A Web Application for Annotating Dependency Parse Treebanks

by

Salih Furkan Akkurt

Submitted to the Department of Computer
Engineering in partial fulfillment of
the requirements for the degree of
Bachelor of Science

Undergraduate Program in Computer Engineering
Boğaziçi University

Spring 2022

A Web Application for Annotating Dependency Parse Treebanks

APPROVED BY:

Prof. Suzan Uskudarli .....
(Project Supervisor)

DATE OF APPROVAL: 14.02.2022

ACKNOWLEDGEMENTS

This project was supported by Boğaziçi University Research Fund Grant Number 16909.

ABSTRACT

A Web Application for Annotating Dependency Parse Treebanks

The value of quality treebanks is increasing due to the important role they play in the development of natural language processing tools. The creation of such treebanks is both labour-intensive and time-consuming. Therefore, tools that support the annotation process are very valuable. This is especially the case for morphologically rich languages (MRLs). The annotation tool BoAT v1 was developed for annotating dependency relations and subsequently was used to create the manually annotated BOUN Treebank (UD_Turkish-BOUN) of Turkish National Corpus (TNC). The extensive annotation experience revealed several opportunities of improvement. Combined with our desire to create a multi-user web-based annotation tool, this led to the development of BoAT v2. The main objectives of the new tool are to: (1) provide further support for creating valid and consistent annotations, (2) significantly improve the user experience of the annotator, and (3) develop an open source and easily deployable web-based annotation tool and an API to benefit the scientific and education community. This report details the requirements, design and implementation of BoAT v2 with annotation examples that highlight the new features.

ÖZET

Ağaç Yapılı Derlemlerin Bağlılık Etiketleriyle Anotasyonu için Web Uygulaması

Kaliteli ağaç yapılı derlemlerin değeri, doğal dil işleme araçlarının geliştirilmesinde oynadıkları rol sebebiyle artıyor. Bu derlemlerin yaratılması hem emek isteyen hem de zaman alan bir iş. Bu yüzden, etiketleme sürecini destekleyen araçlar çok değerli. Bu araçlar, morfolojik olarak zengin diller için ayrıca değerli. Etiketleme aracı BOAT V1 bağlılık ilişkilerini etiketlemek için geliştirilmişti ve ardından Turkish National Corpus'tan (TNC) alınan cümlelerden oluşan BOUN Treebank'i etiketlemek için kullanıldı. Yoğun etiketleme tecrübesi birçok gelişim fırsatları ortaya çıkardı. Çoklu kullanıcı destekleyen bir web uygulaması olacak etiketleme aracını yapma isteğimizle birlikte, bu BOAT V2'yu geliştirmemize sebep oldu. Bu yeni aracın asıl hedefleri: (1) geçerli ve istikrarlı etiketlemeler için daha fazla destek sağlamak, (2) etiketleyicilerin kullanıcı deneyimini epey geliştirmek, (3) eğitim ve bilim alanında çalışan topluluklar için açık kaynak ve kolaylıkla dağıtımı sağlanabilen ve web üzerinde kullanılabilen bir etiketleme aracı ve bu aracı besleyecek bir API geliştirmek. Bu rapor BOAT V2'nun gereksinimlerini, tasarımını ve geliştirilmesini, yeni özellikleri vurgulayan etiketleme örnekleriyle detaylandırıyor.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ACRONYMS/ABBREVIATIONS

CoNLL	Computational Natural Language Learning
UD	Universal Dependencies
BoAT	Bogazici University Annotation Tool
DIP	Doğal Dil İşleme Platformu ("Natural Language Processing Platform")

1. INTRODUCTION AND MOTIVATION

Treebanks are sets of sentences annotated manually by linguists or first automatically and then refined by linguists. They are used in many natural language processing (NLP) tasks, such as machine translation, parsers, etc. For quality purposes, manually annotated treebanks by linguists are very valuable. Even by tools crafted with linguists in mind, manually annotating sentences is hard work and tends to require consistent attention. Therefore, tools that support the annotation process are very important. These tools should be further refined according to user, who are linguists in this case, feedback in order to increase efficiency and reduce cognitive load of annotators. Many of the available annotation tools work well with languages like English or Chinese where words are compact in their dependencies. As Turkish is a morphologically rich language (MRL), tools developed for it should be extra focused on easily splitting tokens.

Recently, there has been a drive to increase quality resources of Turkish, which was considered a low-resource language, in the NLP field. Creating annotated data is a part of this drive. There are several tools for annotating data [1, 2], however, they tend not to be well suited for morphologically rich languages (MRLs) like Turkish. The complex nature of such languages requires the annotator to perform many operations like splitting tokens. Annotation tools often rely on drag-drop interfaces, which seem appealing, however, they tend to disrupt the flow of an annotator. This is especially true for MRLs and when large numbers of sentences are annotated leading to error-prone conditions that increase with passing time.

As a part of the aforementioned drive, BOAT V1 [3] was developed to support the dependency annotation of MRLs to produce treebanks compliant with the Universal Dependency framework [4]. It is a standalone application developed with Python and based on Qt. BOAT V1 was used to create the BOUN Treebank [3, 5, 6] – a manually annotated Turkish dependency treebank comprising 9,761 sentences. Feedback of usage from the annotators that have used the tool motivates this study. The efficiency of the annotator with keyboard-based interactions was validated during this process.

However, several points of improvement were also revealed based on this experience. We believe this tool can be developed further into a web application that allows a network of annotators to collaborate on treebanks.

Given the value of the effort and time of annotators as well as the created data resources, the design of annotation tools should focus on the annotator user experience with features that support quality annotations. This work presents a dependency annotation tool that has been redesigned based on the experiences with BOAT V1 and availability issues. Several improvements have been implemented with focus on improving the user experience and consequently the quality of the produced datasets. The redesign of the tool aims to: (1) improve the user experience of the annotator, (2) reduce errors, (3) create a multi-user web-based application with a supporting API to support flexibility, and (4) create a self-contained and easily deployable tool for the benefit of researchers and educators. The development of the tool was started with requirements elicitation, for which earlier experiences and in-depth interviews with an annotator who is actively annotating hundreds of annotations per week were taken into account.

For this purpose, we are developing a web application that supports multiple users to enable annotation via several annotators. An API is still being designed to render the functionalities accessible to programs with alternative uses. Several user experience improvements were implemented in order to maintain the flow state of an annotator. Finally, the solution will be dockerized in order to make it easily accessible and deployable for others.

The current prototype is being evaluated with positive early feedback. The final version will be made available as a demo on our NLP platform [7] and made available as open source. Additionally, a paper is being written about the project and parts of it can be seen in Appendix B.

The remainder of this report is organized as follows: Section 2 presents related work. Section 3 describes the methods. Section 4 describes the requirements elicitation

process and design. Section 5 presents the implementation with emphasis on the new features. Section 6 describes our current results. We discuss our experiences, point out some future work and make our concluding remarks in Section 7. Section 8 describes the work remaining for the project.

2. STATE OF THE ART

There are various tools for creating dependency annotations such as *brat* [1], *UD Annotatrix* [8] and *DgAnnotator* [2] These tools mostly rely on mouse-based user interaction, which tend to be inefficient for annotating MRLs. Many of them use the UD framework for their formatting of annotations.

There are also tools developed specifically for Turkish. The ITU Treebank Annotation Tool [9] is a tool written in Java that has several versions. It offers semi-automated support for annotators through analyzers for creating new datasets as well as correcting the existing Turkish treebanks. It mostly relies on mouse-based interactions and doesn't support UD.

Our department also have published an annotation tool focused on dependency parsing as an open source desktop application, called BOAT V1 [5]. It has many improvements from the state-of-the-art at the time of its development. A shortcoming of this tool is that it doesn't have a database and is only locally available which makes collaboration between annotators on a single treebank only possible manually. It is a standalone application that supports keyboard-based input while displaying the annotations as a table as well as a dependency graph. It also supports UD [4]. This tool was used to create the BOUN Treebank [3, 6] – a manually annotated Turkish dependency treebank comprising 9,761 sentences. It also uses analyzers to process sentences. The feedback received from this effort indicates that the keyboard based input is much more efficient and takes less time for creating quality treebanks.

The work presented in this report is based on the experiences from BOAT V1 with continuous feedback from an annotator. Like many tools, it supports UD, and it also supports keyword-based input to allow the annotation process to be performed without breaking the flow. Furthermore, the tool is being developed as a web application with an API to support ease of deployment and multi-annotator scenarios.

3. METHODS

In-depth interviews with the annotator of the BOUN Treebank have been conducted for requirements elicitation, requirements validation and prototype feedback. A Software Requirements Specification (SRS) document have been created as a result of these meetings and ample feedback. In these interviews, the annotator showed the tool in a live annotation session. She was asked questions about what the main cause of the loss of annotation flow was. The resultant requirements are displayed in the next section.

3.1. Project Management

(i) We meet every week for discussions related to the previous week's work.

(ii) We also meet at the DIP Meeting of our department on Fridays.

(iii) We communicate through Slack and Zoom.

(iv) We use GitLab for the project repository and create issues there for everything, from bugs to documentation.

(v) The Wiki page on our GitLab repository includes documentation for requirements, implementation, meeting notes, etc.

4. SOFTWARE REQUIREMENTS

This section presents the software requirements elicited and validated through interviews with an annotator who is responsible for the creation of the BOUN Treebank.

4.1. Glossary for the Software Requirements Document

- **Annotation**: The task of parsing and identifying features of a text linguistically.
- **Autocomplete**: A shorthand writing system where a user can write the initials of a value and the system infers the value intended to be written.
- **Column**: A field of the forms chosen to be shown on the table by the annotator.
- **CoNLL-U**: A revised version of the CoNLL-X format[1] .
- **Dependency**: Specifically syntactical dependencies such that one individual form depends upon another.
- **Feature**: A kind of information of a certain word. A list can be found here[2] .
- **Field**: Parts of word lines. The possible fields are ID, FORM, LEMMA, UPOS, XPOS, FEATS, HEAD, DEPREL, DEPS, MISC and all the features individually. They are explained in detail here[3] .
- **Form**: A part of a sentence, whether it be a word or a punctuation symbol.
- **Help Page**: A web page explaining how to use the tool and possible values for features.
- **Index**: A form's order in the sentence (also called Word Index) or a sentence's order in the `conllu` file.
- **Lemma**: The base form of a set of words (e.g. break for the set {break, breaks, breaking, broke}).
- **Note**: Text written for a specific sentence that's to be used to explain the annotation choices.
- **Phrase**: A word or a few words that make sense together that may not make sense by themselves individually.

[1] <http://anthology.aclweb.org/W/W06/W06-2920.pdf>
[2] <https://universaldependencies.org/u/feat/index.html>
[3] <https://universaldependencies.org/format.html>

- **Row**: A form or lemma shown together with its features on the table.
- **Sentence**: One word or several words put together that can be divided into its lemmas to be annotated.
- **Table**: A rectangle divided by rows and columns showing the sentence being annotated, and its fields and forms in tabular form.
- **Tree**: A tree-like graph showing the dependencies annotated by the user for the current sentence.
- **Universal Dependencies**: A framework for annotating grammars of various natural languages. Also known as *UD*.
- **User**: A linguist annotating sentences through the tool.
- **Word**: A string of characters allowed for a certain natural language with no spaces that makes sense together with other words in a sentence.

4.2. Software Requirements Document (SRS)

4.2.1. Functional

1. User
    1. Accounts
        1. User shall be able to sign up or log in.
    2. Annotation
        1. Navigation
            1. User shall be able to select a sentence to be annotated.
            2. User shall be able to navigate to the preceding or succeeding sentences.
            3. User shall be able to navigate to a sentence by index.
            4. User shall be able to customize the annotation table by
                1. showing or hiding columns of the table.
                2. adding or deleting a row of the table, if the change is found to be valid.
        2. Information
            1. User shall be able to see the sentence being annotated at all times.

    2. User shall be able to see a dependency graph of a sentence.
    3. User shall be able to take notes during annotation.
  3. Modifications
    1. User shall be able to annotate a sentence by
      1. navigating to a specific sentence.
      2. modifying the categories and features corresponding to the fields of the CoNLL-U format in tabular format.
    2. User should be able to undo or redo the modifications as much as they want.
    3. User shall be able to reset the fields of the current sentence to the state they were in before the user opened the sentence.
3. Access
  1. User shall be able to access the application through a web browser.
  2. User shall be able to access a help page, related to annotating in general and keyboard shortcuts.
4. Search
  1. User should be able to search the `conllu` files or previous annotations by lemmas, phrases, dependencies & categories.
  2. User should be able to search using regular expressions, if chosen.

2. System
  1. Settings
    1. System should store settings & preferences of a user for use in a subsequent session.
    2. System shall store an annotator's sentences uploaded to the server & annotations done.
  2. Annotation
    1. System shall validate the annotations and check for errors in dependencies.
    2. System shall provide helpful error messages.
  3. Keyboard shortcuts
    1. System shall provide keyboard shortcuts for most tasks.

2. System shall provide autocompleted results for the fields being edited.

4. Database

    1. System shall provide a database to store user information.

4.2.2. Non-functional

1. Errors

    1. Errors given by the validation procedure shall comply with the CoNLL-U format of annotation of UD[4] .

[4] <https://universaldependencies.org/format.html>

5. IMPLEMENTATION

Web application development framework *Django* [10] and Web API toolkit *Django REST Framework* (DRF) [11] and Python [12] library *django-filter* [13] are being used to develop BOAT v2. PostgreSQL [14] is used to provide a database for the server-side applications of the software. This database is responsible for serving the entire data needed to operate the tool. During the implementation phase, decisions regarding models of the database have been directed towards making the tool's UI make use of the API for database queries and searches. The models reflect the UD format of sentences and annotations are saved as fields of word lines at the core.

Errors are displayed for annotators to see invalid edits in real time. They are checked and annotations validated according to the UD framework and the language provided. The scripts used for validation are the ones already provided by the framework as of 2022 [15].

Python library spaCy [16] is used to provide linear dependency graphs. Another JavaScript-based linear dependency graph [17] making use of *brat* [18] is used to provide graphs as well. The preferences of annotators may vary and giving them options in different parts of the screen is important.

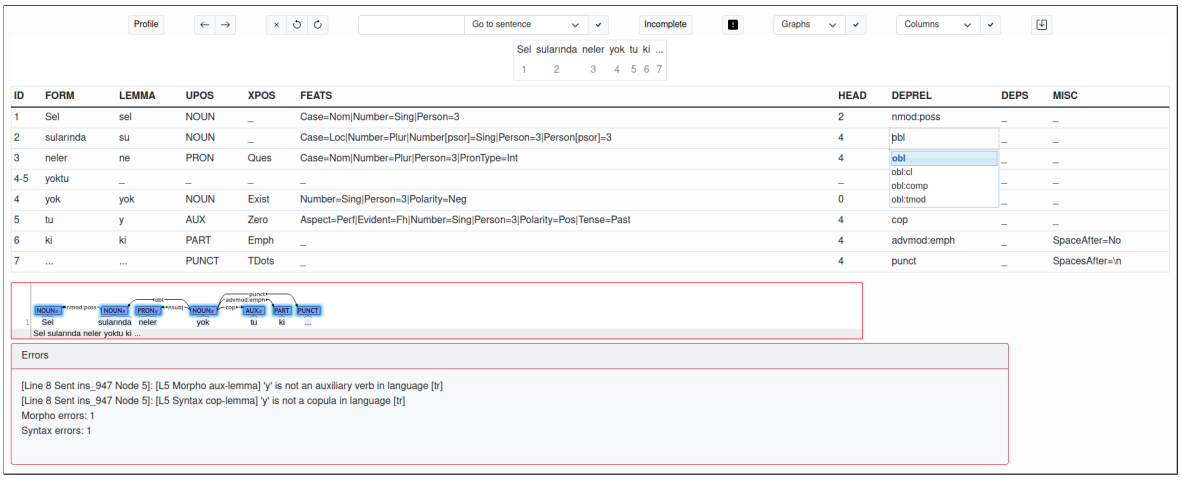

Figure 5.1. Current annotation screen for a sentence

5.0.1. Features

There are many new features and improvements in this tool, as well as the functionality that already existed in BoAT v1.

Treebanks: Treebanks are differentiated and each one has its own sets of sentences. Before uploading a CoNLL-U file, a treebank must be created beforehand.

Loading files: Instead of loading a specific file before every annotation session, an annotator is able to upload a CoNLL-U file for a treebank to the database. The file is parsed and checked for its format. It's rejected if incorrectly formatted, otherwise uploaded to the database. This way, other annotators working with the same treebank don't have to provide the same file.

Annotation view: The annotation page is very similar to BoAT v1, as seen in Figure 5.1. There is an annotation table for editing the word lines of CoNLL-U files. It also includes a dependency graph and an error card, both of which are in sync with the edits done on the table cells. The dependency graph of the initial tool and other 2 graphs have been added to this tool. The annotator has the choice to select a type of graph or none. The other 2 graphs, which are both horizontal and linear, have been selected due to space considerations. The error card displays errors for the current annotation, validating it via the UD validation script.

Network-enabled search: An important feature in this version is the ability to cross-check annotations by implementing a network for annotators where they can review the annotations done by other annotators and if they disagree on some parts, they can interact outside the tool. This can be helpful and a learning

experience for annotators.

For an actual example, the annotator, responsible for the BOUN Treebank's annotation, was annotating a sentence that had a Zodiac sign noun. Not being sure about how to annotate the noun's UPOS tag, she searched the CoNLL-U file in a text editor for similar cases and encountered two different ways nouns of Zodiac signs were annotated previously. Besides not helping how to choose a UPOS tag, this raises a consistency issue within the treebank as well. One of the ways had annotated it as NOUN in its UPOS tag, the other one had as PROPN. The annotator decided to use the one with more cases of annotation and proceeded to replace the inconsistent ones with the current decision. This case can be handled by a simple search of the database. We provide a search page and API in this tool where treebanks can be searched by *sent_id*s, *text*s, all the UD tags and treebank names. With this, an annotator can easily make the treebank more consistent and reduce their frustration.

Another example could be given by the various *-ki* morphemes in Turkish. In the sentence "Evdeki halılar yıkandı." (meaning *The rugs at home were washed.*), the *-ki* acts as an adjectivizer. In another sentence "Benim halılarım yün, Ayşeninkiler sentetik." (meaning *My rugs are woolen. Ayşe's are synthetic.*), it is pronominal. An annotator might not recall how a specific *-ki* morpheme should be annotated, which can be remedied with a simple search. Annotators have informed us that such cases occur frequently when annotating MRLs.

Also one other thing the inter-annotator agreement allows us is the possibility to see some anomalies in the Turkish part of the validation of the UD framework. For example, if

a sentence were annotated a way by many annotators but the UD validation script were finding it invalid, this might indicate the UD validation were lacking in this respect of the Turkish language. Some modifications might be necessary and there could be a case for a proposal of change.

Annotation status: We also provide a feature where an annotation has a status regarding it completeness. There are 3 different statuses and they are cycled through by the annotator in the annotation view. Status of an annotation is also shown in the search view, helping to select an appropriate case. Also there is another view where an annotator can list their completed, drafted or incomplete annotations, helping to keep track of what annotations have already been completed.

6. RESULTS

Initial feedback of the prototype is positive. The constant feedback-development cycle has positive impact on the design, which is aimed to be user-oriented.

7. CONCLUSION AND DISCUSSION

7.1. Discussion

We wanted a tool capable of the functionalities in BOAT V1 and also offer a network for annotators to collaborate on the same treebank. We decided on a web application with an API for this purpose. For the web framework, Django and extensible search capabilities of DRF served our purpose well.

Doğal Dil İşleme Platformu ("Natural Language Processing Platform", DIP) of our department have various NLP tools. The tools mostly output results based on input without a user interface. This is the first tool that has a UI in the platform and accordingly we needed to take some action regarding the integration of this novel tool. Dockerizing the application will allow us to easily integrate it into the DIP platform of the department [7].

Interviews with and feedback from annotators that had used BOAT V1 through the annotation of the BOUN Treebank and also had used other tools have been very helpful. They are shaping every part of the tool, from designing the API to the annotation view in the UI part. It is an extensible tool and can be tailored for other use cases.

7.2. Conclusion

In this work, we aim to develop a web-based dependency annotation tool for the MRL Turkish with a focus on annotators' user experience towards creating quality treebanks. We performed in-depth interviews with annotators who were linguists to gather the requirements. A web-based annotation tool is being designed in accordance with modern software development tools and approaches. The tool is being developed with an API to enable alternative utilization of the annotations and annotation process. The tool will be containerized for easy delivery and deployment. We are encouraged by the development of this tool and anticipate its extensions as well as the development of

similar tools. This tool is in testing phase with encouraging feedback. It will be made available as a demo and a downloadable resource from the Boğaziçi University Natural Language Resources Platform [7].

8. REMAINING WORK

These points need to be taken care of:

(i) API design continues.

(ii) The tool should be dockerized.

(iii) After dockerization, tests will be done and bugs will surely surface. We will continue to receive feedback on our tool's prototype and further refine the tool.

(iv) A user manual needs to be written.

REFERENCES

1. Stenetorp, P., S. Pyysalo, G. Topić, T. Ohta, S. Ananiadou and J. Tsujii, "brat: a Web-based Tool for NLP-Assisted Text Annotation", *Proceedings of the Demonstrations Session at EACL 2012*, Association for Computational Linguistics, Avignon, France, April 2012.

2. Attardi, G., "DgAnnotator", , 2022, `http://medialab.di.unipi.it/Project/QA/Parser/DgAnnotator/`, [Online; last accessed 27 March 2022].

3. Türk, U., F. Atmaca and Şaziye Betül Özateş and Gözde Berk and Seyyit Talha Bedir and Abdullatif Köksal and Balkız Öztürk Başaran and Tunga Güngör and Arzucan Özgür, "Resources for Turkish Dependency Parsing: Introducing the BOUN Treebank and the BoAT Annotation Tool", , 2020.

4. Dependencies, U., "UD tools", , 2022, `https://universaldependencies.org/tools`, [Online; last accessed 22 April 2022].

5. Türk, U., F. Atmaca, Ş. B. Özateş, A. Köksal, B. Ozturk Basaran, T. Gungor and A. Özgür, "Turkish Treebanking: Unifying and Constructing Efforts", *Proceedings of the 13th Linguistic Annotation Workshop*, pp. 166–177, Association for Computational Linguistics, Florence, Italy, Aug. 2019, `https://www.aclweb.org/anthology/W19-4019`.

6. Türk, U., F. Atmaca and Şaziye Betül Özateş and Gözde Berk and Seyyit Talha Bedir and Abdullatif Köksal and Balkız Öztürk Başaran and Tunga Güngör and Arzucan Özgür, "UD Turkish BOUN", , 2020, `https://universaldependencies.org/tr/`, [Online; last accessed 27 March 2022].

7. University, B., "TABILAB Repository Home", , 2022, `https://nlp.cmpe.boun.edu.tr`, [Online; last accessed 24 April 2022].

8. Tyers, F. M., M. Sheyanova and J. N. Washington, "UD Annotatrix: An annotation tool for Universal Dependencies", *Proceedings of the 16th International Workshop on Treebanks and Linguistic Theories (TLT16)*, pp. 10–17, 2018.

9. Pamay, T., U. Sulubacak, D. Torunoğlu-Selamet and G. Eryiğit, "The Annotation Process of the ITU Web Treebank", *Proceedings of The 9th Linguistic Annotation Workshop*, pp. 95–101, Association for Computational Linguistics, Denver, Colorado, USA, Jun. 2015, https://aclanthology.org/W15-1610.

10. Foundation, D. S., "Django documentation — Django documentation — Django", , 2022, https://docs.djangoproject.com/en/4.0/, [Online; last accessed 25 March 2022].

11. Ltd., E. O., "Django REST framework", , 2022, https://www.django-rest-framework.org/, [Online; last accessed 27 March 2022].

12. Foundation, P. S., "Welcome to Python.org", , 2022, https://www.python.org/, [Online; last accessed 24 April 2022].

13. Alex Gaynor, C. G. *et al.*, "django-filter — django-filter 2.4.0 documentation", , 2022, https://docs.djangoproject.com/en/4.0/, [Online; last accessed 24 April 2022].

14. Group, T. P. G. D., "PostgreSQL", , 2022, https://www.postgresql.org/docs/, [Online; last accessed 27 March 2022].

15. Dependencies, U., "UniversalDependencies/tools: Various utilities for processing the data.", , 2022, https://github.com/UniversalDependencies/tools, [Online; last accessed 27 March 2022].

16. Explosion, "spaCy", , 2022, https://spacy.io/, [Online; last accessed 27 March 2022].

17. spyssalo, "conllu.js", , 2022, http://spyysalo.github.io/conllu.js/, [Online; last accessed 25 March 2022].

18. Stenetorp, P., G. Topić, S. Pyysalo, T. Ohta, J.-D. Kim and J. Tsujii, "BioNLP Shared Task 2011: Supporting Resources", *Proceedings of BioNLP Shared Task 2011 Workshop*, pp. 112–120, Association for Computational Linguistics, Portland, Oregon, USA, June 2011, http://www.aclweb.org/anthology/W11-1816.

APPENDIX A: DATA AVAILABILITY STATEMENT

The annotation tool will be made available as open source on GitLab. It will be dockerized and this will make it accessible and easy to reproduce.

APPENDIX B: BoAT v2 - A Web-Based Dependency Annotation Tool with Focus on Agglutinative Languages

Parts of the paper are here presented, being written for the conference AL-TNLP2022: The International Conference on Agglutinative Language Technologies as a challenge of Natural Language Processing. The paper submitted shall have changes.

B.1. Related Work

There are various tools for creating dependency annotations such as *brat* [1], *UD Annotatrix* [8] and *DgAnnotator* [2] These tools mostly rely on mouse-based user interaction, which tend to be inefficient for annotating MRLs.

There are also tools developed specifically for Turkish. The ITU Treebank Annotation Tool [9] is a tool written in Java that has several versions. It offers semi-automated support for annotators through analyzers for creating new datasets as well as correcting the existing Turkish treebanks. It mostly relies on mouse-based interactions and doesn't support UD.

BoAT v1 [5] is a standalone application that supports keyboard-based input while displaying the annotations as a table as well as a dependency graph. It also supports UD [4]. This tool was used to create the BOUN Treebank [3,6] – a manually annotated Turkish dependency treebank comprising 9,761 sentences. It also uses analyzers to process sentences. The feedback received from this effort indicates that the keyboard based input is much more efficient and takes less time for creating quality treebanks.

The work presented in this paper is based on the experiences from BoAT v1 with continuous feedback from an annotator. Like many tools, it supports UD, and it also supports keyword-based input to allow the annotation process to be performed without breaking the flow. Furthermore, the tool is developed as a web application with an

API to support ease of deployment and multi-annotator scenarios.

B.2. Requirements and Design

In-depth interviews and requirements elicitation with annotators that have annotated the BOUN Treebank, which has close to 10 thousand sentences, have been conducted to elicit the software requirements. This section will explain the requirements elicited during the aforementioned process and design decisions based on them.

B.2.1. Requirements

Search: An annotator should be able to search the treebank they are a part of to see other annotations done for a specific sentence. Annotators sometimes struggle with a specific sentence's annotation and want to be able to search for specific annotations linguistically similar to the one they are currently doing to make the treebank consistent within itself and reduce their cognitive load in general. They should be able to search the treebanks in a complex way, by several fields at the same time, such as UD tags, features or plain text.

Focus: The annotation process requires a great deal of attention, therefore, the main priority is to improve the user experience for an annotator so that they can more efficiently produce accurate annotations. Annotators use the tools sometimes for hours on end. Software should make the work easier by reducing distractions and automating tasks. One of the requirements we have gathered for facilitating the focus of an annotator is for the software to offer keyboard input for every task. An annotator should not have to use a mouse for any type of task. Offering keyboard shortcuts for most tasks is required for a keyboard-oriented

approach.

Autocompletion: Since annotators lose focus after some time naturally, automating and validation of entries are high priorities in this respect. Many UD tags have predefined sets of values. Autocompletion should be provided for such UD tags, reducing errors and increasing efficiency.

Clutter: Using the screen space in a compact manner is also important for focus and any part of the screen that's not interacted with frequently should be made compact. Considering this, the previous application's checkboxes, responsible for toggling columns of the table, have been converted into a dropdown menu, leaving more space for the annotation table where most of the annotator's focus is directed towards.

B.2.2. Design

API: An API for search and annotation functionalities is deemed a useful addition to the tool, which would make it possible for the tool to have greater variety of uses outside the tool's prime objective, which is a UI for annotation. In addition to serving the data for the UI generation, the API should also allow a 3rd party agent to use it to get the annotations of a particular treebank to use in their NLP tasks in a systematic way. This opens up many possibilities for the use of the tool beyond the annotator's UI experience.

Docker: Following good software practices, this tool should be dockerized and made available as open source. This means that the tool is is easily deployable and accessible.

B.3. Implementation

Web application development framework *Django* [10] and Web API toolkit *Django REST Framework* (DRF) [11] and Python [12] library *django-filter* [13] were used to develop BOAT V2. PostgreSQL [14] is used to provide a database for the server-side applications of the software. This database is responsible for serving the entire data needed to operate the tool. During the implementation phase, decisions regarding models of the database have been directed towards making the tool's UI make use of the API for database queries and searches. The models reflect the UD format of sentences and annotations are saved as fields of word lines at the core.

The annotation page has all the functionalities of BOAT V1 and more. Errors are displayed for annotators to see invalid edits in real time. They are checked and annotations validated according to the UD framework and the language provided. The scripts used for validation are the ones already provided by the framework as of 2022 [15].

Python library spaCy [16] is used to provide linear dependency graphs. Another JavaScript-based linear dependency graph [17] making use of *brat* [18] is used to provide graphs as well. The preferences of annotators may vary and giving them options in different parts of the screen is important.

B.3.1. Features

There are many new features and improvements in this tool, as well as the functionality that already existed in BOAT V1.

Treebanks: Treebanks are differentiated and each one has its own sets of sentences. Before uploading a CONLL-U file, a treebank must be created beforehand.

Loading files: Instead of loading a specific file before every annotation session, an annotator is able to upload a CONLL-U file for a treebank to the database. The file is parsed and checked

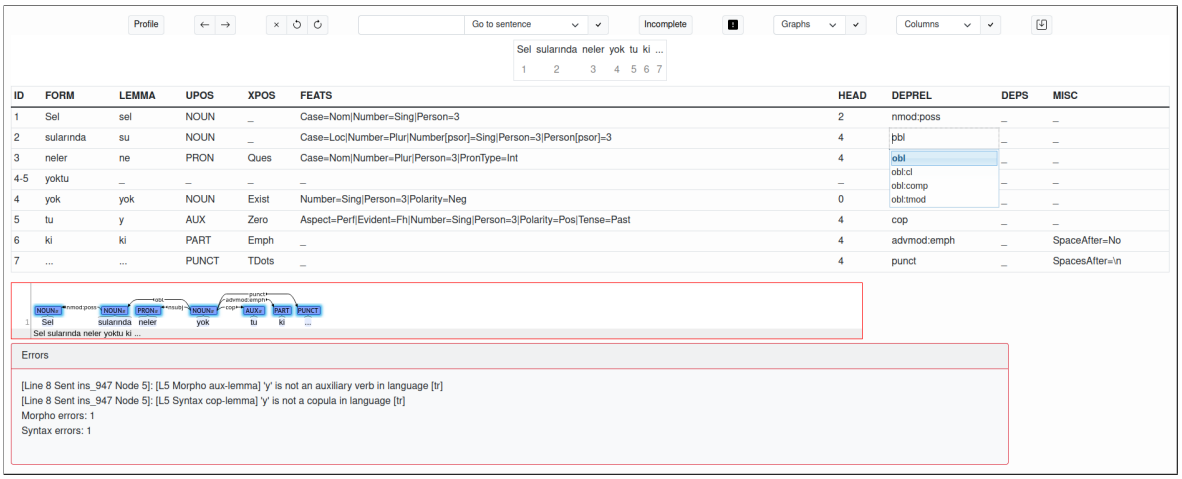

Figure B.1. The annotation screen for the sentence "Sel sularında neler yoktu ki...". The annotator is choosing an annotation for the DEPREL tag of the word "sularında". The valid alternatives pop up based on keystrokes and can be selected via use of arrows.

for its format. It's rejected if incorrectly formatted, otherwise uploaded to the database. This way, other annotators working with the same treebank don't have to provide the same file.

Annotation view: The annotation page is very similar to BOAT V1, as seen in Figure B.1. There is an annotation table for editing the word lines of CONLL-U files. It also includes a dependency graph and an error card, both of which are in sync with the edits done on the table cells. The dependency graph of the initial tool and other 2 graphs have been added to this tool. The annotator has the choice to select a type of graph or none. The other 2 graphs, which are both horizontal and linear, have been selected due to space considerations. The error card displays errors for the current annotation, validating it via the UD validation script.

Network-enabled search: An important feature in this version is the ability to cross-check annotations by implementing a network for annotators where they can review the annotations done by other

annotators and if they disagree on some parts, they can interact outside the tool. This can be helpful and a learning experience for annotators.

For an actual example, the annotator, responsible for the BOUN Treebank's annotation, was annotating a sentence that had a Zodiac sign noun. Not being sure about how to annotate the noun's UPOS tag, she searched the CoNLL-U file in a text editor for similar cases and encountered two different ways nouns of Zodiac signs were annotated previously. Besides not helping how to choose a UPOS tag, this raises a consistency issue within the treebank as well. One of the ways had annotated it as NOUN in its UPOS tag, the other one had as PROPN. The annotator decided to use the one with more cases of annotation and proceeded to replace the inconsistent ones with the current decision. This case can be handled by a simple search of the database. We provide a search page and API in this tool where treebanks can be searched by *sent_id*s, *text*s, all the UD tags and treebank names. With this, an annotator can easily make the treebank more consistent and reduce their frustration.

Another example could be given by the various *-ki* morphemes in Turkish. In the sentence "Evdeki halılar yıkandı." (meaning *The rugs at home were washed.*), the *-ki* acts as an adjectivizer. In another sentence "Benim halılarım yün, Ayşeninkiler sentetik." (meaning *My rugs are woolen. Ayşe's are synthetic.*), it is pronominal. An annotator might not recall how a specific *-ki* morpheme should be annotated, which can be remedied with a simple search. Annotators have informed us that such cases occur frequently when annotating MRLs.

Also one other thing the inter-annotator agreement allows

us is the possibility to see some anomalies in the Turkish part of the validation of the UD framework. For example, if a sentence were annotated a way by many annotators but the UD validation script were finding it invalid, this might indicate the UD validation were lacking in this respect of the Turkish language. Some modifications might be necessary and there could be a case for a proposal of change.

Annotation status: We also provide a feature where an annotation has a status regarding it completeness. There are 3 different statuses and they are cycled through by the annotator in the annotation view. Status of an annotation is also shown in the search view, helping to select an appropriate case. Also there is another view where an annotator can list their completed, drafted or incomplete annotations, helping to keep track of what annotations have already been completed.

B.4. Annotation Procedure

This section describes how an annotation is actually done for a sentence.

An annotator selects a sentence from a treebank, which has previously uploaded sentences coming from a CoNLL-U formatted file. An annotation table appears with the sentence parsed according to the UD format. They fill the cells of the annotation table's fields. During the annotation, an annotator can make use of dependency graphs, error cards, different columns and the search functionality. Dependency graphs are visual cues for how lemmas are dependent upon one another, using HEADs and DEPRELs. They can choose 3 different graphs one at a time or select to hide them completely. Different graphs show the same information with a horizontal or vertical tree. As for errors, they are helpful reminders, coming from a validation script published on the UD's GitHub repository. Also, different columns of the annotation table can be toggled to be shown or hidden. Other than all the columns for every UD tag, there are extra columns for every feature for the FEATS tag. They can also use the search functionality

to search for previous annotations by other annotators or themselves, using various fields (*text*, FEATS, various tags, etc.) for the query. By checking similar annotations, an annotator can ensure consistency and don't lose focus by trying manual methods for the same information. When an annotation is done, they select the annotation to be "Complete".

B.5. Discussion

We wanted a tool capable of the functionalities in BOAT V1 and also offer a network for annotators to collaborate on the same treebank. We decided on a web application with an API for this purpose. For the web framework, Django and extensible search capabilities of DRF served our purpose well.

Doğal Dil İşleme Platformu ("Natural Language Processing Platform") of Bogazici University's Computer Engineering Department have various NLP tools. The tools mostly output results based on input without a user interface. This is the first tool that has a UI in the platform and accordingly we needed to take some action regarding the integration of this novel tool. Dockerizing the application will allow us to easily integrate it into the DIP platform of the department [7].

Interviews with and feedback from annotators that had used BOAT V1 through the annotation of the BOUN Treebank and also had used other tools have been very helpful. They shaped every part of the tool, from designing the API to the annotation view in the UI part. It is an extensible tool and can be tailored for other use cases.

B.6. Conclusion

In this work, we aimed to develop a web-based dependency annotation tool for the MRL Turkish with a focus on annotators' user experience towards creating quality treebanks. We performed in-depth interviews with annotators who were linguists to gather the requirements. A web-based annotation tool was designed in accordance with modern software development tools and approaches. The tool was developed with an

API to enable alternative utilization of the annotations and annotation process. The tool is containerized for easy delivery and deployment. We are encouraged by the development of this tool and anticipate its extensions as well as the development of similar tools. This tool is in testing phase with encouraging feedback. It will be made available as a demo and a downloadable resource from the Boğaziçi University Natural Language Resources Platform [7].