Shivam Syal and Yash Mandavia
CS225, FA23
10/31/2023

CS 225 EC Assignment

Academic Reference

The algorithm we're planning on implementing is the Byte-Pair Encoding (BPE) algorithm (https://doi.org/10.48550/arXiv.2306.16837). Although this paper provides a more formal definition and proof of the algorithm, we wanted to involve ourselves in more application based definitions, so we are using a very detailed and easy to read guide by Philip Gage which provides code snippets to better understand the algorithms implementation (http://www.pennelynn.com/Documents/CUJ/HTML/94HTML/19940045.HTM).

Algorithm Summary

The Byte-Pair Encoding (BPE) algorithm is a text compression and tokenization algorithm which is widely used in small-scale natural language processing applications. With relation to C++, BPE is highly effective due to its small, fast expansion scheme. At its core, BPE is implemented by initializing a unique vocabulary with individual characters which are mapped to common phrases in a text. It iteratively merges the most frequent character pairs to create shorter tokens mapped to these individual characters. C++ is the most ideal framework to implement BPE within due to its efficient memory management and powerful data manipulation capabilities. We will ideally create unique classes and/or functions to manage the vocabulary we create, to perform character pair replacements, and overall data processing and compression.

Function I/O

The algorithm operates through several key functions to transform input .TXT files in various languages into compressed text containing English characters and special symbols. One can think of this string of characters and symbols as a "key" that represents a compressed text file. Firstly, the algorithm detects the language of the input text, ensuring it can distinguish between texts in different languages. The translated text then undergoes a compression process where the algorithm efficiently reduces the text size. During this compression, a combination of English characters and special symbols is utilized to optimize the output. To ensure the correctness of these functions, testing strategies are employed.

We plan to use 3 general functions to do this:

The Language Detection Function in can utilize hash tables or maps to store language-specific features. By analyzing the input text and comparing it with predefined language patterns, this function can efficiently determine the language. C++'s "std::unordered_map" can be used to store language patterns, enabling fast lookup and accurate detection. This function serves as a foundational step, allowing the algorithm to discern the language of each input story and proceed accordingly.

The Translation Function can utilize libraries, where the responses are processed using C++ strings and data structures. The translated text can be stored in "std::string" variables, ensuring compatibility with C++ data structures. By employing HTTP requests and handling responses with C++ string manipulation functions, this function translates non-English texts into a key with English characters and special symbols. The translated key is then stored in appropriately for further processing.

The Compression Function in C++ can work by leveraging C++ data structures such as vectors or arrays.. C++ vectors or arrays can store the compressed data, ensuring space efficiency while retaining the text's meaning. By implementing compression algorithms with C++ data structures, this function generates compressed English texts with special characters, ready for storage or further manipulation within the C++ environment.

Language detection is tested by providing inputs in various languages and validating the accuracy of the detected language. Translation tests involve inputting texts in different languages and verifying the precision of the translation into the key of characters and symbols. Compression tests include scenarios where texts are compressed and then decompressed, ensuring that the output matches the input and that the compression-decompression processes are lossless.

Integration tests are conducted to evaluate the operation of the functions in the specified sequence. These tests involve inputting non-English texts, validating their accurate translation into the key, and then confirming the correctness of the compressed output. Additionally, edge cases and stress tests are employed to assess the algorithm's performance under extreme conditions. This includes testing very short or exceptionally long texts and evaluating the algorithm's handling of special characters and symbols. Through these comprehensive testing methods, the algorithm's correctness and efficiency in processing texts from various languages into the compressed key are rigorously validated.

Data Description

We are using a popular children's story in different languages as our dataset called "The Monkey's Paw." We are trying different languages to see if our algorithm can perform its desired tokenization on various potential inputs. We have this split into newlines to make parsing and preprocessing easier.