PFP Project Proposal: Autocomplete

Michael Jan (mj2886)
Maÿlis Whetsel (mw3391)

Fall 2021

1 Task

Given an unfinished word in context, predict the remainder of the word. The best prediction will depend on the given characters of the current word as well as the previous words. For example, given the input "I love functional prog", our program's output might be "ramming".

Because autocomplete is typically useful only in real time, we would like to speed up the algorithm with parallelization.

2 Steps

2.1 Building up a knowledge base

Construct "n-gram to frequency" maps for $1 \leq n \leq k$ for some $k \geq 1$ out of some corpus (e.g. all Wikipedia articles). Note that for $n = 1$, this would just be a word-frequency map.

We would first clean the data set then extract the n-grams frequencies from the corpus.

A parallel implementation would break the corpus into k sections where k is the number of cores, concurrently create frequency maps for each section, and then combine all maps to generate the final knowledge base. This would run once (possibly writing the output to a file) and then be used for lookup in the interactive portion of the algorithm.

2.2 Analyzing an input

Given an input we need to compute a few things

1. Generate a set S of all valid completions (words) given the current prefix
2. Gather all n-grams $(w_1, w_2, \ldots, w_n)$ such that the first $n-1$ words match the given context, and $w_n \in S$.
3. Use our knowledge base to find the most likely (frequent) n-gram generated by the previous step. Return the word w_n that is part of that most likely n-gram.

This is best explained with an example.

Given the input "the quick f":

1. Generate all valid completions for f from dictionary: ["fox", "ferry", "fire"]
2. For each word, get all n-grams ($1 \leq n \leq$ number of words in input string) using the given context, and

compute a score:

$$\begin{aligned}\text{score}(fox) &= \text{freq}(fox) + \text{freq}(quick, fox) + \text{freq}(the, quick, fox)\\ \text{score}(ferry) &= \text{freq}(ferry) + \text{freq}(quick, ferry) + \text{freq}(the, quick, ferry)\\ \text{score}(fire) &= \text{freq}(fire) + \text{freq}(quick, fire) + \text{freq}(the, quick, fire)\end{aligned}$$

3. Return highest scoring word "fox".

Step 1 can be parallelized if the dictionary (word list) is stored in a trie. First, traverse the trie down according to the given prefix. At this point, every leaf in the subtree corresponds to a valid completion. To find the leaves, we'll have to traverse the entire subtree. At each branching point, we have the option of splitting into parallel jobs.

Step 2 can be parallelized by computing the score of each word concurrently.