From Constituency to UD-Style Dependency: Building the First Conversion Tool of Turkish

Aslı Kuzgun
Starlang Yazılım Danışmanlık
asli@starlangyazilim.com

Oğuz Kerem Yıldız
Ahmet Keleşoğlu High School
oguz@starlangyazilim.com

Neslihan Cesur
Starlang Yazılım Danışmanlık
nesli@starlangyazilim.com

Büşra Marşan
Starlang Yazılım Danışmanlık
busra@starlangyazilim.com

Arife Betül Yenice
Starlang Yazılım Danışmanlık
arife@starlangyazilim.com

Ezgi Sanıyar
Starlang Yazılım Danışmanlık
ezgi@starlangyazilim.com

Oğuzhan Kuyrukçu
Starlang Yazılım Danışmanlık
oguzhan@starlangyazilim.com

Bilge Nas Arıcan
Starlang Yazılım Danışmanlık
bilge@starlangyazilim.com

Olcay Taner Yıldız
Özyeğin University
olcay.yildiz@ozyegin.edu.tr

Abstract

This paper deliberates on the process of building the first constituency-to-dependency conversion tool of Turkish[1]. The starting point of this work is a previous study in which 10,000 phrase structure trees were manually transformed into Turkish from the original Penn Treebank corpus. Within the scope of this project, these Turkish phrase structure trees were automatically converted into UD-style dependency structures, using both a rule-based algorithm and a machine learning algorithm specific to the requirements of the Turkish language. The results of both algorithms were compared and the machine learning approach proved to be more accurate than the rule-based algorithm. The output was revised by a team of linguists. The refined versions were taken as gold standard annotations for the evaluation of the algorithms. In addition to its contribution to the UD Project with a large dataset of 10,000 Turkish dependency trees, this project also fulfills the important gap of a Turkish conversion tool, enabling the quick compilation of dependency corpora which can be used for the training of better dependency parsers.

1 Introduction

There are two types of annotated treebanks used in natural language processing (NLP) systems: constituency treebanks and dependency treebanks. They are used to represent syntactic relations, argument structures, and other hierarchical relations. Constituency treebanks display groups of phrases as trees and dependency treebanks marks head-dependent relations for each item. Today, dependency parsers are expected to adapt to new data from a variety of genres. The Universal Dependencies (UD) Project (Nivre et al., 2020) provides convenient corpora for these parsers. However, this kind of an adaptation requires a huge amount of data that is not suitable for manual annotation. Thus, the existent constituency treebank corpora are being automatically converted into dependency treebanks as the need for dependency treebanks increased in order to train dependency parsers (Marneffe et al., 2006); (Johansson and Nugues, 2007); (Choi and Palmer, 2012).

Constituency to dependency conversion requires a big corpus and a coherent annotation framework. UD helps to provide a framework for annotated treebanks to be used in different languages. There have been several attempts made to represent Turkish language in this project and to create a common framework in Turkish language treebank studies (Sulubacak et al., 2016); (Oflazer et al., 2003); (Çöltekin, 2015) (Kuzgun et al., 2020). However, Turkish language

[1] https://github.com/StarlangSoftware/StructureConverter

Proceedings of Recent Advances in Natural Language Processing, pages 761–769
Sep 1–3, 2021.
https://doi.org/10.26615/978-954-452-072-4_087

has flexible word order as a consequence of its agglutinative morphology. Such features complicates the annotation scheme for Turkish and renders the language harder to parse while giving rise to problems such as non-projectivity. In their work, Türk et al. (2019) edited the IMST Treebank (Sulubacak et al., 2016) according to the UD framework considering the needs of the Turkish language but its corpus does not provide a parallel treebank in terms of a cross-linguistic attribution to the literature and even this refinement could not help the IMST Treebank to adapt the new versions of the UD framework. In spite of all the attempts and refinements which have been mentioned above, Turkish still lags behind compared to similar languages in the area. For instance, a dependency conversion tool is yet to be developed for Turkish while there are studies present for similar languages such as Hungarian (Simkó et al., 2014).

We have applied our conversion algorithms to the biggest Turkish Constituency Treebank which is the Turkish counterpart of the original Penn TreeBank corpus (Marcus et al., 2002), the first annotated treebank of English that set the gold standard for other treebanks. There have been several studies done for the parallel constituency treebank creation in Turkish. Yıldız et al. (2014) developed an automatic translation process to translate trees from the Penn Treebank. They created a tool that learned from the 5000 manually translated sentences and offered possible translations. Yıldız et al. (2015) fine tuned this work by deleting empty projections, and rearranging the two word constructions that were appearing in one node due to the cross-linguistic consequences. In their study, Bakay et al. (2019) translated sentences using a tree-based approach to deal with the long distance dependencies that are not present in the corresponding English sentences. These long distance dependencies occur as a result of the difference between English and Turkish in having a fixed word order and a flexible word order respectively.

The Turkish Penn Constituency Treebank used in this conversion study, however, is distinct from the previous parallel Turkish constituency treebank studies as the most recent one is translated and annotated manually. It consists of 10.000 annotated sentences[2] translated from the original version of the Penn Treebank (Kara et al., 2020). In order to make the translations suitable for the Turkish language, a team of linguists deleted, changed some tags from the original Penn Treebank, and also they introduced new tags that are necessary for a better syntactic representation of the Turkish language. We preferred using The Turkish Penn Treebank in our conversion study as it offered an accurate and less complicated constituency treebank that is necessary for the demands of our conversion tool.

Our main objective in this study is to introduce the first Turkish constituency to dependency converter by using the parallel Turkish constituency treebank introduced above. Another contribution of this work to the literature comes from its cross-linguistic nature. Since most of the conversion studies used the Penn Treebank corpus (Marneffe et al., 2006); (Johansson and Nugues, 2007); (Choi and Palmer, 2012), our choice of employing a translated corpus results in a Turkish-English parallel constituency to dependency conversion study.

All the previous converter studies employed a rule-based approach. Johansson and Nugues (2007) differs from these studies in that they included extra labels to include semantic information and to handle syntactic phenomena such as topicalization, clefting, gapping, and so on. They put the semantic annotations in the Penn Treebank in use to achieve a more semantically rich dependency treebank. However, as the labels were more complicated, the employment of semantic information decreased the parsing accuracy. The Stanford dependency approach, on the other hand, did not use these function labels that were manually annotated over the Penn Treebank corpus. In their study, Choi and Palmer (2012) combined the Stanford approach with the CoNLL dependency approach which uses different labels and relation rules. This combination allowed them to achieve better accuracy without eliminating the semantic information encoded in the Penn Treebank corpus. In our study, the translated version of the Penn Treebank does not include these semantic labels, and therefore our conversion rules does not employ these semantic information as in the Stanford approach.

[2]https://github.com/olcaytaner/TurkishAnnotatedTreeBank-15

Hierarchy	Tag List
1	VP, NOMP
2	S, NP, ADJP, ADVP
3	PP
4	DP, NUM
5	QP, NEG, CONJP, INTJ, WP

Table 1: Hierarchical head table.

2 Conversion Process

2.1 Turkish Penn Treebank as Input

Using the Penn Treebank as an input for converter algorithms is a very common procedure. So as not to break this tradition, we have also decided to use the Turkish version of Penn Treebank. As mentioned above, this widely-used constituency treebank was first adapted to Turkish by Yıldız et al. (2014). We found the latest version of the dataset to be simplistic and yet an adequate representation of the Turkish language, especially thanks to its being manually edited by a team of linguists. The tree in Figure 1 illustrates the final version of a sentence in the Turkish Penn Treebank corpus.

The input we used follows the main principles of the Penn Treebank annotation. However, there are differences in some aspects. In their study, Kara et al. (2020) excluded the bar level projections such as NONE, VBN, NNP, IN, NSBJ-1. They aimed for a more minimal approach to reduce the number of branches while keeping the necessary constituency information. In addition to this difference, this version has additional tags such as NEG, NOMP, and QP. In conversion, we used word level non-terminal nodes while benefiting from the headedness of Turkish language.

2.2 Algorithms

2.2.1 Rule-based Approach

While constructing an algorithm for constituency to dependency converters, there are some basic and simple steps that should be followed. The real hardship lies in creating specific rules for the needs of each language. Turkish has a rich morphology, with many suffixes and different cases carrying semantic or syntactic information. This made it compulsory to include in the algorithm some information from a morphological analyzer. As our input dataset already had each word morphologically analyzed through a semi-automatic process (Yıldız et al., 2019), we had the opportunity to extract information from their analyses.

The first problem to tackle is determining a head for each phrase in the phrase structure trees. Whereas in phrase structure trees the head of a constituent is not transparent due to its non-binary representation and linking style, in dependency structures the head and its dependents are always marked. This requires a head to be determined for each constituent. The most efficient way to achieve this is by constructing a hierarchical table which allocates a number to each type of phrase and listing them according to their prioritization. Table 1 contains the hierarchical head table used for the Turkish Penn Treebank. Hierarchical head table allocates priority numbers (from 1 to 5, with 1 having the top priority and 5 having the lowest) to different phrase labels. In a node with more than one daughter, the one with the lowest number becomes the head. In a case where phrases with the same priority number are sisters, the one on the right becomes the head.

The hierarchy of the tags are decided by considering the behaviors of the UD tags and the head final structure of Turkish. For instance words with VP/NOMP labels are always the root of the sentence. Therefore, they have the highest priority for being labeled as the head. The following group is the adjunct category. When these types of words are linked to the root, they are always dependents. The third category is the post positions and they can be linked to a word that is in the first or in the second category in the hierarchy, either case it is always a dependent. The fourth category consists of determiners and numeral modifiers. These are always dependents to noun phrases, therefore it is crucial that they are lower in the hierarchy. The last category of the hierarchy consists of the tags that are always dependents.

This table allows for an accurate head determination, especially in exceptional cases. Due to its head-final nature, in Turkish, heads of phrases generally appear on the right, thus at the end of the phrase. To write such an algorithm would be perfectly simple if it weren't for some exceptions such as postpositions and negation. For instance, in the phrase *senin için* ("for you") the postposition *için* is the rightmost word which should be the head according to a head-final algorithm. However, in UD annotations, the adpositions should be dependents of NPs, so the leftmost element *senin* (NP) should be head and the postposition

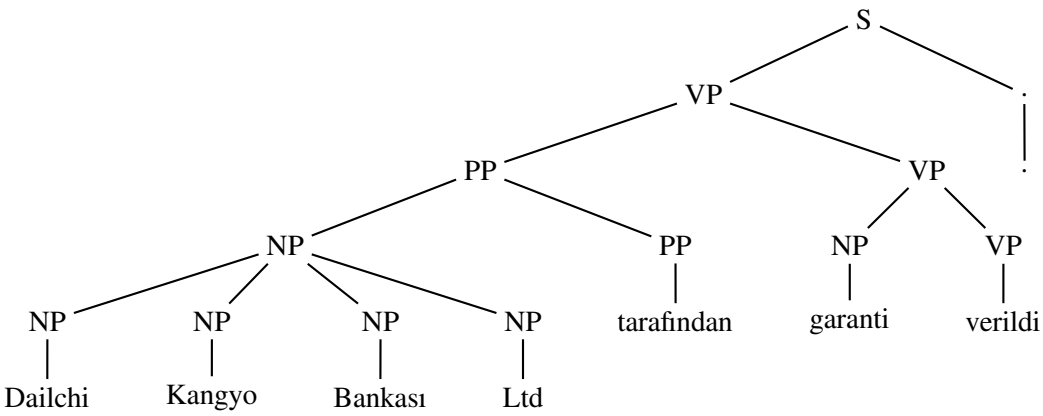

Figure 1: Representation of the sentence achieved by new methodology

için should be linked to the NP with the tag CASE. The solution to this lies in the hierarchical head table, where NP is given a priority value of 2 and PP a priority value of 3. In this case, wherever an NP is found as a sister to a PP, the NP has the priority to be the head. Another example to this can be the labels NOMP (nominal predicate) and NEG (negation), which appear as sisters with this respective order. However, the negation, even though it appears on the right of the main predicate, cannot be the head according to UD annotations. By allocating the number 5 to NEG and 1 to NOMP, we make sure that whenever NOMP appears with NEG, the algorithm will choose NOMP as the head and NEG as its dependent.

After the heads were determined, certain rules had to be written to link structures and phrase labels of the constituency trees with UD annotation tags. Figure 2 is a list which illustrates some of these conversion rules. There are 21 conversion rules in total, excluding the DEP rule which is used whenever the listed conversion rules fail to apply. The rules take the POS information of the head and dependent nodes in the constituency tree along with the morphological information of the word tokens. For instance, the sentence *"hızlı koşuyor"* meaning "s/he is running fast." consists of an adverb and a verb. The tag hierarchy marks the verb *"koşuyor"* as the head and the adverb *"hızlı"* as the dependent. Then, the rules in Figure 2 apply and determine the relation between them. There are two labeling options for the relation between an adverb and a verb: ADVCL or ADVMOD. The difference between the two relations is that the ADVCL relation is used when the dependent is clausal, otherwise the adverb relation is labeled with ADVMOD. The clausal adverbials have a verbal root in their morphology. Therefore, the algorithm questions the existence of a verbal root in an adverb to determine the correct dependency relation. The adverb *"hızlı"* does not have a verbal root, so it is marked as ADVMOD. If it had a verbal root as in *"hızlanarak"*, meaning "in an increasing speed", then, it would create an adverbial clause. Therefore, the algorithm would mark it as the ADVCL.

2.2.2 Machine Learning Approach

In training a machine learning algorithm, there are two main issues to consider: determining the head and how it is linked to its dependents, and determining the dependency relations between the head and its dependents. In order to achieve this, first, the algorithm sees a tree as a whole. Then, it finds the correct constituents by scanning the nodes of the tree. Once a node is matched, it cannot be overwritten by the upper nodes. This procedure continues until there are no more nodes left in the tree. Once the algorithm has established the constituents, we use an oracle which is the mechanism that determines the correct linkings between the heads and their dependents as well as the dependency relation types between them. We use two different oracles to find these constituents. One of them is the rule-based basic oracle and the other is a classifier oracle which is used as a machine learning model. After the constituents have been determined, they are put together to form a sentence.

The classifier oracle forms an instance list by taking the Parts of Speech (POS) tags of the words in the constituent and assigns them an index according to its position in the constituent. In another iteration, the head of the constituent is not included in the ranking. For instance, in a con-

```
if dependentNode == ADVP:
    if isVerbal(dependentWord):
        return ADVCL
    elif isNominal(dependentWord):
        return NMOD
    else:
        return ADVMOD
elif dependentNode == ADJP:
    if headNode == NP:
        if isVerbal(dependentWord):
            return ACL
        else:
            return AMOD
    return ADVMOD
elif dependentNode == PP:
    if headNode == NP:
        return CASE
    else:
        if isNominal(dependentWord):
            return NMOD
        else:
            return ADVMOD
elif dependentNode == NP:
    if isCompound(dependentWord)
    and isCompound(headWord):
        return COMPOUND
    if headNode == NP:
        if isProper(dependentWord)
        or isProper(headWord):
            return FLAT
        return NMOD
    elif headNode == VP:
        if isAccOrNom(dependentWord):
            return OBJ
        return OBL
elif dependentNode == S:
    if headNode == VP:
        return CCOMP
elif dependentNode == DP:
    return DET
elif dependentNode == NUM:
    return NUMMOD
elif dependentNode == INTJ:
    return DISCOURSE
elif dependentNode == NEG:
    return NEG
elif dependentNode == CONJP:
    return CC
```

Figure 2: Python rules for linking structures and phrase labels of the constituency trees with UD annotation tags

Length	Percentage
2	1.32
3	6.11
4	6.77
5	8.86
6	12.57
7	11.66

Table 2: Error rates according to constituency length (Random Forest, ensemble size = 50)

stituent that includes an adjective and a noun, we have a 2 word long instance as the following: ADJ NOUN 0 1. Here, the ADJ marks the POS tag of the first word of the constituent, the NOUN marks the second word, and the numbers mark their indexes. The instance lists are classified according to the length of the constituent. Later, the index of the head of the constituent is converted into class information. Thus, an instance like "ADJ NOUN 0 1" turns into ADJ NOUN 1 given that the head of this pair is the NOUN.

Subsequently, we have tested decision tree, Naive Bayes, KNN, and Random Forest models by using constituency and dependency treebanks of the same corpus. Running the models separately and as ensemble resulted in similar error rates. Thus, we decided to use the Random Forest which had the least errors. We tested the Random Forest model with several parameters and chose the one that minimizes the error rates. Table 2 shows error rates according to the constituency length. The error rates get higher as the constituency length increases. Therefore, we did not include constituents bigger than 7 words length and applied a rule-based algorithm for them.

As for the second problem of choosing dependency relations, the first task was to assign the POS-tags and class information flags of heads and its dependents for the entirety of the instance list. For instance, in a constituent with an adjective and a noun, the instance would be ADJ NOUN AMOD. Here, the first two tags mark the POS tag of the two words in linear order, and the last one marks the type of the dependency relation by considering the information provided by the POS tags.

However, the first results achieved by the random forest model were not satisfying. So as to achieve better performance, each word, including the head, was evaluated according to the following questions concerning their morphological

structure:
What is the POS-tag of the root?
Does the word have an ablative tag?
Does the word have a dative tag?
Does the word have a genitive tag?
Does the word have a nominative tag?
Does the word have an accusative tag?
Does the word have a proper tag?
Does the first word in the constituent share the same sense ID in Turkish WordNet with the head of the constituent?

These questions result in a sequence of true/-false outputs along with the POS-tag information of the head of the constituent and its dependents, in addition to the class information provided at the end of the instance. The performance was satisfying after the implementation of the questions. The following string illustrates an instance of an ADJ-NOUN constituent which consists of two words. The string begins with the POS tag information of the first word of the constituent and it is followed by the true/false outputs relevant to this word. Then, it continues with the POS tag information of the second constituent and its applicable true/false outputs. The last "false" output stands for the last question and shows that the two words do not share the same word ID. The ADJP and NOMP marks the node names of the constituents in the phrase tree. Lastly, the AMOD label shows the relation between the two words.
ADJ false false false false false false
NOUN false false false true false false
false
ADJP NOMP
AMOD

2.3 Setting the Gold Standard

In order to evaluate the algorithm, we needed gold-standard dependency annotations for all 10,000 sentences in the Penn treebank corpus. This work was carried out by a team of 7 linguists and a UD-style annotation scheme was adopted. The annotated data is freely available in the UD web page[3]. The tags were adapted to the requirements of Turkish, so that they would successfully capture important distinctions while avoiding over-specifications which hinder the training of any prospective dependency parsers.

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

Metric	Percentage
LAS	66.92
UAS	85.32
LA	72.32

Table 3: LAS, UAS, and LA parsing scores of the rule-based algorithm

The annotations for the gold-standard were carried out with the help of an original interface. The sentences appear in a linear way and the annotator can mark the head-dependent relations by drawing an arrow from the dependent towards the head and choosing the suitable tag from a pop-up window. After the manual tagging process, the annotations are checked and corrected on another window where the sentences appear as a list. The fact that they can be grouped according to their tags or the annotated words enables an effective and quick control process for the annotators.

3 Evaluation

3.1 Evaluation Tool

We used three metrics in order to measure the performance of our converter: Labelled Attachment Score (LAS), unlabelled attachment score (UAS), and label accuracy (LA). These are the metrics that show the highest correlation with the human judgements respectively (Plank et al., 2015). The LAS score showed us the percentage of the words that are connected to the right head with the right dependency label. The UAS score determined the accuracy of our head finding algorithm by displaying the proportion of the words that are connected to their correct head. Lastly, the LA score showed us the proportion of the words with the correct tags.

3.2 Results & Discussion

Table 3 and 4 show the LAS, UAS and LA scores of our converted dependency treebank for the two approaches. The scores are calculated according to the amount of changes made by the human annotators on the converted dependency treebank. It should also be noted that the output sentences were all projective in both conversion methods.

Since this is the first converter for the Turkish language, we were unable to compare our results with other Turkish conversion tools. However, the results show that the rule-based approach lags behind the conversion studies held in other languages. The relatively lower LA score of the

Metric	Percentage
LAS	72.83
UAS	84.82
LA	78.18

Table 4: LAS, UAS, and LA parsing scores of the Machine Learning algorithm

rule-based approach reflects the fact that confusion is existent between similar tags which cause debates even among the human annotators. On the other hand, the relatively higher UAS score of the rule-based algorithm shows that our head finding algorithm works better than the overall performance of the converter. This might be the outcome of the corpus we used since it consisted of formal sentences with mostly canonical order. A less formal corpus could give different results since the amount of scrambling would be different.

Nevertheless, the rule-based approach is not as successful as the machine learning algorithm. This is an interesting finding given that the machine learning approach resulted in better scores even though the rule-based algorithm was specifically tailored for the requirements of the Turkish Language. The results indicate that dividing the conversion problem into two phases as head finding and determining the dependency relation is a better methodology for conversion studies because it not only results in better scores but also provides a more adaptable algorithm that can be used for other languages. This supports the idea that there is confusion between similar tags, and this affects the performance of the converter. The main reasons for the confusions have been explained below.

Tables 5 and 6 shows the confusion matrices of the 13 most frequent tags used in the data for rule-based and machine learning algorithms respectively. From this table, some of our mistakes can be explained as a result of the co-occurrence of the same lexical items with different functions in Turkish. For instance, the word *yumuşak* meaning "soft" can occur as an adjective as in *yumuşak yastık* meaning "soft pillow", as an adverb modifying a verb in a sentence like *"Yumuşak davrandı."* meaning "S/he acted softly.". It can also function as an adjectival predicate to be labelled as a root as in *"Onun kumaşı yumuşak."* meaning "Its cloth is soft". Notice that the three words with different functions do not show any morphological variance on the word. These kinds of structures affect the labeling of ADVMOD, AMOD, OBL, and NMOD the most. Table 5 illustrates this effect. For example, 5.6% of OBL had been mistakenly labelled as ADVMOD and another 34.8% of them labelled as NMOD. Considering the syntactic position helps to differentiate between these tags to some extent, especially in OBL-ADVMOD cases. However, understanding the function of morphologically identical structures is still an important issue because of the flexible word order. Assumptions on syntactic position often cause problems as a result of the flexible word order of Turkish language. An example is the NSUBJ tag. The table shows that 25.3% of the total mistaken tags for NSUBJ consists of the NMOD tags and another 16.4% consists of the OBJ tags. Some of these mistakes can be explained by scrambling. In the canonical word order, we would expect the first NP of a sentence in Turkish to be the subject as in the treebank all the complements of the verb were put under the predicate node. However, there were sentences where other NP structures occurred before the subject. As a result, both flexible word order and the morphological reasons mentioned above gave rise to the confusion among these tags in the rule-based algorithm.

Table 6 shows that the machine learning algorithm prioritizes the learning of the more frequent tags such as AMOD, ADVMOD, NMOD and so on in that it shows higher performance than the rule-based algorithm over these tags. This proves that the machine learning approach does not tackle the same confusion problems as the rule-based model stated above. On the other hand, there are areas in which the machine learning algorithm performed poorer than the rule-based algorithm such as identifying FLAT and CASE relations. These tags share the common property of reversed attachment type. This uniqueness of these relations can easily be applied in rule-based algorithm but apparently it causes confusion to the machine learning models.

All of these examples show that machine learning approach is helpful to increase the parsing accuracy for languages with rich morphology and flexible word order since it takes each sample as an input, and evaluates the patterns according to the relevant information provided by the data rather than trying to fit the data into a structure. Both the rule-based model and the machine learning model

Tag	Advmod	Amod	Case	Cc	Ccomp	Compd	Det	Flat	Nmod	Nsubj	Nummod	Obj	Obl
Advmod	**81.3**	1.7	1.9	3.0	0.1	0.1	0.7	0.0	3.9	0.2	1.4	0.3	2.0
Amod	11.7	**74.4**	1.4	0.0	0.0	0.1	0.8	0.2	4.7	0.1	0.5	0.2	0.1
Case	12.4	1.0	**75.1**	6.8	0.1	0.1	0.1	0.0	1.0	0.5	0.0	0.0	1.0
Cc	0.5	0.1	1.4	**97.5**	0.0	0.0	0.0	0.0	0.1	0.0	0.1	0.0	0.0
Ccomp	3.6	1.1	0.1	0.5	**47.2**	0.5	0.0	0.1	7.1	1.5	0.1	25.4	8.3
Compd	6.5	0.6	0.0	0.3	0.2	**50.2**	1.3	1.3	16.1	0.0	14.8	4.8	2.5
Det	0.5	0.3	0.0	0.3	0.0	0.0	**97.3**	0.0	0.5	0.3	0.4	0.0	0.1
Flat	0.0	0.8	0.5	0.2	0.1	0.7	0.0	**32.6**	34.0	13.3	0.8	3.1	0.9
Nmod	0.8	0.3	1.7	0.1	0.1	6.3	0.0	7.6	**73.5**	1.3	2.1	1.7	4.2
Nsubj	0.2	0.1	0.1	0.1	0.1	0.1	0.4	3.1	25.3	**52.8**	0.2	16.4	0.7
Nummod	0.2	0.1	0.0	0.1	0.0	0.0	1.0	0.0	1.4	0.0	**95.3**	0.1	1.6
Obj	0.9	0.0	0.2	0.0	0.9	3.2	0.0	0.7	27.3	5.1	0.3	**52.4**	8.2
Obl	5.6	0.3	6.8	0.2	0.4	0.8	0.1	1.1	34.8	1.1	2.9	4.1	**41.1**

Table 5: Confusion matrix of the most frequent tags in the rule-based algorithm

Tag	Advmod	Amod	Case	Cc	Ccomp	Compd	Det	Flat	Nmod	Nsubj	Nummod	Obj	Obl
Advmod	**80.6**	4.9	1.7	0.7	0.1	1.2	0.8	0.0	2.1	0.6	0.4	0.5	2.4
Amod	3.1	**82.6**	0.5	0.0	0.1	0.7	0.5	0.0	6.1	0.3	0.1	0.3	0.6
Case	10.1	1.2	**74.5**	1.4	0.3	0.1	0.0	0.0	4.2	0.1	0.0	0.1	4.5
Cc	2.9	0.1	2.5	**83.3**	0.0	0.2	0.0	0.0	4.6	0.3	0.0	0.0	0.0
Ccomp	1.5	0.7	0.1	0.2	**71.6**	0.8	0.0	0.0	3.4	1.6	0.0	8.1	4.6
Compd	3.3	1.4	0.2	0.0	0.3	**58.6**	1.1	0.0	12.7	0.6	9.3	7.1	3.5
Det	0.6	1.4	0.1	0.0	0.0	0.3	**94.2**	0.0	1.2	0.9	1.1	0.0	0.1
Flat	0.0	1.1	1.7	0.2	0.1	0.3	0.0	**1.4**	59.1	18.1	0.3	1.3	2.2
Nmod	0.9	0.9	1.5	0.0	0.2	1.0	0.0	0.2	**80.7**	2.5	1.5	3.2	6.1
Nsubj	0.3	0.2	0.1	0.0	0.1	0.2	0.1	0.1	7.7	**87.3**	0.1	2.1	0.7
Nummod	1.8	0.4	0.1	0.0	0.0	4.0	1.7	0.1	2.7	0.1	**82.5**	0.1	0.3
Obj	0.5	0.2	0.3	0.0	4.5	2.3	0.1	0.0	11.0	9.3	0.2	**59.9**	9.6
Obl	2.5	1.0	2.1	0.0	1.4	0.9	0.0	0.1	21.0	4.0	1.1	7.4	**55.0**

Table 6: Confusion matrix of the most frequent tags in the machine learning algorithm

employ the use of dependency relations which are standardized according to the UD annotation scheme. In a different domain, the frequency of the labels used can change slightly. However, as the edge cases are a result of the structure of the language, a domain change would not significantly affect the accuracy difference between the two models. Therefore, the scores presented in this study are not domain specific. In addition, it should be noted that the comparison of the models are based solely on the accuracy. The two models require different kinds of prior work. For instance, machine learning approach requires an annotated corpus to be trained on while the rule-based model does not need this kind of a training data. This is one of the reasons that all of the previous conversion studies employed a rule-based approach as there was not any data to train the converter in most languages. However, there are now enough training corpora to employ machine learning approach for further studies.

4 Conclusion

Overall, this project presents a remarkable contribution to UD studies in Turkish, in that it introduces the first constituency to the dependency conversion tool. Furthermore, the performance comparison which was provided shows that machine learning algorithms are able to achieve better accuracy scores compared to rule-based approaches. We hope that this tool will prove to be useful in future studies on Turkish dependency corpora as well. Thanks to the rapid transformation from phrase structure trees, it will allow for the creation of more dependency data, paving the way for better Turkish dependency parsers.

Moreover, as a result of this conversion, a corpus of 10,000 new sentences has been added to the Turkish dependency corpora, constituting one of the largest dependency corpora in Turkish. Besides, considering that the Penn Treebank constituency trees have previously been transformed into dependency trees with the help of many conversion tools such as the Stanford conversion tool (Marneffe et al., 2006) and ClearNLP (Choi and Palmer, 2012) the output we provide represents an impeccable overview of comparative structures of English and Turkish UD trees, thus offering a cross-linguistic perspective.

References

Ö. Bakay, B. Avar, and O. T. Yıldız. 2019. A tree-based approach for English-to-Turkish translation. *Turkish Journal Of Electrical Engineering And Computer Sciences*, 27:437–452.

J. D. Choi and M. Palmer. 2012. Guidelines for the clear style constituent to dependency conversion.

R. Johansson and P. Nugues. 2007. Extended constituent-to-dependency conversion for English. In *Proceedings of the 16th Nordic Conference of Computational Linguistics (NODALIDA 2007)*, pages 105–112, Tartu, Estonia. University of Tartu, Estonia.

N. Kara, B. Marşan, M. Özçelik, B. N. Arıcan, A. Kuzgun, N. Cesur, D. B. Aslan, and O. T. Yıldız. 2020. Creating a syntactically felicitous constituency treebank for Turkish. In *2020 Innovations in Intelligent Systems and Applications Conference (ASYU)*, pages 1–6.

Aslı Kuzgun, Neslihan Cesur, Bilge Nas Arıcan, Merve Özçelik, Büşra Marşan, Neslihan Kara, Deniz Baran Aslan, and Olcay Taner Yıldız. 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.

M. Marcus, M. Marcinkiewicz, and B. Santorini. 2002. Building a large annotated corpus of English: The penn treebank. *Computational Linguistics*, 19:313–330.

M. C. De Marneffe, B. MacCartney, and C. D. Manning. 2006. Generating typed dependency parses from phrase structure parses. In *Proceedings of the LREC*.

J. Nivre, M. de Marneffe, F. Ginter, J. Hajič, C. D. Manning, S. Pyysalo, S. Schuster, F. Tyers, and D. Zeman. 2020. Universal dependencies v2: An evergrowing multilingual treebank collection. *arXiv preprint arXiv:2004.10643*.

K. Oflazer, B. Say, D. Z. Hakkani-Tür, and G. Tur. 2003. Building a Turkish treebank. In *Treebanks*, pages 261–277. Springer, Dordrecht.

B. Plank, Héctor Martínez A., Z. Agić, D. Merkler, and A. Søgaard. 2015. Do dependency parsing metrics correlate with human judgments? In *Proceedings of the Nineteenth Conference on Computational Natural Language Learning*, pages 315–320, Beijing, China. Association for Computational Linguistics.

K. I. Simkó, V. Vincze, Z. Szántó, and R. Farkas. 2014. An empirical evaluation of automatic conversion from constituency to dependency in Hungarian. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 1392–1401, Dublin, Ireland. Dublin City University and Association for Computational Linguistics.

U. Sulubacak, M. Gökırmak, F. Tyers, C. Çöltekin, J. Nivre, and G. Eryiğit. 2016. Universal dependencies for turkish. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 3444–3454.

U. Türk, F. Atmaca, S. B. Özateş, A. Köksal, O. B. Başaran, T. Güngör, and A. Özgür. 2019. Turkish treebanking: Unifying and constructing efforts. In *Proceedings of the 13th Linguistic Annotation Workshop*, pages 166–177.

O. T. Yıldız, B. Avar, and G. Ercan. 2019. An open, extendible, and fast Turkish morphological analyzer. In *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2019)*, pages 1364–1372, Varna, Bulgaria. INCOMA Ltd.

O. T. Yıldız, E. Solak, O. Görgün, and R. Ehsani. 2014. Constructing a Turkish-English parallel TreeBank. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 112–117, Baltimore, Maryland. Association for Computational Linguistics.

O. T. Yıldız, E. Solak, Ş. Çandır, E. Ehsani, and O. Görgün. 2015. Constructing a Turkish constituency parse treebank. volume 363, pages 339–347.

Ç. Çöltekin. 2015. Improved transition-based parsing by modeling characters instead of words with lstms. In *Proceedings of the 14th workshop on Treebanks and Linguistic Theories (TLT 14)*, pages 35–49.