Hangman Challenge Strategy

Author: Teng Changqing

I basically refer to the implementation of

> https://github.com/rakshit176/Trexquant-Hangman-Challenge-
>
> by rakshit176

and using the strategy of n**-sliding window** with details as follows.

STEP 1: Statistical analysis

The letter frequency of the training set is calculated and the 7 most frequently occurred letter is `freq= ["e", "i", "a", "n", "o", "r", "s"]`. The distribution of the ratio of these letters to the length of word is displayed. From a statistics point of view, if the number of letters in `freq` are greater than 0.7 times of its length, then guessing a letter in `freq` will not be a good choice.

STEP 2: Useful functions

1. Build `n_word_dictionary`: creates a dictionary `n_word_dictionary` where `key` represents the length of the word (from 2 to 29, where 29 is the maximum length of word in the training set), the corresponding `value` are lists of substrings of the specific length from the word in the training set.
2. `counter_freq(new_dictionary)` : a function to find the number of times a letter shows up in whole `new_dictionary`, return the corresponding `Counter`.
3. `substring(n_word_dictionary, clean_word)`: a function to generate a list of words from `n_word_dictionary`, where each word is of same length as `clean_word`

STEP 3: Guess

For the `guess(word)` function, there are two cases

Case I: **No correct guesses has been made so far.** The guess will start with the most frequently occurred letters (which has not been guessed) in the dictionary `full_dictionary_common_letter_sorted`.

Case II: At least one correct guess has been made. The guess will start from the substrings in `n_word_dictionary` whose length is same as `len(clean_word)`. The most frequently occurred letter in the list of substrings will be picked. If there is no appropriate letter, the guess will be made based on the substrings in `n_word_dictionary` whose length equals `len(clean_word) - 1`. So and so forth, until the length of the substrings reaches minimum `2`. If no guess has been determined, guess will start with the most frequently occurred letters (which has not been guessed) in the dictionary `full_dictionary_common_letter_sorted`.

Key modifications compared with "rakshit176"

1. Use the `freq` list rather than `vowels = ["a", "e", "i", "o", "u"].`
2. Extend the `n_word_dictionary` whose keys are `3-29` to `2-29`.
3. Initial guess is based on the most frequently occurred letters rather than iterating through all of the words in the old plausible dictionary.
4. Use n-sliding window to compute the probabilities of most frequently occurred letters with `n = len(clean_word), len(clean_word) -1, ..., 2` rather than using `len(clean_word), len(clean_word)/2, len(clean_word)/3`.

Summary

The algorithm displayed here is pure statistic-based and very simple but could achieve relatively accurate predictions.