CMPE561 Natural Language Processing Application Project I

Final Report

Nur Bengisu Çam and Ezgi Paket
Boğaziçi University
Department of Computer Engineering
34342 Bebek, Istanbul, Turkey
{bengisu.cam, ezgi.paket}@boun.edu.tr

1 Introduction

This final report shows the detailed explanations of completed tasks for creating our preprocessing toolkit for Turkish. We completed all the tasks in the project.

This report is divided into six main sections except from this introduction part. These divisions are tokenization, sentence splitting, stopword elimination, normalizaiton, stemmer and conclusion. In conclusion part, the overall flow of this project is summarized in detail.

Dataset: We have used the Turkish dependency treebank annotated in UD style (Türk et al., 2021) as our corpus.

2 Tokenization

Natural language processing projects mainly include preprocessing stage, before the modelling stages are started. One of these steps in preprocessing can be defined as tokenization (Webster and Kit, 1992). Text documents should be divided into smaller chunks such as sentences, subwords, words and characters. These smaller chunks can be easily processed instead of using larger documents. Moreover, they provide more accurate insights, since they focus on more detailed meanings and relationships.

There are many tokenization techniques in the literature, some of which are white space tokenization, dictionary based tokenization, rule based tokenization, machine learning based tokenization and spacy tokenization. In this project, both rule based and machine learning based tokenizations are implemented to divide sentences into words. This process is called as word tokenization.

2.1 Rule Based Tokenization

In most of the situation, splitting the sentence only from the white spaces may not be useful enough to obtain great text segmentation. Therefore, various extra rules should be defined. These rules may handle the abbreviations, punctuations, domain (Akkasi et al., 2016) and language specific terms (Attia, 2007).

In this project, we have implemented different rules by using their regex explanations.

URL rule: $\hat{}(https:|http:|www\setminus.)\setminus S*$

URL's or websites may start with different abbreviations such as https, http, and www. While some of these websites include https or http at the beginning of the URL before the "www" notation, others can directly start with "www" expression. Therefore, we have added a check for these expressions at the beginning of above regex rule.

Punctuation rules: [“\,”, “\;”, “\!”, “\?”]

Tokens should not concatenated with punctuation after them. For instance, if there is a comma after a word, this comma should be separated from the word. One token for the word and other token for the comma should be created. These processes are implemented for all of the comma, semi colon, exclamation and question mark notations.

Hashtag rule: $/\hat{}\#\backslash w+\$/$

Hashtags are generally used in social media platforms such as Twitter, Instagram and Facebook to emphasize some trending topics. Therefore, they should be processed in the raw text inputs. We have defined a rule to identify words start with # sign.

E-mail rule: ^[a-z0-9]+[\._]?$[a-z0-9]+[@]\setminus w+[.]\setminus w\{2,\}\$$

E-mails can be defined as one of the other exceptions in the raw text inputs. They have their own specific format. In the tokenization part, we have used a rule for e-mail format handling (Goyvaerts, 2021). They can start with any group of letters or digits. Some of these e-mails include a dot to separate name and surnames. Question mark at the right side of the dot sign controls this situation in above regex rule. Moreover, @ sign before the domain name and the last two or three characters such as ".com", ".com.tr" are checked.

Multi-word Expression (MWEs): Although some words can be used alone to form individual tokens, some should be merged together to provide more meaningful insights. For this aim, we have benefited from MWE Tokenizer of NLTK (Bird et al., 2009) library. We have used PARSEME 1/train.cupt (Berna Erden and Gungor, 2018) file to obtain list of MWE lexicons. We have extracted the PARSEME:MWE column from this dataset by using cuptlib library. Then, we read MWE lexicons from this list and related tokens are merged together using "_" separator.

An example from the rule based tokenizer is demonstrated below. As we desired, both e-mail adress 'ezgi.bengisu@hotmail.com' and url 'www.abcd.com' are perceived as separate tokens. Furthermore, punctuation symbols (commas and dots) are separated from the words. In the end, the MWE lexicon 'yardımcı olabilirler', which is included in PARSEME 1/train.cupt (Berna Erden and Gungor, 2018) file, is linked to build 'yardımcı_olabilirler' as one token.

Input Sentence: ' Merhaba, mail adresim ezgi.bengisu@hotmail.com. Internet sitesi www.abcd.com . Senin? Onlar sana yardımcı olabilirler.'

Output Sentence: '['Merhaba', ',', 'mail', 'adresim', 'ezgi.bengisu@hotmail.com', '.', 'Internet', 'sitesi', 'www.abcd.com', '', '.', 'Senin', '?', 'Onlar', 'sana', 'yardımcı_olabilirler', '.']

2.2 Machine Learning Based Tokenization

Model: Apart from the rule based tokenization, both unsupervised or supervised machine learning algorithms can be implemented to divide sentences into tokens. Wrenn et al. (2007) have proposed an unsupervised system by using only statistical knowledge of tokens and sentence boundaries.

We have searched the literature to find the related statistical features. Typological classes, letter-case situations, checks for following white spaces and punctuation etc. can be used as features in the models (Jurish and Würzner, 2013). Apart from this idea, the features which are used now in our model are listed below:

- Next character check (Is it a white space?)
- Previous character check (Is it a white space?)
- Next character check (Is it a dot sign?)
- Previous character check (Is it a dot sign?)
- Next character check (Is it punctuation?)
- Previous character check (Is it punctuation?)
- Next character check (Is it numeric character?)
- Previous character check (Is it numeric character?)

Since tokenization processes are mainly based on existence of dots and white spaces, we have defined separate features for them. The other punctuation symbols [':', '!', '?', ',',';','(',')'] are grouped together to create one feature. Furthermore, we have defined numeric character controls as our features. The main reason of controlling these numeric characters is that there is a need for checking dates and time slots to not separate them from the dot sign.

After these features are extracted in the boolean format, labels are determined according to location indexes of the ' ', '.', and ',' signs. Both features and labels are fitted to Logistic Regression model of sklearn (Pedregosa et al., 2011) library to predict results.

We have trained our model by using training set of Turkish dependency treebank annotated in UD style (Türk et al., 2021) corpus.

An example for the ML based tokenizer is shared below. Model is able to separate tokens according to white spaces and punctuation symbols. On the other hand, it is not good at MWE lexicon detection, since it is not benefited from any annotated MWE list. For instance, 'başarılı olabilirler' is not detected by ML based tokenizer.

Input Sentence: ' Merhaba, ben istanbulda yaşıyorum. Sen? Onlar çok başarılı olabilirler.'

Output Sentence: ['Merhaba', ',', ' ben', ' istanbulda', ' yaşıyorum', '.', ' Sen', '?', ' Onlar', ' çok', ' başarılı', ' olabilirler', '.']

3 Sentence Splitting

Given a text with multiple sentences, sentence splitting is the process of extracting each sentence from the text. In order to split the whole text into sentences, sentence endings are considered. These sentence endings are period, question mark and exclamation mark. However, there are some challenges in this task. For example, in many languages there are abbreviations and usually they are written with a dot followed. In such cases, rather than just checking the occurrences of the sentence endings, one should consider checking the abbreviations.

3.1 Rule Based Sentence Splitting

In the rule based sentence splitting, the sentences are detected according to the provided rules. In the project, we used regular expressions to create our rules. We considered sentence endings, as well as these inside the parenthesis and quotation marks since there can be sub sentences. There another case where we should pay extra attention, the triple dots. In order to detect a sentence, which ends with "..." punctuation, first we replaced them with a single dot (period). Again, to replace the "..." with a period, we used another regular expression. At the moment, our rule based sentence splitting system cannot correctly split a sentence if there are any abbreviations. For example, if there is "Av." in the sentence, which is an abbreviation of "avukat", our rule based splitter divides the sentence after the occurrence of such abbreviations. Here, in order to handle such cases, we will be using an abbreviations list that we have already extracted from a source, and try to look at the word before splitting. If the word is in the abbreviations list, we will not split the sentence, we will continue until it is the sentence ending. One approach can be checking the letter after the punctuation. However, since there can be proper nouns after an abbreviation, this approach will not be that effective and could produce wrong results. For example, after "Av." there can be "Ayşe" as a proper noun and checking the capital letter after the period. Thus, in such a rule based system, it is better to use a predefined list of abbreviations. There are not only abbreviations which builds the non splitting words list. For examples ordinals, Latin numbers are also non splitting words.

Replacing "..." with a period rule:
regex.sub(r'\.+', ".", input_text)

Sentence Splitting rule:
regex.compile(r'([A-Z][\

\.!?]*[\.!?])', regex.M)

Here is a toy example we used as input text to our rule based sentence splitter:

'Bugün kaçta geleceksin? Ben de ona göre hazırlanayım. Belki gelmem... (Veya gelirim.) "Gelmeyebilirim."

Result list after applying the sentence splitting rule:
['Bugün kaçta geleceksin?', 'Ben de ona göre hazırlanayım.', 'Belki gelmem.', 'Veya gelirim.', 'Gelmeyebilirim.']

3.2 Machine Learning Based Sentence Splitting

As we said in the previous report, we were considering using <BOS> and <EOS> tags to indicate the beginning of the sentence and end of the sentence. Then we can calculate how many times a word occurred before the <EOS> tag. We applied a similar idea in this part. We created a synthetic dataset which contains sentences at each line in a file. We also created a dataset which contains tags for each sentence. We tagged each word in the sentence with NS (non-splitting) and S (splitting) tags. Then we implemented Naive-Bayes algorithm where we applied Laplace smooting. We calculated the posterior probability as follows:
$P(T\|w) = P(w\|T).P(T)$,
where $P(w\|T)$ is the likelihood of a word w to have a tag T. $P(T)$ is the prior probability of the tag T. $P(w\|T)$ can be calculated using the joint probability $C(T,w)/C(T)$. For our quite small synthetic dataset, we calculated the probability of the given word having tag NS(non-splitting word) and S(splitting word). Whichever posterior probability is higher, we assigned the tag accordingly. If a word has a NS tag, we do not create a new sentence, rather we add the current word to the previous words. If a word has a S tag, we create a new sentence. Since our synthetic dataset is very small, we could not extracted enough propability values. But the results are promising and we know that if the data size increases, Naive Bayes will capture the distribution of the data.

4 Stop Word Elimination

In any language, stop words are usually the words which have less meaning and have many occurrences. In many language processing tasks, as a pre-processing step, removing the stop words are applied. Stop words removal allows to deal with less number of words, thus helps o reduce the dimension of the features and as a result increases the processing time. In this project, we will be implementing both static and dynamic ways to remove the stop words in Turkish language.

4.1 Static Stop Word Elimination

In this task, we used different sources to create a stop words list. First of all, we used a list of stop words created by Aksoy (2018). Here, there are many stop words which we did not expect to be a stop word. This list contains the number names, conjunctions, prepositions, pronouns and some verbs. Another source we used is created by Xuan Xia (). This github repository contains different stop words list for many languages. The number of Turkish stop words listed here is much more less compared to the previous source we talked about. Furthermore, there are no verbs neither number names. This list contains only the conjunctions, prepositions and pronouns. Thus, this list seems more reliable.
First of all, in order to remove the stop words, we split our text into sentences. Then we tokanize each sentence. Next, we analyze each token whether it is in the stop words list that we created using previously mentioned sources. If it matches, we remove the token.

4.2 Dynamic Stop Word Elimination

In the dynamic stop word elimination task, we used the idea of the IDF(inverse document frequencies). As we said, we will be using the (Türk et al., 2021) corpus. In this corpus, there are many documents which are seperated by "sent_id = document_name". In each document there are multiple sentences,

some of the documents are shorther compared to the others. In the idea of IDF, for each word in the total corpus the total number of corpus which contains that word counted and total number of corpus is divided to that term. If the result is high, then we counted that word in only a few documets. So it is a rare word, thus can contain more information about the text. However if the result is low, it means we counted that word in many of the documents in our corpus. Thus, that word contains less information and we can accept it as a stop word. In order to implement this IDF value for each word in our corpus, we used the TfidfVectorizer which comes within the sklearn library (Pedregosa et al., 2011). This vectorizer, takes the corpus (where the documents are separated) as an input, learns the vocabulary and calculates the IDF values. So, by using the sklearn library (Pedregosa et al., 2011), we calculated the IDF values. IDF values are sorted in a list and we looked them up. Only the first 120 of them seemed to be a cantidate for stop words. Thus, our dynamic stop word list contatins the most common 120 words in our corpus. These words are very similar to the static stop words. Here are the dynamic stop words that we extracted:
'bir', 've', 'bu', 'de', 'da', 'için', 'çok', 'gibi', 'daha', 'her', 'sonra', 'en', 'ama', 'ile', 'ne', 'olarak', 'kadar', 'ki', 'olan', 'in', 'ben', 'ya', 'nin', 'büyük', 'diye', 'var', 'türkiye', 'ın', 'zaman', 'gün', 'nın', 'ise', 'değil', 'yeni', 'hiç', 'şey', 'iki', 'önce', 'dedi', 'içinde', 'olduğu', 'olduğunu', 'ilk', 'iyi', 'benim', 'bütün', 'yok', 'bile', 'türk', 'bana', 'ancak', 'kendi', 'göre', 'onun', 'yıl', 'böyle', 'artık', 'üç', 'son', 'karşı', 'yer', 'oldu', 'birlikte', 'çünkü', 'aynı', 'mi', 'biri', 'onu', 'bunu', 'vardı', 'ye', 'üzerine', 'doğru', 'güzel', 'arasında', 'başka', 'küçük', 'tek', 'tüm', 'nasıl', 'beni', 'bunun', 'yi', 'yine', 'dünya', 'stanbul', 'gelen', 'hep', 'tarafından', 'un', 'hiçbir', 'insan', 'nun', 'kez', 'biz', 'ilgili', 'şu', 'etti', 'hemen', 'pek', 'bazı', 'ona', 'su', 'yani', 'kabul', 'uzun', 'adam', 'olur', 'sadece', 'söyledi', 'söz', 'önemli', 'birkaç', 'eden', 'ifade', 'biraz', 'den', 'mı', 'ortaya', 'te'

5 Normalization

In general, texts documents include randomness and they do not have a standard form. Therefore, these irregularities may cause to decrease in model performances. Normalization steps are implemented in order to convert text documents into standard and predetermined format.

There are various normalization processes. In this project, we have applied four of them. Firstly, we have converted uppercase letters to lowercase ones. Then, we have excluded punctuation symbols except the words and spaces from the texts. For this aim, we have benefited from the $[\wedge \setminus w \setminus s]$ regex rule. Thirdly, stop words are eliminated from the sentences. As we mentioned previously, a list of stop words is taken from the Aksoy (2018). If an input word is included in this list, it is removed from the input text. Lastly, numbers are converted into their word formats, since they do not provide any meaningful information to models with their number format. They should be converted into standard word format.

An example for normalization is showed below to be more clear. Firstly, the uppercase letters in 'MerHABAlar' are converted into lowercase letters and 'merhabalar' is obtained. Secondly, comma after the word 'MerHABAlar,' is removed. Then, stop words included in list Aksoy (2018) are removed from this sentence, which are 'ben', 'benim', 've', 'var'. Finally, 26 number is converted into its word form 'yirmialtı'.

Input Sentence: 'MerHABAlar, ben 26 yaşındayım. Benim kedim ve köpeğim var.'

Output Sentence: 'merhabalar yirmialtı yaşındayım kedim köpeğim.'

6 Stemmer

Stemmer takes derivational and inflectional rules into account. In Turkish, there are lots of suffixes. In this project we tried to find many of them: First of all, we created files for the derivational and inflectional rules.

First of all, our algorithm combines inflectional and derivative suffixes into a list where the inflectional rules appear before the derivational rules. usually inflectional rules are the ones that appear at the end of a word. We created a Turkish dictionary for the most common 1000 words in Turkish. This dictionary is

ndan—ntan—nden—nten	dık—dik—duk—dük
ları—leri	nız—niz—nuz—nüz
ecek—acak	mız—miz—muz—müz
meli—malı	miş—mış—muş—müş
dan—tan—den—ten	yor
dır—dir—dur—dür	lar—ler
ta—te—da—de	tı—ti—tu—tü
ları—leri	ar,er,ir,ır,ür,ur
di—dı—du—dü	se—sa
i—ı—u—ü—e—a	se—sa—sı—si—su—sü

Table 1: Inflectional rules

lık—lik—luk—lük	ma—me
siz—sız—suz—süz	ım—im—um—üm
cil—cul—cül—cıl	un—ün—ın—in
sal—sel	ış—iş—uş—üş
cık—cik—cuk—cük	ca—ce—cu—cü—cı—ci
lı—li—lu—lü—la—le	

Table 2: Derivational rules

helpful to identify words and not the split them even if they contain the suffix's characters in the syllable, not as a suffix. For example, if we do not check the word ”yüzünden” from the dictionary we use, the stemmer will remove the ”den” part. Another example, ”şeker” will be stemmed as ”şek” if we do not check its existence from the dictionary. In order to handle such cases, we used the dictionary. While creating this dictionary, we used the 1000 common words in the (most common words, 2021). However, the words had suffiex so we needed to go through all and convert them in to the dictionary format. The algorithm is as follows:
For each suffix in the suffix list, we looked if the given word contains that suffix. We also kept track of the length of the suffixes and tried to find the longest suffixed first. If a word contains the suffix and if the word has more than 2 characters, we remove the suffix. Here, we also wanted to check the length of the word at each step because in Turkish the shortest word can contain 2 characters so we do not want to exract words which are shorter than 2 characters. Next, we check the word if the word exists in the dictionary of 1000 common Turkish words after removing the suffix. We do not remove the suffix, if such a word does not exist. We remove it, if it is in the dictionary and continue looking for other suffixes in the remaining word. While checking form the dictionary, we also tried to handle cases where ”p,ç,t,k” changes into ”b,c,d,g”. So after removing the suffix, if the remaining word's last character is on of the ”b,c,d,g” characters, we change the last character into ”p,ç,t,k” and look up from the dictionary.
Our stemmer can handle the cases where a word contains more than 2 of the suffixes, both derivational and inflectional.

Input Word	**Stemming Result**
çocuklardanmış	çocuk
şekerliksiz	şeker
kalemliklerdendir	kalem
geldiler	gel
varmışlardır	var

Table 3: Results of the stemmer

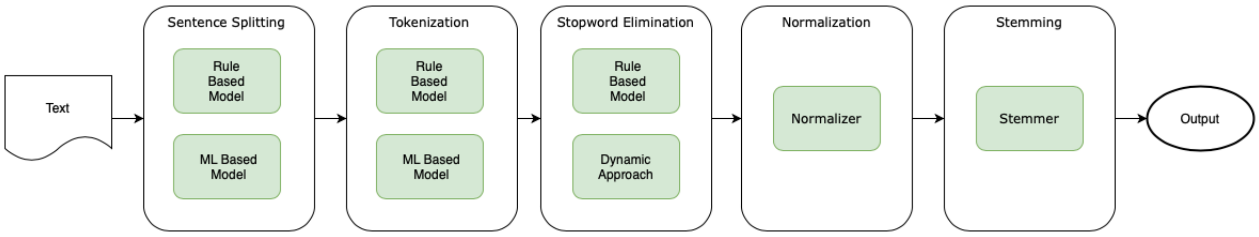

Figure 1: All stages of this project are completed.

7 Conclusion

In this project, we are trying to design a preprocessing toolkit for Turkish. While input of this toolkit is raw text data, processed texts are returned after various preprocessing stages. These preprocessing stages are crucial for building models in the following implementations.

In Figure 1, all steps within this project are demonstrated. Rule based model for sentence splitting, rule based model for tokenization, machine learning based model for tokenization, rule based model for stop word elimination and dynamic approach for stop word elimination processes are done in progress report. In the final, we have developed ML based sentence splitter by using Naive Bayes from scratch, normalizer and stemmer.

We will implement a Naive Bayes model for a sentence splitting process from scratch. Furthermore, a generalizable normalizer and a stemmer for both inflectional and derivational suffixes will be implemented in the second phase of this project. In the end, this toolkit will be tested by various challenging tasks.

References

Abbas Akkasi, Ekrem Varoğlu, and Nazife Dimililer. 2016. Chemtok: a new rule based tokenizer for chemical named entity recognition. *BioMed research international*, 2016.

Ahmet Aksoy. 2018. trstop. https://github.com/ahmetax/trstop/blob/master/dosyalar/turkce-stop-words.

Mohammed Attia. 2007. Arabic tokenization system. In *Proceedings of the 2007 workshop on computational approaches to semitic languages: Common issues and resources*, pages 65–72.

Gozde Berk Berna Erden and Tunga Gungor. 2018. Turkish verbal multiword expressions corpus. In *26th IEEE Signal Processing and Communications Applications Conference, SIU 2018*, İzmir, Turkey, May.

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.".

Jan Goyvaerts. 2021. Regular Expressions Tutorial. https://www.regular-expressions.info/index.html.

Bryan Jurish and Kay-Michael Würzner. 2013. Word and sentence tokenization with hidden markov models. *J. Lang. Technol. Comput. Linguistics*, 28(2):61–83.

1000 most common words. 2021. 1000 most common words. https://1000mostcommonwords.com/1000-most-common-turkish-words/.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830.

Utku Türk, Furkan Atmaca, Şaziye Betül Özateş, Gözde Berk, Seyyit Talha Bedir, Abdullatif Köksal, Balkız Öztürk Başaran, Tunga Güngör, and Arzucan Özgür. 2021. Resources for turkish dependency parsing: Introducing the boun treebank and the boat annotation tool. *Language Resources and Evaluation*, pages 1–49.

Jonathan J Webster and Chunyu Kit. 1992. Tokenization as the initial phase in nlp. In *COLING 1992 Volume 4: The 14th International Conference on Computational Linguistics*.

Jesse O Wrenn, Peter D Stetson, and Stephen B Johnson. 2007. An unsupervised machine learning approach to segmentation of clinician-entered free text. In *AMIA Annual Symposium Proceedings*, volume 2007, page 811. American Medical Informatics Association.

Meng Xuan Xia.