Trexquant Interview Project (The Hangman Game)

Gaurish Bansal

2024-11-26

Introduction

Hangman, a popular word-guessing game, is a compelling challenge due to its linguistic and computational intricacies. As part of the TrexQuant internship selection process, I was tasked with developing an algorithm that outperforms a basic model in playing Hangman. The target was an accuracy rate of over **50%**.

This report outlines the development of my Hangman-playing algorithm, which utilizes linguistic patterns, conditional probabilities, and statistical techniques to improve guessing accuracy.

Intuition

Hangman involves guessing a hidden word represented by blanks (`_`). The game ends when the word is correctly guessed or **6 incorrect guesses** are made.

Key Observations:

1. **Frequent Letter Patterns**:
   - Common patterns like **'Q' followed by 'U'** or word endings such as **'TION', 'MENT', and 'NESS'** improve predictive accuracy.
   - Assumption: Letters in a word are related to their neighbors.
2. **Word Length**:
   - Short words (e.g., 5–6 letters) are harder to guess due to fewer identifiable patterns.
3. **Dummy Variables**:
   - Dummy characters `{` (start) and `|` (end) were added to dictionary words to capture positional patterns, e.g., endings like **'ING'** or **'TION'**.
4. **N-grams**:
   - N-grams are contiguous sequences of `n` letters. For instance, the sentence "The cat sat" has 2-grams like **'The cat'** and **'cat sat'**.
   - Patterns up to 5-grams were considered for this algorithm to balance accuracy and computational complexity.

Methodology

Step 1: Initialization

- Start with the hidden word (e.g., **APPLE**) represented as blanks: `_ _ _ _ _`.
- Add dummy variables: `{ _ _ _ _ _ |`.

Step 2: Guessing Letters

- Use previously guessed letters (e.g., P, E) to refine the word pattern: `{ _ P P _ E |`.

Step 3: Pattern Breakdown

- Break down the current word pattern into 1-grams, 2-grams, 3-grams, 4-grams, and 5-grams components. For example:

Table 1: Pattern components of the string ("{ _ P P _ E |") with one space.

<table>
<tr><th>Length (α_1)</th><th>Length (α_2)</th><th>Length (α_3)</th><th>Length (α_4)</th><th>Length (α_5)</th></tr>
<tr><td>_</td><td>{_</td><td>{_P</td><td>{_PP</td><td>PP_E|</td></tr>
<tr><td>_</td><td>_P</td><td>_PP</td><td>PP_E</td><td></td></tr>
<tr><td></td><td>P_</td><td>PP_</td><td>P_E|</td><td></td></tr>
<tr><td></td><td>_E</td><td>P_E</td><td></td><td></td></tr>
<tr><td></td><td></td><td>_E|</td><td></td><td></td></tr>
</table>

Table 2: Pattern components of the string ("{ _ P P _ E |") with two spaces.

Length (β_1)	Length (β_2)
_ P P _ (β_{11})	{ _ P P _ (β_{21})
	_ P P _ E (β_{22})

Step 4: Calculate Weights and Scores

1. For each n-gram, compute **conditional probabilities** of unguessed letters.
2. Assign **weights** to each letter based on its likelihood in the pattern.
3. The **score** for a letter K is calculated as:

$$\text{Score}(K) = \sum_{i=1}^{5} C_i \sum_{j=1}^{n_i} P(\gamma_{ij} = K \mid \alpha_{ij}, K \notin \text{guessed}) +$$

$$\sum_{i=1}^{5} C_i \sum_{j=1}^{m_i} \Big[P(\gamma_{ij1} = K \mid \beta_{ij}, K \notin \text{guessed}) +$$

$$P(\gamma_{ij2} = K \mid \beta_{ij}, K \notin \text{guessed}) -$$

$$P(\gamma_{ij1} = K \wedge \gamma_{ij2} = K \mid \beta_{ij}, K \notin \text{guessed}) \Big]$$

Where:

- C_i: Weight assigned to patterns of length i.
- n_i: Number of patterns of length i with one blank.
- m_i: Number of patterns of length i with two blanks.
- α_{ij}: The j-th pattern of length i with one blank.
- β_{ij}: The j-th pattern of length i with two blanks.
- γ_{ij}: The letter filling the blank in α_{ij}.

- $\gamma_{ij1}, \gamma_{ij2}$: Letters filling the first and second blanks in β_{ij}.

Step 5: Conditional Probabilities

Single Blank:

$$P(\gamma_{ij} = K \mid \gamma_{ij} \in P\_E) = \frac{N(PKE) \cdot \mathbb{I}(K \notin \text{guessed})}{\sum_{\delta \in \{A,B,\ldots,Z\}} N(P\delta E) \cdot \mathbb{I}(\delta \notin \text{guessed})}$$

First Blank in Two-Blank Pattern:

$$P(\gamma_{ij1} = K \mid \gamma_{ij} \in \_PP\_) = \frac{N(KPP\delta)}{\sum_{\delta_1,\delta_2} N(\delta_1 PP\delta_2)}$$

Second Blank in Two-Blank Pattern:

$$P(\gamma_{ij2} = K \mid \gamma_{ij} \in \_PP\_) = \frac{N(\delta PPK)}{\sum_{\delta_1,\delta_2} N(\delta_1 PP\delta_2)}$$

Joint Probability for Two Blanks:

$$P(\gamma_{ij1} = K \text{ and } \gamma_{ij2} = K \mid \gamma_{ij} \in \_PP\_) = \frac{N(KPPK)}{\sum_{\delta_1,\delta_2} N(\delta_1 PP\delta_2)}$$

Step 6: Letter Selection

- Choose the letter with the **highest score**.

Step 7: Update Pattern

- Incorporate the guessed letter and repeat steps 3–6 until the word is guessed or 6 incorrect guesses are made.

Functions

Function	Description
`guess(word)`	Combines probabilities to determine the most likely letter.
`find_score(word, ngram)`	Calculates the score for each letter based on n-grams.
`Conditional_Prob1(pat)`	Computes probabilities for patterns with 1 blank.
`Conditional_Prob2(pat)`	Computes probabilities for patterns with 2 blanks.
`normalize1(pat)`	Converts raw frequencies into probabilities for 1-blank patterns.
`normalize2(pat)`	Converts raw frequencies into probabilities for 2-blank patterns.
`build_freq_tables()`	Precomputes pattern frequencies in the dictionary for constant-time lookups.

Results

- The algorithm achieved an **accuracy of 66.5%**, surpassing the target of 50%.
- This demonstrates the effectiveness of combining linguistic insights, statistical techniques, and computational strategies.

Conclusion

This project illustrates how statistical modeling and linguistic patterns can significantly enhance a Hangman-playing algorithm. By leveraging n-gram components and conditional probabilities, the model outperformed a basic implementation, achieving a success rate well above the set benchmark.