NLP HOMEWORK 1 REPORT

BATUHAN BASAK

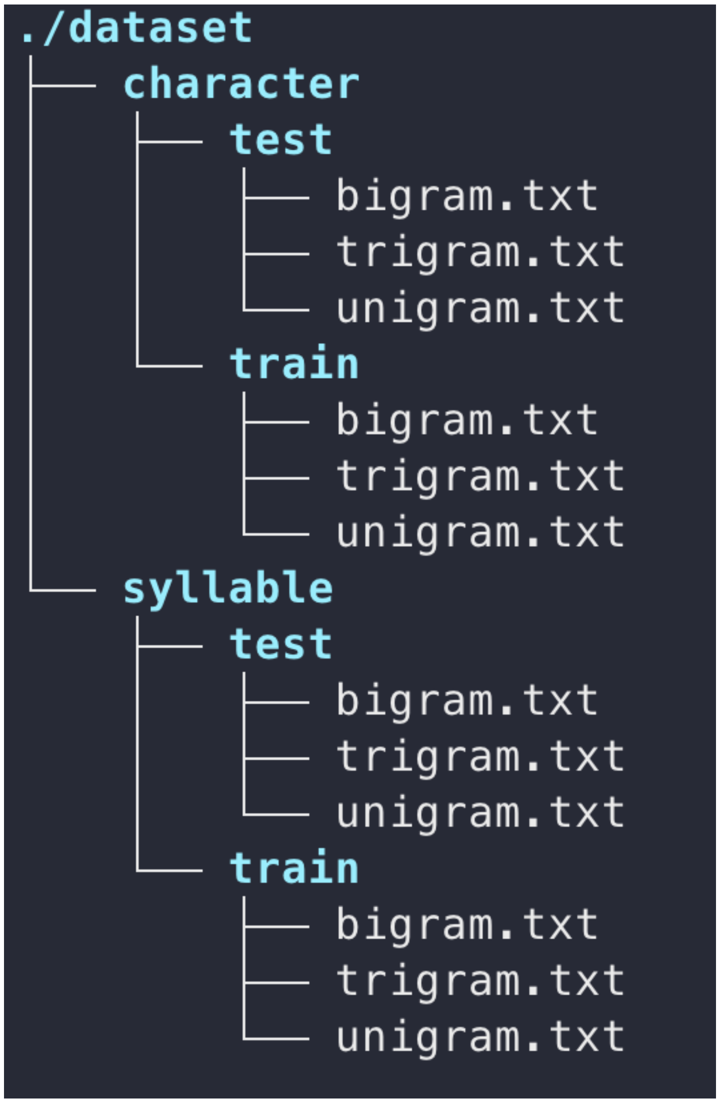
161044021

1. INTRODUCTION

An n-gram is a sequence of n tokens, and an n-gram model is a model that estimates nth token of sequence respect to n-1 previous tokens[1]. A 1-gram is also called Unigram, a 2-gram is also called Bigram model, and a 3-gram is also called Trigram. The tokens of an N-gram models can be either characters, syllables, or words. In this homework, I have implemented n-gram models which are character-based unigram model, character-based bigram model, character-based trigram model, syllable-based unigram model, syllable-based bigram model, and syllable-based trigram model. I have generated sentences by using each model. And I compute the perplexity of each model using testing data. To increase the readability of the code, I have implemented each process in an Interactive Python Notebook, ipynb, files. The notebook files are *preprocessing.ipynb*, *creating_models.ipynb*, *generate_sentence.ipynb*, and *compute_perplexity.ipynb*.
To use the power of Object-Oriented-Programming paradigm, the model has been implemented as a class, which is called *NGramModel*, in the file *ngram_model.py*. The models are stored in the folder *models*. And the datasets are in the folder *dataset*. If user wants to recreate them from scratch, then user must delete dataset and models directories first, then he/she must run all scripts in files *preprocessing.ipynb* and *creating_models.ipynb* consecutively.

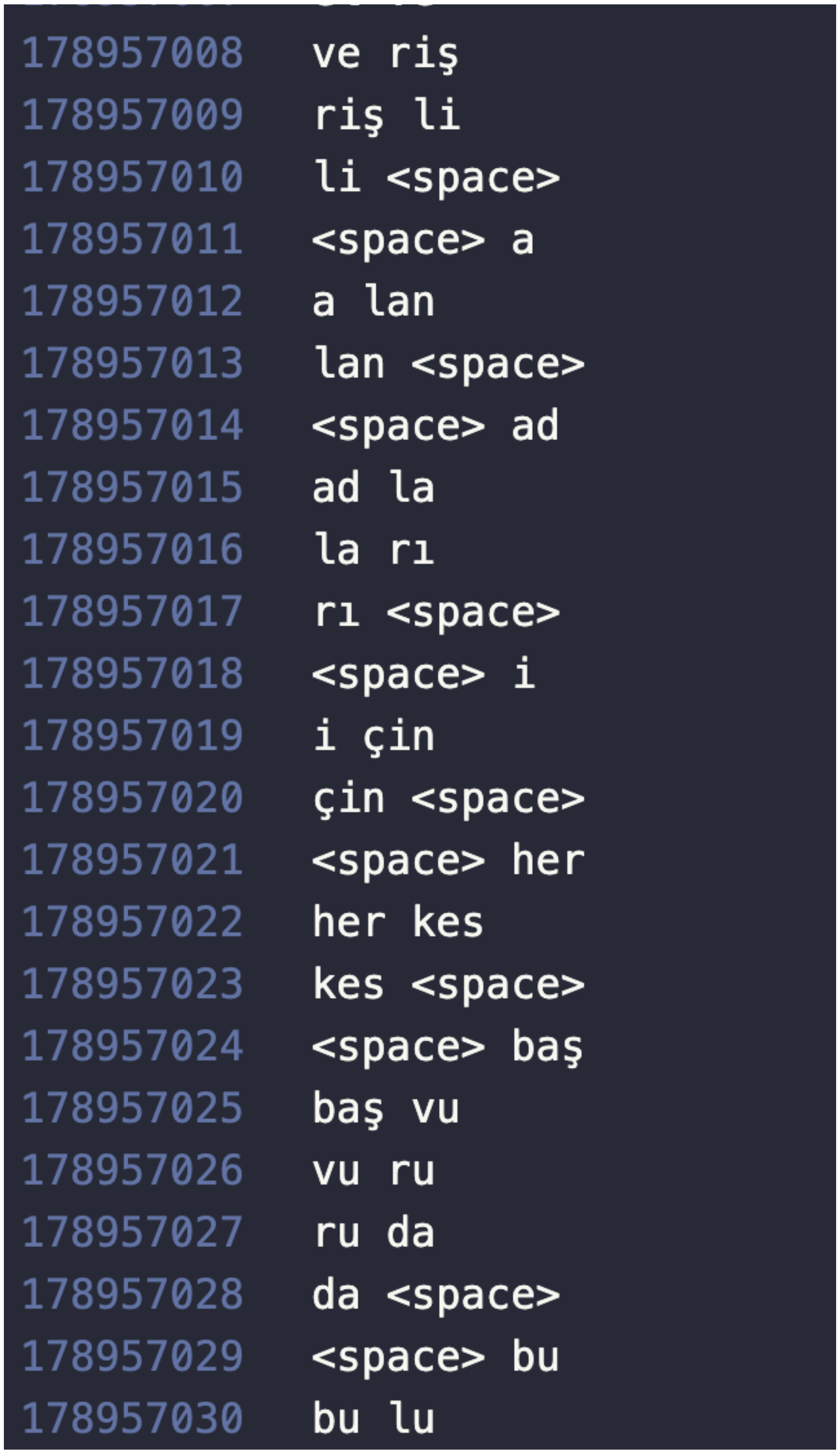
2. CREATING DATASET

The Turkish Wikipedia Dump dataset[2] is used as our corpus. The dataset file processed line by line and each word of the line is tokenized, and each token written into another file. For tokenizing each word for syllable-based model, the syllabification library, which is called turkishnlp[3], has been used. The file consists of tokens of character-based model and the file consists of tokens of syllable-based model are different files. The empty lines and lines that consist of html tag have been ignored. The character-based unigrams, bigrams, and trigrams were generated by using character-based tokens, which were stored in a file at previous step. This process has been done for syllable-based tokens also to generate syllable-based unigrams, bigrams, and trigrams. And each of them was stored in a separate file. After that, each of them was split into training and testing files. The character-based data was stored in a subdirectory which is called *character*. And in *character* subdirectory, training data was stored in subdirectory *train*, testing data was stored in subdirectory *test*. The same process has been done for syllable-based data also. The tree structure of dataset directory is in Figure 1. A screenshot of a training bigram dataset for syllable-based model is in Figure 2. One more important thing is to use turkishnlp library we need to have a directory which is called *Trnlpdata*. This one musn’t be deleted.



```
./dataset
├── character
│   ├── test
│   │   ├── bigram.txt
│   │   ├── trigram.txt
│   │   └── unigram.txt
│   └── train
│       ├── bigram.txt
│       ├── trigram.txt
│       └── unigram.txt
└── syllable
    ├── test
    │   ├── bigram.txt
    │   ├── trigram.txt
    │   └── unigram.txt
    └── train
        ├── bigram.txt
        ├── trigram.txt
        └── unigram.txt
```

Figure 1: The tree structure of directory dataset

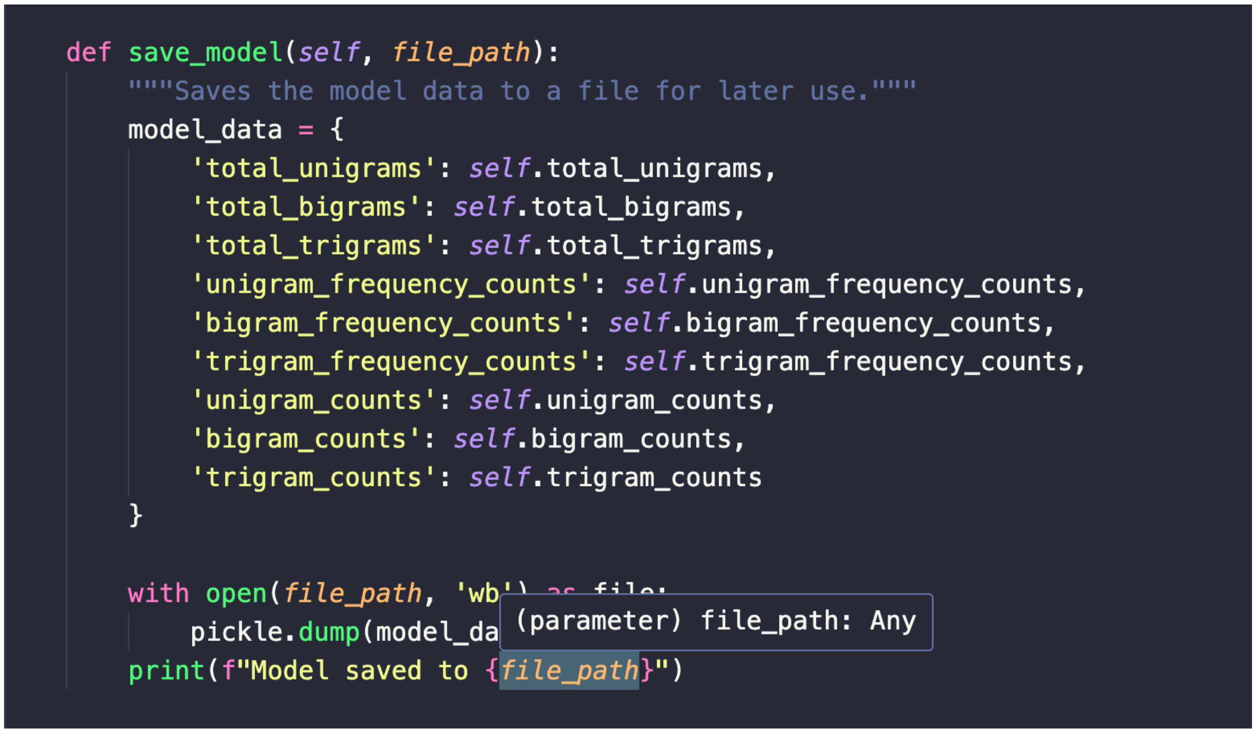


```
ve riş
riş li
li <space>
<space> a
a lan
lan <space>
<space> ad
ad la
la rı
rı <space>
<space> i
i çin
çin <space>
<space> her
her kes
kes <space>
<space> baş
baş vu
vu ru
ru da
da <space>
<space> bu
bu lu
```

Figure 2: Bigram training file for syllable-based model

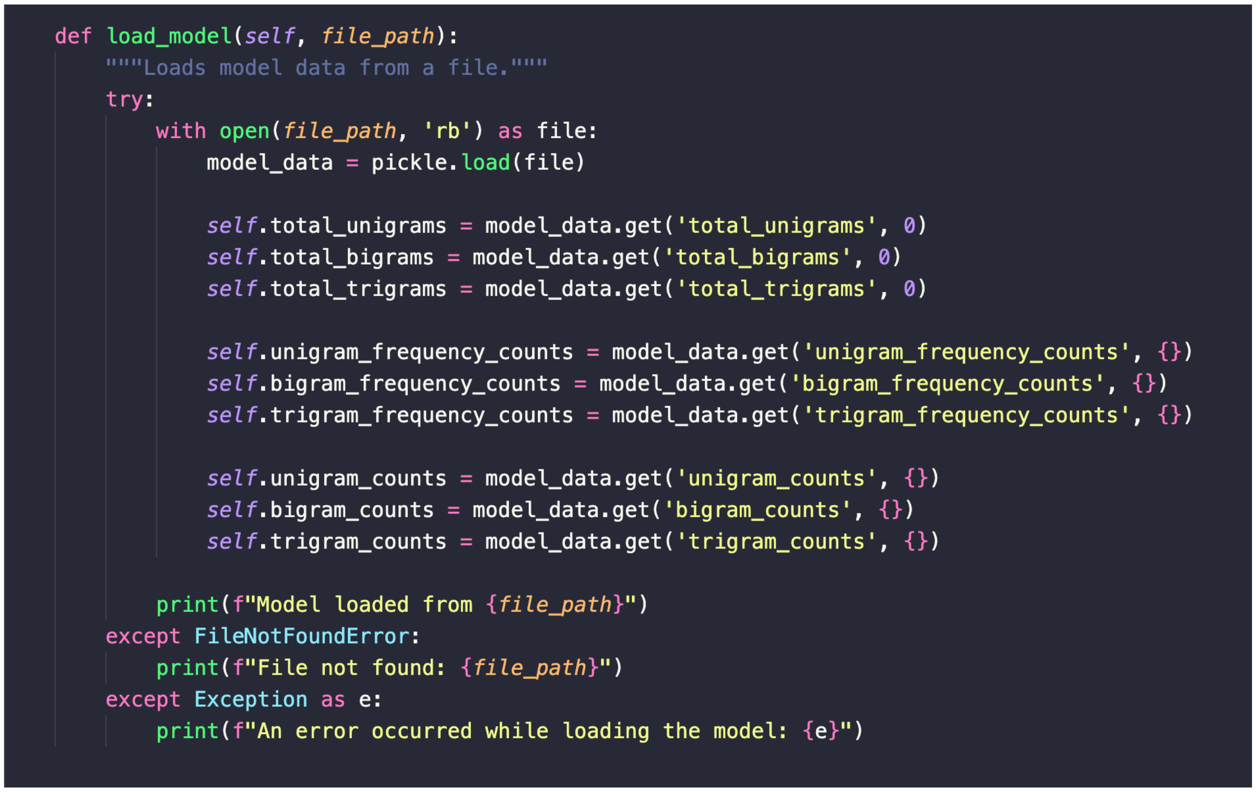
3. DESIGN & IMPLEMENTATION

The NGram class contains both unigram, bigram, trigram model at the same time. It uses three dictionaries to stores the information for a given key that how many times does it appear in given training data. The dictionary of holds counts of this information for unigrams is called *self.unigram_counts*. So, *self.unigram_counts['a']* holds count of token 'a' in given training data for a character-based model. Another example for a unigram is *self.unigram_counts['ar']* holds count of token 'ar' in given training data for a syllable-based model. The bigrams counts are stored in dictonary *self.bigram_counts*, and the trigram counts are stored in dictionary *self.trigram_counts*. The *self.bigram_counts[(w0, w1)]* holds the count of bigram [w0, w1] in given training data, where w0 and w1 are tokens. When the *train_model* method has been called, this counts are counted. There 3 more dictionaries which are required for implementing good-turing smoothing algorithm that holds frequency of frequencies. These are *self.unigram_frequency_counts*, *self.bigram_frequency_counts*, and *self.trigram_frequency_counts*. We also need to store number of samples for unigrams, bigrams, and trigrams in order to compute the probabality. These are stored in *self.total_unigrams*, *self.total_bigrams*, and *self.total_trigrams*. The model can be stored in a file, to load the mode and use later on, in a file with given file name. To save the model, there is a method which is called *save_model* that takes the file name with full path. The *save_model* method is in Figure 3. The method to load model from a given file is *load_model* is in Figure 4. To generate a sentence, there is a method which is called *generate_sentence*. The method *generate_sentence* takes three inputs which are *n*, *num_probables*, and *max_len*. The n indicates which ngram model user want to use to generate a sentence. If n is 1, the model uses unigram to generate a sentence. If n is 2, then the model uses bigram, else it uses trigram to generate sentence. The method *generate_sentence* picks a random item from list that consists of most probable next tokens estimated by the model. The number of items is adjusted by the second parameter of the method *generate_sentence* which Is called *num_probables.* The sentences end with a special token <*end*>. But the model may not be good at estimating the next one and this may cause to never get an <*end*>. To solve this, I have also restricting the length of the sentence, which is the third parameter of the method *generate_sentence* which is called *max_len*. The method *generate_sentece* is in Figure 5. To compute a perplexity of a given test data, the method *compute_perplexity* exists. The *compute_perplexity* method takes two parameters, which are *input_file* and *n*. The *input_file* is the name of the file with full path that contains test data. The *n* indicates which n-gram to use to compute the perplexity. If the n is 1, then the model uses unigram. If n is 2, then the model uses bigram. Else the model uses trigram model. The method *compute_perplexity* is in Figure 6.



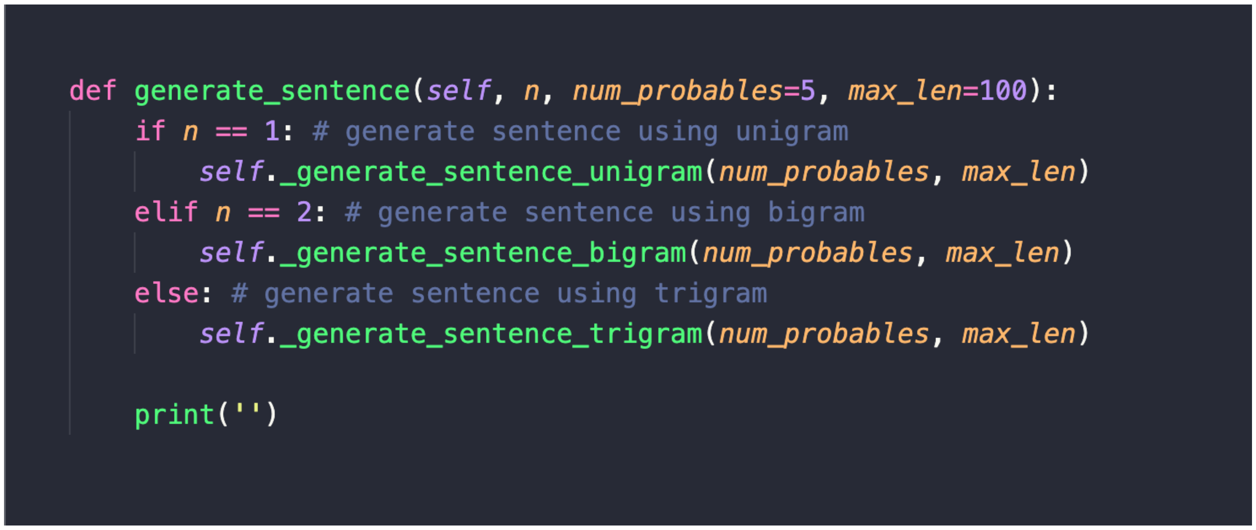
```python
def save_model(self, file_path):
    """Saves the model data to a file for later use."""
    model_data = {
        'total_unigrams': self.total_unigrams,
        'total_bigrams': self.total_bigrams,
        'total_trigrams': self.total_trigrams,
        'unigram_frequency_counts': self.unigram_frequency_counts,
        'bigram_frequency_counts': self.bigram_frequency_counts,
        'trigram_frequency_counts': self.trigram_frequency_counts,
        'unigram_counts': self.unigram_counts,
        'bigram_counts': self.bigram_counts,
        'trigram_counts': self.trigram_counts
    }

    with open(file_path, 'wb') as file:
        pickle.dump(model_da (parameter) file_path: Any
    print(f"Model saved to {file_path}")
```

Figure 3: The method save_model



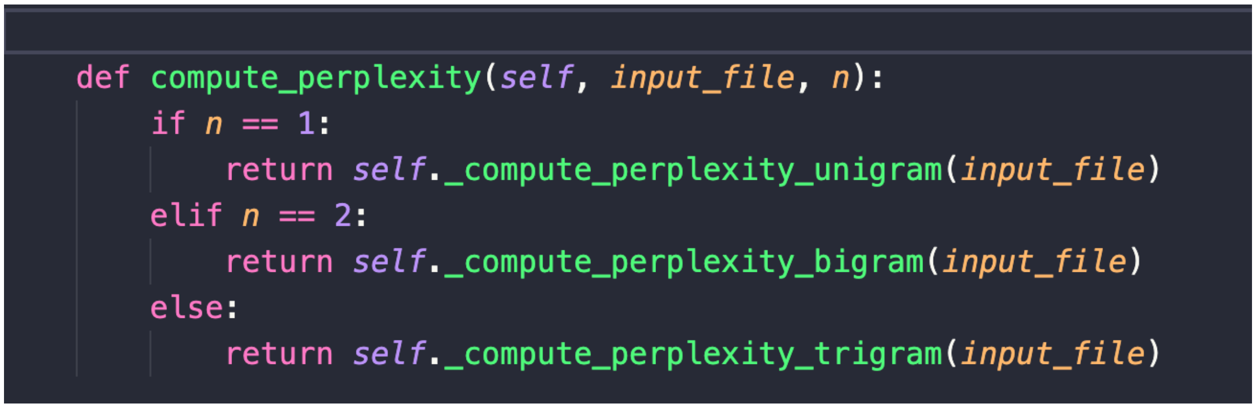
```python
def load_model(self, file_path):
    """Loads model data from a file."""
    try:
        with open(file_path, 'rb') as file:
            model_data = pickle.load(file)

            self.total_unigrams = model_data.get('total_unigrams', 0)
            self.total_bigrams = model_data.get('total_bigrams', 0)
            self.total_trigrams = model_data.get('total_trigrams', 0)

            self.unigram_frequency_counts = model_data.get('unigram_frequency_counts', {})
            self.bigram_frequency_counts = model_data.get('bigram_frequency_counts', {})
            self.trigram_frequency_counts = model_data.get('trigram_frequency_counts', {})

            self.unigram_counts = model_data.get('unigram_counts', {})
            self.bigram_counts = model_data.get('bigram_counts', {})
            self.trigram_counts = model_data.get('trigram_counts', {})

        print(f"Model loaded from {file_path}")
    except FileNotFoundError:
        print(f"File not found: {file_path}")
    except Exception as e:
        print(f"An error occurred while loading the model: {e}")
```

Figure 4: The method load_model



```python
def generate_sentence(self, n, num_probables=5, max_len=100):
    if n == 1: # generate sentence using unigram
        self._generate_sentence_unigram(num_probables, max_len)
    elif n == 2: # generate sentence using bigram
        self._generate_sentence_bigram(num_probables, max_len)
    else: # generate sentence using trigram
        self._generate_sentence_trigram(num_probables, max_len)

    print('')
```

Figure 5: The generate_sentence method



```python
def compute_perplexity(self, input_file, n):
    if n == 1:
        return self._compute_perplexity_unigram(input_file)
    elif n == 2:
        return self._compute_perplexity_bigram(input_file)
    else:
        return self._compute_perplexity_trigram(input_file)
```

Figure 6: The method compute_perplexity

4. RESULTS

The perplexity scores are in Table 1. The bar plot of perplexities is in Figure 7. The sentences generated by character-based models are in Table 2. The sentences generated by syllable-based models are in the Table 3. According to the Table 1 and Figure 7, we can say that character-based models have less perplexity scores than corresponding syllable-based models. According to the perplexity scores, for both character-based and syllable-based models, bigram models become the models that have the least perplexity, and the trigram models become the models that have the most perplexity scores.

Generated sentences, respect to the human evaluation, shows that, for both character-based model and syllable-based model, when the n is increased, the generated sentence becomes more successful, where n is the number of previous tokens considered by the n-gram model (E.g. n=0 is a unigram model, n=1 is a bigram model, and n=3 is a trigram model).

Table 1: The perplexity scores

	Unigram	Bigram	Trigram
Character-Based	25.91	13.56	126.46
Syllable-Based	184.59	28.31	340.17

Table 2: The sentences generated by character-based models

N-gram Model	The Sentence Generated
Unigram	aaaaa a a aa aa a a aa a aaa a aa a a aa a aaaa aaaaa a a a aaaaaaaa
Bigram	0 kan kan bi karararar bi kanderandandan k bar k bi k kar k k k bander bi barararararan binde ban
Trigram	19990 ye başamından bir varaktanınde kuları bağlan kanı ileri biliklan bağın bağlı isi kan kurulan

Table 3: The sentences generated by syllable-based models

N-gram Model	The Sentence Generated
Unigram	.dadada.dale ledalalalelalalale.leladalela .lele.laledadale lela.la...leledaladale da . ..da.ladala.lelele.lale le da da le ..dale.la. le..lele.le da dalela
Bigram	ile içinde ileri içinde ve veyapılandı. ilerinede ve ve ileri veya ile ilerine veya ile veya veya ilerinede içinde veya ve için için ve ile ve ile içindeki i
Trigram	için kullanılmaktadır. 18 tarihi ve 1980 tarihi ileri arasından olabilir bir arasındakikadaşınıncayarak görevi yoktur fakültatifler veyahut ve 1990 yılı bir arasındandır.

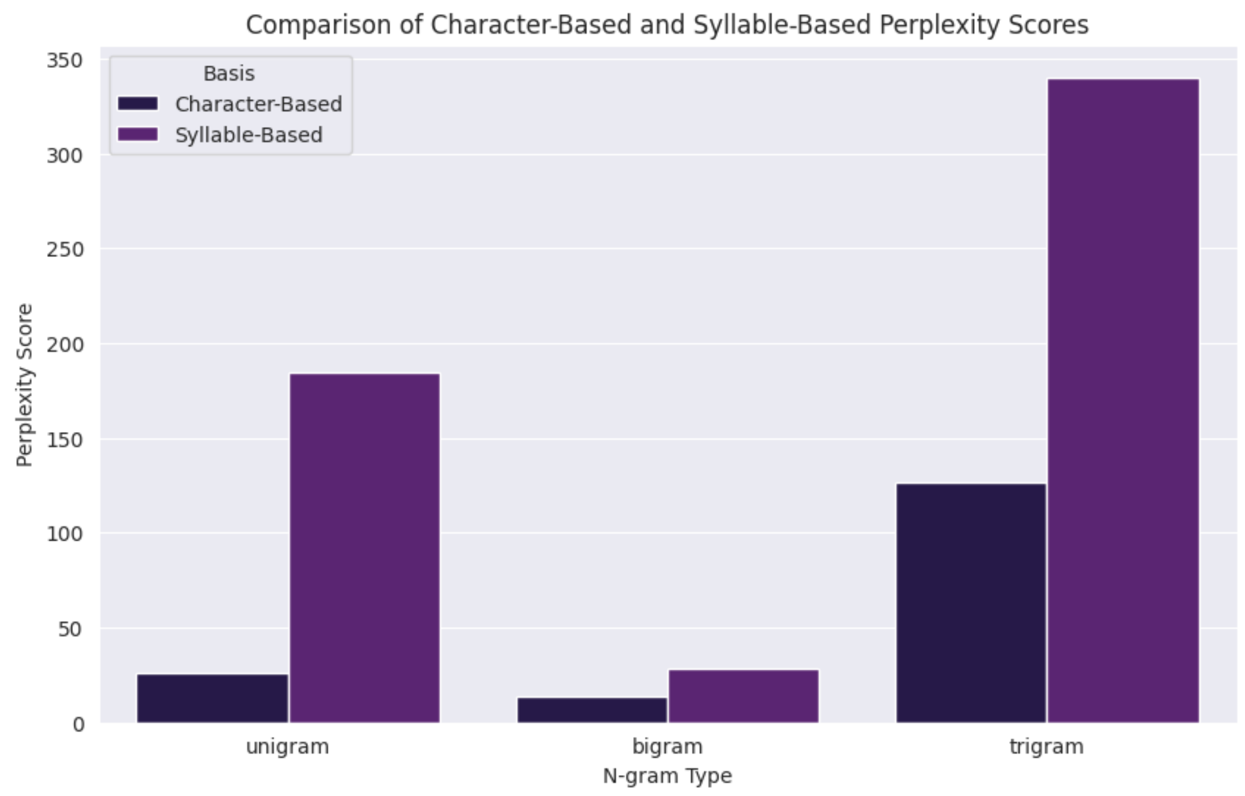


Figure 7: Comparison of Character-Based and Syllable-Based Perplexity Scores

5. LLM USAGE

The ChatGPT[4] has been used in this assignment mainly for two things - code optimization and bug fixing. The code optimization has been required during creating dataset, storing dataset, and storing models' stats. And bug fixing has been required during implementing some internal methods of class NGramModel, especially for computing perplexity using Good-Turing smoothing algorithm.

6. REFERENCES

1. Dan Jurafsky, James H. Martin. Speech and Language Processing. 2024.
2. Mustafa Keskin. https://www.kaggle.com/datasets/mustfkeskin/turkish-wikipedia-dump
3. Metehan Cetinkaya. https://pypi.org/project/turkishnlp/
4. Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever. Improving Language Understanding by Generative Pre-training. 2018.