HG2051 Project 1

Lim Zhe Xun (U2121481J)

Introduction

This project aims to develop a program to perform multi-level search on Pnar interlinear text. Pnar is a Mon-Khmer language spoken in northeast India, consisting of 7 vowels and 21 consonants (Hiram, 2012).

Objectives

The objectives of this project are as follows:

1. Read, preprocess, and store interlinear text
2. Enable support for full dataset from original repository
3. Execute search using three levels of annotation (text, gloss, POS)
4. Implement forward/backward search for POS
5. Implement a simple user interface (UI) for users to perform lookups
6. Write lookup results to output file

Repository Structure

- data directory: contains the raw Pnar text files
- utils.py: contains essential functions for data processing and interlinear search
- app.py: code to run interactive application on web browser
- project1.py: code to run interlinear search via terminal
- requirements.txt: dependencies required to run the codes

Data

This section describes how the data is read, pre-processed and stored for further use.

Preprocessing

Upon studying the format of the data file, I observe that they are split into individual clips, with each clip containing various features. Each feature is a line of text, starting with their label followed by the feature data. I first define global variables corresponding to the different features to be extracted (e.g. “tx” for original text, “ps” for part of speech). Defining these variables at the start allows me to easily reference them whenever needed. To split the text into individual clips, I decided to use Python's split function to separate text based on the "\ref" string.

Pseudo Code:

```
1. Read text file as string
2. Initialize empty list L
3. Separate data into individual clips by splitting string on
   "\ref".
4. For each clip:
     a. Initialize empty dictionary D
     b. Extract relevant lines from each clip using their
        corresponding start-of-line markers (e.g. \tx for
        original text, \ps for part of speech). Skip this clip if
        the contents are labelled as "EMPTY".
     c. Clean and assign the extracted lines to dictionary D
     d. Append dictionary D to list L
```

Data Structure

The final data is stored as a list of dictionaries, with each dictionary representing one clip in the dataset. Text 1 and text 2 contain 16 and 9 non-empty clips respectively. The original dataset (Hiram, 2017) consists of 2619 non-empty clips. By using a list data structure to contain the individual clip data, this allows me to read multiple text files and concatenate them together, allowing me to search across multiple text files simultaneously.

Process

This section describes the key functionalities used for the project.

Text Lookup

The text lookup function takes a string query provided by the user and identifies clips which native text contains one or more occurrence of the query, utilizing Python's inbuilt string-matching function. This functionality was relatively easy to implement as it simply iterates through all clips, and checks if their corresponding "tx" strings contains the user query. By default, this function checks the "tx" data of each clip, but has a parameter to allow users to flexibly check other features instead.

Gloss Lookup

The gloss lookup function takes a string query provided by the user and identifies clips which gloss contains one or more occurrence of the query. Unlike text lookup, this function utilizes a different search algorithm as it also tracks the indexes of matching queries.

POS Lookup

The Part of Speech (POS) lookup is an extension of Gloss lookup, taking in a gloss string query, as well as POS terminology from the user. Upon identifying a matching gloss, this function also checks if the gloss's POS matches the POS terminology provided by the user. In addition, the function also supports forward and backwards search (i.e. "All words with POS 'n' preceded by gloss 'dog').

Given the indexes of matching gloss obtained from the gloss lookup function, I utilized index slicing to obtain the parts of speech for the matching gloss, as well as the previous and next gloss. In the event of the edge case where the matching gloss is either the first or last word in the text, the respective backward and forward searches will be skipped.

Overall, this function was the most difficult to implement, as it involves searching over two levels of annotation simultaneously.

Combining Lookup Results

To support searching of two or more levels of annotation, results from multiple searches need to be merged. This was done by implementing a function to find and return the intersection of multiple lists. For example, my project implementation first performs text lookup and gloss lookup independently, and then checks for clips that occur in both lookup results.

Saving Results

After each search is performed, the results will be saved to "output.txt" in a format similar to the original text files. While the function has a parameter to name the result file, it does not check if an existing result file exists, and simply overwrites it. Hence, users have to manually relabel/rename result files if they wish to save results from multiple searches.

User Interface

This project offers two ways to interact with the code listed below:

app.py

This Python file utilizes Streamlit, a lightweight framework for deploying web-based user interface. Here, users will be able to perform multi-level search, and the web page will dynamically display the resulting matches.

```
streamlit run app.py
```

project1.py

As an alternative fallback, this Python file performs the same functionalities, albeit with a less visual user interface. To conduct multi-level search, users will need to directly modify the search conditions in the file, and run the following command in the terminal:

```
python project1.py
```

Results

Overall, the functions implemented work as intended, meeting all the project's objectives. The following are some areas that could be improved:

1. Additional support for interlinear search: the project currently supports text, gloss and pos levels of annotation, but other annotations such as pronunciation can be added.
2. Dynamic reading and writing files: the project is currently hardcoded to read and write to specific files. Additional user interface designs can be implemented to allow users to dynamically change the files to be read and saved.

References

Ring, Hiram. 2012. A phonetic description and phonemic analysis of Jowai-Pnar. Mon-Khmer Studies Journal, Volume 40, 133–175.

Ring, Hiram, 2017, "Replication Data for: A grammar of Pnar", https://doi.org/10.21979/N9/KVFGBZ, DR-NTU (Data), V1