**N-Gram Algorithm Implementation**

Aim of this project is to implement an N-Gram algorithm and analyze that algorithm for unigram, bigram and trigram. To use the algorithm, a corpus is needed. Therefore, for this assignment, a small corpus that includes 5 Turkish novels have been used.

**File Operations and Data Preparation**

To use data in the corpus, all texts files needs to be rea. So, all files in the corpus have been read one by one as whole text, not line by line. As the files are in Turkish, “ISO-8859-9” encoding has been used. After that, all delimiters and some multiple spaces in the text have been cleaned. These two operations have been done by two different Regex that replace a space for all delimiters and multiple spaces. After cleaning process, whole text is ready to send to N-Gram algorithm.

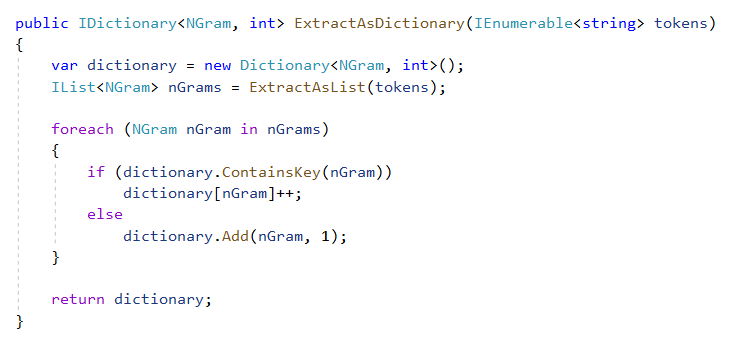
**N-Gram Algorithm**

As a N-Gram algorithm, “Nuve” [1] library that is the one of the natural language processing libraries for Turkish language has been used. 3 classes named “NGramDictionary”, ”NGramExtractor” and ”NGram” have been added to project. All required processes will be explained later in this report, but shortly “NGram” is used for creating objects that represents n-grams consist of n tokens, “NGramDictionary” is used for creating a dictionary for any gram that holds “NGram” objects as a key and their frequencies as a value in the dictionary, ”NGramExtractor” is used for extract tokens for the desired n-gram from text.

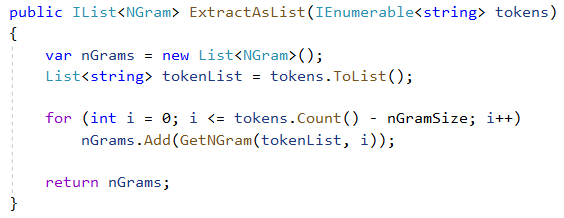
These 3 classes taken from Nuve library has not been used originally. Some changes made on those classes. Unused procedures and functions have been removed, some required functions have been added and some functions have been modified. Normally, the algorithm takes minimum and maximum values (in NGramExtractor class) for n-gram and places all different types of the n-grams in one dictionary. That was changed and the algorithm now takes just only one n-gram value and all the same value n-grams were found in the dictionary. In that way, the most 100 frequent n-grams could be sorted easily and be used in the UI. Other way, that could still be made, but some more redundant functions needed to be added. In NGramExtractor, “Validate”, “ExtractAsSet” functions removed. “GetNGram” function have been changed, because the algorithm now just takes one n-gram value. That’s why, “GetNGrams” function removed totally. In NGramDictionary, “Validate”, “ToString”, “GetFrequency” and “DeserializeFrom” functions removed, since they are not being using in our program.

When we come to flow and working phases in algorithm, the first is that a NGramDictionary is created for each n-gram type (unigram, bigram, trigram). NGramDictionary gets NGramExtractor object in its constructer. NGramExtractor is took n-gram number in its constructer while creating. That extractor object is used, after a new text file cleaned and sent to dictionary. The extractor gets the all clean text and separate them in to tokens according to the n-gram and adds them to a list. So, we need to understand NGramExtractor initially. Procedures and functions of NGramExtractor are listed below with their explanations:

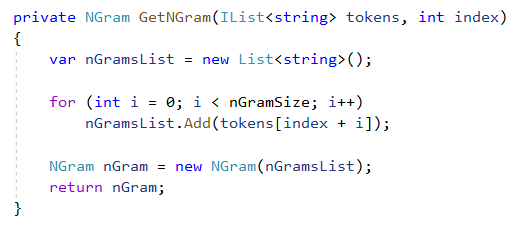
* **ExtractAsDictionary:** This function gets list of tokens and returns as an output a dictionary. The aim of this function is to extract all n-grams as a map from unique n-grams to their frequencies. To do this, all n-grams have to be listed. Therefore, “ExtractAsList” function is used in this function.



* **ExtractAsList:** This function gets input a string list that holds the all tokens and gives a NGram object list that holds all n-grams. The aim of this function is to extract all n-grams from tokens according to the number of n. To get all n-grams one by one, GetNGram function is used in this function in a for loop that traverse all list.

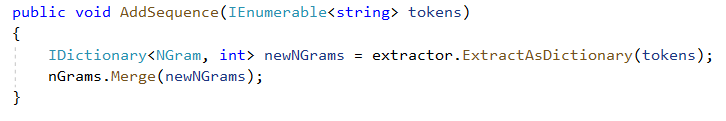


* **GetNGram:** This function gets all taken list, value of n and index number and gives output a NGram object. Index value helps us where should we start to get tokens from list and n value helps us how many token we need to add to NGram object in front of that token.

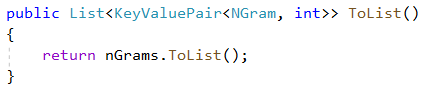


After we know that the all functions and working of the NGramExtractor class. NGramDictionary needs to be explained. As explained before, NGramDictionary gets NGramExtractor in its constructor. Besides, in this class, there is one dictionary object that key is NGram object and value is the frequency of that n-gram. Procedures and functions of NGramDictionary are listed below with their explanations:

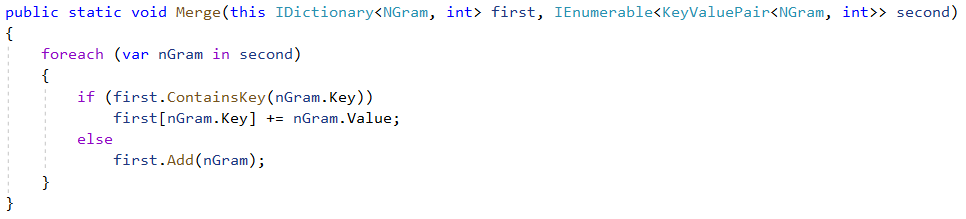
* **AddSequence:** When a text file read from corpus, whole text is parsed to tokens and sent to this procedure as an input. Then, using the ExtractAsDictionary function of the NGramExtractor object in this class, all tokens are converted to a dictionary. Lastly, with “Merge” function, that dictionary returned from the extractor is merged with the main dictionary in this class.



* **Merge:** This procedure merged the dictionary that constructed when new file read and the main dictionary for all of the corpus.



* **ToList:** This function is made conversion the main dictionary to the list. This list is made easy the sorting process for the find the most frequent n-grams.



**Results**

|  |  |
| --- | --- |
| kitap 3 bölüm | 65 |
| kitap 1 bölüm | 56 |
| kitap 2 bölüm | 55 |
| başka bir şey | 18 |
| diye karşılık verdi | 18 |
| 2 bölüm 12 | 17 |
| ne yazık ki | 11 |
| 3 bölüm 9 | 11 |
| ne var ki | 11 |
| o kadar ki | 10 |
| 1 bölüm 26 | 10 |

The most frequent 10 n-grams are listed below for unigrams, bigrams and trigrams. You can see the most frequent 100 n-grams from the program.

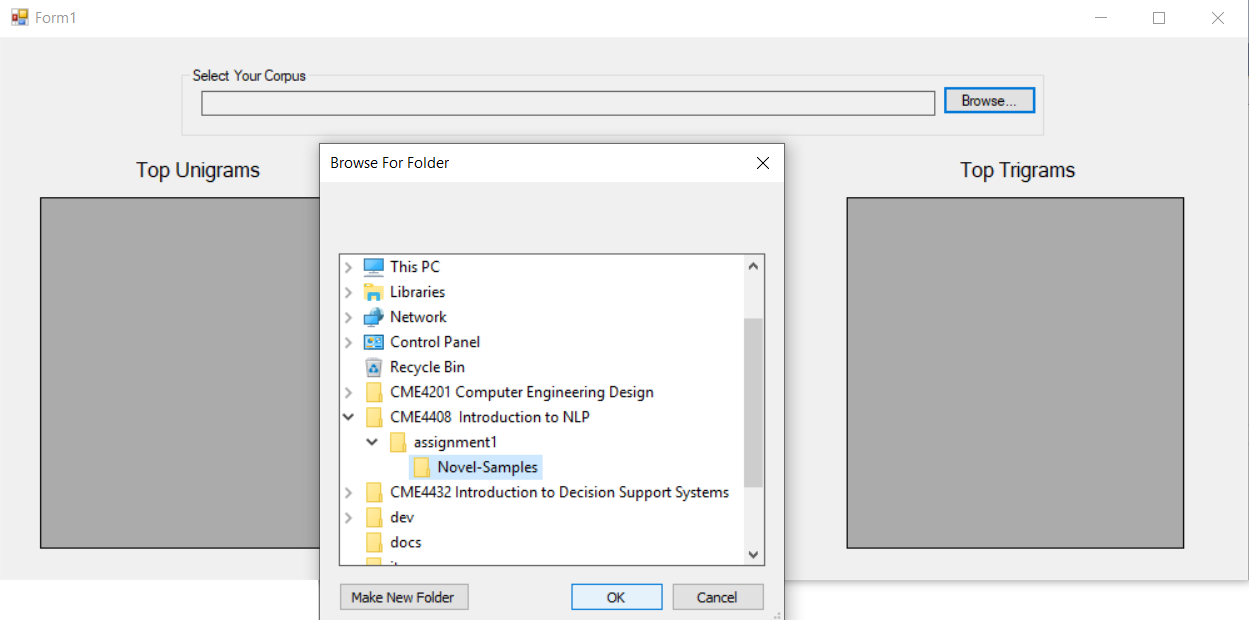
|  |  |
| --- | --- |
| bir | 3497 |
| ve | 1730 |
| bu | 1123 |
| de | 903 |
| da | 788 |
| daha | 697 |
| için | 677 |
| ne | 615 |
| gibi | 515 |
| ki | 512 |
| kadar | 494 |

|  |  |
| --- | --- |
| bir şey | 126 |
| o kadar | 94 |
| ya da | 84 |
| ne kadar | 76 |
| büyük bir | 65 |
| 3 bölüm | 65 |
| kitap 3 | 65 |
| başka bir | 60 |
| hiç de | 58 |
| kitap 1 | 56 |
| 1 bölüm | 56 |

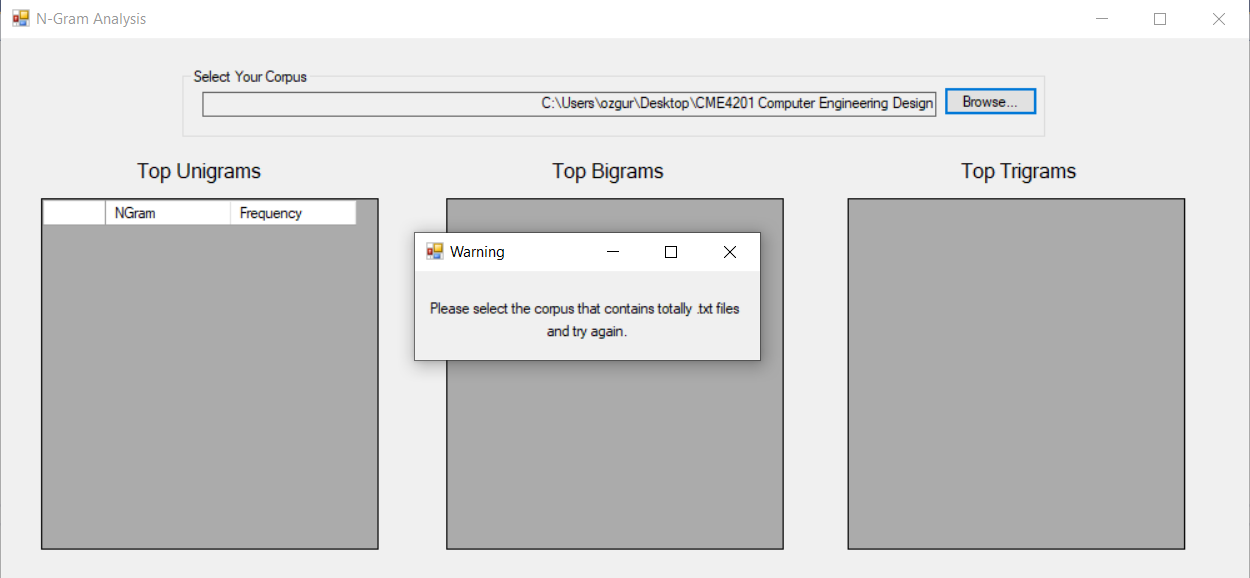
Creating unigrams took 211 milliseconds, bigrams took 189 milliseconds and trigrams took 208 milliseconds.

**Screenshots**

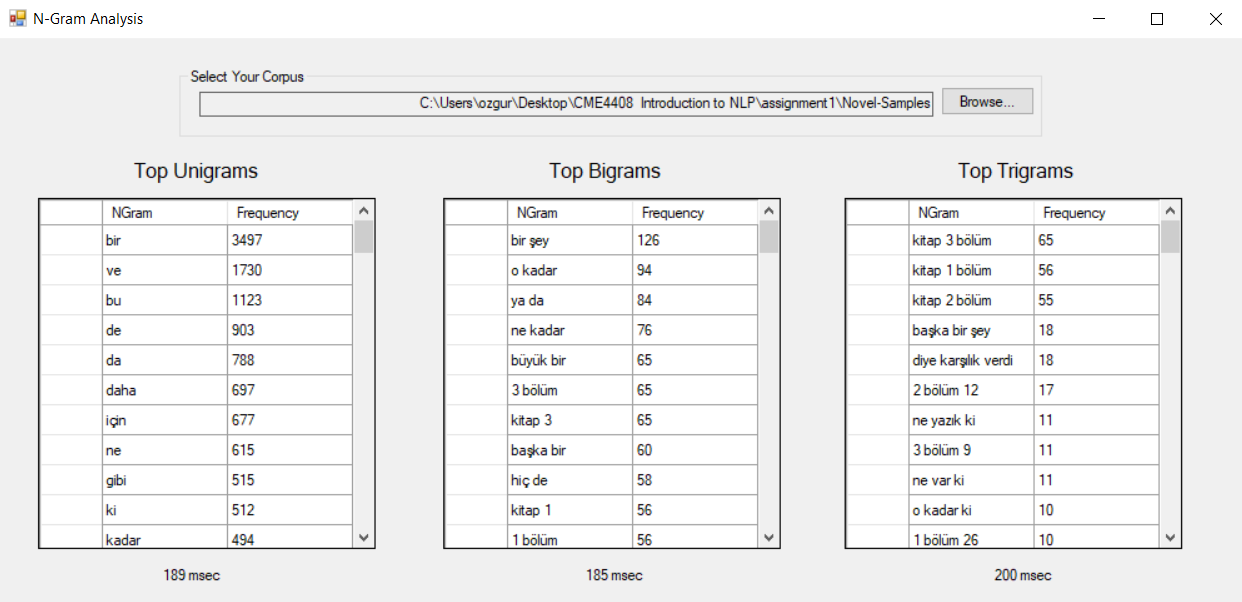
When the program is opened, first thing the user have to do is to browse the folders and select the corpus. Once the user clicks “OK” button, top 100 frequent unigrams, bigrams and trigrams are going to be listed below.



If the user selects the corpus folder that not contains full of .txt files. The program gives a warning and the user should select the correct project corpus again.



Once the user selects the corpus correctly, the most frequent 100 unigrams, bigrams and trigrams are listed in a few seconds in grid views below. The user also can see how long it takes for each n-gram results.



**References**

1. <https://github.com/hrzafer/nuve>