A Preprocessing Toolkit for Turkish NLP Applications

Karahan Şahin and Büşra Marşan
Boğaziçi University
Department of Linguistics
34342 Bebek, Istanbul, Turkey
{karahan.sahin, busra.marsan}@boun.edu.tr

Abstract

In this paper, we aim to describe the development process of a capable preprocessing toolkit for Turkish Natural Language Processing applications. This toolkit contains a tokenizer, sentence splitter, normalization algorithm, stemmer, and stopword elimination algorithm. Developed in consideration of linguistic and typological particularities of Turkish, our toolkit intends to yield accurate results even with rather challenging data.

1 Introduction

Especially with the rise of quantitative research trend in humanities, Natural Language Processing techniques and applications have gained an immense popularity which has been increasing every day. From political science to linguistics, a wide range of different disciplines require and employ NLP systems to conduct research. As a result, the subject matter, quantity, characteristics, and range of input texts to NLP systems show great variance. That is why preprocessing constitutes a vital step to any NLP application.

In this study, we aim to offer a comprehensive toolkit that is capable of dealing with the entirety of the preprocessing steps: Tokenization, sentence splitting, normalization, stemming and stop word elimination. In the development process of our toolkit, we considered linguistic and typological particularities of Turkish in order to obtain better results. One of the guiding works that was of particular use in this project was Liu (2018) which attempts to offer a unique normalization task for Swedish language. Compared to other Indo-European languages, Swedish has relatively richer morphology which makes normalization tasks rather challenging. In order to deal with that, Liu (2018) opts for employing a more localization-based approach which taps into the linguistic features of the language. Inspired by this approach, we aimed to consider affix stacking features of Turkish to offer a higher performance preprocessing toolkit.

Aiming to explain the development process and unique features of our toolkit, this paper is organized as follows: Strategies adopted in the tokenization, sentence splitting, normalization, stemming and stopword elimination processes are explained in the System Description section, and the performance of the toolkit is examined in the Results section.

2 System Description

2.1 Tokenization

For the tokenization task, two different models are adopted: A machine-learning based model and a rule-based model. First, a comprehensive Multi Word Expression list was created. For this purpose, Turkish PARSEME 1.2 corpus[1] (see Sak, Gungor, & Saraclar, 2011, for a detailed discussion) was used as a lemma list. Since it consists of 22,311 sentences (LVC.full: 3582, VID: 4139, MVC: 5), it served as a great and extensive resource yet it required some sifting and sorting due to its one particular quirk: It includes various inflectional forms of each word due to voicing alternations (passive, causative and so forth). Having such variations in the corpus was not very efficient. Instead, voicing alternations and similar divergent forms were removed from the 2874-item multiword expression list offered by Turkish PARSEME 1.2 and morphographemic alternations due to voicing were encoded in Regular Expressions.

[1] https://gitlab.com/parseme/sharedtask-data/-/tree/master/1.2/TR

The main motivation behind this approach is efficiency. In addition, it allows a sophisticated syntactic and phonological representation of verbs which allow voicing alternation.

Using regular expressions, consonant changes triggered by both voicing alternation and affixation were handled. For instance, "etmek" *("do", active)* and "edilmek" *("do", passive)* are encoded as "e[d/t]" while consonant changes that can be observed in expressions like "boyunduru**ğ**u altına girmek" are encoded as "boyunduru[kğ]".

A challenge of tokenization process is punctuation and clitics. In order to circumvent potential problems that could arise due to quotation marks, punctuation marks that occur sentence finally, or clitics, following features were used in the rule-based tokenizer:

Rule 1: Put white-space around unambiguous punctuation (? ( ) ; etc.)
Rule 2: Put white-space around commas that aren't inside numbers
Rule 3: Put white-space after single quotes not preceded by a letter
Rule 4: Put white-space after clitics not followed by a letter

For the machine-learning based tokenizer, Logistic Regression was opted for due to the large quantities of data the tokenizer has to handle. To tackle the challenge posed by the punctuation marks and clitics, LR algorithm uses the following features:

```
"is_capital": 0, # Followed by capitalized letter
"prev_space": 0, # preceeded by space
"forw_space": 0, # Followed by space
"is_number": 0, # Followed by space
"is_punct": 0, # Is it a punct or not
"forw_punct": 0, # Is it followed by one char before punct or not
"is_acroynm": 0, # Is the preceeding token an acronym
"class": 0,
```

Table 1: Logistic Regression features.

2.2 Sentence Splitting

The purpose of sentence splitting is to detect where one sentence ends and another starts. Often punctuation marks provide the necessary cues for this task but certain orthographic conventions such as using full stops in abbreviations create noise. Hence, distinguishing abbreviations from sentence endings is a great challenge for sentence splitting (see Mikheev, 2002; Stevenson & Gaizauskas, 2000, for a detailed discussion). In order to handle this problem, an extensive abbreviation list was created using TDK Kısaltmalar Dizini[2] [3].

Using the abbreviation list, regular expressions were created to cover all abbreviations. Then these regular expressions were used to extract relevant features to distinguish full stops in abbreviations from full stops at sentence final position. Full set of the features can be seen below.

Features	**Description**
is_capital	Whether the following token starts with capital letter
is_space	Whether the punctuation followed by space character
is_number	Whether the preceded or followed by a number
is_alpha	Whether preceded by a one alphabetic character
is_acronym	Whether preceded by an acronym
length_prev	Total number of preceded characters between another non-alphanumeric character

Table 2: Description of the features.

[2] https://www.tdk.gov.tr/icerik/yazim-kurallari/kisaltmalar-dizini/
[3] https://tdk.gov.tr/wp-content/uploads/2019/01/Ksaltmalar_Dizini.pdf

When the sentence splitter algorithm faces a full stop, it refers to is_abbr feature to decide whether that instance is a sentence ending or an abbreviation. If there is no occurance of an abbreviation preceding the full stop, that instance is classified as a sentence boundary.

2.3 Normalization

For the normalizer, Zemberek's lexicon[4], MWE list extracted from Turkish PARSEME 1.2 corpus (Sak et al., 2011) and regular expressions extracted from TDK's Kısaltmalar Dizini were used.

Similar to the case in sentence splitting, abbreviations posed a serious challenge for normalization. In order to distinguish abbreviations and handle them properly, normalizer refers to the previously mentioned regular expressions to classify certain instances as abbreviations or not-abbreviations. For not-abbreviations, uppercase letters are changed into lowercase letters. Also, line breaks are eliminated to create uniformed strings.

In an attempt to enhance the normalizer's performance, Levenshtein Distance was first employed as a similarity metric to compare problematic instances with the ones in the lexicon. The goal was to find the closest match in the dataset and replace the problematic instance with that, yet this approach was not fruitful thus it was discarded after the first testing round.

Most normalization algorithms also include a component to remove accents and diacritics. Since Turkish doesn't have diacritics, implementing such a component was not of question. On the other hand, accent removal yielded rather problematic results since Turkish alphabet has graphemes like "ç" and "ş" which have accents. Thus, the normalizer offered as part of this study doesn't remove accents.

One distinguishing feature of the normalization algorithm we offer is that it is able to normalize contractions like "Ali'den" to yield results like "aliden". Such normalizations enhance the performance of the tokenizer whose performance directly affects stemming and stopword elimination.

Other than those, there is also regularization of special tokens as the set of ["<URL>","<HASHTAG>","<EMAIL>"] with the corresponding regular expressions.

2.4 Stemming

As an agglutinative language, Turkish has a very complex morphology compared to analytical languages like Indo-European languages. That is why developing a stemmer with a high success rate is a formidable task. Moreover, the fuzzy distinction between verbal and nominal morphology adds to the difficulty of this challenge:

- Turkish employs many affixes to derive nouns from verbs, and verbs from nouns.
- Same set of person and number agreement affixes are employed in both verbal domain and nominal domain.

As a result of these typological peculiarities of Turkish, making a clear-cut distinction between nominal and verbal morphology is close to impossible.

In order to overcome difficulties posed by Turkish morphology, the stemmer offered by this study takes a more linguistically sound approach and refers to the affix stacking rules of Turkish discussed in Taylan (2015) *(see below).*-

	Voice	**Negation**	**Modality**	**TAM I**	**TAM II**	**Agreement**	**Mod**
Verb +	-Il, -DIr, -In, -Iş	-mA	-(y)abil	-DI, -mIş, -Iyor, -(y)AcAk, -Ar/Ir, -sA, -mAlı, -(y)A, -mAkta	idi, imiş, ise	person	-DIR

Table 3: Affix stacking hierarchy in Turkish.

[4]https://github.com/ahmetaa/zemberek-nlp/tree/master/normalization/src/main/resources/normalization

The stemmer starts slicing the word from the right and refers to the stacking hierarchy of the morphemes. Thus, it can correctly stem potentially confusing words like “hayatım” which can be stemmed as “haya” when considered as a verbal predicate and “hayat” when considered as a nominal.

In order to give the correct output, the stemmer has to distinguish nominal and verbal roots. Following the heuristics below, it classifies a word as a noun or a verb and proceeds with their respective morpheme hierarchy to remove all morphemes starting from the right end of the string.

If Mod -DIR is present, it is a verbal predicate.

Follow the morpheme hierarchy for verbal morphology.

If the final morpheme is a person agreement marker, look to the left to make a decision.

If the left hand side bears nominal morphology (such as -lAr), treat the word as a noun and slice morphemes while referring to nominal morphology.

If the left hand side bears,

TAM-II

Follow the morpheme hierarchy for verbal morphology.

TAM I

Follow the morpheme hierarchy for verbal morphology.

(. . .)

To extract the entire morphological feature paradigm of Turkish language, UD morphology (see de Marneffe, Manning, Nivre, & Zeman, 2021, for a detailed discussion) was referred to and corresponding regular expressions were created. A comprehensive list of regular expressions pertaining to derivational morphemes was also added.

A sample output of the stemmer can be seen below.

```
Token: "etmektedir"
Stem: "et"
Morphology: ['Copula=GenCop', 'Case=Loc', 'PersonNumber=V1pl', 'Polarity=Neg']
```

Table 4: Stemmer output.

2.5 Stopword Elimination

Stopword elimination algorithm follows two different methodologies: Dynamic and static. For the static stopword elimination algorithm, NLTK’s Turkish stop words list[5] was used. Adapted by Jayaweera, Senanayake, and Haddela (2019), dynamic approach makes use of a stop word lexicon and calculates TF-IDF score. Top k words can be selected, in this case k=100. See below for the details of the algorithm.

```
Procedure Avg_TF_IDF_Score (input):
       1. Calculate the term frequency (TFt, d) for each word in the document.
       2. Parallelly calculate the document frequency (DF) of each word in the
          corpus.
       3. After calculating the document frequency, calculate the Inverse
          document frequency (IDFt) with the count of documents N in the given
          source text.
       4. Calculate TF-IDF score for each word.
       5. Average_TF-IDFt = TF-IDFt, d / count (t)
       6. Order Average_TF-IDFt score in ascending order.
```

3 Results

The tokenizer offered in this study has a very promising performance with 0.84 precision score and 0.91 F1 score. In order to raise its performance, a corpus with wider coverage can be implemented in the future studies and additional features can be added to Logistic Regression model.

[5](nltk.stopwords(”turkish”))

Sentence splitter, on the other hand, requires certain improvements as its precision is lower than satisfactory. In order to enhance performance, a different approach may be taken in the follow-up studies.

	Precision	**Recall**	**F1**	**Support**
0	0.84	1.00	0.91	81326
1	0.00	0.00	0.00	15981

Accuracy	0.84			9737
Macro Avg	0.42	0.50	0.46	97307
Weighted Avg	0.70	0.84	0.76	97307

Table 5: Tokenizer performance.

	Precision	**Recall**	**F1**	**Support**
0	0.33	1.00	0.50	2
1	1.00	1.00	1.00	1647

Accuracy	1			1649
Macro Avg	0.67	1.00	0.75	1649
Weighted Avg	1.00	1.00	1.00	1649

Table 6: Splitter performance.

Additional Notes

The following libraries were used in the development of our preprocessing toolkit: regex, pandas, numpy, sklearn, nltk

To train machine learning models, KeNet UD style dependency corpus[6] (see Bakay et al., 2021, for a detailed discussion of WordNet KeNet) (see de Marneffe et al., 2021, for a recent discussion of Universal dependencies) was used. The motivation behind this choice was that KeNet dependency corpus was annotated and tokenized, thus provided valuable training data.

References

Bakay, Ö., Ergelen, Ö., Sarmış, E., Yıldırım, S., Arıcan, B. N., Kocabalcıoğlu, A., ... others (2021). Turkish wordnet kenet. In *Proceedings of the 11th global wordnet conference* (pp. 166–174).

de Marneffe, M.-C., Manning, C. D., Nivre, J., & Zeman, D. (2021). Universal dependencies. *Computational linguistics*, *47*(2), 255–308.

Jayaweera, A., Senanayake, Y., & Haddela, P. S. (2019). Dynamic stopword removal for sinhala language. In *2019 national information technology conference (nitc)* (pp. 1–6).

Liu, Y. (2018). *A pipeline for automatic lexical normalization of swedish student writings.*

Mikheev, A. (2002). Periods, capitalized words, etc. *Computational Linguistics*, *28*(3), 289–318.

Sak, H., Gungor, T., & Saraclar, M. (2011). Resources for turkish morphological processing. *Language resources and evaluation*, *45*(2), 249–261.

Stevenson, M., & Gaizauskas, R. (2000). Experiments on sentence boundary detection. In *Sixth applied natural language processing conference* (pp. 84–89).

Taylan, E. E. (2015). *Phonology and morphology of turkish.* Bogazici Universitesi.

[6] https://universaldependencies.org/treebanks/tr_kenet/index.html