

EC504 Final Project

Group 7: LanguageCorrection

Manuel Segimón, Moi Bensadón, Leon Long, Tejas Singh



Background

Objective: Develop an application to evaluate text accuracy and suggest corrections, utilizing an optimal data structure for efficiency

Three Main Components:

Crawler: Fetches data from input (web, files etc.)

Checker: Analyzes texts and identifies unusual language features

Corrector: Suggests grammatical corrections

Crawler

- Crawls an arbitrary number of English-language web pages.
- Stores compressed meta-data about word usage, limited to 1Kb per page.
- Command-line interface to start crawl from a user-designated list of URLs.

Checker

- Processes texts to detect grammatically unusual structures.
- Outputs results in JSON format, detailing the level of suspicion for each phrase and sentence.
- Command-line interface to analyze specified files.

Corrector

- Analyzes a sentence and suggests more grammatically correct alternatives.
- Command-line interface to correct files.

Features

- Real-Time Crawler Status & Statistics
 - Displays current page, links found, processing rate, and metadata size.
- Suspicious Text Correction
 - Provides ranked corrections based on difference from original text.
- GUI Text Highlighting
 - Highlights suspicious textual elements in the GUI.

Features

- Social Media Crawling (Reddit)
 - Extends crawler to fetch and process social media posts.
- Multi-Language Support
 - System functions in multiple languages in which no team member had fluency.
 - Languages: German, Portuguese, and Italian.
- Graphical User Feedback for Phrase Corrections
 - Users can expand the knowledge base by providing urls or subreddits which we will crawl and add to the trie.
 - After providing feedback, the user can select corrections from interactive pop-up and improve future suggestions.

Features Suggestions

Implemented Suggested Features:

- Full Language Name Display (Sergio Emanuel Rodriguez Rivera)
 - Displays complete language names in dropdown menus for clarity.
- Correct Badly Spelt Words (Sergio Emanuel Rodriguez Rivera)
 - Improves spelling suggestions based on Levenshtein Distance.
- Smart Suggestion Management (Sergio Emanuel Rodriguez Rivera)
 - Corrector and checker withhold suggestions when the sentence is correct.
- Executable Bash Script (Phuong Khanh Tran)
 - Simplifies starting the application with a bash script for the GUI and CLI components.

Additional Features:

- Progress Bar for Crawling
 - Visual representation of crawling progress in the GUI.
- Comprehensive Language Selection in GUI
 - Allows easy switching between languages directly from the GUI.

Design

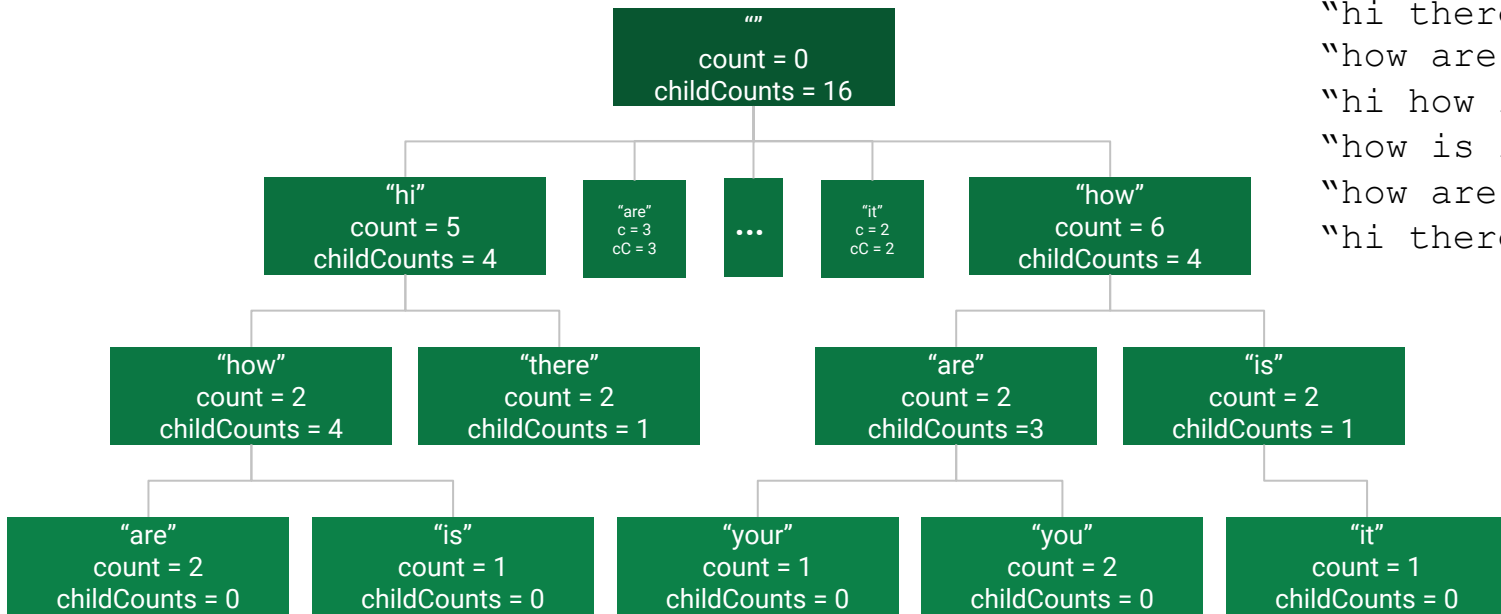
Our n-gram Trie will store subphrases of sentences in order to understand language syntax, grammatical order and the underlying probability distribution of words.

Our crawler scans the text on a given page, splits it into sentences and adds n-grams of each phrase to the Trie. We extended the web crawler to be a social media crawler for Reddit that takes in a subreddit name and scans it.

Our checker uses the Trie to analyze the the perplexity of a sentence and all its substrings. The GUI for the checker displays and highlights the worst substring.

The corrector suggests changes to an input sentence, by reordering the words and calculating the new perplexities. It also suggests corrections for misspelled words.

Design - Trie Node



Strings:

"Hi!"

"How?"

"hi how are you doing"

"hi there"

"how are your dogs?"

"hi how is your dad?"

"how is it going"

"how are you doing"

"hi there is it raining?"

Analysis - Crawler

- Data Structures:
 - URL queue: Constant time/memory to enqueue/dequeue
- Algorithms:
 - Text Retrieval:
 - Read all HTML info. into a Document - linear time wrt. # lines of HTML
 - Compression:
 - Used zlib - linear time
 - Link retrieval:
 - Iterate through previously stored HTML and look for tags - linear time
 - Copy string and add to URL queue - constant time

Analysis - TrieNode

We provide a popup box on startup that allows users to choose a language corpus to base their initial Trie on. If they choose to do so, the Trie ends up being very wide, in fact as wide as the number of words, and only, at most, 3 levels deep.

The children of each TrieNode are stored with a HashMap allowing for quick insertion and lookup when we apply probabilistic functions.

Probability and Perplexity:

- $P(W) = P(w_1, w_2, \dots, w_n)$; Ex: $P(\text{"a red fox"}) = P(\text{"a"}) * P(\text{"red"} | \text{"a"}) * P(\text{"fox"} | \text{"a red"})$
- $PP(W) = (1/P(W))^{(1/n)} = 2^{((\sum -\log(P(w_i | W_{i-1}))/n)} = \dots = 2^{(H(W))}$

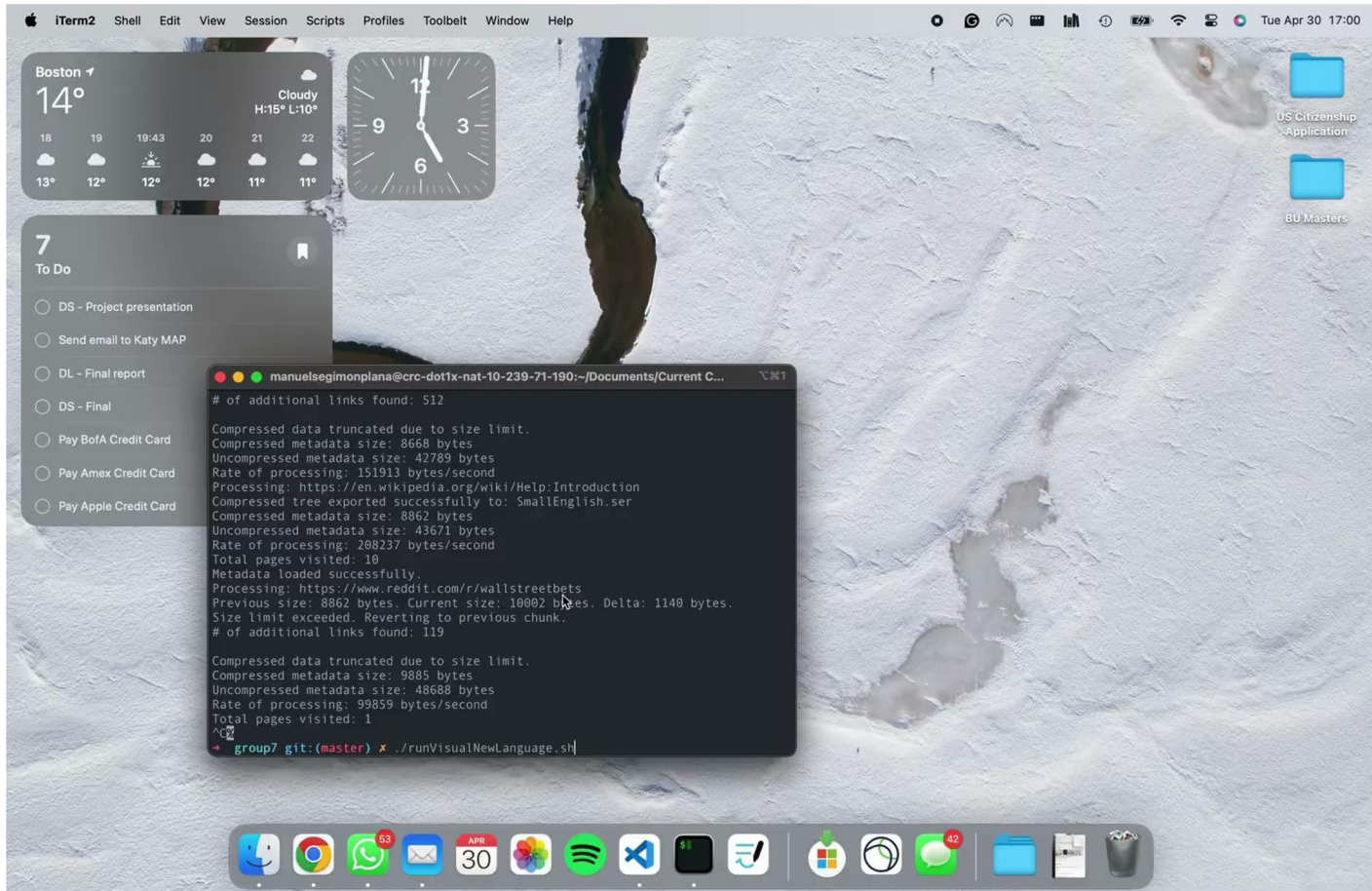
Analysis - Checker + Corrector

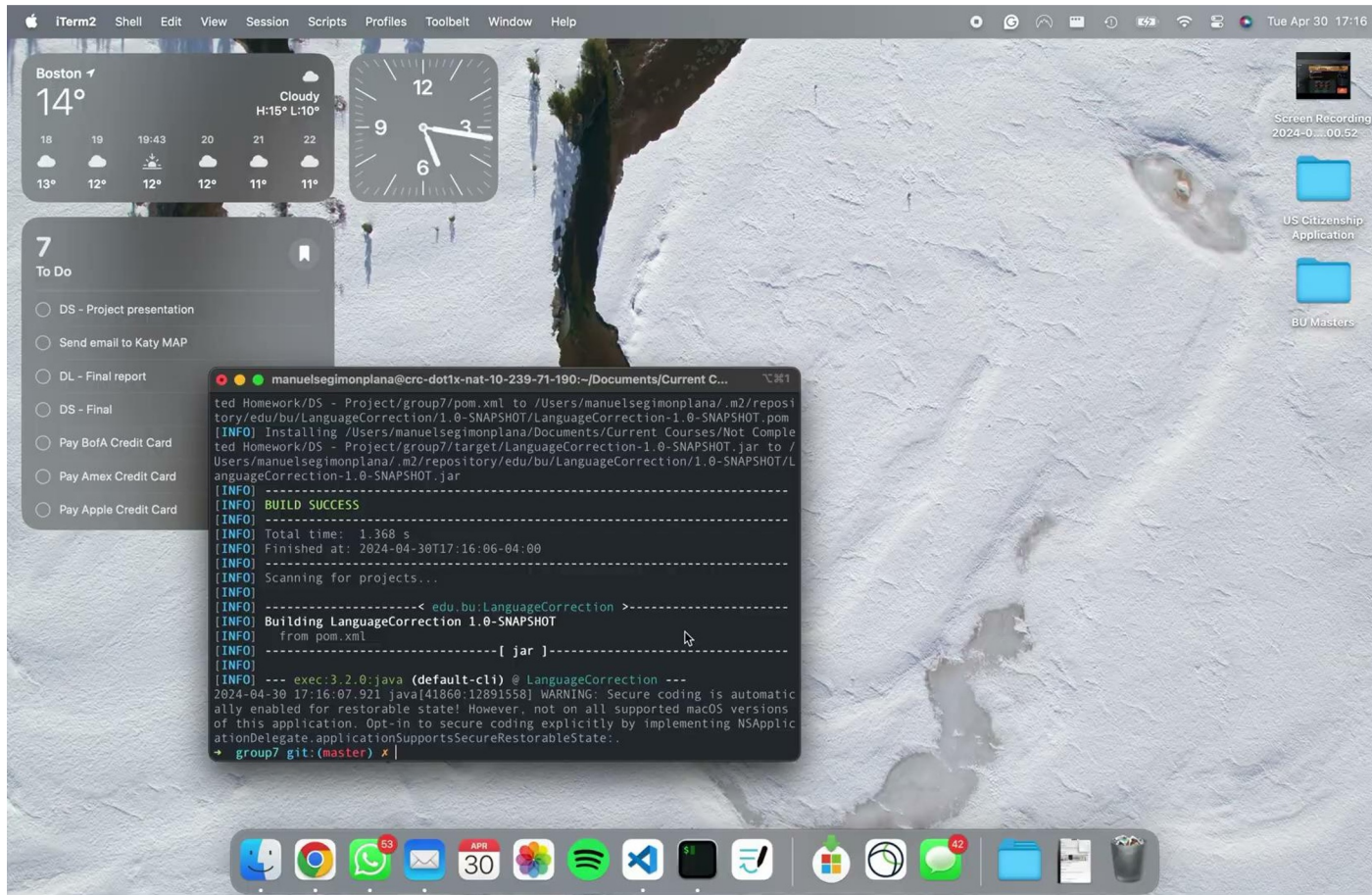
The Checker simply makes a call to Perplexity and evaluates how “perplexing” each phrase is. If it has the greatest perplexity of the sentence, it’ll highlight it.

The Corrector works in a somewhat brute force manner. We generate all possible combinations of the words in the sentence and evaluate the perplexity of each. We then list the top 5 results by lowest perplexity and suggest them to the user.

If the user chooses one of the corrections the corrected version will be added to the Trie, incrementing necessary count values, thereby decreasing its perplexity.

Demo Time





Link to video: https://drive.google.com/file/d/1_r1v0Th2U39hFSmrdlzAxTrNAZRdpA8Z/view?usp=drive_link

Future Improvements

If we had more time we would love to:

- Seek a more efficient way to implement checker
- Remove constraint of 1KB per crawl page - institute max number of sentences instead
- Start with cached Tries instead of cached corporii that require building on first startup of application OR filter corporii to improve performance.

Questions?