TLN Part 3 Lab 06 - Ghigliottina

Paletto Andrea, Tuninetti André

September 12, 2023

Contents

1 Introduction

The goal of this exercise is to simulate the game of "Ghigliottina," in which, given 5 words, the player has to find a sixth word that is somehow semantically related to the other 5. To find the word we implemented two different strategies, one that involves similarity measures and another one that instead uses the frequency of synsets.

2 Code Overview

2.1 Project Structure

- The ./res folder contains the data.csv file. In this file the first column contains 78 polysemic english words while the second column contains a set of 10 sentences for each word. This sentences will later on be used to disambiguate the 5 words used in the game. This dataset was generated with the help of OpenAi's ChatGPT.
- The "ghigliottina.ipynb" is the Jupyter Notebook where the game is implemented

2.2 Libraries Used

- `NLTK`: Natural Language Toolkit for NLP tasks.
- `csv`: Module for reading CSV files.
- `pandas`: Library for data manipulation.

2.3 Functions

`load_csv_to_dict(csv_filename)` Loads a CSV file containing word-sentence pairs into a Python dictionary.

`read_first_column(csv_file)` Reads the first column of a CSV file and returns a list of words in lowercase.

`disambiguate_senses_for_word(word, word_sentences_dict)` Disambiguates word senses in sentences using the Lesk algorithm and returns a set of disambiguated senses.

`get_related_synsets(synset)` Returns hypernyms, hyponyms and meronyms of a given synset.

`retrieve_related_synsets(word_list, word_sentences_dict)` Retrieves related synsets for a list of words using disambiguated senses.

`find_closest_synset(synsets)` Finds the closest synset based on similarity measures and returns its name and average similarity.

`find_most_frequent_synset(synsets, five_words)` Finds the most frequent synset among a list of synsets, excluding specified words.

3 Usage

3.1 Loading Data

We use the `load_csv_to_dict()` function in order to load the data and store it in a dictionary where the keys are the words contained in the first column and the values are the lists of sentences contained in the second column. We then select 5 random words that will be used to play the game from the first column

3.2 Find related synsets

The idea is that in order for a word to be semantically linked to five other words there must be some overlap in the synset space. To find that we implemented two different strategies, one that involves similarity measures and another one that instead uses the frequency of synsets. In both cases we have to find all the related synsets first. To do that we use the Lesk algorithm on each one of the 5 words using the sentences associated to each word as disambiguation context. After that we extend the related synsets list including every hyponym, hypernym and meronyms.

3.3 Finding Best synset using Similarity Measures

To find the closest synset among a list of synsets, use the `find_closest_synset()` function. In this approach we iterate through the list of related synset and measure the Wu-Palmer similarity, the Leacock-Chodorow similarity and the Shortest Path similarity of every synset. We then compute the average of the three measures and return the synset with the highest average similarity score. This solution is far from optimal because it takes a relatively long time to compute all the scores.

3.4 Finding Best Synset using Similarity

To find the most frequent synset among the list of related synsets, we use the `find_most_frequent_synset()` function. This approach is much more straight forword. It's based on the assumption that the synset with the strongest link to the 5 given words will have the highest number of occurrences in the related synsets list. Therefore all we need to do is to find the synset with the highest frequency in the list.

4 Examples

In this section, there are the results of a few runs.

4.1 Run 1

1. ***part***
2. ***spring***
3. ***light***
4. ***bark***
5. ***park***

Similarity-based approach: ***jump***
Frequency-based approach: ***end***

4.2 Run 2

1. ***order***
2. ***key***
3. ***rock***
4. ***scale***
5. ***bark***

Similarity-based approach: ***arrange***
Frequency-based approach: ***arrange***

4.3 Run 3

1. ***file***
2. ***stick***
3. ***part***
4. ***nail***
5. ***box***

Similarity-based approach: ***touch***
Frequency-based approach: ***end***

4.4 Run 4

1. ***bowl***
2. ***mark***
3. ***spring***
4. ***range***
5. ***row***

Similarity-based approach: ***jump***
Frequency-based approach: ***bounce***

4.5 Run 5

1. ***part***
2. ***time***
3. ***play***
4. ***order***
5. ***tank***

Similarity-based approach: *break*
Frequency-based approach: *chukker*

5 Conclusion

Both strategies compute the results in a reasonable amount of time but the frequency based approach is much faster. In some instances they both converge to the same solution (Run 2), while in other scenarios (Run 4) the words found are synonyms. Most of the times though the words obtained are unrelated to eachother.