GLOBAL ALPHA RESEARCHER HANGMAN CHALLENGE

PROJECT DOCUMENTATION

HARSH VERMA, IIT KANPUR

Phone: +91-9669988459

hverma21@iitk.ac.in

2007harshverma@gmail.com

Abstract

This documentation details the strategy behind the guess function for a Hangman game, developed for TrexQuant's Global Alpha Challenge. The function aims to maximize correct guesses using statistical analysis of letter frequencies and patterns from a large dictionary. The report covers initialization, guess strategy, preliminary analysis, rejected strategies, and code snippets. This first-principles approach achieves a 64.6% success rate in guessing Hangman words from an unseen lexicon within 8 attempts.

Objective

The primary objective of this project is to develop an intelligent guessing strategy for the Hangman game that maximizes the chances of correctly identifying the hidden word within a limited number of attempts. The strategy leverages frequency analysis of letter sequences, prefixes, and suffixes from a comprehensive dictionary to make informed guesses.

Preliminary Analysis

Before settling on the final strategy, several analyses and considerations were made:

1. **Dictionary Analysis**:
    - **Word Length Distribution**: Analyzed the distribution of word lengths in the dictionary to determine common lengths and inform the decision on pattern lengths (n).
    - **Letter Frequency**: Analyzed the frequency of individual letters across the entire dictionary to identify common letters.
    - **Pattern Frequency**: Computed the frequency of n-letter sequences, prefixes, and suffixes to understand their occurrence and potential informativeness.
2. **Optimal Pattern Length**:
    - Preliminary tests revealed that considering patterns up to a length of 7 provided a good balance between computational efficiency and accuracy. Longer patterns yielded only marginal improvements.
3. **Vowel-Consonant Ratio**:
    - Analyzed the impact of vowel-heavy words and adjusted the strategy to penalize vowel guesses when the ratio of vowels to consonants was high, improving overall performance by avoiding common traps.

Rejected Strategies

Several alternative strategies were considered and ultimately rejected in favor of the current approach. These strategies include:

1. Random Letter Guessing

- **Description**: Guessing letters randomly without any statistical basis.
- **Reason for Rejection**: This approach led to a high number of incorrect guesses and did not leverage any available information, resulting in poor performance.

2. Sequential Letter Guessing

- **Description**: Guessing letters in a fixed sequence (e.g., alphabetically).
- **Reason for Rejection**: Similar to random guessing, this approach did not take into account the specific word pattern and thus had low accuracy.

3. Fixed Frequency-Based Guessing

- **Description**: Always guessing the most frequent letter in the dictionary regardless of the current word state.
- **Reason for Rejection**: While better than random guessing, this approach did not adapt to the information revealed in each round and often led to redundant or incorrect guesses.

4. Single Pattern Matching

- **Description**: Using only single-letter frequency without considering n-letter sequences, prefixes, or suffixes.
- **Reason for Rejection**: This approach lacked the depth of pattern analysis and was less effective in identifying correct letters in more complex word structures.

5. Machine Learning-Based Approach

- **Description**: Implementing a machine learning model to predict the next letter based on the masked word and previous guesses. This could involve training a neural network or other supervised learning algorithm on a labeled dataset of Hangman games.
- **Reason for Rejection**: While potentially powerful, this approach requires a large labeled dataset and significant computational resources for training. Additionally, the model's performance is heavily dependent on the quality and size of the training data, which may not be readily available.

6. Bayesian Inference

- **Description**: Using Bayesian inference to update the probability of each letter being the correct guess based on prior guesses and the current word state. This would involve maintaining a probability distribution over the possible words and updating it as new guesses are made.
- **Reason for Rejection**: Although this method could provide a theoretically sound approach, it is computationally intensive and complex to implement. The need for continuous probability updates and handling large word lists makes it less practical for real-time guessing in Hangman.

7. Markov Chain Model

- **Description**: Using a Markov Chain to model the transitions between letters in words. This would involve constructing a state transition matrix based on letter sequences and using it to predict the next letter.
- **Reason for Rejection**: The complexity of constructing and utilizing a Markov Chain for all possible word transitions is high. Additionally, this method requires significant memory and computational power, making it less efficient than the current approach.

Strategy Points

1. Initial Guesses

When more than 70% of the word is unknown, the strategy guesses the most frequent letter in the dictionary. This is based on the idea that, initially, we have little information about the word. By guessing the most common letter, we maximize our chances of a correct guess, which provides more information for subsequent guesses. This approach reduces computational time because it avoids complex analysis in the early stages when it's less likely to be informative.

```
if blanks_pct >= 0.70 and new_dictionary:
    freq_table_init =
collections.Counter("".join(self.current_dictionary)).most_common()
    for letter, freq in freq_table_init:
        if letter not in self.guessed_letters:
            return letter
```

2. Pattern Length Limitation

Based on preliminary analysis, the strategy limits the maximum length of patterns considered to 7. This decision is a balance between accuracy and computational efficiency. Patterns longer than 7 letters were found to provide only marginally better results but significantly increased computational complexity. By focusing on patterns up to 7 letters, the strategy remains efficient while still leveraging meaningful patterns.

```
self.max_sequence_length = 7
for i in range(self.max_sequence_length):
    temp1, temp2, temp3 = create_freq(self.full_dictionary, i+1)
    self.nseq_freq_list.append(temp1)
    self.prefix_freq_list.append(temp2)
    self.suffix_freq_list.append(temp3)
```

3. Dictionary Recalibration

The strategy continuously filters out words from the current dictionary that do not match the current mask. This refinement process ensures that the set of possible words is as accurate as possible, improving the chances of making correct guesses in subsequent rounds.

```
current_dictionary = self.current_dictionary
new_dictionary = []

for dict_word in current_dictionary:
    if len(dict_word) != len_word:
        continue
    if re.match(clean_word, dict_word):
        add = True
        for char in guessed_but_not_found_letters:
            if char in dict_word:
                add = False
                break
        if add:
            new_dictionary.append(dict_word)
self.current_dictionary = new_dictionary
```

4. Weighted Patterns

The strategy assigns specific weights to n-letter sequences, prefixes, and suffixes. Longer patterns are prioritized as they provide more information. By weighting these patterns, the strategy can make more informed guesses based on the structure and common patterns within the dictionary.

```
n_seq_weights = [1, 10, 100, 1000, 10000, 100000, 1000000]
prefix_weights = [1, 10, 100, 1000, 10000, 100000, 1000000]
suffix_weights = [1, 10, 100, 1000, 10000, 100000, 1000000]
```

5. Vowel-consonant ratio Adjustment

If the vowel-consonant ratio in the masked word exceeds 1.5, the strategy penalizes vowel guesses by reducing their weight by a factor of 10. This adjustment helps avoid over-guessing vowels in words that are likely to be consonant-heavy, improving the efficiency and accuracy of the guessing process.

```
if vowel_consonant_ratio > 1.5:
    for vowel in vowels:
        if vowel not in self.guessed_letters:
            freq_table_lat[vowel] *= 0.1
```

6. Frequency-Based Guessing

The strategy always guesses the letter with the highest weighted frequency that has not been guessed yet. This approach ensures that each guess is as informed as possible, leveraging the accumulated frequency data and weights to make the most probable guess.

```
for letter, _ in sorted(freq_table_lat.items(), key=lambda x: (x[1], x[0]), 
reverse=True):
    if letter not in self.guessed_letters:
        return letter
```

These strategy points together form a comprehensive approach to efficiently and accurately guess letters in the Hangman game, leveraging statistical analysis and pattern recognition to improve the chances of winning.

Conclusion

This strategy leverages statistical analysis and pattern recognition to make informed guesses in the Hangman game. By dynamically adjusting to the information revealed in each round, the guess function aims to maximize the likelihood of correctly identifying the hidden word in the fewest possible guesses. This approach combines efficiency with accuracy, making it a robust solution for the Hangman game.