoetxea, Yevgeni Berzak, Irshad Ahmad Bhat, Riyaz Ahmad Bhat, Erica Biagetti, Eckhard Bick, Rogier Blokland, Victoria Bobicev, Carl Börstell, Cristina Bosco, Gosse Bouma, Sam Bowman, Adriane Boyd, Aljoscha Bur-

chardt, Marie Candito, Bernard Caron, Gauthier Caron, Gülşen Cebiroğlu Eryiğit, Flavio Massimiliano Cecchini, Giuseppe G. A. Celano, Slavomír Čéplö, Savas Cetin, Fabricio Chalub, Jinho Choi, Yongseok Cho, Jayeol Chun, Silvie Cinková, Aurélie Collomb, Çağrı Çöltekin, Miriam Connor, Marine Courtin, Elizabeth Davidson, Marie-Catherine de Marneffe, Valeria de Paiva, Arantza Diaz de Ilarraza, Carly Dickerson, Peter Dirix, Kaja Dobrovoljc, Timothy Dozat, Kira Droganova, Puneet Dwivedi, Marhaba Eli, Ali Elkahky, Binyam Ephrem, Tomaž Erjavec, Aline Etienne, Richárd Farkas, Hector Fernandez Alcalde, Jennifer Foster, Cláudia Freitas, Katarína Gajdošová, Daniel Galbraith, Marcos Garcia, Moa Gärdenfors, Sebastian Garza, Kim Gerdes, Filip Ginter, Iakes Goenaga, Koldo Gojenola, Memduh Gökırmak, Yoav Goldberg, Xavier Gómez Guinovart, Berta Gonzáles Saavedra, Matias Grioni, Normunds Grūzītis, Bruno Guillaume, Céline Guillot Barbance, Nizar Habash, Jan Hajič, Jan Hajič jr., Linh Hà Mỹ, Na-Rae Han, Kim Harris, Dag Haug, Barbora Hladká, Jaroslava Hlaváčová, Florinel Hociung, Petter Hohle, Jena Hwang, Radu Ion, Elena Irimia, Ọlájídé Ishola, Tomáš Jelínek, Anders Johannsen, Fredrik Jørgensen, Hüner Kaşıkara, Sylvain Kahane, Hiroshi Kanayama, Jenna Kanerva, Boris Katz, Tolga Kayadelen, Jessica Kenney, Václava Kettnerová, Jesse Kirchner, Kamil Kopacewicz, Natalia Kotsyba, Simon Krek, Sookyoung Kwak, Veronika Laippala, Lorenzo Lambertino, Lucia Lam, Tatiana Lando, Septina Dian Larasati, Alexei Lavrentiev, John Lee, Phương Lê Hồng, Alessandro Lenci, Saran Lertpradit, Herman Leung, Cheuk Ying Li, Josie Li, Keying Li, KyungTae Lim, Nikola Ljubešić, Olga Loginova, Olga Lyashevskaya, Teresa Lynn, Vivien Macketanz, Aibek Makazhanov, Michael Mandl, Christopher Manning, Ruli Manurung, Cătălina Mărănduc, David Mareček, Katrin Marheinecke, Héctor Martínez Alonso, André Martins, Jan Mašek, Yuji Matsumoto, Ryan McDonald, Gustavo Mendonça, Niko Miekka, Margarita Misirpashayeva, Anna Missilä, Cătălin Mititelu, Yusuke Miyao, Simonetta Montemagni, Amir More, Laura Moreno Romero, Keiko Sophie Mori, Shinsuke Mori, Bjartur Mortensen, Bohdan Moskalevskyi, Kadri Muischnek, Yugo Murawaki, Kaili Müürisep, Pinkey Nainwani, Juan Ignacio Navarro Horñiacek, Anna Nedoluzhko, Gunta Nešpore Bērzkalne, Lương Nguyễn Thị, Huyền Nguyễn Thị Minh, Vitaly Nikolaev, Rattima Nitisaroj, Hanna Nurmi, Stina Ojala, Adédayọ̀ Olúòkun, Mai Omura, Petya Osenova, Robert Östling, Lilja Øvrelid, Niko Partanen, Elena Pascual, Marco Passarotti, Agnieszka Patejuk, Guilherme Paulino Passos, Siyao Peng, Cenel-Augusto Perez, Guy Perrier, Slav Petrov, Jussi Piitulainen, Emily Pitler, Barbara Plank, Thierry Poibeau, Martin Popel, Lauma Pretkalniņa, Sophie Prévost, Prokopis Prokopidis, Adam Przepiórkowski, Tiina Puolakainen, Sampo Pyysalo, Andriela Rääbis, Alexandre Rademaker, Loganathan Ramasamy, Taraka Rama, Carlos Ramisch, Vinit Ravishankar, Livy Real, Siva Reddy, Georg Rehm, Michael Rießler, Larissa Rinaldi, Laura Rituma, Luisa Rocha, Mykhailo Romanenko, Rudolf Rosa, Davide Rovati, Valentin Roșca, Olga Rudina, Jack Rueter, Shoval Sadde, Benoît Sagot, Shadi Saleh, Tanja Samardžić, Stephanie Samson, Manuela Sanguinetti, Baiba Saulīte, Yanin Sawanakunanon, Nathan Schneider, Sebastian Schuster, Djamé Seddah, Wolfgang Seeker, Mojgan Seraji, Mo Shen, Atsuko Shimada, Muh Shohibussirri, Dmitry Sichinava, Natalia Silveira, Maria Simi, Radu Simionescu, Katalin Simkó, Mária Šimková, Kiril Simov, Aaron

Smith, Isabela Soares Bastos, Carolyn Spadine, Antonio Stella, Milan Straka, Jana Strnadová, Alane Suhr, Umut Sulubacak, Zsolt Szántó, Dima Taji, Yuta Takahashi, Takaaki Tanaka, Isabelle Tellier, Trond Trosterud, Anna Trukhina, Reut Tsarfaty, Francis Tyers, Sumire Uematsu, Zdeňka Urešová, Larraitz Uria, Hans Uszkoreit, Sowmya Vajjala, Daniel van Niekerk, Gertjan van Noord, Viktor Varga, Eric Villemonte de la Clergerie, Veronika Vincze, Lars Wallin, Jing Xian Wang, Jonathan North Washington, Seyi Williams, Mats Wirén, Tsegay Woldemariam, Tak-sum Wong, Chunxiao Yan, Marat M. Yavrumyan, Zhuoran Yu, Zdeněk Žabokrtský, Amir Zeldes, Daniel Zeman, Manying Zhang, and Hanzhi Zhu. Universal Dependencies 2.3, 2018. URL `http://hdl.handle.net/11234/1-2895`. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Joakim Nivre, Mitchell Abrams, Željko Agić, Lars Ahrenberg, Gabrielė Aleksandravičiūtė, Lene Antonsen, Katya Aplonova, Maria Jesus Aranzabe, Gashaw Arutie, Masayuki Asahara, Luma Ateyah, Mohammed Attia, Aitziber Atutxa, Liesbeth Augustinus, Elena Badmaeva, Miguel Ballesteros, Esha Banerjee, Sebastian Bank, Verginica Barbu Mititelu, Victoria Basmov, John Bauer, Sandra Bellato, Kepa Bengoetxea, Yevgeni Berzak, Irshad Ahmad Bhat, Riyaz Ahmad Bhat, Erica Biagetti, Eckhard Bick, Agnė Bielinskienė, Rogier Blokland, Victoria Bobicev, Loïc Boizou, Emanuel Borges Völker, Carl Börstell, Cristina Bosco, Gosse Bouma, Sam Bowman, Adriane Boyd, Kristina Brokaitė, Aljoscha Burchardt, Marie Candito, Bernard Caron, Gauthier Caron, Gülşen Cebiroğlu Eryiğit, Flavio Massimiliano Cecchini, Giuseppe G. A. Celano, Slavomír Čéplö, Savas Cetin, Fabricio Chalub, Jinho Choi, Yongseok Cho, Jayeol Chun, Silvie Cinková, Aurélie Collomb, Çağrı Çöltekin, Miriam Connor, Marine Courtin, Elizabeth Davidson, Marie-Catherine de Marneffe, Valeria de Paiva, Arantza Diaz de Ilarraza, Carly Dickerson, Bamba Dione, Peter Dirix, Kaja Dobrovoljc, Timothy Dozat, Kira Droganova, Puneet Dwivedi, Hanne Eckhoff, Marhaba Eli, Ali Elkahky, Binyam Ephrem, Tomaž Erjavec, Aline Etienne, Richárd Farkas, Hector Fernandez Alcalde, Jennifer Foster, Cláudia Freitas, Kazunori Fujita, Katarína Gajdošová, Daniel Galbraith, Marcos Garcia, Moa Gärdenfors, Sebastian Garza, Kim Gerdes, Filip Ginter, Iakes Goenaga, Koldo Gojenola, Memduh Gökırmak, Yoav Goldberg, Xavier Gómez Guinovart, Berta González Saavedra, Matias Grioni, Normunds Grūzītis, Bruno Guillaume, Céline Guillot Barbance, Nizar Habash, Jan Hajič, Jan Hajič jr., Linh Hà Mỹ, Na-Rae Han, Kim Harris, Dag Haug, Johannes Heinecke, Felix Hennig, Barbora Hladká, Jaroslava Hlaváčová, Florinel Hociung, Petter Hohle, Jena Hwang, Takumi Ikeda, Radu Ion, Elena Irimia, Ọlájídé Ishola, Tomáš Jelínek, Anders Johannsen, Fredrik Jørgensen, Hüner Kaşıkara, Andre Kaasen, Sylvain Kahane, Hiroshi Kanayama, Jenna Kanerva, Boris Katz, Tolga Kayadelen, Jessica Kenney, Václava Kettnerová, Jesse Kirchner, Arne Köhn, Kamil Kopacewicz, Natalia Kotsyba, Jolanta Kovalevskaitė, Simon Krek, Sookyoung Kwak, Veronika Laippala, Lorenzo Lambertino, Lucia Lam, Tatiana Lando, Septina Dian Larasati, Alexei Lavrentiev, John Lee, Phương Lê Hồng, Alessandro Lenci, Saran Lertpradit, Herman Leung, Cheuk Ying Li, Josie Li, Keying Li, KyungTae Lim, Yuan Li, Nikola Ljubešić, Olga Loginova, Olga Lyashevskaya, Teresa Lynn, Vivien Macketanz, Aibek Makazhanov, Michael

Mandl, Christopher Manning, Ruli Manurung, Cătălina Mărănduc, David Mareček, Katrin Marheinecke, Héctor Martínez Alonso, André Martins, Jan Mašek, Yuji Matsumoto, Ryan McDonald, Gustavo Mendonça, Niko Miekka, Margarita Misirpashayeva, Anna Missilä, Cătălin Mititelu, Yusuke Miyao, Simonetta Montemagni, Amir More, Laura Moreno Romero, Keiko Sophie Mori, Tomohiko Morioka, Shinsuke Mori, Shigeki Moro, Bjartur Mortensen, Bohdan Moskalevskyi, Kadri Muischnek, Yugo Murawaki, Kaili Müürisep, Pinkey Nainwani, Juan Ignacio Navarro Horñiacek, Anna Nedoluzhko, Gunta Nešpore Bērzkalne, Lương Nguyễn Thị, Huyền Nguyễn Thị Minh, Yoshihiro Nikaido, Vitaly Nikolaev, Rattima Nitisaroj, Hanna Nurmi, Stina Ojala, Adédayọ̀ Olúòkun, Mai Omura, Petya Osenova, Robert Östling, Lilja Øvrelid, Niko Partanen, Elena Pascual, Marco Passarotti, Agnieszka Patejuk, Guilherme Paulino Passos, Angelika Peljak Łapińska, Siyao Peng, Cenel-Augusto Perez, Guy Perrier, Daria Petrova, Slav Petrov, Jussi Piitulainen, Tommi A Pirinen, Emily Pitler, Barbara Plank, Thierry Poibeau, Martin Popel, Lauma Pretkalniņa, Sophie Prévost, Prokopis Prokopidis, Adam Przepiórkowski, Tiina Puolakainen, Sampo Pyysalo, Andriela Rääbis, Alexandre Rademaker, Loganathan Ramasamy, Taraka Rama, Carlos Ramisch, Vinit Ravishankar, Livy Real, Siva Reddy, Georg Rehm, Michael Rießler, Erika Rimkutė, Larissa Rinaldi, Laura Rituma, Luisa Rocha, Mykhailo Romanenko, Rudolf Rosa, Davide Rovati, Valentin Roșca, Olga Rudina, Jack Rueter, Shoval Sadde, Benoît Sagot, Shadi Saleh, Alessio Salomoni, Tanja Samardžić, Stephanie Samson, Manuela Sanguinetti, Abigail Walsh Sarah McGuinness, Dage Särg, Baiba Saulīte, Yanin Sawanakunanon, Nathan Schneider, Sebastian Schuster, Djamé Seddah, Wolfgang Seeker, Mojgan Seraji, Mo Shen, Atsuko Shimada, Hiroyuki Shirasu, Muh Shohibussirri, Dmitry Sichinava, Natalia Silveira, Maria Simi, Radu Simionescu, Katalin Simkó, Mária Šimková, Kiril Simov, Aaron Smith, Isabela Soares Bastos, Carolyn Spadine, Antonio Stella, Milan Straka, Jana Strnadová, Alane Suhr, Umut Sulubacak, Shingo Suzuki, Zsolt Szántó, Dima Taji, Yuta Takahashi, Fabio Tamburini, Takaaki Tanaka, Isabelle Tellier, Guillaume Thomas, Liisi Torga, Trond Trosterud, Anna Trukhina, Reut Tsarfaty, Francis Tyers, Sumire Uematsu, Zdeňka Urešová, Larraitz Uria, Hans Uszkoreit, Sowmya Vajjala, Daniel van Niekerk, Gertjan van Noord, Viktor Varga, Eric Villemonte de la Clergerie, Veronika Vincze, Lars Wallin, Jing Xian Wang, Jonathan North Washington, Maximilan Wendt, Seyi Williams, Mats Wirén, Christian Wittern, Tsegay Woldemariam, Tak-sum Wong, Alina Wróblewska, Mary Yako, Naoki Yamazaki, Chunxiao Yan, Koichi Yasuoka, Marat M. Yavrumyan, Zhuoran Yu, Zdeněk Žabokrtský, Amir Zeldes, Daniel Zeman, Manying Zhang, and Hanzhi Zhu. Universal Dependencies 2.4, 2019. URL `http://hdl.handle.net/11234/1-2988`. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Elias Pampalk, Arthur Flexer, and Gerhard Widmer. Improvements of Audio-Based Music Similarity and Genre Classificaton. In *ISMIR*, volume 5, pages 634–637. London, UK, 2005.

Slav Petrov, Dipanjan Das, and Ryan McDonald. A Universal Part-of-Speech Tagset. In *Proceedings of the Eighth International Conference on Language*

Resources and Evaluation (LREC-2012), pages 2089–2096, Istanbul, Turkey, May 2012. European Languages Resources Association (ELRA). URL http://www.lrec-conf.org/proceedings/lrec2012/pdf/274_Paper.pdf.

Martin Popel, Zdeněk Žabokrtský, and Martin Vojtek. Udapi: Universal API for Universal Dependencies. In *Proceedings of the NoDaLiDa 2017 Workshop on Universal Dependencies (UDW 2017)*, pages 96–101, Gothenburg, Sweden, May 2017. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/W17-0412.

Rudolf Rosa and Zdeněk Žabokrtský. KLcpos3 - a Language Similarity Measure for Delexicalized Parser Transfer. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 243–249, Beijing, China, July 2015. Association for Computational Linguistics. doi: 10.3115/v1/P15-2040. URL https://www.aclweb.org/anthology/P15-2040.

Benoît Sagot and Éric de la Clergerie. Error Mining in Parsing Results. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 329–336, Sydney, Australia, July 2006. Association for Computational Linguistics. doi: 10.3115/1220175.1220217. URL https://www.aclweb.org/anthology/P06-1042.

Timothy Shopen. *Language Typology and Syntactic Description*, volume 1, pages 40–59. Cambridge University Press, 2 edition, 2007. ISBN 0-511-36671-X. doi: 10.1017/CBO9780511619427. URL https://doi.org/10.1017/CBO9780511619427.

Leon Stassen. AND-languages and WITH-languages. *Linguistic Typology*, 4(1): 1–54, 2000. doi: https://doi.org/10.1515/lity.2000.4.1.1. URL https://www.degruyter.com/view/journals/lity/4/1/article-p1.xml.

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

Gertjan van Noord. Error Mining for Wide-Coverage Grammar Engineering. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL-04)*, pages 446–453, Barcelona, Spain, July 2004. doi: 10.3115/1218955.1219012. URL https://www.aclweb.org/anthology/P04-1057.

Himanshu Yadav, Ashwini Vaidya, and Samar Husain. Understanding Constraints on Non-Projectivity Using Novel Measures. In *Proceedings of the Fourth International Conference on Dependency Linguistics (Depling 2017)*, pages 276–286, Pisa, Italy, September 2017. Linköping University Electronic Press. URL https://www.aclweb.org/anthology/W17-6531.

Daniel Zeman. Reusable Tagset Conversion Using Tagset Drivers. In *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*, Marrakech, Morocco, May 2008. European Language Resources Association (ELRA). URL http://www.lrec-conf.org/proceedings/lrec2008/pdf/66_paper.pdf.

Daniel Zeman, Ondřej Dušek, David Mareček, Martin Popel, Loganathan Ramasamy, Jan Štěpánek, Zdeněk Žabokrtský, and Jan Hajič. HamleDT: Harmonized Multi-Language Dependency Treebank. *Language Resources and Evaluation*, 48(4):601–637, 2014a. ISSN 1574-0218. doi: 10.1007/s10579-014-9275-2. URL https://doi.org/10.1007/s10579-014-9275-2.

Daniel Zeman, David Mareček, Jan Mašek, Martin Popel, Loganathan Ramasamy, Rudolf Rosa, Jan Štěpánek, and Zdeněk Žabokrtský. HamleDT 2.0, 2014b. URL http://hdl.handle.net/11858/00-097C-0000-0023-9551-4. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Daniel Zeman, Martin Popel, Milan Straka, Jan Hajič, Joakim Nivre, Filip Ginter, Juhani Luotolahti, Sampo Pyysalo, Slav Petrov, Martin Potthast, Francis Tyers, Elena Badmaeva, Memduh Gokirmak, Anna Nedoluzhko, Silvie Cinková, Jan Hajič jr., Jaroslava Hlaváčová, Václava Kettnerová, Zdeňka Urešová, Jenna Kanerva, Stina Ojala, Anna Missilä, Christopher D. Manning, Sebastian Schuster, Siva Reddy, Dima Taji, Nizar Habash, Herman Leung, Marie-Catherine de Marneffe, Manuela Sanguinetti, Maria Simi, Hiroshi Kanayama, Valeria de Paiva, Kira Droganova, Héctor Martínez Alonso, Çağrı Çöltekin, Umut Sulubacak, Hans Uszkoreit, Vivien Macketanz, Aljoscha Burchardt, Kim Harris, Katrin Marheinecke, Georg Rehm, Tolga Kayadelen, Mohammed Attia, Ali Elkahky, Zhuoran Yu, Emily Pitler, Saran Lertpradit, Michael Mandl, Jesse Kirchner, Hector Fernandez Alcalde, Jana Strnadová, Esha Banerjee, Ruli Manurung, Antonio Stella, Atsuko Shimada, Sookyoung Kwak, Gustavo Mendonça, Tatiana Lando, Rattima Nitisaroj, and Josie Li. CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 1–19, Vancouver, Canada, August 2017. Association for Computational Linguistics. doi: 10.18653/v1/K17-3001. URL https://www.aclweb.org/anthology/K17-3001.

Daniel Zeman, Jan Hajič, Martin Popel, Martin Potthast, Milan Straka, Filip Ginter, Joakim Nivre, and Slav Petrov. CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 1–21, Brussels, Belgium, October 2018. Association for Computational Linguistics. doi: 10.18653/v1/K18-2001. URL https://www.aclweb.org/anthology/K18-2001.

Daniel Zeman, Joakim Nivre, Mitchell Abrams, Noëmi Aepli, Željko Agić, Lars Ahrenberg, Gabrielė Aleksandravičiūtė, Lene Antonsen, Katya Aplonova, Maria Jesus Aranzabe, Gashaw Arutie, Masayuki Asahara, Luma Ateyah, Mohammed Attia, Aitziber Atutxa, Liesbeth Augustinus, Elena Badmaeva,

Miguel Ballesteros, Esha Banerjee, Sebastian Bank, Verginica Barbu Mititelu, Victoria Basmov, Colin Batchelor, John Bauer, Sandra Bellato, Kepa Bengoetxea, Yevgeni Berzak, Irshad Ahmad Bhat, Riyaz Ahmad Bhat, Erica Biagetti, Eckhard Bick, Agnė Bielinskienė, Rogier Blokland, Victoria Bobicev, Loïc Boizou, Emanuel Borges Völker, Carl Börstell, Cristina Bosco, Gosse Bouma, Sam Bowman, Adriane Boyd, Kristina Brokaitė, Aljoscha Burchardt, Marie Candito, Bernard Caron, Gauthier Caron, Tatiana Cavalcanti, Gülşen Cebiroğlu Eryiğit, Flavio Massimiliano Cecchini, Giuseppe G. A. Celano, Slavomír Čéplö, Savas Cetin, Fabricio Chalub, Jinho Choi, Yongseok Cho, Jayeol Chun, Alessandra T. Cignarella, Silvie Cinková, Aurélie Collomb, Çağrı Çöltekin, Miriam Connor, Marine Courtin, Elizabeth Davidson, Marie-Catherine de Marneffe, Valeria de Paiva, Elvis de Souza, Arantza Diaz de Ilarraza, Carly Dickerson, Bamba Dione, Peter Dirix, Kaja Dobrovoljc, Timothy Dozat, Kira Droganova, Puneet Dwivedi, Hanne Eckhoff, Marhaba Eli, Ali Elkahky, Binyam Ephrem, Olga Erina, Tomaž Erjavec, Aline Etienne, Wograine Evelyn, Richárd Farkas, Hector Fernandez Alcalde, Jennifer Foster, Cláudia Freitas, Kazunori Fujita, Katarína Gajdošová, Daniel Galbraith, Marcos Garcia, Moa Gärdenfors, Sebastian Garza, Kim Gerdes, Filip Ginter, Iakes Goenaga, Koldo Gojenola, Memduh Gökırmak, Yoav Goldberg, Xavier Gómez Guinovart, Berta González Saavedra, Bernadeta Griciūtė, Matias Grioni, Normunds Grūzītis, Bruno Guillaume, Céline Guillot-Barbance, Nizar Habash, Jan Hajič, Jan Hajič jr., Mika Hämäläinen, Linh Hà Mỹ, Na-Rae Han, Kim Harris, Dag Haug, Johannes Heinecke, Felix Hennig, Barbora Hladká, Jaroslava Hlaváčová, Florinel Hociung, Petter Hohle, Jena Hwang, Takumi Ikeda, Radu Ion, Elena Irimia, Ọlájídé Ishola, Tomáš Jelínek, Anders Johannsen, Fredrik Jørgensen, Markus Juutinen, Hüner Kaşıkara, Andre Kaasen, Nadezhda Kabaeva, Sylvain Kahane, Hiroshi Kanayama, Jenna Kanerva, Boris Katz, Tolga Kayadelen, Jessica Kenney, Václava Kettnerová, Jesse Kirchner, Elena Klementieva, Arne Köhn, Kamil Kopacewicz, Natalia Kotsyba, Jolanta Kovalevskaitė, Simon Krek, Sookyoung Kwak, Veronika Laippala, Lorenzo Lambertino, Lucia Lam, Tatiana Lando, Septina Dian Larasati, Alexei Lavrentiev, John Lee, Phương Lê Hồng, Alessandro Lenci, Saran Lertpradit, Herman Leung, Cheuk Ying Li, Josie Li, Keying Li, Kyung-Tae Lim, Maria Liovina, Yuan Li, Nikola Ljubešić, Olga Loginova, Olga Lyashevskaya, Teresa Lynn, Vivien Macketanz, Aibek Makazhanov, Michael Mandl, Christopher Manning, Ruli Manurung, Cătălina Mărănduc, David Mareček, Katrin Marheinecke, Héctor Martínez Alonso, André Martins, Jan Mašek, Yuji Matsumoto, Ryan McDonald, Sarah McGuinness, Gustavo Mendonça, Niko Miekka, Margarita Misirpashayeva, Anna Missilä, Cătălin Mititelu, Maria Mitrofan, Yusuke Miyao, Simonetta Montemagni, Amir More, Laura Moreno Romero, Keiko Sophie Mori, Tomohiko Morioka, Shinsuke Mori, Shigeki Moro, Bjartur Mortensen, Bohdan Moskalevskyi, Kadri Muischnek, Robert Munro, Yugo Murawaki, Kaili Müürisep, Pinkey Nainwani, Juan Ignacio Navarro Horñiacek, Anna Nedoluzhko, Gunta Nešpore-Bērzkalne, Lương Nguyễn Thị, Huyền Nguyễn Thị Minh, Yoshihiro Nikaido, Vitaly Nikolaev, Rattima Nitisaroj, Hanna Nurmi, Stina Ojala, Atul Kr. Ojha, Adédayọ̀ Olúòkun, Mai Omura, Petya Osenova, Robert Östling, Lilja Øvrelid, Niko Partanen, Elena Pascual, Marco Passarotti, Agnieszka Patejuk, Guilherme

Paulino-Passos, Angelika Peljak-Łapińska, Siyao Peng, Cenel-Augusto Perez, Guy Perrier, Daria Petrova, Slav Petrov, Jason Phelan, Jussi Piitulainen, Tommi A Pirinen, Emily Pitler, Barbara Plank, Thierry Poibeau, Larisa Ponomareva, Martin Popel, Lauma Pretkalniņa, Sophie Prévost, Prokopis Prokopidis, Adam Przepiórkowski, Tiina Puolakainen, Sampo Pyysalo, Peng Qi, Andriela Rääbis, Alexandre Rademaker, Loganathan Ramasamy, Taraka Rama, Carlos Ramisch, Vinit Ravishankar, Livy Real, Siva Reddy, Georg Rehm, Ivan Riabov, Michael Rießler, Erika Rimkutė, Larissa Rinaldi, Laura Rituma, Luisa Rocha, Mykhailo Romanenko, Rudolf Rosa, Davide Rovati, Valentin Roșca, Olga Rudina, Jack Rueter, Shoval Sadde, Benoît Sagot, Shadi Saleh, Alessio Salomoni, Tanja Samardžić, Stephanie Samson, Manuela Sanguinetti, Dage Särg, Baiba Saulīte, Yanin Sawanakunanon, Nathan Schneider, Sebastian Schuster, Djamé Seddah, Wolfgang Seeker, Mojgan Seraji, Mo Shen, Atsuko Shimada, Hiroyuki Shirasu, Muh Shohibussirri, Dmitry Sichinava, Aline Silveira, Natalia Silveira, Maria Simi, Radu Simionescu, Katalin Simkó, Mária Šimková, Kiril Simov, Aaron Smith, Isabela Soares-Bastos, Carolyn Spadine, Antonio Stella, Milan Straka, Jana Strnadová, Alane Suhr, Umut Sulubacak, Shingo Suzuki, Zsolt Szántó, Dima Taji, Yuta Takahashi, Fabio Tamburini, Takaaki Tanaka, Isabelle Tellier, Guillaume Thomas, Liisi Torga, Trond Trosterud, Anna Trukhina, Reut Tsarfaty, Francis Tyers, Sumire Uematsu, Zdeňka Urešová, Larraitz Uria, Hans Uszkoreit, Andrius Utka, Sowmya Vajjala, Daniel van Niekerk, Gertjan van Noord, Viktor Varga, Eric Villemonte de la Clergerie, Veronika Vincze, Lars Wallin, Abigail Walsh, Jing Xian Wang, Jonathan North Washington, Maximilan Wendt, Seyi Williams, Mats Wirén, Christian Wittern, Tsegay Woldemariam, Tak-sum Wong, Alina Wróblewska, Mary Yako, Naoki Yamazaki, Chunxiao Yan, Koichi Yasuoka, Marat M. Yavrumyan, Zhuoran Yu, Zdeněk Žabokrtský, Amir Zeldes, Manying Zhang, and Hanzhi Zhu. Universal Dependencies 2.5, 2019. URL `http://hdl.handle.net/11234/1-3105`. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

List of Figures

2.1 Sample Non-projective Tree 14
2.2 Punctuation Node Attached Non-Projectively 15
2.3 Punctuation Node Causing Non-Projectivity 16

3.1 Features Used by LISCA to Calculate Plausibility Score for an Arc 20

4.1 Growth of POS Trigrams in PDT with Increase in Dataset Size . 28
4.2 Growth of POS Trigrams in EDT with Increase in Dataset Size . 28
4.3 Interaction of Genres in Treebanks A and B, such that $|G_A| \leq |G_B|$ 35

5.1 Possible Wrong Attachments of a Coordinating Conjunction: Correct Head as Wrong Head's Sibling 52
5.2 Possible Wrong Attachments of a Coordinating Conjunction . . . 52
5.3 Original Annotation for Example 9 53
5.4 Dependency Tree for Example 10 54
5.5 Dependency Tree for Example 11 56
5.6 Change in Annotation for Example 12 61
5.7 *attachToSibling()*: Single Sibling Available 62
5.8 Change in Annotation for Example 13 66
5.9 Change in Annotation for Example 14 67
5.10 Annotation Error in Example 15 70
5.11 Dependency Trees for Example 16 72
5.12 Dependency Trees for Example 17 73
5.13 Introduced Conjunction Sandwich in `ar` 75
5.14 Introduced Conjunction Sandwich in `af` 76
5.15 Change in Annotation for Example 20 77

6.1 `Case Error` in Example 21 90
6.2 `MWE Error` in Example 22 91
6.3 `Reported Speech Error` in Example 23 92
6.4 Head Identification Error in Example 24 93
6.5 `Naming Error` in Example 25 94
6.6 `Subtree Error` in Example 26 95
6.7 Correction of `Subtree Error` in Example 26 96

7.1 Rug plot with Distribution of Predictions with low confidence score 101

8.1 Dependency Tree showcasing association of `PART` with `mark` deprel 107
8.2 Example tree from Alzetta et al. [2017] showcasing `auxHead` error type 108

List of Tables

4.1 Sentence Counts in cs-PDT and et-EDT Treebanks 26
4.2 θ_{pos} and Coverage of POS Trigram Scores (± sd) Averaged over 100 Different Runs to Highlight the Effect of Size Disparity 27
4.3 Average Sentence Lengths in ar Treebanks 29
4.4 Identified Dimensions for Comparison of Genres in Biber [1991] . 31
4.5 Genre Distribution in UDv2.5 pl-LFG treebank 32
4.6 Genre Distribution in UDv2.5 fi-TDT treebank 32
4.7 θ_{pos} (± sd) Scores Averaged Over 100 Different Runs for Different Genres in UDv2.5 fi-TDT Treebank To Show Intra-Genre Annotation Consistency 33
4.8 Counts of Sentences for Different Genres in Downsampled Data from UDv2.5 fi-TDT and pl-LFG Treebanks 33
4.9 θ_{pos} Scores (± sd) Averaged over 100 runs for Inter-Genre Analysis in Downsampled UDv2.5 pl-LFG Data 34
4.10 θ_{pos} Scores (± sd) Averaged over 100 runs for Inter-Genre Analysis in Downsampled UDv2.5 fi-TDT Data 34
4.11 Datasets Compared when $G_A \subset G_B$ and $|G_A| < |G_B|$ 36
4.12 Datasets Compared when $G_A \not\subseteq G_B$; $G_A \cap G_B \neq \phi$ and $|G_A| \leq |G_B|$ 36
4.13 Datasets Compared when $G_A \not\subseteq G_B$; $G_A \cap G_B = \phi$ and $|G_A| \leq |G_B|$ 36
4.14 θ_{pos} (± sd) Scores Averaged over 100 Runs, Reported for Case When $G_A \subset G_B$ and $|G_A| < |G_B|$ 37
4.15 θ_{pos} (± sd) Scores Averaged over 100 Runs, Reported for Case When $G_A \not\subseteq G_B$; $G_A \cap G_B \neq \phi$ and $|G_A| \leq |G_B|$ 37
4.16 θ_{pos} (± sd) Scores Averaged over 100 Runs, Reported for Case When $G_A \not\subseteq G_B$; $G_A \cap G_B = \phi$ and $|G_A| \leq |G_B|$ 37
4.17 θ_{pos} Scores (± sd) Averaged over 100 Different Runs With Adulterant Genres are Present in pl-LFG Data 39
4.18 Color Codes Used to Mark Consistent or Inconsistent Treebanks based on θ_{pos} Scores in Table 4.19 42
4.19 θ_{pos} Scores in UDv2.5 Marked for Consistency or Inconsistency in POS Annotation 43
4.20 Comparison of θ_{pos} Score and Θ_{pos} Limit for Pairs of Treebanks Marked as Inconsistent in Table 4.19 43

5.1 Possible Syndetion Typologies across Languages 47
5.2 Misdirected Coordinating Conjunctions in UDv2.4 PUD Treebanks 49
5.3 Misdirected Coordinating Conjunctions in UDv2.4 Treebanks . . . 50
5.4 Misdirected Coordinating Conjunctions in UDv2.4 Treebanks for af and ar 57
5.5 Average Runtime (± sd) for Udapy Python Block Implementation 68
5.6 Nodes Affected: Non Projective Attachment 69
5.7 Evaluation: *projTempFix()* 71
5.8 Nodes Affected: Overall 74
5.9 Overall Evaluation of Affected Nodes on Randomly Sampled Instances 74

5.10 Misdirected Dependencies: Before and After 79

6.1 Size of hi-HDTB treebank 82
6.2 Counts of Sentences and Tokens from Individual Splits, Before and After Downsampling 83
6.3 Statistics for Arc Scores in Baseline Run 85
6.4 Classification of Errors in Baseline Run 85
6.5 Statistics for Arc Scores in Experimental Runs 85
6.6 Commonly Flagged Instances from Downsampled Test Data in Baseline and Experimental Runs 86
6.7 Error Counts in Downsampled Test Data across Different Runs . 86
6.8 Typology of Errors in Downsampled Test Data across Different Runs 87
6.9 Error Frequencies for Experimental Runs, Normalized Over 1000 Flagged Arcs 88

7.1 Hyper-Parameters for Neural Network 100
7.2 Categories of Error Patterns 100
7.3 Metrics of Best Model trained over UDv2.4 hi-HDTB Treebank . 102
7.4 Results of Manual Annotation 103
7.5 Statistics for hi-HDTB Treebank 103

A.1 Languages in UDv2.5, identified with their ISO Codes 134
A.2 Multiple Treebanks in Different Languages, UDv2.5 135
A.3 Languages with PUD Treebanks, UDv2.5 136
A.4 Non-Projectivity and Relaxations in UDv2.5 Data (% of # Trees) 141

A. Appendix

A.1 Terminology Pertaining to UD

This appendix is meant primarily for the offline/hard copy readers of the document. A better (and official) explanation of the terms can be accessed online[1,2].

A.1.1 CoNLL-U Format

UD uses an extension of CoNLL-X format [Buchholz and Marsi, 2006], referred to as CoNLL-U format. The CoNLL-U format is used for the annotation procedure, with three types of lines. Each line is delimited by LF character as line break, written in UTF-8 encoding. The details of the line types are as follows:

1. **Blank Line**: A line without any content, used as a separator for annotations of different sentences in the treebank.
2. **Comment Line**: A line starting with hash (#) symbol, typically contains details about the annotated sentence. The details that are common across all treebanks are 'sent_id' (a unique ID associated with each sentence in the treebank), and 'text' (the text of the annotated sentence). The comment can also include any other details like paragraph id, document id, etc.
3. **Word Line**: Each Word Line contains the annotation of a single word, in a 10-column TSV (tab-separated values) format. The columns, in order, and their explanation are as follows:
   (a) **ID**: Word Index in the sentence, starts at 1. Can be a ranged value for fused tokens and multiword tokens; decimal value for empty nodes. The ID of a token can be only greater than 0.
   (b) **FORM**: Word Form, as it appears in the sentence.
   (c) **LEMMA**: Lemma or Stem of Word Form.
   (d) **UPOS**: The Universal POS tag of the word, as per UD Tagset.
   (e) **XPOS**: The language-specific POS tag of the word. Generally comes from the original tagset that was converted into UD.
   (f) **FEATS**: List of morphological features from UD feature inventory, or a language specific version thereof.
   (g) **HEAD**: Head of the current word in dependency relation. Contains 'ID' of the parent word, or 0 if the parent word is 'Root' (explained later).
   (h) **DEPREL**: Universal Dependency Relation, extendable with language specific extension thereof (cf. Section A.1.2).
   (i) **DEPS**: Enhanced Dependency Relation in form of head:deprel pairs.

[1] `https://universaldependencies.org/format.html`
[2] `https://universaldependencies.org/u/overview/morphology.html`

(j) **MISC**: Any other annotation.

Of the different columns (referred to as Fields), there are associated restrictions, briefed as follows:

- Fields must not be empty. An unspecified value is represented by an underscore (_) symbol.
- Fields other than FORM and LEMMA cannot contain space characters.
- UPOS, HEAD, DEPREL are not allowed to be left unspecified.

A.1.2 UD Annotation

There are some additional points with respect to UD Annotation that must be clarified.

1. For the dependency tree, UD annotates the global root of a sentence as a token with ID=0, referred to as `ROOT`. The root in the sentence is always a singular unit, and is a direct child of this `ROOT` node.

2. A dependency relation is expressed in a format that combines the universal deprel and language specific part of deprel with a colon mark (:). The language specific extension is optional, but is present in a lot of cases nonetheless. We refer to the universal relation as udeprel, and the language specific extension as xdeprel. Following example illustrates the same.

   Example 29. In DEPREL Field value as `acl:relcl`, `acl` is the universal dependency relation (referred to as udeprel, as per Udapi nomenclature) while `relcl` is the language specific extension of `acl` udeprel (referred to as xdeprel, as per Udapi nomenclature).

   As mentioned earlier, we refer to udeprel when we talk about deprels in this document, unless otherwise stated.

A.2 List of Language Codes

This appendix contains the list of languages along with their identification codes, as used in the different treebanks of UDv2.5. A full list of ISO 639-3 language codes can also be accessed online[3].

Table A.1 indicates languages where the ISO codes (ISO 639-1 or ISO 639-3) is used as an identifier, arranged in alphabetical order. The only exception is `qhe` for UD_Hindi_English-HIENCS code-switching treebank, where the ISO code being employed is a reserved code for local use.

Note:

- * against a language name indicates lack of a treebank corresponding to the language in UDv2.4.

Code	Language Name
af	Afrikaans
aii	Assyrian
akk	Akkadian
am	Amharic
ar	Arabic
be	Belarusian
bg	Bulgarian
bho	Bhojpuri*
bm	Bambara
br	Breton
bxr	Buryat
ca	Catalan
cop	Coptic
cs	Czech
cu	Old Church Slavonic
cy	Welsh
da	Danish
de	German
el	Greek
en	English
es	Spanish
et	Estonian
eu	Basque
fa	Persian
fi	Finnish
fo	Faroese
fr	French
fro	Old French
ga	Irish
gd	Scottish Gaelic*
gl	Galician

Continued on next page

[3] https://iso639-3.sil.org/code_tables/639/data/all

Code	Language Name
got	Gothic
grc	Ancient Greek
gsw	Swiss German*
gun	Mbya Guarani
he	Hebrew
hi	Hindi
hr	Croatian
hu	Hungarian
hsb	Upper Sorbian
hy	Armenian
id	Indonesian
it	Italian
ja	Japanese
kk	Kazakh
kmr	Kurmanji
ko	Korean
koi	Komi Permyak*
kpv	Komi Zyrian
krl	Karelian
la	Latin
lt	Lithuanian
lv	Latvian
lzh	Classical Chinese
mdf	Moksha*
mr	Marathi
mt	Maltese
myv	Erzya
no	Norwegian
nl	Dutch
olo	Livvi*
orv	Old Russian
pcm	Naija
pl	Polish
pt	Portuguese
ro	Romanian
ru	Russian
sa	Sanskrit
sk	Slovak
sl	Slovenian
sme	North Sami
sms	Skolt Sami*
sr	Serbian
sv	Swedish
swl	Swedish Sign Language
ta	Tamil
te	Telugu

Continued on next page

Code	Language Name
th	Thai
tl	Tagalog
tr	Turkish
ug	Uyghur
uk	Ukrainian
ur	Urdu
vi	Vietnamese
wbp	Warlpiri
wo	Wolof
yo	Yoruba
yue	Cantonese
zh	Chinese

Table A.1: Languages in UDv2.5, identified with their ISO Codes

A.3 Multiple Treebanks in Languages (UDv2.5)

Table A.2 contains the different languages in UDv2.5 such that they contain multiple treebanks. The second column of the table corresponds to the count of the different treebanks, and the last column contains the name of the treebanks. Notice that PUD treebanks are not included. A list of PUD treebanks can be accessed in Appendix A.4.

Language	Count	Treebank Names
ar	2	NYUAD, PADT
cs	4	CAC, CLTT, FicTree, PDT
de	3	GSD, HDT, LIT
en	6	ESL, EWT, GUM, LinES, ParTUT, Pronouns^{+}
es	2	AnCora, GSD
et	2	EDT, EWT
fi	2	FTB, TDT
fr	6	FQB, FTB, GSD, ParTUT, Sequoia, Spoken
gl	2	CTG, TreeGal
grc	2	Perseus, PROIEL
gun	2	Dooley, Thomas
it	5	ISDT, ParTUT, PoSTWITA, TWITTIRO^{+}, VIT
ja	3	BCCWJ, GSD, Modern
ko	2	GSD, Kaist
kpv	2	IKDP, Lattice
la	3	ITTB, Perseus, PROIEL
lt	2	ALKSNIS, HSE
nl	2	Alpino, LassySmall
no	3	Bokmaal, Nynorsk, NynorskLIA
orv	2	RNC, TOROT
pl	2	LFG, PDB
pt	2	Bosque, GSD
ro	3	Nonstandard, RRT, SiMoNERo^{+}
ru	3	GSD, SynTagRus, Taiga
sl	2	SSJ, SST
sv	2	LinES, Talbanken
tr	2	GB, IMST
zh	4	CFL, GSD, GSDSimp^{+}, HK

Table A.2: Multiple Treebanks in Different Languages, UDv2.5
Note: Superscript + against a treebank name indicates treebank not present in UDv2.4

A.4 PUD Treebanks

PUD treebanks were formed as a part of CoNLL 2017 Shared Task [Zeman et al., 2018]. Across different languages, the PUD treebanks contain the same 1000 sentences, from news genre, and from Wikipedia. Of these sentences, the first 750 sentences were originally in `en`, whereas the others were originally in `de`, `es`, `fr` or `it` and were translated to other languages via `en`. The translation into majority of the languages have been performed by professional translators. The treebanks for the languages were first annotated as per Google universal annotation guidelines Petrov et al. [2012], and then to UDv2 guidelines. The treebanks for `cs`, `fi`, `pl` and `sv` were translated by local teams responsible for the language, and were annotated directly as per UDv2 guidelines.

Table A.3 contains a list of languages which contain a PUD treebank. Notice that PUD treebanks contain only the test set, and are devoid of train and dev data. The official recommended usage of PUD treebanks is with a 10-fold cross validation for training purpose, or using the whole treebank as testing data, as the case may be.

Code	Language Name
ar	Arabic
cs	Czech
de	German
en	English
es	Spanish
fi	Finnish
fr	French
hi	Hindi
id	Indonesian
it	Italian
ja	Japanese
ko	Korean
pl	Polish
pt	Portuguese
ru	Russian
sv	Swedish
th	Thai
tr	Turkish
zh	Chinese

Table A.3: Languages with PUD Treebanks, UDv2.5

A.5 Relaxations to Non-Projectivity

The condition of projectivity is a strict constraint for natural languages, exhibited by very few constructions in most languages of the world. To better account for linguistic processes, several relaxations to the definition of projectivity were defined. A discussion of all such relaxations is out of scope of this work. However, we define 3 most widely used relaxations here.

1. **Planarity**
   The given tree is said to be planar, if it does not have any edges that overlap. Formally speaking, given two undirected edges $i_1 \leftrightarrow j_1$ and $i_2 \leftrightarrow j_2$; if $i_1 < i_2 < j_1 < j_2$ or $i_1 > i_2 > j_1 > j_2$, the edges are said to overlap. Therefore, a given tree is called non-planar if there exists a pair of edges $i_1 \leftrightarrow j_1$ and $i_2 \leftrightarrow j_2$ such that the edges overlap.

2. **Ill-Nestedness**
   It is easier to define the condition of ill-nestedness rather than to define the well-nestedness. A given (sub)tree is called ill-nested, if for given undirected edges $i_1 \leftrightarrow j_1$ and $i_2 \leftrightarrow j_2$; $i_1 \in Gap(i_2, j_2)$ & $i_2 \in Gap(i_1, j_1)$. It is worth noting that projective trees are always well-nested, but a well-nested tree is not always projective.

3. **Mild Non-Projectivity**
   A tree is said to be mildly non-projective if

   (a) It is well-nested.

   (b) The gap degree of the tree is bound by any constant k. Essentially, gap degree of tree $\leq k$.

A.5.1 Statistics of Non-Projectivities in UDv2.5

Treebank	**# Trees**	**Non-Proj.**		**Non-Planar**		**Ill-Nested**	
		Trees	%	Trees	%	Trees	%
af-afribooms	1934	432	22.34	19	0.98	1	0.05
aii-as	57	-	-	-	-	-	-
akk-pisandub	101	7	6.93	-	-	-	-
am-att	1074	26	2.42	-	-	-	-
ar-nyuad	19738	122	0.62	-	-	-	-
ar-padt	7664	638	8.32	19	0.25	11	0.14
ar-pud	1000	38	3.80	1	0.10	-	-
be-hse	637	46	7.22	-	-	-	-
bg-btb	11138	342	3.07	2	0.02	1	0.01
bho-bhtb	254	35	13.78	7	2.76	1	0.39
bm-crb	1026	33	3.22	-	-	-	-
br-keb	888	24	2.70	1	0.11	1	0.11
bxr-bdt	927	145	15.64	12	1.29	1	0.11
ca-ancora	16678	746	4.473	5	0.03	-	-
cop-scriptorium	1575	206	13.08	-	-	-	-

Continued on next page

Treebank	# Trees	Non-Proj.		Non-Planar		Ill-Nested	
		Trees	%	Trees	%	Trees	%
cs-cac	24709	3143	12.72	50	0.20	14	0.06
cs-cltt	1125	163	14.49	7	0.62	6	0.53
cs-fictree	12760	1455	11.40	32	0.25	3	0.02
cs-pdt	87913	10098	11.49	157	0.18	47	0.05
cs-pud	1000	104	10.40	2	0.20	1	0.10
cu-proiel	6338	1034	16.31	32	0.50	5	0.08
cy-ccg	956	18	1.88	1	0.10	-	-
da-ddt	5512	1185	21.50	55	1.00	19	0.34
de-gsd	15590	1451	9.31	24	0.15	4	0.03
de-hdt	189928	12871	6.78	588	0.31	37	0.02
de-lit	1922	150	7.80	10	0.52	1	0.05
de-pud	1000	137	13.70	6	0.60	1	0.10
el-gdt	2521	142	5.63	-	-	-	-
en-esl	5124	208	4.06	7	0.14	4	0.08
en-ewt	16622	767	4.61	22	0.13	6	0.04
en-gum	5427	410	7.55	10	0.18	1	0.02
en-lines	5243	459	8.75	24	0.46	13	0.25
en-partut	2090	39	1.87	2	0.10	1	0.05
en-pronouns	285	5	1.75	-	-	-	-
en-pud	1000	45	4.50	1	0.10	-	-
es-ancora	17680	928	5.25	5	0.03	-	-
es-gsd	16013	937	5.85	16	0.10	2	0.01
es-pud	1000	45	4.50	1	0.10	-	-
et-edt	30972	993	3.21	9	0.03	3	0.01
et-ewt	1662	111	6.68	2	0.12	1	0.06
eu-bdt	8993	2983	33.17	424	4.71	92	1.02
fa-seraji	5997	401	6.69	25	0.42	1	0.02
fi-ftb	18723	1444	7.71	150	0.80	73	0.39
fi-pud	1000	36	3.60	-	-	-	-
fi-tdt	15136	931	6.15	9	0.06	-	-
fo-oft	1208	33	2.73	2	0.17	1	0.08
fr-fqb	2289	75	3.28	1	0.04	-	-
fr-ftb	18535	2019	10.89	69	0.37	21	0.11
fr-gsd	16342	428	2.62	6	0.04	-	-
fr-partut	1020	45	4.41	-	-	-	-
fr-pud	1000	17	1.70	-	-	-	-
fr-sequoia	3099	66	2.13	-	-	-	-
fr-spoken	2789	340	12.19	8	0.29	1	0.04
fro-srcmf	17678	2726	15.42	290	1.64	82	0.46
ga-idt	1763	272	15.43	22	1.25	9	0.51
gd-arcosg	2193	259	11.81	14	0.64	8	0.36
gl-ctg	3993	-	-	-	-	-	-
gl-treegal	1000	113	11.30	7	0.70	2	0.20
got-proiel	5401	949	17.57	32	0.59	5	0.09
grc-perseus	13919	8890	63.87	1275	9.16	150	1.08

Continued on next page

Treebank	# Trees	Non-Proj.		Non-Planar		Ill-Nested	
		Trees	%	Trees	%	Trees	%
grc-proiel	17080	6409	37.52	392	2.30	38	0.22
gsw-uzh	100	4	4.00	-	-	-	-
gun-dooley	1046	-	-	-	-	-	-
gun-thomas	98	4	4.08	-	-	-	-
he-htb	6216	472	7.59	6	0.10	-	-
hi-hdtb	16647	2264	13.60	116	0.70	13	0.08
hi-pud	1000	257	25.70	16	1.60	1	0.10
hr-set	9010	810	8.99	20	0.22	9	0.10
hsb-ufal	646	73	11.30	2	0.31	-	-
hu-szeged	1800	488	27.11	38	2.11	17	0.94
hy-armtdp	2502	179	7.15	4	0.16	-	-
id-gsd	5593	291	5.20	11	0.20	2	0.04
id-pud	1000	13	1.30	-	-	-	-
it-isdt	14167	196	1.38	9	0.06	5	0.04
it-partut	2090	42	2.01	2	0.10	2	0.10
it-postwita	6713	86	1.28	2	0.03	2	0.03
it-pud	1000	8	0.80	-	-	-	-
it-twittiro	1424	17	1.19	1	0.07	-	-
it-vit	10087	353	3.50	18	0.18	7	0.07
ja-bccwj	57109	163	0.29	1	0.00	-	-
ja-gsd	8186	-	-	-	-	-	-
ja-modern	822	5	0.61	-	-	-	-
ja-pud	1000	-	-	-	-	-	-
kk-ktb	1078	130	12.06	3	0.28	1	0.09
kmr-mg	754	130	17.24	5	0.66	4	0.53
ko-gsd	6339	1006	15.87	22	0.35	3	0.05
ko-kaist	27363	5938	21.70	89	0.33	-	-
ko-pud	1000	66	6.60	-	-	-	-
koi-uh	49	1	2.04	-	-	-	-
kpv-ikdp	117	3	2.56	-	-	-	-
kpv-lattice	210	4	1.90	1	0.48	-	-
krl-kkpp	228	45	19.74	3	1.32	-	-
la-ittb	21011	7771	36.99	357	1.70	39	0.19
la-perseus	2273	1094	48.13	201	8.84	64	2.82
la-proiel	18411	5227	28.39	448	2.43	38	0.21
lt-alksnis	3642	441	12.11	7	0.19	1	0.03
lt-hse	263	38	14.45	2	0.76	1	0.38
lv-lvtb	13643	888	6.51	7	0.05	-	-
lzh-kyoto	15115	-	-	-	-	-	-
mdf-jr	65	2	3.08	-	-	-	-
mr-ufal	466	28	6.01	1	0.21	1	0.21
mt-mudt	2074	81	3.91	1	0.05	-	-
myv-jr	1550	79	5.10	4	0.26	3	0.19
nl-alpino	13578	1961	14.44	129	0.95	-	-
nl-lassysmall	7338	447	6.09	25	0.34	1	0.01

Continued on next page

Treebank	# Trees	Non-Proj.		Non-Planar		Ill-Nested	
		Trees	%	Trees	%	Trees	%
no-bokmaal	20044	1495	7.46	32	0.16	-	-
no-nynorsk	17575	1361	7.74	27	0.15	4	0.02
no-nynorsklia	5250	495	9.43	37	0.70	3	0.06
olo-kkpp	125	17	13.60	2	1.60	2	1.60
orv-rnc	604	189	31.29	10	1.66	3	0.50
orv-torot	16944	2575	15.20	71	0.42	4	0.02
pcm-nsc	948	6	0.63	-	-	-	-
pl-lfg	17246	111	0.64	3	0.02	1	0.01
pl-pdb	22152	1390	6.27	20	0.09	2	0.01
pl-pud	1000	52	5.20	-	-	-	-
pt-bosque	9365	2862	30.56	307	3.28	72	0.77
pt-gsd	12078	684	5.66	11	0.09	6	0.05
pt-pud	1000	33	3.30	-	-	-	-
qhe-hiencs	1898	192	10.12	7	0.37	4	0.21
ro-nonstandard	15843	819	5.17	9	0.06	1	0.01
ro-rrt	9524	864	9.07	21	0.22	10	0.11
ro-simonero	491	54	11.00	4	0.81	1	0.20
ru-gsd	5030	318	6.32	10	0.20	2	0.04
ru-pud	1000	24	2.40	-	-	-	-
ru-syntagrus	61889	4658	7.53	58	0.09	13	0.02
ru-taiga	3264	277	8.49	12	0.37	5	0.15
sa-ufal	230	40	17.39	3	1.30	-	-
sk-snk	10604	347	3.27	4	0.04	2	0.02
sl-ssj	8000	960	12.00	11	0.14	2	0.03
sl-sst	3188	144	4.52	1	0.03	-	-
sme-giella	3122	338	10.83	21	0.67	5	0.16
sms-giellagas	36	2	5.56	-	-	-	-
sr-set	4384	172	3.92	5	0.11	1	0.02
sv-lines	5243	305	5.82	13	0.25	4	0.08
sv-pud	1000	38	3.80	-	-	-	-
sv-talbanken	6026	181	3.00	-	-	-	-
swl-sslc	203	67	33.00	6	2.96	-	-
ta-ttb	600	9	1.50	-	-	-	-
te-mtg	1328	2	0.15	-	-	-	-
th-pud	1000	28	2.80	-	-	-	-
tl-trg	55	-	-	-	-	-	-
tr-gb	2802	28	1.00	-	-	-	-
tr-imst	5635	646	11.46	65	1.15	26	0.46
tr-pud	1000	149	14.90	4	0.40	-	-
ug-udt	3456	172	4.98	1	0.03	-	-
uk-iu	7060	547	7.75	9	0.13	1	0.01
ur-udtb	5130	1158	22.57	98	1.91	27	0.53
vi-vtb	3000	87	2.90	1	0.03	-	-
wbp-ufal	55	6	10.91	-	-	-	-
wo-wtb	2107	63	2.99	1	0.05	1	0.05

Continued on next page

Treebank	**# Trees**	**Non-Proj.**		**Non-Planar**		**Ill-Nested**	
		Trees	%	Trees	%	Trees	%
yo-ytb	100	9	9.00	-	-	-	-
yue-hk	1004	126	12.55	13	1.29	5	0.50
zh-cfl	451	4	0.89	-	-	-	-
zh-gsd	4997	117	2.34	1	0.02	-	-
zh-gsdsimp	4997	-	-	-	-	-	-
zh-hk	1004	43	4.28	-	-	-	-
zh-pud	1000	7	0.70	-	-	-	-

Table A.4: Non-Projectivity and Relaxations in UDv2.5 Data (% of # Trees)