Proceedings of the 16th Linguistic Annotation Workshop (LAW-XVI) @LREC 2022, pages 79–84
Marseille, 24 June 2022

Introducing StarDust: A UD-based Dependency Annotation Tool

Arife Betül Yenice[♡], Neslihan Cesur[♡], Aslı Kuzgun[♡], Olcay Taner Yıldız[♢]
Starlang Yazılım Danışmanlık[♡], Özyeğin University[♢]
Istanbul, Turkey
{arife, neslihan, asli}@starlangyazilim.com, olcay.yildiz@ozyegin.edu.tr

Abstract

This paper aims to introduce StarDust, a new, open-source annotation tool designed for NLP studies. StarDust is designed specifically to be intuitive and simple for the annotators while also supporting the annotation of multiple languages with different morphological typologies, e.g. Turkish and English. This demonstration will mainly focus on our UD-based annotation tool for dependency syntax. Linked to a morphological analyzer, the tool can detect certain annotator mistakes and limit undesired dependency relations as well as offering annotators a quick and effective annotation process thanks to its new simple interface. Our tool can be downloaded from the Github.

Keywords: Dependency parsing, Annotation tool, Turkish

1. Introduction

With recent developments in Natural Language Processing (NLP) studies and tools such as parsers, the demand for datasets is constantly increasing. The quality of these corpora, which are used to train and evaluate parsers, mostly lies in an efficient annotation process. User-friendly and effective annotation tools enable human annotators to have a better and easier experience.

Our aim in creating StarDust is to develop a simple and easy-to-learn interface, which can be used by anyone with minimal instruction and regardless of prior experience. Our interface offers a multi-layered structure with many different tools for different purposes. These include tools for semantic and morphological analysis, dependency annotation and verb frame annotation.We also seek to minimize annotator errors and increase inter-annotator agreement by embedding the rules of Universal Dependencies (UD) (Nivre et al., 2016) annotation scheme as a restriction for possible head-dependent relations. In this paper we will firstly introduce related work and our motivation for a new tool and elaborate on the features of our interface and how it is used. Then, we will briefly talk about its implementation and some technical details. Finally, we will refer to the treebanks annotated using this tool.

2. Related Work and Motivation

There are currently several annotation tools for dependency annotation, all of which promise different advantages for different tasks. Before introducing StarDust, it is useful to give an overview of some of these tools and explain why they did not suffice to use for our goals personally. The one which is used for the UD documentation system is BRAT (Stenetorp et al., 2012), which is a web-based tool that requires user log in to do annotations. It has a good range of features; however, it was not ideal for us to opt for a web-based tool and it does not support word tokenization. The most recent one is Palmyra (Habash and D.Taji, 2020). It is suitable for morphologically rich languages and its ability to annotate various linguistic features are attractive. However, its dependency tree representation is not ideal for our purposes. Other annotation tools include Prodigy (Montani and Honnibal, 2018) which is a great tool for those who are experienced in the field and it is a desktop tool which is what we desire; however our aim in creating a new tool was to provide a beginner friendly, easy annotation process, therefore it did not suit our needs. ConlluEditor (Heinecke, 2019), WebAnno (de Castilho et al., 2016) , UD Annotatrix (Tyers et al., 2017), and Arborator (Gerdes, 2013) are some of the other tools used for annotations. WebAnno, UD Annotatrix and Arborator permit collaborative work and they are web-based tools. A full list of tools can be found on UD's website. [1] For the functions we needed and our desire for simplicity, we needed another tool.

We need an annotation tool that is both multifaceted and user-friendly for precise annotations in languages of different typologies such as Turkish, an agglutinative language with rich morphology and free word order. For this reason, we present StarDust, an open source annotation tool that aims to simplify the main editing window while only keeping the main information such as POS-tags and dependency relations. In doing so, we still made sure to offer some crucial functions such as editing and deleting tokens, finding and grouping tokens based on their initializer tags, and preventing invalid tags or tree structures. We have opted for a linear representation of the sentences instead of using tree representations to make the annotation more intuitive for inexperienced or first-time annotators. By linking a semi-automatic disambiguation tool to our dependency annotator, we wanted to make sure that human annotators can immediately correct the morphological analysis of tokens. Overall, this has increased the accuracy of the annotations and reduced the time spent for cor-

[1] https://universaldependencies.org/tools.html

Figure 1: A screenshot of the StarDust's morphological disambiguation layer

rection considerably. StarDust is efficient in annotating compounding languages like English as well as morphologically rich languages like Turkish. It can also be adapted to annotate other languages, if there is a morphological analyzer for the language we can intergrate.

3. Features

StarDust is a desktop annotation tool which can be used offline. To keep the interface clean, we have opted for a linear representation and used a layered architecture where only two layers are necessary for dependency annotation purposes. Nonetheless, extra layers could be implemented for more in-depth annotations. For dependency annotations, the first layer of the tool is for the morphological disambiguation layer and the second layer is dependency annotation layer .

The annotators navigate between layers freely during the annotation process while keeping both layers uncluttered yet functional. Even though morphological features cannot be edited in dependency annotation layer, switching back to morphological disambiguation layer to fix the errors is possible. This prevents any errors from accumulating and yield higher accuracy in annotations. Opting for a desktop tool rather than a web based tool comes with a few challenges such as collaboration issues and ensuring inter-annotator agreement. These can be easily overcome if the annotators are working online. They can easily see other annotators working on the data and check with their annotations. However, when working offline, the annotator can only see the last synchronised version, which can still be helpful with their decisions but it would require them to check their annotations when everything is synchronised. In the future versions, we can optimize it for web platforms to make collaborative work easier.

Changes are saved automatically for each token; however, the change history can be seen in Dropbox and the token can be reverted back to its original form. This might be useful for those who wish to see the original token for any reason. A change history feature that is internal to the tool might be added in the future. If there are changes made to a specific word form - POS tag combination at any level, you can go back to morphological analyzer and list all the annotations from View Annotations tab (See Figure 2) to apply the changes to all of the same word form - POS Tag combination in the dataset consistently. The features of both layers will be demonstrated in the following sections.

4. Pos Tagging

Typically, the burden of annotating each word, its root, its features, and its POS-tag and falls on the dependency annotator. Thanks to our multi-layered interface, all this information is provided by the morphological analyzer. All words are parsed automatically, and to enable alternations, all possible features that might constitute their internal morphology are listed for annotators to choose from. It uses a rule-based method to parse the words and their features.

Figure 1 shows a screenshot of our interface. The automatic annotation checkbox on the top of the program automatically parses the word in all possible derivations. The annotators can also select the relevant derivation manually. This configuration leads to a consistent annotation for the morphological analyses and saves time. For the morphological annotation, our current editor makes use of the format introduced by (Oflazer, 1994). However, we are working on converting this format into the CoNLL-U format. Each sentence is stored as a different file, therefore, the annotators can work on different files simultaneously. The arrows on the control panel allows the annotators to go back and forth between files in different distances.

Our morphological analyzer tool is suited for the analysis of languages with different morphological typologies such as English, an analytic language, and Turkish, an agglutinative language. Agglutinative languages need more in-depth analyses in morphology for dependency parsing because the grammar of such languages is encoded at the word level rather than at sentence level. Also, in agglutinating derivation systems the same word forms can have multiple different meanings depending on their internal morphology. Previous dependency tools developed for agglutinative languages such as Hungarian and Turkish address this fact (Zsibrita et al., 2013); (Türk et al., 2020). At this layer of StarDust, morphological disambiguation for Turkish and Pos tagging for English are automatically derived.

4.1. Turkish Morphological Disambiguation

Due to its high reliance on affixation, Turkish word forms bear great complexity; thus, Turkish morphology needs to be analyzed and disambiguated before

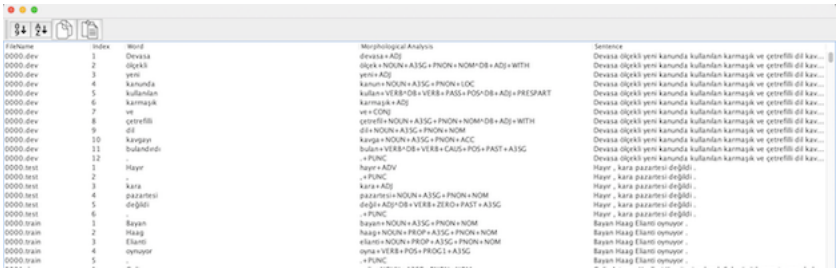

Figure 2: Viewing All Annotations in Morphological Analyzer

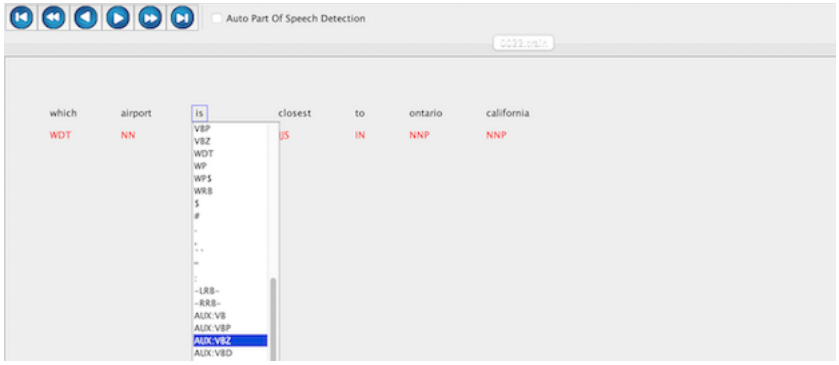

Figure 3: A screenshot of the StarDust's Pos-tagging layer for English

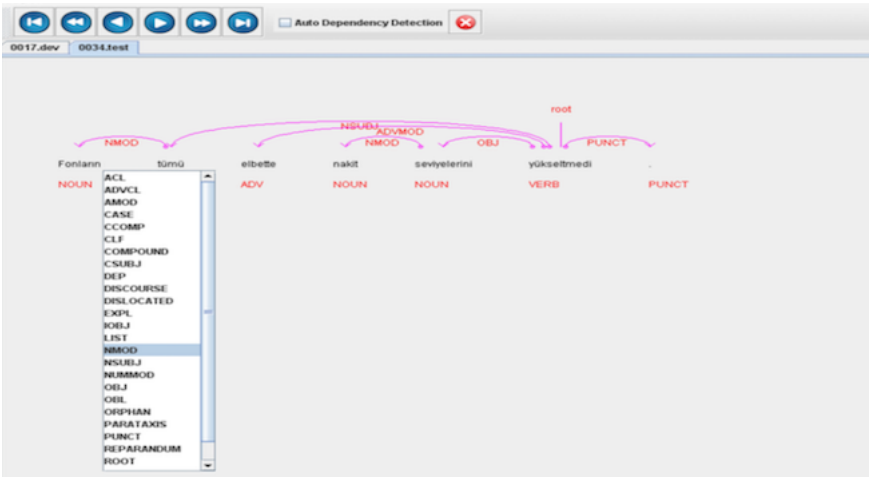

Figure 4: Dependency Layer of StarDust's Interface

dependency relations can be established. Our method for automatic annotation of words finds the roots of the words and annotates their POS-tags and features. For each word, it takes the word with its annotation layers and sets the corresponding morphological layers. For each annotation layer, the method divides the layers and derives all possible internal morphologies layer by layer. If the language is Turkish, it directly calls Universal Dependency POS tags of the parse. Next, it returns the features of the Universal Dependency relations of the word. For instance all possible derivations of "kalemi" are shown in Figure 1. As can be seen, the internal morphology of Turkish is so complicated that one word form with minimal affixation can have three different meanings encoded in it, and our method can derive them automatically and successfully.

4.2. English Pos Tagging

English is an analytical language; therefore, internal morphologies of the words in English are simpler. English mostly depends on functional words instead of multiple suffixes on words. Thus, our method follows different rules for English words. Our method takes the word and returns its annotation layers and sets corresponding layers. When setting the dependency layer of the word, If the language is English, it returns Universal Dependency POS tags based on the Penn tag of the word from Penn Treebank Project. For instance, if

the Penn tag of the word is “VB”, “VBD” or “VBN”, it returns the POS tag “VERB”. There is not always a one-to-one mapping between POS tags and Penn tags, in these cases, extended versions of POS tags are used. Then, it returns the features of the UD relations of the word. After this, dependency features are established based on the Penn tag of the word (Figure 3)

5. The Dependency Annotation Layer

The front-end of our dependency editor helps the annotators to visualize the head-dependent relationships between the words. Its simplistic structure has enabled some untrained annotators to make annotations by only watching basic annotation videos [2] created by our team. Figure 4 shows our interface. The buttons on the top left corner in here make it possible to browse the data by skipping different amounts of files. The most embedded arrows skip one file, the two headed arrows skip 10 files, and the ones with a vertical stroke skip 100 files at once. Each word token has its POS tag shown below. This information comes from the previous layer. The annotators click on a word and drag the cursor from the dependent to its head. Upon this, a box pops up displaying all possible dependency tags between those two words based on the UD annotation framework. Possible dependency tags are listed for annotators to choose according to the frequency rate of their use with the chosen dependent and head. Another feature is displaying examples for each dependency tag listed. (See Figure 5) When the annotator holds the cursor on the tag, a few example sentences highlighting the words between which the tag is used appear. These features are available for both Turkish and English tags. Once the relevant dependency tag is chosen, the relation is shown with arrows. The little circle on the tail of the arrow marks the head word of the constituent while the head of the arrow indicates the dependent word. The Automatic Dependency Annotation checkbox automatically annotates some certain structures by using the information that comes from the previous layer. In the most basic two layered structure, nominal modifiers, punctuations, and the root nodes can be annotated automatically with the information provided from the morphological analyzer. It is possible to edit the word tokens during the annotation. Ctrl+click to the word token enables the annotators to edit or delete the words individually. Whenever a change is made for a token during the dependency annotation, the changed token is also updated on the other levels of the editor. However, the annotator should make sure to check and if needed update the morphological analysis of the token. Since the tool was designed to be used with a translated corpus, the annotators are able to see the original sentence that corresponds to the sentence being annotated as represented in the bottom left corner of the Figure 4 . StarDust allows the users to see all the annotations sorted as in Figure 7. The annotations can be sorted according to the alphabetical order of the word tokens, or they can be sorted according to the number of the data types. This function mainly helps the annotators to check the annotations. Another feature of the StarDust is the error warnings. It has been mentioned that this editor is designed for UD style annotations. The editor prevents the annotators from doing any annotation that conflicts with the UD annotation framework by giving an error as shown in Figure 6. In Figure 6 the black cautions shows the earlier mistakes made in the annotation by stating which node causes the error.

[2]The videos are available here (in Turkish): `https://tinyurl.com/y2jq5lrw`

6. Implementation

Our tool is compatible with all platforms on Desktop (Windows, OS, Linux) with its implementation in Java. We have not opted for a browser-based system to ensure that annotators can work offline, when needed. For the projects which were carried out so far, the edited files and .jar editors were all kept in Dropbox, ensuring immediate synchronization of the data. The back-end is supported in many other languages such as Python, Cython, C#, Swift, Javascript, and C++. The back-end of the morphological analyzer has been discussed briefly in Section 4. Our morphological analyzer follows different methods for Turkish and English. It creates the roots, the features and POS tags of each token following from rules written specific for the specific language. There are rules and methods for the morphological analysis; for example, methods for possesives, plurality etc. For inter-annotator agreement and gold standard annotations, we implemented rule-based methods of restrictions and controls that are independent of the language and follow rules of Universal Dependencies (UD). These controls can check the dependency relations and rule out the impossible ones.
The front-end is currently only available in Java but it can be adapted to any desired language. Before dependency annotations are done, all other annotated layers are stored on each token within a .txt file. Each text file contains one sentence or phrase. In order to store dependency annotations, these .txt files are processed by Annotated Sentence Library and transformed into CoNLL-u format. This library also contains information from Turkish WordNet and FrameNet, which can provide automatic annotation for certain compounds.

7. Annotated Corpora

So far, our tool has been used in five different Turkish Treebanks, and one English Treebank project. The annotated sentences of the Turkish FrameNet Project are already available on Github.[3] **English Atis** : This treebank is taken from English ATIS corpus[4]. It con-

[3]`https://github.com/UniversalDependencies/UD_Turkish-FrameNet`

[4]`https://github.com/howl-anderson/ATIS_dataset/blob/master/README.en-US.md`

Figure 5: Example sentences feature in dependency layer

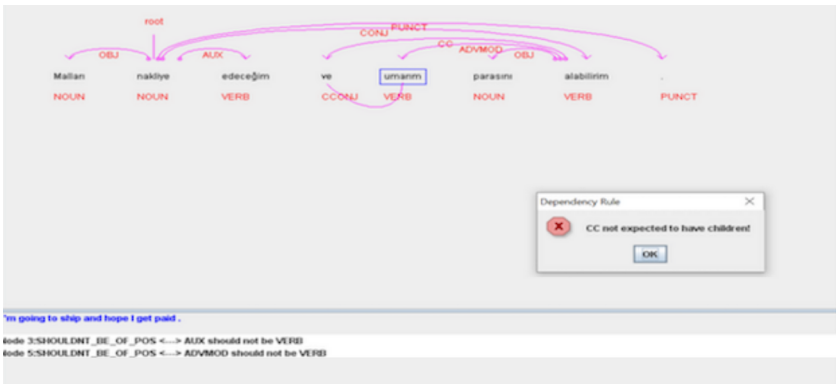

Figure 6: UD-based errors and limitations

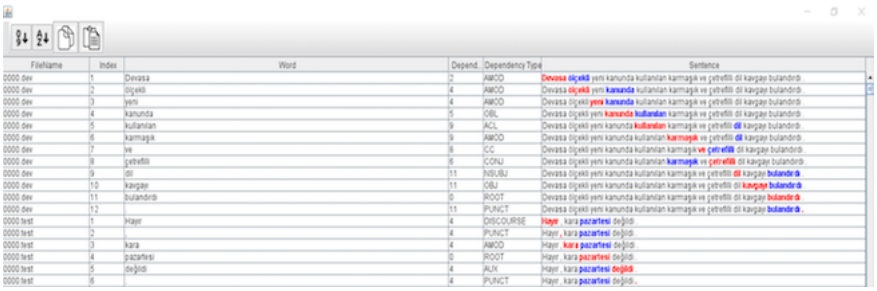

Figure 7: Show Annotations feature of dependency layer

sists of 5,432 sentences[5]. **Turkish Atis** : This treebank is a translation of English ATIS corpus. It consists of 5,432 sentences[6]. **Turkish FrameNet Project**: In this project, about 2,500 example sentences were manually annotated with the help of our annotation tool (Marsan et al., 2021). **Turkish WordNet** : This project contains 18,700 example sentences from the Turkish Wordnet (Bakay et al., 2021). **Turkish Penn TreeBank** : The Turkish version of Penn Treebank (Kuzgun et al., 2020) includes the translation of 17,000 sentences retrieved from the original Penn Treebank (Marcus et al., 1993). **Tourism** : This is a domain-specific corpus that contains around 20,000 sentences. (Arıcan et al., 2021).

[5] https://github.com/UniversalDependencies/UD_English-Atis

[6] https://github.com/UniversalDependencies/UD_Turkish-Atis

8. Conclusion and Future Work

Overall, the convenience of our annotation tool lies in its approachable interface with the basic functions. Its easy-to-learn and easy-to-use interface makes it usable by anyone without lengthy instructions or learning periods. The tool could be improved by embedding the morphological layer into the dependency annotator to facilitate any necessary changes in POS tags, without navigating back to morphological analyzer. Currently, any mistake done during the annotation process can be arranged simply by rearranging the arrows. So far, there has been no reported problems about this but an "undo" button could also be implemented in the future. For more feedback from the annotators, we plan to conduct user studies in the future. Moreover, even though we have used a linear representation for the sentences to make the annotation more intuitive, an option to view the sentences as tree representations could be added. This would allow different annotators with different preferences to choose the view that suits them.

9. Bibliographical References

Arıcan, B. N., Ozçelik, M., Aslan, D. B., Sarmıs, E., Parlar, S., and Yıldız, O. T. (2021). Creating domain dependent turkish wordnet and sentinet.

Bakay, O., Ergelen, O., Sarmis, E., Yildirim, S., Kocabalcioglu, A., Arican, B., Ozcelik, M., Saniyar, E., Kuyrukcu, O., Avar, B., et al. (2021). Turkish wordnet kenet. In *Proceedings of GWC 2021*.

de Castilho, R. E., Mujdricza-Maydt, E., Yimam, S. M., Hartmann, S., Gurevych, I., Frank, A., and Biemann, C. (2016). A web-based tool for the integrated annotation of semantic and syntactic structures. In *Proceedings of the Workshop on Language Technology Resources and Tools for Digital Humanities (LT4DH)*, pages 76–84.

Gerdes, K. (2013). Collaborative dependency annotation. In *Proceedings of the second international conference on dependency linguistics (DepLing 2013)*, pages 88–97.

Habash, N. and D.Taji. (2020). Palmyra 2.0: A configurable multilingual platform independent tool for morphology and syntax annotation. In *In Proceedings of Universal Dependencies Workshop (UDW) 2020*.

Heinecke, J. (2019). Conllueditor: a fully graphical editor for universal dependencies treebank files. In *Proceedings of the Third Workshop on Universal Dependencies (UDW, SyntaxFest 2019)*, pages 87–93.

Kuzgun, A., Cesur, N., Arıcan, B. N., Özçelik, M., Marşan, B., Kara, N., Aslan, D. B., and Yıldız, O. T. (2020). On building the largest and cross-linguistic turkish dependency corpus. In *2020 Innovations in Intelligent Systems and Applications Conference (ASYU)*, pages 1–6. IEEE.

Marcus, M., Santorini, B., and Marcinkiewicz, M. A. (1993). Building a large annotated corpus of english: The penn treebank.

Marsan, B., Kara, N., Ozcelik, M., Arican, B. N., Cesur, N., Kuzgun, A., Saniyar, E., Kuyrukcu, O., and Yıldız, O. T. (2021). Building the Turkish FrameNet. In *Proceedings of GWC 2021*.

Montani, I. and Honnibal, M. (2018). Prodigy:a new annotation tool for radically efficient machineteaching. Artificial Intelligence.

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

Oflazer, K. (1994). Two-level description of turkish morphology. *Literary and linguistic computing*, 9(2):137–148.

Stenetorp, P., Pyysalo, S., Topić, G., Ohta, T., Ananiadou, S., and Tsujii, J. (2012). Brat: a web-based tool for nlp-assisted text annotation. In *Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 102–107.

Türk, U., Atmaca, F., Özateş, Ş. B., Berk, G., Bedir, S. T., Köksal, A., Başaran, B. Ö., Güngör, T., and Özgür, A. (2020). Resources for turkish dependency parsing: Introducing the boun treebank and the boat annotation tool. *arXiv preprint arXiv:2002.10416*.

Tyers, F., Sheyanova, M., and Washington, J. (2017). Ud annotatrix: An annotation tool for universal dependencies. In *Proceedings of the 16th International Workshop on Treebanks and Linguistic Theories*, pages 10–17.

Zsibrita, J., Vincze, V., and Farkas, R. (2013). magyarlanc: A tool for morphological and dependency parsing of hungarian. In *Proceedings of the International Conference Recent Advances in Natural Language Processing RANLP 2013*, pages 763–771.