A Modern Preprocessing Toolkit for Turkish

Atahan Uz and Gizem Yılmaz
Boğaziçi University
Department of Computer Engineering
34342 Bebek, Istanbul, Turkey
{atahan.uz, gizem.yilmaz1}@std.bogazici.edu.tr

Abstract

Preprocessing is a crucial first step in Natural Language Processing. This paper introduces a modern preprocessing toolkit developed for the Turkish language, including modules for tokenization, normalization, stemming, sentence splitting, and stop-word elimination.Together, these components provide a robust foundation for developing high-quality Turkish NLP applications.

1 Introduction

For the successful operation of Natural Language Processing (NLP) systems, cleaned and formatted input is essential. Before advanced tasks such as part-of-speech tagging, information extraction, and text classification, text must be converted into a format that is easily understandable by computers. This crucial step is called preprocessing, and it is the first stage of almost all NLP workflows. In this study, we designed a tokenizer, normalizer, stemmer, sentence splitter, and stop-word eliminator specially tailored for the Turkish language.

The Turkish language presents specific challenges for natural language processing. Due to its agglutinative structure, a single word can have a large number of suffixes attached to it. For example, the word *"kitaplarımızdan" (kitap+lar+ımız+dan)* has three suffixes. In addition, the Turkish language has a rich history of words from various eras, such as Ottoman and Republican eras. To solve this complexity, our system uses both rule-based and machine learning–based methods such as Naive Bayes and Logistic Regression. We also incorporated and corpora, techniques and methodologies from previous studies.

This toolkit aims to address a gap in Turkish NLP resources by combining linguistic rule systems with machine learning techniques. In addition to its advanced algorithms, the toolkit includes a user-friendly React.js interface that allows users to interactively analyze words, phrases, or entire texts. This design makes it easy for both researchers and practitioners to quickly experiment with different preprocessing options.

2 System Description

We programmed the application in Python and developed a React.js based graphical user interface where users can easily interact with. After entering a word, sentence or text, users can click the appropriate module and the output will be generated by the program and will be shown on the screen.

2.1 Tokenizer

Tokenization is the process of dividing the raw text input into meaningful parts to be processed in NLP applications. We developed two tokenizers based on two different paradigms, a rule based tokenizer and a machine learning based tokenizer.

Rule Based Tokenizer The rule-based tokenizer operates by segmenting the text input according to predetermined rules. By default, it separates the input on a word-by-word basis.
Certain entities are preserved as single units during tokenization, including URLs (e.g., *https://example.com*), e-mail addresses (e.g., *example@gmail.com*), hashtags (e.g., *trump2028*), and user mentions (e.g., *@atahanuz*). Detection of these entities is performed using regular expressions adapted

to informal digital Turkish text.
Using a corpus of Turkish multi-word expressions (MWEs), the tokenizer identifies MWEs and assigns them as single tokens. For example, the Turkish MWE "değer vermek," meaning "considering something high value," is output as a single token.
For each whitespace-level token, the tokenizer separates punctuation from the core alphanumeric content. Leading punctuation (e.g., "(", "[", "…") is extracted character-by-character. Trailing punctuation (e.g., ".", ",", "?") is separated in the same manner. Tokens corresponding to placeholders for protected entities are excluded from this process to prevent corruption. For example: *"güzel."* → [*"güzel"*, *"."*] , *"örnek",* → [*"örnek"*, *","*] Final output example: Given the sentence "Web sitemiz *https://example.com* adresinde." the tokenizer produces the following output: [*"Web"*, *"sitemiz"*, *"https://example.com"*, *"adresinde"*, *"."*].
The tokenizer has several important features:

Deterministic Behavior: Tokenization decisions follow clear rules, ensuring consistency across different datasets and domains.

Robust Handling of Digital Turkish: The method accurately processes social media content that includes URLs, mentions, and hashtags.

Linguistic Awareness through MWEs: MWEs are preserved as single tokens. This supports tasks such as semantic similarity, syntactic parsing, information extraction, and phrase-based representation learning.

Machine Learning Based Tokenizer This study uses a supervised machine learning method for tokenization (Jurish and Würzner, 2013). The task is implemented as a binary classification problem. The classification is performed at the character level (Palmer and Hearst, 1997). The main goal is to predict whether each character in the input text marks the start of a new token. The logistic regression classifier (Manning et al., 2008) determines the decisions based on local contextual features, which are taken from the location around each character position.

Feature Representation The feature set includes different aspects of character context to assist with boundary decisions. Character-level features consist of the character and its attributes, like whitespace, punctuation, digits, letters, and case (Guo et al., 2016). Contextual features extend two positions in both directions, creating a five-character window around each decision point (Bengio et al., 2003). The surrounding characters are represented individually and as n-grams, which consist of bigrams and trigrams to capture common sequences that indicate token transitions (Manning et al., 2008).

Training and Inference The BOUN Treebank is used for training and testing purposes. The original text was recovered from the treebank, and the feature representation methods described in this paper were applied to the training set for use in the Logistic Regression Classifier's training phase. Training data consists of text and token pairs with the appropriate tokenization. The algorithm establishes token boundaries by locating each designated token in the raw text and determining its last character position. This results in labeled examples, where boundary positions get positive labels and all other positions receive negative labels. The class imbalance is dealt with by using balanced class weighting during model training (Pedregosa et al., 2011).

This method enables the model to learn tokenization patterns directly from annotated data instead of depending on manually created rules (Jurish and Würzner, 2013). The character level approach helps the system handle complex boundary cases and language-specific features through learned feature weights (Guo et al., 2016). The model's performance is evaluated using standard classification metrics including precision, recall, and F1-score (Sokolova and Lapalme, 2009).

2.2 Stemming

To address the complex agglutinative structure of Turkish, we developed a stemming algorithm that reduces words to their root forms. The system utilizes a four-step methodology to ensure morphotactic and phonological validity.

Category-Based Suffix Classification Suffixes are classified into three distinct categories: *derivational*, *inflectional*, and *functional*. This classification enforces morphotactic constraints, controlling the

order in which suffixes are processed to improve stem identification accuracy.

Application of Phonological Rules The algorithm applies systematic rules to reverse sound changes introduced during suffixation. The system handles: *consonant softening* ($p \rightarrow b$ or $k \rightarrow ğ$), *vowel dropping* ($ı, i, u, ü$), *buffer consonants and doubling*.

Vowel Harmony Check Every derived suffix combination is validated against Turkish major and minor vowel harmony rules. Candidates violating these acoustic constraints are discarded.

Backtracking Algorithm A breadth-first search (BFS) generates all possible root candidates by iteratively stripping suffixes. Each candidate is validated against dictionary existence, minimum root length thresholds, and suffix order validity.

Scoring System

To resolve ambiguity among multiple valid root candidates, a weighted scoring metric selects the optimal result: Dictionary Validity *(+10 points if the root exists in the dictionary)*, Optimal Length *(roots closer to an ideal length of 5 characters are favored)*, Suffix Penalty *(each stripped suffix incurs a penalty of –0.2 points)*, Morphotactic Validity *(+2 points for correct suffix order)*, Suffix Category Bonus *(functional (+0.5), inflectional (+0.3), and derivational (+0.1))*, Short Root Penalty *(roots with fewer than 3 characters incur a penalty of –5 points)*.

2.3 Normalization

This study introduces a six-stage normalization pipeline for Turkish digital text. It changes non-standard spelling and form variations into standard versions. The system processes tokens one by one through abbreviation extension, letter case transformation, diacritic restoration, vowel restoration, spelling correction, and accent normalization. We evaluated the pipeline using 200 syntactically generated erroneous-correct data pairs.

Abbreviation Extension This module expands common Turkish abbreviations using a dictionary lookup. It maps shortened forms to their full versions. Standardizing abbreviations reduces vocabulary fragmentation in models (Hakkani-Tür et al., 2000; Sarikaya et al., 2009).

Letter Case Transformation The module performs the mappings of *'i/İ', 'ı/I',* and other characters that deviate from standard ASCII norms using case rules unique to Turkish language. It keeps significant instances, such as proper nouns with apostrophes, while standardizing incorrect capitalization. In this module, we used BOUN TULAP's morphological parser to detect proper nouns.

Diacritic Restoration This module restores Turkish-specific characters *('ş', 'ç', 'ğ', 'ü', 'ö', 'ı')*, which are frequently replaced with standard ASCII characters. The algorithm generates candidate forms by converting confusing characters back to their Turkish equivalents, then selects the best choice based on dictionary checks and character frequency (Adalı and Eryiğit, 2014; Tür, 2000; Yüret and de la Maza, 2006).

Vowel Restoration The module reconstructs missing vowels by comparing consonant patterns with lexicon entries (Adalı and Eryiğit, 2014). It takes consonant sequences from both the input and lexicon words, identifies possible matches, and selects alternatives that need the fewest vowel insertions. Structural rules prevent unrealistic vowel adds.

Spelling Correction This module creates correction options by applying edit operations (deletion, insertion, substitution, transposition) within an edit distance of two (Wang et al., 2011; Torunoğlu-Selamet and Eryiğit, 2016). Changes related to diacritics get lower costs (0.3) compared to arbitrary changes (1.0) to reflect common error trends in Turkish text. The system ranks options by merging transformation costs with word frequency (Lindén and Pirinen, 2014), choosing forms that balance minimal changes with likelihood.

Accent Normalization The module uses regex-based modifications to standardize informal morphological patterns (McKean, 2005; Eisenstein, 2013). It transforms reduced verbal suffixes into standard forms. For example, present continuous *'-iyo/-ıyo'* becomes *'-iyor/-ıyor'*, future markers *'-cak/-cek'* becomes *'-acak/-ecek'*, and negative futures revert to standard forms.

2.4 Stopword Elimination

This section describes the stopword removal methodology implemented for Turkish text processing. The approach enables systematic comparison between lexicon-based and corpus-driven stopword identification strategies, both of which are evaluated on the same document collection.

Lexicon-Based Stopword Removal (Satatic Approach) The first removal approach uses a fixed stopword list from existing Turkish language resources. This method illustrates the traditional way of identifying stopwords by using language knowledge and manual selection. The filtering algorithm looks at each document and checks its tokens against the stopword list. It removes any token that matches a stopword, while the tokens that do not match stay in the text.

Statistical Stopword Identification (Dynamic Approach) The second method identifies stopwords dynamically using statistical analysis of the corpus. This method is based on the idea that words that appear frequently in multiple documents but contribute minimal discriminative information can be identified based on their TF-IDF aspects (Manning et al., 2008).

TF-IDF Computation The system calculates term frequency-inverse document frequency scores using a traditional information retrieval algorithm. For each document, term frequency is calculated as the normalized occurrence count: $TF(t,d) = \frac{f(t,d)}{|d|}$, where f(t,d) is the raw frequency of the term t in document d, and `|d|` is the total number of terms in the document. This normalization accommodates for differences in document length.

Document frequency is calculated by counting the number of documents that contain each unique term. The inverse document frequency is using logarithmic smoothing: $\mathrm{IDF}(t) = \log(\frac{N}{\mathrm{DF}(t)})$, where N is the total number of documents in the corpus and DF(t) is the document frequency of term t.

Threshold-Based Selection The stopword candidates are found by setting a threshold for IDF values. Words with IDF scores below a certain threshold are considered stopwords. This is based on the idea that low IDF values show high frequency across documents and a limited ability to distinguish meaning (Jones, 1972).

The threshold of 5.65 was found through examining the distribution of the corpus. This method strikes a balance between removing truly common function words and keeping important terms that may appear less often. The IDF-based approach remains the primary way to identify stopwords, as IDF directly measures term specificity independent of within-document frequency patterns (Robertson, 2004).

2.5 Sentence Splitting

Sentence splitting divides text into boundaries, a foundational NLP step ensuring subsequent processing receives well-formed inputs.

Rule-Based Sentence Splitting This method simulates human reading by scanning text sequentially. It buffers characters and triggers a context check upon encountering potential terminators (e.g., *., !, ?*). A split is confirmed only if specific rules are satisfied.

The algorithm handles complex scenarios to prevent false positives using contextual rules:

Quotations: A stack-based logic tracks opening and closing quotes, preventing splits within quoted speech unless a clear boundary exists immediately after the closing mark.

Abbreviations & Numbers: The system checks a predefined list of abbreviations (e.g., *"Dr.", "bkz."*) and detects numeric patterns (e.g., *"3.14"*) or ellipses to suppress splitting at their dots.

Lookahead: After a punctuation mark, the algorithm examines the next character. Uppercase letters or digits suggest a new sentence. In Turkish, specific lowercase conjunctions (e.g., *ve, ama, fakat*) following punctuation also trigger a split to handle informal text.

Machine Learning-Based Sentence Splitting This approach treats splitting as a binary classification problem, learning from data rather than manual rules. We treat every punctuation mark as a decision point labeled either *boundary* (end of sentence) or *non_boundary*.

Training: A Naive Bayes classifier is trained on labeled sentences. To simulate real-world errors, training pairs are joined both with and without spaces.

Features: Instead of linguistic parsing, the model extracts local features around the punctuation: the type of preceding/succeeding characters (uppercase, digit, space), capitalization of the next word, and specific token types (abbreviations/numbers).

Execution: The system scans the text; at each punctuation mark, the classifier calculates the probability of a boundary based on feature occurrence counts (using add-one smoothing).

3 Results and Discussion

We make validation runs on the training and development data, and compare our three neural models for each language. Afterwards, we report the official results of the selected systems on the blind test set.

Tokenizer Results A separate test set of 979 Turkish text samples yielded strong generalization results. Table 1 summarizes the complete dataset statistics and performance metrics.
The ML-based logistic regression tokenizer was trained using 7,803 Turkish sentences, yielding 608,711 character-level training samples. The class distribution was significantly imbalanced, with 97,797 positive examples (boundaries, 16.1%) and 510,914 negative examples (non-boundaries, 83.9%), which was rectified through balanced class weighting. The trained model achieved a training accuracy of 99.94%.
The ML-based method outperformed in every parameter as shown in Table 1. It performed an 99.54% F1-score, 99.63% recall, and 99.47% precision. While the rule-based approach also produced impressive outcomes as precision of 94.22%, recall of 94.95%, and F1-score of 94.49%.

Table 1: Performance Comparison of Rule-Based and ML-Based Turkish Tokenizers

Metric	**Rule-Based**	**ML-Based (LR)**
Test samples	979	979
Training sentences	–	7,803
Training examples	–	608,711
Training accuracy	–	99.94%
Precision	94.22%	99.47%
Recall	94.95%	99.63%
F1-Score	94.49%	99.54%

Stemmer Results The stemmer was tested on 1,000 random word–stem pairs from the BOUN Treebank test dataset. Some words were already in stem form, so the stemmer should not have changed them to different words. The performance results are summarized in Table 2.

Table 2: Performance metrics of the stemmer on the BOUN Treebank test dataset.

Metric	**Value**
Accuracy	84.2%
Precision	89.5%
Recall	95.0%
F1 Score	92.1%

Normalization Results Accuracy of the normalizer is tested on a dataset of 200 syntactically generated erroneous-correct data pairs (e.g. guzell - güzel). The Δ Accuracy column shows the absolute decrease in accuracy (percentage points) when the corresponding component is removed, relative to the full pipeline accuracy of 91%. We also made a comprehensive analysis, removed each element of the pipeline one at a time and tested the system's accuracy. Based on these results, diacritic_restoration ended up the most essential step of the pipeline.

Table 3: A comprehensive analysis of the normalization pipeline.

Component removed	Δ Accuracy	Accuracy
diacritic_restoration	-13.50	77.50%
spelling_correction	-12.00	79.00%
vowel_restoration	-8.50	89.50%
accent_normalization	-4.50	86.50%
letter_case_transformation	-3.00	88.00%
abbreviation_extension	-1.00	90.00%

Stopword Elimination Results This study evaluated three stopword removal approaches for Turkish text processing using the TR BOUN-UD training corpus: a static stopword list, a TF-IDF-based list, and the NLTK Turkish stopword library.

Table 4: Evaluation Results of Stopword Removal Methods

Metric	Original	Static	TF-IDF	NLTK
Vocabulary Reduction Analysis				
Vocabulary Size	28,395	28,123	28,118	28,343
Reduction	–	0.096	0.098	0.018
Document Length Reduction Analysis				
Avg. Tokens/Doc	10.50	7.92	7.20	9.26
Total Tokens	81,917	61,769	56,176	72,255
Reduction	–	24.60	31.42	11.79
Stopword Coverage Analysis				
Coverage	–	24.60	31.42	11.79
Stopword Tokens	–	20,148	25,741	9,662

Table 4 shows that all three techniques had small impact on vocabulary size, ranging from 0.18% to 0.98%. However, significant differences were observed in document length reduction and stopword coverage.

Sentence Splitting Results We evaluated the performance of the sentence splitting on its ability to correctly decide the sentence boundries. Our evaluation dataset was 1000 random samples derived from the test set of the BOUN Treebank.

Table 5: Performance comparison of sentence splitting methods on Turkish text

Metric	Rule-Based	Naive Bayes
Accuracy (%)	65.70	**90.70**
Precision (%)	**92.25**	91.41
Recall (%)	70.20	**99.00**
F1 Score (%)	79.73	**95.06**

The Naive Bayes approach significantly outperforms the rule-based method across most metrics, as it was able to learn nuances rule based method couldn't catch.

4 Conclusion

In this paper, we presented a modular preprocessing toolkit tailored for Turkish, integrating rule-based and machine-learning–based methods for tokenization, stemming, normalization, stopword elimination, and sentence splitting. The toolkit is designed to handle the challenges of Turkish morphology and informal digital text, and it is made easily accessible through a practical web interface. In future work, we plan to improve the performance of the modules, add new preprocessing components, and provide a more comprehensive evaluation on real-life use cases.

Acknowledgements

We've used the NLP toolkits of ITU NLP group (University, 2014) and Boğaziçi University TULAP platform (University, 2016) extensively in this study.

References

Adalı, K. and Eryiğit, G. (2014). Vowel and diacritic restoration for social media texts. In *Proceedings of the 5th Workshop on Language Analysis for Social Media (LASM)*, pages 53–61.

Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155.

Eisenstein, J. (2013). Phonological factors in social media writing. *Proceedings of the Workshop on Language Analysis in Social Media*, pages 11–19.

Guo, J., Che, W., Wang, H., and Liu, T. (2016). A universal framework for inductive transfer parsing across multi-typed treebanks. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 772–782.

Hakkani-Tür, D., Oflazer, K., and Tür, G. (2000). Statistical language modeling for turkish. In *Proceedings of the 18th Conference on Computational Linguistics*, volume 2, pages 1214–1218.

Jones, K. S. (1972). A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28(1):11–21.

Jurish, B. and Würzner, K.-M. (2013). Word and sentence tokenization with hidden markov models. *Journal for Language Technology and Computational Linguistics*, 28(2):61–83.

Lindén, K. and Pirinen, T. (2014). Weighting finite-state morphological analyzers using hfst tools. In *Proceedings of the 9th International Conference on Finite State Methods and Natural Language Processing*, pages 1–9.

Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press, Cambridge, UK.

McKean, E. (2005). *The Oxford Dictionary of English*. Oxford University Press.

Palmer, D. D. and Hearst, M. A. (1997). Adaptive multilingual sentence boundary disambiguation. *Computational Linguistics*, 23(2):241–267.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830.

Robertson, S. (2004). Understanding inverse document frequency: On theoretical arguments for idf. *Journal of Documentation*, 60(5):503–520.

Sarikaya, R., Gravano, A., and Gao, Y. (2009). Rapid language model development using external resources for new spoken dialog domains. *IEEE Transactions on Audio, Speech, and Language Processing*, 17(1):182–192.

Sokolova, M. and Lapalme, G. (2009). A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, 45(4):427–437.

Torunoğlu-Selamet, D. and Eryiğit, G. (2016). A cascaded approach for social media text normalization of turkish. In *Proceedings of the 5th Workshop on Language Technology for Closely Related Languages, Varieties and Dialects*, pages 62–70.

Tür, G. (2000). A statistical approach to deasciification. In *Proceedings of the 18th International Conference on Computational Linguistics*, volume 2, pages 849–853.

University, B. (2016). TULAP: Turkish Language Analysis Platform. `https://tulap.cmpe.boun.edu.tr`. Accessed: 2025-11-23.

University, I. T. (2014). ITU Turkish NLP Web Service. `http://tools.nlp.itu.edu.tr/index.jsp`. Accessed: 2025-11-23.

Wang, W., Xiao, C., Lin, X., and Zhang, C. (2011). A fast and accurate method for approximate string search. In *Proceedings of the VLDB Endowment*, volume 4, pages 535–546.

Yüret, D. and de la Maza, M. (2006). Learning morphological disambiguation rules for turkish. In *Proceedings of the Human Language Technology Conference of the NAACL*, pages 328–334.