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

Salih Furkan Akkurt[1], Büşra Marşan[2] and Susan Uskudarli[1]

[1] *Department of Computer Engineering, Boğaziçi University, İstanbul, Turkey*
[2] *Department of Linguistics, Boğaziçi University, İstanbul, Turkey*

Abstract

The value of quality treebanks is steadily increasing due to the crucial role they play in the development of natural language processing tools. The creation of such treebanks is enormously labor-intensive and time-consuming. Especially when the size of treebanks is considered, tools that support the annotation process are essential. Various annotation tools have been proposed, however, they are often not suitable for agglutinative languages such as Turkish. BOAT-v1 was developed for annotating dependency relations and was subsequently used to create the manually annotated BOUN Treebank (UD_Turkish-BOUN). In this work, we report on the design and implementation of a dependency annotation tool (BOAT-v2) based on the experiences gained from the use of BOAT-v1, which revealed several opportunities for improvement. BOAT-v2 is a multi-user and web-based dependency annotation tool that is designed with a focus on the annotator user experience to yield valid annotations. The main objectives of the tool are to: (1) support creating valid and consistent annotations with increased speed, (2) significantly improve the user experience of the annotator, (3) support collaboration among annotators, and (4) provide an open-source and easily deployable web-based annotation tool with an API to benefit the scientific community. This paper discusses the requirements elicitation, design, and implementation of BOAT-v2 along with examples.

Keywords

natural language processing, linguistic annotation, annotation tool, web application, dependency parsing, Universal Dependencies

1. Introduction

Treebanks are important resources in the development of natural language processing tools. Quality tools need treebanks that are manually annotated by linguistic experts. This is especially true for agglutinative languages due to their complex morphologies. The creation of such treebanks is highly labor-intensive and time-consuming due to the meticulous attention required. Thus, tools that support this process are essential.

In recent years, there have been significant efforts to bridge the gap in data resources available for agglutinative low-resource languages. Dependency annotation involves annotating each token of a sentence with linguistically relevant values. CONLL-U [1] defines a set of linguistic tags for annotation purposes (i.e. UPOS to annotate core part-of-speech categories). For agglutinative languages, it is common to provide annotations for most tags. Moreover, the FEATS tag is typically

ALTNLP2022: The International Conference on Agglutinative Language Technologies as a challenge of Natural Language Processing, June 6-8, 2022, Koper, Slovenia
furkan.akkurt@boun.edu.tr (S. F. Akkurt); busra.marsan@boun.edu.tr (B. Marşan); suzan.uskudarli@boun.edu.tr (S. Uskudarli)

annotated with multiple values. The effort required to annotate agglutinative languages is significantly higher due to the complex morphology of such languages. Annotation tools with drag-drop and mouse-based interfaces, while may seem appealing, are not well suited for agglutinative languages as they require alternating among input modalities that disrupts the flow.

BOAT-v1 [2] is an annotation tool that was developed to support dependency annotation of MRLs to produce treebanks compliant with the Universal Dependencies framework [1]. It is a standalone application developed using Python and Qt for the user interface. Its main input modality is via the keyboard at the request of the annotators. It was used to create the BOUN Treebank [3, 2, 4] – a manually annotated Turkish dependency treebank comprising thousands of sentences. This experience revealed several points of improvement for such annotation tools. The main takeaway was a much better understanding of the time, effort, cognitive load, and extra information requirements of the annotation process. Improvements regarding these aspects should, consequently, produce higher quality data resources.

This work presents a dependency annotation tool (BOAT-v2) that has been designed based on the experience with BOAT-v1. The design and implementation of the tool aimed to: (1) support creating valid and consistent annotations with increased speed, (2) significantly improve the user experience of the annotator, (3) allow collaboration among annotators during the annotation process, and (4) provide an open-source and easily deployable web-based annotation tool with an API to benefit the scientific community. The development started with requirements elicitation, for which earlier experiences and in-depth interviews with an experienced annotator were taken into account.

BOAT-v2 is a web-based collaborative dependency annotation tool that focuses on the user experience of annotators. The current prototype is being evaluated with positive feedback. This feedback indicated that the ability to collaborate within the tool is needed to increase the efficiency of multi-annotator treebank creation. In light of the feedback, we developed a web application that supports multiple users to enable a collaborative environment for annotations. It has an application programming interface (API) for programmatic access and extensibility. Several user experience improvements were implemented to enhance the flow of an annotation session. The tool is dockerized to facilitate accessibility and ease of deployment. The final version will be made available on Boğaziçi University's NLP platform [5] and provided as an open-source resource.

The main contributions of this work are:

- Design of a dependency annotation tool that is based on requirements elicited from experienced annotators who are linguists,
- The development of a tool that takes into account the annotator experience to improve resulting annotations,
- Support for multi-users to provide individual spaces for annotations, computation of inter-annotator agreements, and other potential collaboration,
- The development of a web-based annotation tool based on a supporting API, and
- Packaging of the tool to support easy access by virtualizing it using Docker [6] and providing the code as an open-source resource.

The remainder of this paper is organized as follows: Section 2 presents related work, Section 3 describes the requirements and design, Section 4 presents the implementation and new features, Section 5 presents a use case of annotation, and Section 6 provides a discussion along with our conclusions.

2. Related Work

Annotation tools may be characterized in terms of the support they provide for languages, user interface modalities, standards, and multiple annotators. Adherence to standards is recommended to get the most benefit from the annotated data sources. Universal Dependencies [1] is the most widely accepted standard for dependency annotations. It is an actively growing standard that intends to cover all languages. The support for agglutinative languages is evolving.

Several dependency annotation tools have been proposed such as *brat* [7], *UD Annotatrix* [8] and *DgAnnotator* [9, 10]. These tools mostly rely on mouse-based user interaction, which is inefficient for annotating agglutinative languages due to the need for annotation of numerous features for most tokens.

The ITU Treebank Annotation Tool [11] is developed for Turkish. It is written in Java as a standalone tool and 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 the UD framework.

BOAT-v1 [3] is a standalone annotation tool for treebanks compatible with the UD framework [1]. Although mouse-based interaction is supported, for almost all functionality, keyboard input is supported in accordance with annotator demands. This tool was used to create the BOUN Treebank [2, 4] – a manually annotated Turkish dependency treebank comprising nearly ten thousand sentences. It also uses analyzers to process treebanks with the CONLL-U format of UD.

WebAnno [12] is a web-based annotation tool that supports multiple users. To annotate features of a token, it requires several mouse clicks which is impractical for MRLs. It does not display a dependency graph like BOAT-v1 and BOAT-v2. It presents several sentences simultaneously, which can be distracting.

The work presented in this paper (BOAT-v2) is based on the experiences gained during the use of BOAT-v1. Like its predecessor and many other tools, it supports UD. The primary interaction modality is the keyboard to enable a flow state. It is a web application with a backend that provides an API. A database is used for persisting the annotations created by various annotators for various treebanks. It provides rich search functionality that supports collaborative annotations. It is an open-source resource that is containerized for ease of distribution and deployment.

3. Requirements and Design

Requirements elicitation and validation meetings were held with annotators who worked on the BOUN Treebank involving thousands of sentences to identify the requirements. The main requirements that emerged are:

Collaborative annotation: The sheer size of treebanks necessitates that they are annotated via multiple annotators. The tool must support multiple annotators working on the same

treebank. The ability to share annotation experiences is vital for reference and consistency. Hence, the tool must provide means for registering multiple annotators and keeping track of their respective annotations.

Search: While annotating a sentence, annotators may need to refer to previous annotations for guidance. An annotator should be able to search a treebank according to surface forms, and importantly, according to linguistic features. They should be able to perform complex searches that involve several criteria such as UD tags, individual features of the FEATS tag, and surface forms. This requirement reduces the cognitive load as well as supports consistency among annotations.

Keyboard-oriented input: Annotations of agglutinative languages require annotation of many features for individual tokens in comparison to analytical languages like English. While drag-and-drop interfaces can work well for analytical languages, they don't for MRLs as they require more toggles between mouse and keyboard interactions which is frustrating. Keyboard-based interaction must be offered for all possible tasks.

Support for Sentence annotation: To support the speed and accurate annotations, *autocompletion* should be supported whenever possible. Due to typologies of and syncretism being generally higher in MRLs, automatic parsing of such treebanks is difficult and they more often than not fail to create valid parsings of tokens. Thus, refining automatic parsing is important for agglutinative languages, which requires splitting of lemmas. Words in MRLs tend to have morphemes stacked on roots. Annotators must be able to refine/correct automatically parsed entries, which for agglutinative languages includes *splitting of lemmas*.

Use of screen real estate and customization: Annotators annotate numerous sentences every session. For agglutinative languages, sentences tend to be long and complicated. The view of such sentences can cover a considerable part of the screen. The annotator must focus on the sentence being annotated, the lemmas, and all the features, which requires much concentration. The annotation process must not involve scrolling, if possible. For very long sentences, this may not be possible; however, the sentence being annotated should never go out of vision. Furthermore, each annotator has their unique preferences for how they annotate a sentence. They must have some control regarding showing or hiding certain information according to their preferences. Overall efficient use of the screen real estate is important to convey the context needed to annotate long sentences.

Given these requirements, we decided that a web-based application that supports multiple authenticated users would support a collaborative platform for annotators. The treebank and user annotations will persist in a database which makes managing the data and searches much more reliable and efficient. Also, an API is developed to support flexibility and extensibility. Finally, in addition to making the tool open source, we decided to containerize the application using Docker[6] to support its accessibility.

4. Implementation

The annotation tool is implemented using Python [13], the web application development framework *Django* [14] with the API framework *Django REST Framework* (DRF) [15]. The webpages use Bootstrap [16]. PostgreSQL [17] is used as a database. The models reflect the UD format of sentences. Most of the sentence annotation functionalities are similar to BOAT-v1. User input is validated according to UD and errors are reported on the annotation page. Three alternative forms of dependency graph visualizations are supported, two of which are compact horizontal to reduce the required screen real estate [18, 19].

The following features have been implemented to support the creation of valid annotations with increased speed in a collaborative manner:

Treebanks handling: Users can work on multiple treebanks. They can upload new treebanks as CONLL-U files. Treebanks and their annotations are stored in a database.

Sentence annotation: An annotator selects the sentence they want to annotate. The sentence annotation page is very similar to BOAT-v1. It consists of three main parts: (1) A table with rows for every token in the sentence and columns, which represent the UD tags of a sentence (FORM, LEMMA, DEPREL, etc.), corresponding to their annotations; (2) the dependency graph of the sentence; and (3) errors from validation according to the UD framework. The dependency graph and errors are synchronized with the annotations. Several dependency graph presentations are supported to suit the annotator's preference. Vertical graphs can consume a significant amount of screen real estate, which can lead to loss of focus in long sentences that are common in agglutinative languages.

An annotator may need to stop the annotation of a sentence for some reason (i.e. complexity or external interruption). To capture the state of an annotation, a status is introduced whose values may be: "New", "Draft" and "Complete". The status of a sentence that has not been annotated is "New". An annotator can set the status to "Draft" or "Complete".

The annotator is able to perform almost all operations via keyboard action, based on the demand of the annotators (see Section 3 for details). Upon the experience of annotating a Turkish treebank, the annotators have reported being very pleased with the convenience and speed resulting from keyboard-based interaction.

Improved searching for reference and consistency: Users are able to search for previously annotated sentences in combinations of surface text, tags, and features. Without a good search feature, an annotator would have to manually search the CONLL-U file for relevant cases (e.g. how to annotate some surface form's UPOS tag). The surface form might have been inconsistently annotated, alas it is unlikely that a manual search would reveal this case. In such a case, the annotator would likely use the first encountered as a reference. The situation gets more complicated for syncretic morphemes such as *-ki* in Turkish. For example, in the sentence "Evdeki halılar yıkandı." (*The rugs at home were washed.*), the *-ki* acts as an adjectivizer. However, in "Benim halılarım yün, Ayşeninkiler sentetik." (*My rugs are woolen. Ayşe's are synthetic.*), it is pronominal. Searching for sentences where the *-ki* morpheme occurs via text search would be hopeless as there would be too many hits since they occur very frequently.

To facilitate effective searching, we have implemented search functionality based on

combinations of text and UD tags. Regular expression-based search is also supported. This feature supports annotators to share experiences, which consequently is expected to result in more accurate and consistent treebanks.

Inter-annotator agreement: The consistency of annotations among annotators is an indicator of the quality of the resulting resource. Inter-annotator agreement computes the consistency among annotators. Since this tool keeps track of annotator actions, such computations are straightforward.

5. Using BoAT-v2

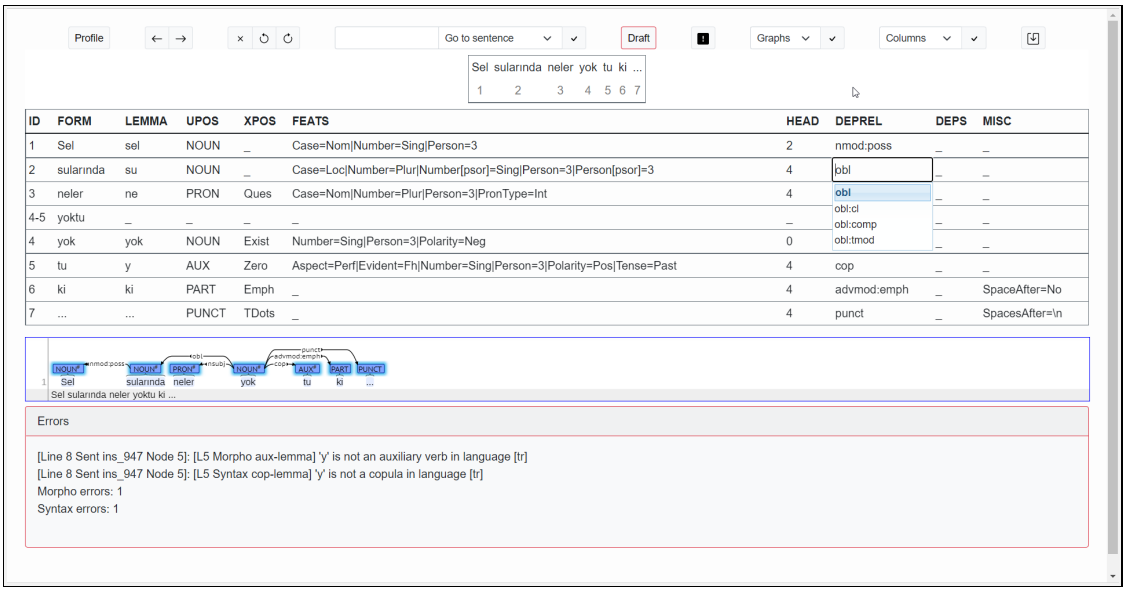

Figure 1: The annotation screen captured while the sentence "Sel sularında neler yoktu ki..." is being annotated. The DEPREL tag for the surface form "sularında" is being annotated by selecting among the valid alternatives that appear in the pop-up. Selections can be made with the use of arrow keys.

An annotator selects a sentence from a treebank. An annotation table appears with the sentence parsed according to the UD format. Each row corresponds to a token and its annotations. Figure 1 shows the annotation view while an annotator is annotating a Turkish sentence "Sel sularında neler yoktu ki..." (translation: *What wasn't in the flood waters...*).

An annotator can make use of dependency graphs, errors, and search during annotation. Dependency graphs are visual cues for how lemmas are dependent upon one another. Errors are helpful reminders compliant with UD.

Annotators can customize the columns and dependency graph in accordance with their preferences. When an annotation is finished, its status can be set to "Complete". The annotator can search for previously made annotations in combinations of text and feature values to refer to previously made annotations.

6. Discussion and Conclusions

BOAT-v2 aimed to extend the functionality of BOAT-v1 as a collaborative web-based application to support the annotation process based on previous experiences. We developed a web application that supports agglutinative languages as described in Section 4.

The implementation choices served our goals well. We believe that having experts in linguistics and experienced annotators in agglutinative treebank creation was instrumental in understanding the requirements and design process. We held numerous elicitation interviews and further meetings for clarifications and feedback requests.

We used modern software development tools and management practices during the development lifecycle of this tool. The development of an API enables various extensions of this tool and access to the treebanks. The containerization with Docker [6] has facilitated easy delivery and deployment. It will be made available on Boğaziçi University's NLP platform DIP [5] as a demo as well as an open-source resource. The availability of the tool as a user as well as a developer is valuable for future use and developments.

This tool is in the testing phase and has had encouraging early feedback. We are encouraged by the early responses to this tool and anticipate its extensions. We plan to conduct further tests with the linguistics department. We also see potentials for customizing this tool to support annotation of rare languages.

Acknowledgments

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

References

[1] Universal Dependencies, UD, 2022. URL: https://universaldependencies.org, [Online; last accessed 2 May 2022].

[2] U. Türk, F. Atmaca, Ş. B. Özateş, G. Berk, S. T. Bedir, A. Köksal, B. Ö. Başaran, T. Güngör, A. Özgür, Resources for turkish dependency parsing: Introducing the boun treebank and the boat annotation tool, Language Resources and Evaluation (2021) 1–49.

[3] U. Türk, F. Atmaca, Ş. B. Özateş, A. Köksal, B. Ozturk Basaran, T. Gungor, A. Özgür, Turkish treebanking: Unifying and constructing efforts, in: Proceedings of the 13th Linguistic Annotation Workshop, Association for Computational Linguistics, Florence, Italy, 2019, pp. 166–177. URL: https://www.aclweb.org/anthology/W19-4019. doi:10.18653/v1/W19-4019.

[4] U. Türk, F. Atmaca, 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. URL: https://universaldependencies.org/tr, [Online; last accessed 27 March 2022].

[5] Boğaziçi University, TABILAB Repository Home, 2022. URL: https://nlp.cmpe.boun.edu.tr, [Online; last accessed 24 April 2022].

[6] Docker Inc., Home - Docker, 2022. URL: https://www.docker.com, [Online; last accessed 2 May 2022].

[7] P. Stenetorp, S. Pyysalo, G. Topić, T. Ohta, S. Ananiadou, J. Tsujii, brat: a web-based tool for NLP-assisted text annotation, in: Proceedings of the Demonstrations Session at EACL 2012, Association for Computational Linguistics, Avignon, France, 2012.

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

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

[10] Universal Dependencies, UD tools, 2022. URL: https://universaldependencies.org/tools, [Online; last accessed 22 April 2022].

[11] T. Pamay, U. Sulubacak, D. Torunoğlu-Selamet, G. Eryiğit, The annotation process of the ITU web treebank, in: Proceedings of The 9th Linguistic Annotation Workshop, Association for Computational Linguistics, Denver, Colorado, USA, 2015, pp. 95–101. URL: https://aclanthology.org/W15-1610. doi:10.3115/v1/W15-1610.

[12] WebAnno, WebAnno - Documentation, 2022. URL: https://webanno.github.io/webanno/documentation, [Online; last accessed 5 May 2022].

[13] Python Software Foundation, Welcome to python.org, 2022. URL: https://www.python.org, [Online; last accessed 24 April 2022].

[14] Django Software Foundation, Django documentation, 2022. URL: https://docs.djangoproject.com/en/4.0, [Online; last accessed 25 March 2022].

[15] Encode OSS Ltd., Django REST framework, 2022. URL: https://www.django-rest-framework.org, [Online; last accessed 27 March 2022].

[16] Bootstrap, Bootstrap v5.1 Documentation, 2022. URL: https://getbootstrap.com/docs/5.1, [Online; last accessed 5 May 2022].

[17] The PostgreSQL Global Development Group, PostgreSQL, 2022. URL: https://www.postgresql.org/docs, [Online; last accessed 27 March 2022].

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

[19] S. Pyysalo, conllu.js, 2022. URL: http://spyysalo.github.io/conllu.js, [Online; last accessed 25 March 2022].