Progress Report

The preprocessing operations such as tokenization, sentence splitting, and normalization are necessary to apply to the given text before advanced NLP tasks. Tokenization is a process of identifying tokens which are usually words within a text. In order to determine named entities or part-of-speech tags of the words, the words of the text should be extracted priorly. In addition to that, to obtain meaningful information from a given text such as a blog post that contains words formed by special characters instead of regular characters, these characters should be converted into meaningful ones, in other words, the text should be normalized. Furthermore, the identification of the end of sentences makes it possible to translate languages. The project aims to build a text processing pipeline that consists of tokenization, sentence splitting, normalization, stemming, and stopword elimination operations. Our progress is stored in https://github.com/OnurSefa/preprocessTR repository.

The training and evaluation steps of the operations utilize the .conllu files under the Universal Dependencies Turkish BOUN Treebank. BOUN Treebank datasets contain sentences belonging to different domains and taken from the Turkish National Corpus. BOUN Treebank consists of more data compared to other Turkish treebanks. Moreover, usage of this dataset may be helpful for improving the dataset and the developed preprocessing applications may support future NLP projects developed under Boğaziçi University.

1.Tokenization

Tokenization is the process of division of a given text into words or terms. As stated above, to analyze a document or a text, words are the most important components such that they can be interpreted as the features of a document. At the end of this project, there will be a rule-based tokenizer and a machine learning-based tokenizer. We are inspired by the tokenizing methods of Stanford NLP Group's Stanza and Charles University's UDPipe tools where they use whitespace as an indicator of a token boundary. UDPipe Also, the course slides are used as a guide.

1.1.Rule-based Tokenizer

The rule-based tokenizer's workflow consists of two steps: first, identifying the punctuations such as '.', ',', '?', '"' and modifying the text accordingly, and second, splitting the texts by whitespaces. A significant exceptional case is when the iterator encounters a period ('.'). The period may signal the end of the sentence and be considered as a separate token or the period may belong to an abbreviation such as 'Dr.'. To discriminate these different duties of period character, we first thought of building an abbreviation dictionary. However, we recognized that only checking if the period is at the last position of the sentence is sufficient to distinguish different types of period characters. Moreover, the '.' character in URLs and emails is also not modified since the dots usually stand in between other characters. The algorithm is designed as when an end of sentence dot or other punctuations is encountered, the algorithm modifies that character of the text by adding whitespace to its sides. Following this step, the modified text is split into its components by whitespace. As a remark, since the dataset we have used does not include multi-word tokenization, our algorithm does not perform tokenization to identify multi-words. For future work, the dataset might be extended to capture multi-words as a single token.

tr_boun-ud-test.conllu	0.9418342652090461
tr_boun-ud-dev.conllu	0.9256722899615835

Figure1. Accuracy of tokenizer

As stated in Figure1, the accuracy of the tokenizer might be improved. Incorrectness in some cases is because of our tokenizer's inability to tokenize separately the '...' punctuation. Moreover, the development and test data contain some mislabeled data where '"' or '"' characters belong to the token, e.g. 'dedi"'. To fix this, the different datasets might be considered.

1.2.Machine Learning-based Tokenizer

The machine learning-based tokenizer will perform using logistic regression classifier. In order to utilize a logistic regression classifier, the task is converted into a binary classification task. Characters of training and test data are labeled as 'end of token' and 'non', 1 and 0 respectively. When the classifier is trained, it is expected to predict a label for each character.

2. Sentence Splitter

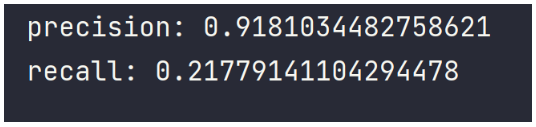
2.1 Initial Implementation

First task was to detect the identifiers which indicate the end of the sentence. We found out that every sentence must end with a punctuation. Base intuition was finding which punctuations identifies end of the sentence. Triple dots, dot, question mark, exclamation point, or quotation marks may indicate the end of the sentences. In the fist implementation of this task, we also implemented some difficult cases like having dot inside the exclamation points.

After the implementation of the base code, we imported chosen dataset to extract data. To achieve this, we used pyconll library which enables reading conll formatted files from python code. We connected every sentence together to prepare a mock document. This process gave us the input string. To test the performance of our algorithm we also needed to implement correctly partitioned version of this input. Thus, we created a string in which ~ symbol appears in between sentences to differentiate the sentences.

Last part was preparing a correctly designed evaluation function to test whether our application works well. Every character in the generated output string and wanted string holds positive or negative value. Basically, ~ character is positive, and all other characters are negative. Because of the abundancy of negative valued characters, we believe that using accuracy value will give us misleading information. We value to detect the ~ symbols, not other characters in the given input.

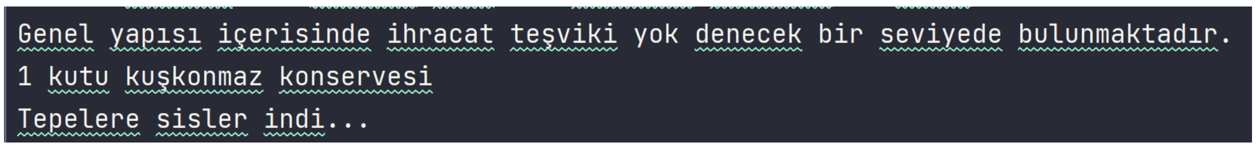
After the evaluation process, we see that our program ridiculously low results. First action was to inspect the correctness of the evaluation function. Before, we were comparing strings index by index. Apparently, this technique was not working well. So that, we created an evaluation method which thinks strings as tapes and decides whether step on both of them or only one of them. This abstract definition does not give clear methodology, but it would be too technical otherwise. Even though we have evolved our evaluation function, we still got very low recall values (Figure 1).



```
precision: 0.9181034482758621
recall: 0.21779141104294478
```

Figure 1: Initial Algorithm Results

This result indicates that when our program says there is a sentence ending, it finds it correctly. Unfortunately, capability of our program on finding the sentence endings from the given input is not efficient. Thus, we examined our dataset to find out the main reason. Our findings surprised us because of the erroneous set of sentences. In Figure 2, we can see that “1 kutu kuşkonmaz konservesi” is not a complete sentence but our dataset disagrees.



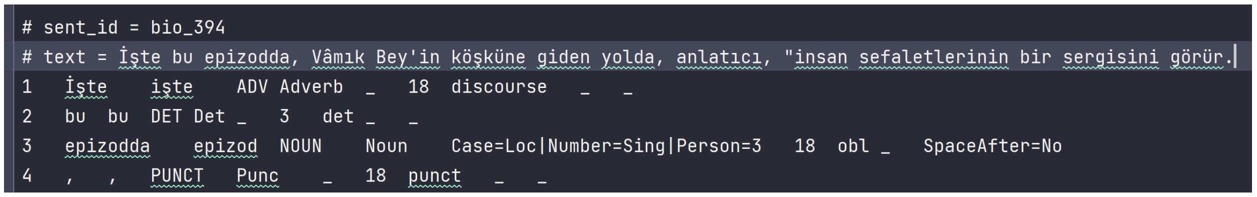
```
Genel yapısı içerisinde ihracat teşviki yok denecek bir seviyede bulunmaktadır.
1 kutu kuşkonmaz konservesi
Tepelere sisler indi...
```

Figure 2: Erroneous Sentence Example from Our Dataset

When we returned to our base assumption “Every sentence should end with a punctuation.”, we believe that these erroneous cases should be removed from our dataset. So that, we implemented our code again.

2.2 Second Implementation

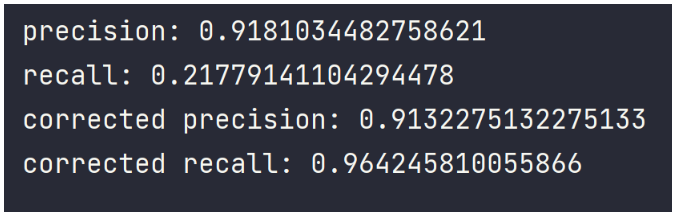
In the second phase, we developed a function which chooses sentences which includes at least one punctuation at the end of their index and gives to our algorithm. After that, we see slight improvement on both recall and precision, but it was still in the same level. We decided to continue our investigation on the chosen dataset. At the end, we found out that there are other error types in the dataset. For instance, in the Figure 3, the sentence includes opening quota but not the closing one. This causes our program to not detect this as a finished sentence. So, we discarded hard case detection system from our algorithm.



```
# sent_id = bio_394
# text = İşte bu epizodda, Vâmık Bey'in köşküne giden yolda, anlatıcı, "insan sefaletlerinin bir sergisini görür.
1   İşte    işte    ADV Adverb  _   18  discourse   _   _
2   bu  bu  DET Det _   3   det _   _
3   epizodda    epizod  NOUN    Noun    Case=Loc|Number=Sing|Person=3   18  obl _   SpaceAfter=No
4   ,   ,   PUNCT   Punc    _   18  punct   _   _
```

Figure 3: Example Erroneous Sentence from Dataset

After updating our algorithm and evaluation function we tested our findings. In the Figure 4 we clearly see that corrected version outperforms the previous one. But we want to keep previous architecture too. It uses correct way to find sentence endings. New version just suits the used dataset.



```
precision: 0.9181034482758621
recall: 0.21779141104294478
corrected precision: 0.9132275132275133
corrected recall: 0.964245810055866
```

Figure 4: Recall and Precision Scores of Old and New Algorithms

2.3 Future Work

We understand the ability to create well algorithms depends on using correct inputs. So that, we will search better datasets to test our algorithm. After evaluation process, we might want to update our algorithm. Also, we will solve this problem using ML based algorithms.

3.Stopword Elimination

Stopword elimination is one of the most necessary preprocessing operations especially in topic classification and frequency finding tasks. For this purpose, a stopword lexicon is created by adding a repository we found on Github.

References

https://github.com/ufal/udpipe

https://github.com/stanfordnlp/stanza

https://github.com/tkorkunckaya/Turkish-Stopwords