Hungarian Text Generator for Children's Novels

Szilárd Novoth, Zsombor Páncsics, Ádám Varga

***Abstract*— This work explores the field of Natural Language Generation (NLG) and develops a novel text generation algorithm for the Hungarian language using the Transformer model. The work uses data collected from Magyar Elektronikus Könyvtár (MEK) and introduces numerous language models. The results of the models are compared and matched against our state-of-the art Transformer based Hungarian Language Generator (HLG) method.**

***Index Terms*— Hungarian, Natural Language Generation, NLG, text generation, Transformer, GPT, BERT**

I. Introduction

TODO

II. Related Work

TODO

III. Preparing data

Collecting a good set of data is an essential part of any Machine Learning (ML) task. Language models, trained on "good quality" text corpus achieve better results and work more reliably. However, being able to obtain data that lacks any noise is rarely possible. The goal of this section is to introduce the reader to the most common methods to prepare text before it can be used to train the ML model.

A. Collecting the data

For many tasks there are various datasets online, readily available to train the ML model. The goal of this work is to create a Hungarian text generator algorithm which is able to output children stories when supplied with the appropriate input sequence. There is however no dataset specifically for generating children's novels in Hungarian language. For this reason a sample of youth books were collected from the Hungarian database: MEK [1]. While the obtained files were relatively clean, still, there were steps needed to be taken, before the text corpus could be fed into our HLG algorithm. In the following the most common data-cleaning methods will be introduced and we will highlight the ones applied in our algorithm.

B. Preprocessing

Preprocessing the data is an essential step in Natural Language Processing (NLP) tasks. Its purpose it to transform the text so that the ML algorithms can perform better. Historically there are three main components: 1) splitting, 2) normalization and 3) noise removal.

S. Novoth, Z. Páncsics, Á. Varga are with the Department of Electrical Engineering, Budapest University of Technology and Economics, Budapest, 1111, Hungary (email: novothsz@gmail.com, zsombor.pancsics@gmail.com, adamvarga0103@gmail.com)

Splitting is a method for stripping text into smaller components. A book can be split into chapters, sentences and words. Even words can be further stripped to subwords and to characters. In the case of normalization, the words are converted into the same format: they are lower cased and in some cases punctuations are removed. Numbers can be converted into their textual representation and contractions can be expanded. Normalization typically refers to removing unwanted elements from the text, like extra whitespaces and HTML tags. The word-art frequency representation of the words in the used text corpus is shown in Figure 1.

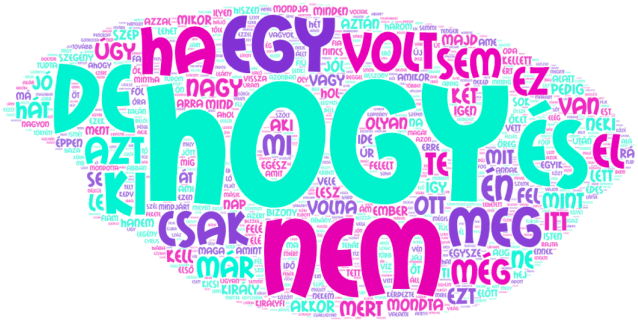

Fig. 1: A word-art frequency representation of words in the used text corpus [2]

IV. Input representations

Once the text has been preprocessed, it must be converted into numerical values before it can be fed into the ML algorithm. There are various ways the input can be generated and for each application the most appropriate method must be chosen. In the following three of the most popular methods will be presented: one-hot encoding, count vectorizer, word embedding.

A. One-hot encoding

One-hot encoding is useful when working with categorical features in a dataset. It uses sparse vectors where each element is zero, except those indices, which categorize the given data. In the case of text documents, tokens are replaced with their one-hot vectors. These vectors have the length of the vocabulary and contain zeros except at the index of the specific word where the value is one. As such, sentences can be turned into two dimensional matrices of size (n, m), n being the number of tokens and m the length of the vocabulary.

While the method is intuitive, it has some disadvantages. We can see, that with increasing vocabulary size the vectors become increasingly long and sparse. This can be avoided by

using character-level representation. In this case, the length of the vectors is limited by the number of characters present in the given language. Still, the length of the sentence is a function of the number of tokens it contains, making it less ideal for many of the ML models. The problem can be circumvented by truncating or padding the vectors.
To make one-hot encoding robust against the appearance of unknown words (or characters for that matter), an artificially enlarged vocabulary can be used, which can be filled with out-of-vocabulary tokens, when necessarily.

B. Counter vectorizer

Using the count vectorizer method, sentences are replaced by their count vectors. The length of these vectors is equal to the length of the vocabulary and each number at the specific index specifies how many times the given word has appeared in the sentence.

This technique reduces the size at which a given sentence is stored by representing it by a single vector instead of the large albeit sparse matrix present in the case of one-hot encoding. This however comes at a price: information about the sequential arrangement of the words is lost, only their count remains. This hinders the methods applicability to many NLP tasks, such as text generation.

C. Word embeddings

While representations like one-hot vector and count vectors are great for some problems like text classification, their usefulness in more complex NLP tasks is limited. One major disadvantage of one-hot vectors is that no comparison, no relationship between words can be made. Each word is represented by a vector which is orthogonal to all the other vectors in the multi-dimensional space, hence no distance or cosine-similarity can be calculated between them [3].

Word embeddings are capable of representing semantic meaning of words. Each word is characterised by a dense vector of length n. The size n of these vectors is dependent on how many corresponding features we are interested in, but usually ranges between 50 to 300. Each index corresponds to a specific feature like *"soft"*, *"dark"*, *"age"*. The numbers at these indicies show how representative the given feature is.

REFERENCES

[1] "Magyar Elektronikus Könyvtár."
[2] "Word Art."
[3] J. Hu, "An Overview for Text Representations in NLP," Mar. 2020.