Next Word Prediction System

Raghda Al taei

2024

1 Introduction

This document explains the R code used for processing and analyzing text data from the SwiftKey dataset. The code involves loading, cleaning, and extracting n-grams (unigrams, bigrams, trigrams, fourgrams, fivegrams, and sixgrams) from the dataset, and N- Gram Next word prediction function.

2 Code Explanation

2.1 Loading the Data

The first step involves creating a folder for the datasets and downloading the SwiftKey dataset if it has not already been downloaded. The data is read from three different text files: blogs, news, and Twitter data.

2.2 Sampling the Data

The cleaned data is then split into training and testing datasets using an 80-20 split. This is done to prepare for model training and testing.

2.3 Ensemble Training Data

The training data is then split into eight datasets and stored in a list. This approach helped manage memory usage, as the system was crashing during the n-grams extraction stage. While the system ran smoothly for unigrams and bigrams, it started crashing as the size of the data for higher-order n-grams increased. Splitting the data into smaller subsets provided an efficient solution to this problem.

2.4 Cleaning the Data

Once the data is loaded and split, it is combined into a single list that has 8 text strings. The data is then cleaned by:

- Converting all text to lowercase.

- Removing punctuation, numbers, and extra whitespace.
- Removing URLs, email addresses, hashtags, and mentions.
- Retaining only alphabetic characters.

For this dataset, I noticed that removing the stop words decreased the accuracy. so I eliminated this step and kept them. Additionally, some words were misspelled, for Example (I looooooooove yooooooooooou) these words are not frequent and when counting their grams they will get a value of one. To save space on your computer you can eliminate them from your N-gram tables.

2.5 Extracting N-grams

The cleaned training data is now taken and used to extract the N-Grames from (1 to 6) each gram is a list that contains 8 sub-datasets :

- Unigrams, bigrams, trigrams, fourgrams, fivegrams, and sixgrams are extracted using the `unnest_tokens` function from the `tidytext` package.
- Each n-gram is counted and stored in separate lists.

2.6 N-Gram Prediction Function

The input text is Reprocessed: it is converted to lowercase, punctuation and numbers are removed, and extra whitespace is stripped. The cleaned text is then split into individual words (tokens).

N-gram Search: The function attempts to predict the next word by progressively checking n-gram models:

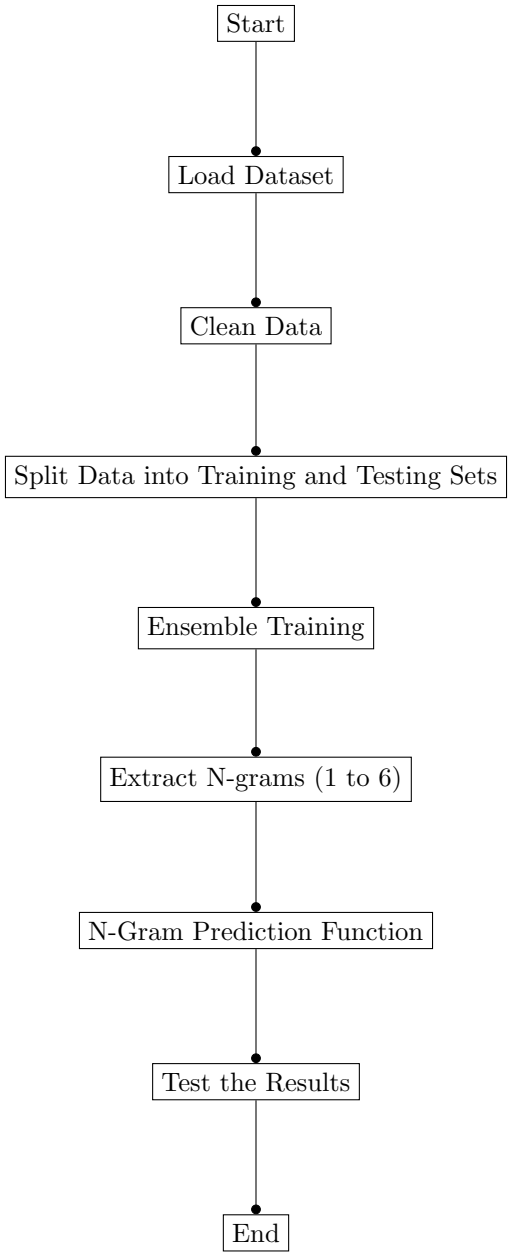
Sixgrams: If the input text has at least five words, it uses the last five words to search for predictions in the 6-gram list. Fivegrams: If no predictions were found in the sixgram and the input has at least four words, it searches in the 5-gram list using the last four words. Quadgrams: If no fivegram predictions are found and the input has at least three words, it checks the 4-gram list. Trigrams: If quadgrams fail, it checks the 3-gram list if the input has at least two words. Bigrams: Finally, if no trigram predictions are available and the input has at least one word, it uses the bigram list (last word) for predictions.

Prediction Output: Once predictions are found, the function groups the results by word and ranks them by frequency. If no predictions are available, it returns a message: "No prediction available."

In each case, it retrieves the top 10 predictions (this can be changed by modifying the slice(1:10) part) from each n-gram model based on word frequency.

3 Workflow Overview

The overall workflow of the code is summarized in the following diagram:



4 Conclusion

This document outlines the key steps in the R code for processing text data from the SwiftKey dataset. The resulting n-grams can be used for various natural language processing tasks. An app was created to represent the results and here is the link NLP App. Additionally, the presentation can be accessed here: Presentation.