Final Project Documentation

Section 1: An overview of the code functionality

The project is a proof of concept for a children-friendly vocabulary learning app to help children memorize words easily by returning relevant example sentences ranking by difficulty after a user keys in a word. We believe it's very beneficial for parents to utilize this app to help educate their children based on feedback from our team member who is a parent.

The App allows users to enter a word, and return its definitions, and relevant example sentences that are age appropriate and children friendly. The example sentences are extracted from the children's story book collection, and are ranked based on difficulty as we would like to return the easiest for better learning experiences.

Our code consists of Jupyter notebooks and python code:

- Data exploration and showcase notebook (under folder "notebooks"):

  Notebook 1.1 to 4.1

- Data cleaning and index code:

  Index.py, cbfile.py, config.toml

- Difficulty ranking object:

  Difficulty_ranker.py, pos_difficulty_ranker.py, sentence_ranker.py, sentence.py

- Main application object:

  Word_app.py, word_definition.py, word_difficulty_ranker.py, word_frequency.py, word_result.py

- Demo (under "notebooks" folder):

  5.1 User.ipynb

Section 2: Implementation Details

Source Data

There are three data components play key roles in the project:

- Top 5,000 english vocabulary (wordFrequency.xlsx)
- English word dictionary (word_def.json)
- Children's story book collection (cbtest_NE_train_cleaned.csv)

The top 5,000 English vocabulary data source contains a list of popular words ranked based on their frequency. The most frequent words such as "the" and "of" appear at the top of the list.

The English word definition dataset is a word and definition pair. It was downloaded from the API provided by *dictionaryapi.dev*. Since the data provider has set a limit on the number of requests for a free account, we had only pulled the definition for the top 5,000 frequent english vocabulary mentioned above.

The children's story book collection includes over one hundred bedtime stories for children. One of our goals is to clean and extract sentences from the collection, and use those sentences as examples for children to learn and remember English words in a fun way. Extensive data exploratory analysis has been performed based on the story book collection data, please refer to Jupyter notebooks 1.1 to 2.3 for the analysis.

Building Index

Since we need to match the English word to sentence, we have built an inverted index for the story book collection dataset by using metaPy introduced in the class. Instead of matching a multi-word query with the contents like what we've learnt in the class, it only needs to match a single word. This makes the job much easier and most MetaPy built-in ranking function was able to return the expected results.

Sentence Ranking - Word difficulty & Part-of-speech tag difficulty

In order to rank the sentences based on difficulty, we came up with two scoring functions, word difficulty and Part-of-speech tag (POS) difficulty.

In word difficulty, a score is given to each individual English word in a sentence. The score is determined by the frequency rank in the wordFrequency.xlsx dataset. Let's use "My child is learning English" as an example sentence. Higher score means it is less frequent. To get the overall word difficulty score of a sentence, we calculate the mean score and divide it by the average word difficulty score of the entire corpus to scale it down.

Sentence	my	child	is	learning	english
Word Difficulty Score	36	140	2	1435	1896

In POS difficulty, as in the example below, a POS tag is given to each word in a sentence. Each POS tag has a corresponding difficulty score defined in the "pos_difficulty.json" dataset. Instead of taking the average value, we sum up the POS tag difficulty score to penalize longer sentences. In the end, the sum is divided by the average POS difficulty score of the entire corpus to scale it down.

Sentence	my	child	is	learning	english
POS Tag	PRP$	NN	VBZ	VBG	NN
POS Difficulty Score	1	1	1	1	1

In the end, we combined word difficulty and POS difficulty by taking the arithmetic mean value to calculate the overall difficulty of the resulting sentences.

Sampling and Evaluation

To ensure the ranked sentences accuracy, we compare the App ranked sentences with human ranked sentences by using nGDC score as an evaluation metric. We have reviewed the words to sentence distribution analyzed in Jupyter notebooks 1.2 and decided to sample from the words with at least 10 and no more than 1000 sentences in our source data.

To evaluate the sentences ranking accuracy, the top 5 example sentences ranked by the App were being used to compare with the top 5 sentences ranked by human. The nGDC relevance level scales from 1 to 3, the top sentence ranked by human has relevance level 3, while the relevance level for the fifth sentence is 1. Manual rank has the ideal nGDC score, and the calculation is shown below:

Ideal nGDC:

$$3 + \frac{3}{log(3)} + \frac{2}{log(4)} + \frac{2}{log(5)} + \frac{1}{log(6)}$$

The nGDC of the App ranked sentences should always be less or equal to the ideal value. We use the ratio of nGDC over ideal nGDC as a metric to evaluate the sentence ranking algorithm. We have evaluated example sentences for 5 different words, the evaluation results are listed in the table below. Please refer to Python notebook "*4.1 Evaluation"* for details.

Word	**nGDC over ideal nGDC ratio**
Value	100%
Spirit	83.9%
lying	76.9%
Sandy	85.3%
Stake	66.7%

The score of the App ranked sentences are close to the ideal value, and the score of one selected word even reached ideal value, indicating the difficulty ranking algorithm aligns with human intuition.

Software Demo - Usage and instruction

We've created a notebook (5.1 User.ipynb) to demo how to query the words and show the word definition and relevant sentences as results.

word_app.py: Word app entry class.

- index.py: Load indexed data and search matched sentences by word.
- sentence_ranker.py: Rank sentence by linear interpolation of different ranking results.
    - word_difficulty_ranker.py: Rank sentence by word difficulties according to word ranking data.
    - pos_difficulty_ranker.py: Rank sentence by a customized POS difficulty table.
- word_definition.py: It contains word definitions.

Our demo is built with the Jupyter notebook and the data and index is locally stored, so the user just needs to download our code package which includes the cleansed data, built index, web scraped word definition, Jupyter notebooks, python scripts as well as the environment yml file. The user will need to create the conda environment as specified in the environment yml file then will be able to run the jupyter notebooks and demo.

To test the App, please use the predefined “search_and_display_word(word: string, n_results: int)” where the word is a single english word in string format, and you can specify the number of example sentences to return in the “n_results” parameter.

Since there is limited free data source available online, the App is only focusing on the top 5,000 frequent words listed in the “data/wordFrequency.xlsx”, please use only the word from that list for testing purposes.

Tutorial presentation link: https://www.youtube.com/watch?v=Hgc5yuiNiM8

Section 3: Team member contribution

Cheng Wei leads the code development by creating the environment and framework for the team and also helps with consolidating the code and Jupyter notebook. Xiaoyu Zhu provides technical support by researching different sentence difficulty methods, building web scraping for word definition and recording the demo. Jiaxi Wu helps with cleansing and compiling the data for building indexes and the documentation for the project.