# Hangman Game Strategy using Statistical Analysis

Mayank Bharati, Indian Institute of Information Technology, Allahabad

2024-01-7

## Contents

# Introduction

The game of Hangman, a classic word-guessing game, has been a subject of interest in various fields due to its inherent linguistic and computational challenges. As part of the selection process for a summer internship at TrexQuant, the task was presented to create an algorithm that significantly outperforms a base model in playing Hangman. The objective of this challenge was to design an algorithm to achieve an accuracy rate exceeding 50%.

In this report, I present a comprehensive analysis of the development process, methodology, and results achieved in designing my Hangman-playing algorithm. My approach focused on exploiting patterns in En-glish words, utilizing conditional probabilities, and employing statistical techniques to create a sophisticated model capable of achieving a higher success rate in guessing the correct letters. By combining these strate-gies and rigorously evaluating the algorithm's performance through cross-validation and confidence interval estimation, I was able to demonstrate the effectiveness of our approach.

# Intuition

Hangman is a word-guessing game that challenges players to deduce a hidden word by guessing individual letters. The game typically involves two participants: one player who thinks of a word and another player who tries to guess it. The word to be guessed is represented by a series of dashes, each dash representing a letter in the word. The guessing player's task is to sequentially suggest letters they believe are in the word.

The game continues until the guessing player either successfully guesses the entire word or the guessing player has made 6 incorrect guesses, indicating a loss.

The guessing player can use strategies based on the frequency of letters in the English language, common letter combinations, and other linguistic patterns to narrow down potential letters and maximize the chances of guessing the word correctly within a limited number of attempts.

**Key Assumption:** Letters appearing in a word are correlated with their adjacent letter.

This assumption was made by observing various patterns in English words. For example the letter 'Q' is almost always followed by the letter 'U' and many words end with the letters 'TION'. All the analysis is done using the words in the dictionary provided in the challenge.

**Note 1 :** Our chances of guessing the word correctly also depend significantly on the number of unique letters in the word ( which depends on word length ). If the length of the word is very small, say 4 or 5, we are very unlikely to guess it correctly because the number of correct letters out of 26 are small and there are very few patterns to observe. But fortunately the such words are few in number (See Figure-2).

**Note 2:** Some letter combinations occur at the end of a word like "NESS" or "ER". In order to exploit this fact I added dummy letters "{" and "|" at the beginning and end of each word in the dictionary. This allows us to identify such patterns and adjust our prediction accordingly.

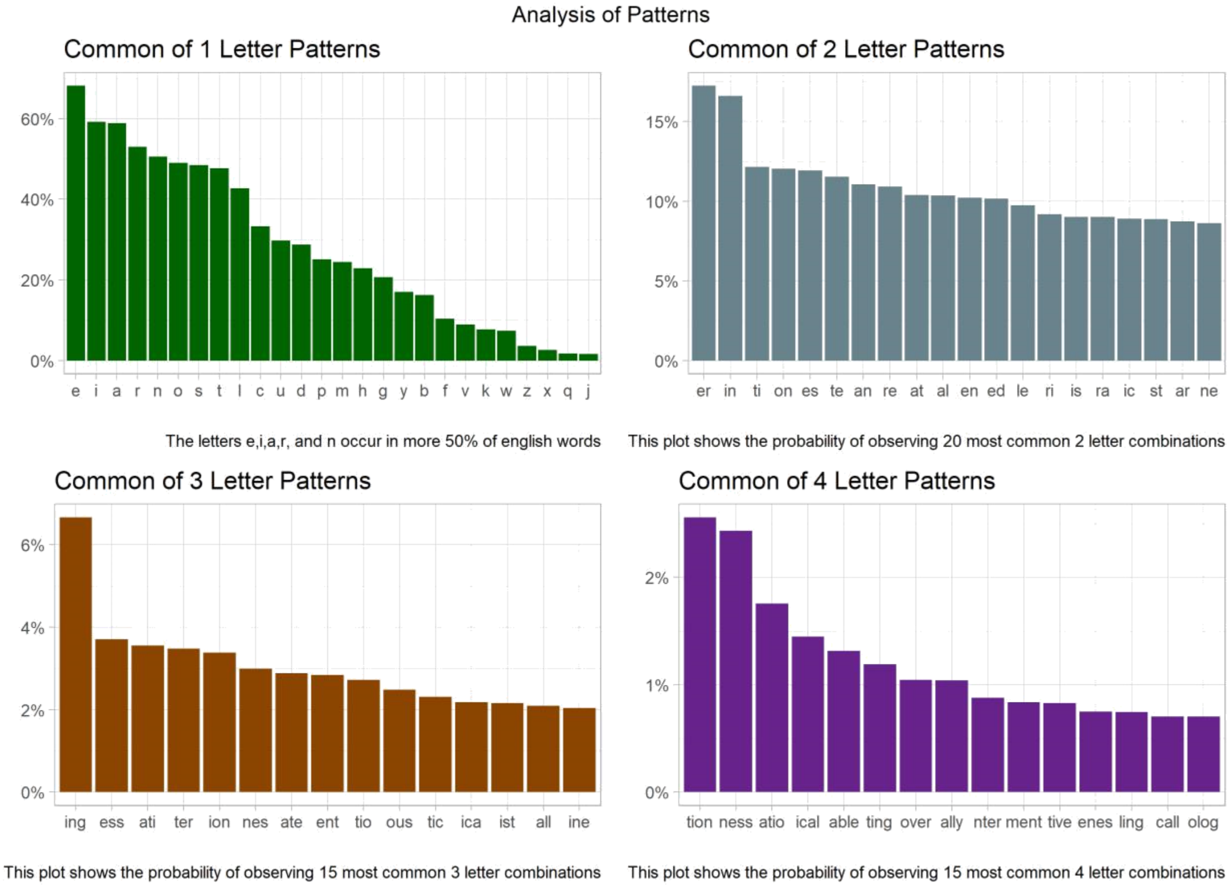
Keeping these points in mind I designed an algorithm which predicts the most likely letter in the word.

# Methodology

The idea is to break the word into components and for each component find the probability of observing each letter and then use this probability as score and return the letter with maximum score. The algorithm is given below for the specific case of guessing the word FINISH

Suppose we have already guessed the letters I, E, A, and S and thus the input sequence is **_ I _ I S _** .

First, we add the dummy letters { and | to the input to get **{ _ I _ I S _ |** . This allows us to identify that the last blank space is the last letter in the word.



The letters e,i,a,r, and n occur in more 50% of english words

This plot shows the probability of observing 20 most common 2 letter combinations

This plot shows the probability of observing 15 most common 3 letter combinations

This plot shows the probability of observing 15 most common 4 letter combinations

Figure 1: Probability that a random English word contains a given pattern

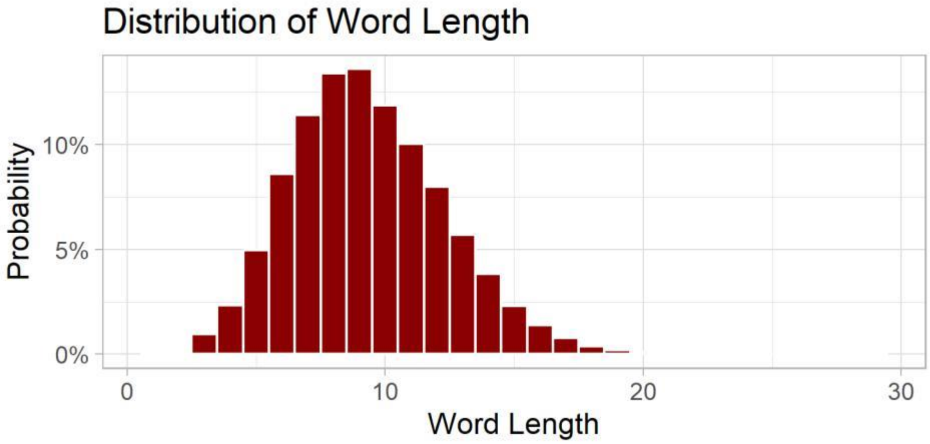


Figure 2: Plot showing the distribution of the length of English word.

Second, we break the input into patterns components containing exactly one blank space to get the following patterns.

Table 1: Pattern components of the example string

| Length 1 ( $\Pi_1$ ) | Length 2 ( $\Pi_2$ ) | Length 3 ( $\Pi_3$ ) | Length 4 ( $\Pi_4$ ) |
|---|---|---|---|
| _($\Pi_{11}$) | {_($\Pi_{21}$) | {_I($\Pi_{31}$) | I_IS($\Pi_{41}$) |
| _($\Pi_{12}$) | I_($\Pi_{22}$) | I_I($\Pi_{32}$) | IS_\|($\Pi_{42}$) |
| _($\Pi_{13}$) | _I($\Pi_{23}$) | _IS($\Pi_{33}$) | |
| | S_($\Pi_{24}$) | IS_($\Pi_{34}$) | |
| | _\|($\Pi_{25}$) | S_\|($\Pi_{35}$) | |

Now we want to calculate a score for each letter in the alphabet. If the score for a given letter is high, it signifies that the letter is more likely to appear in the word. Let $\Pi_i = \{\Pi_{i1}, \Pi_{i1}, . . . \Pi_{in_i} , \}$ denote the set of patterns of length $i$ in the above table. Lets denote the letter in the blank space of the pattern $\Pi_{ij}$ by $\gamma_{ij}$. The score for the letter T is calculated in the following manner

$$score(T) = \sum_{i=1}^{5} \beta_i \sum_{j=1}^{n_i} P\ robability\ (\gamma_{ij} = T \mid \gamma_{ij} \in \Pi_{ij}, \gamma_{ij} \notin \{A, E, I, S\})$$

Here denotes the weights assigned to the probabilities of each according to the length of the pattern i.e. $\beta_i \in \{1, 2, 3, 4, 5\}$ as I am considering patterns of length up to 5.

$\beta_i$ is a parameter which controls the influence of each pattern on the score. It depends only on the length of the pattern. The reason why we are not just adding the raw probabilities is that the probability from a longer patterns is more important than the probability obtained from a smaller pattern. The values of $\beta_i$ were found by trial and error.

Table 2: Values of factors for different length patterns

| $\beta_1$ | $\beta_2$ | $\beta_3$ | $\beta_4$ | $\beta_5$ |
|---|---|---|---|---|
| 1 | 1 | 4 | 10 | 20 |

The conditional probability used in the above formula is calculated using the following formula. Focusing on a particular pattern, say **I _ I.**

$$P(\gamma = T \mid \gamma \in I\_I, \gamma \notin \{A,E,I,S\}) = \frac{N(ITI) \cdot \mathrm{I}(T \notin \{A, E, I, S\})}{\sum_{\delta \in \{A,B,\dots Z\}} N(I\delta I) \cdot \mathrm{I}(\delta \notin \{A, E, I, S\})}$$

Here I($x$) is the indicator function which return 1 if $x$ is a true statement otherwise it returns 0.

$N(\pi)$ is the number of occurences of the pattern $\pi$ in our word dictionary. Figure 3 shows the the conditional probability distributions for some patterns.

The score is calculated for each letter in the alphabet and our prediction is the letter with the highest score.

The implementation of this strategy requires the following functions.

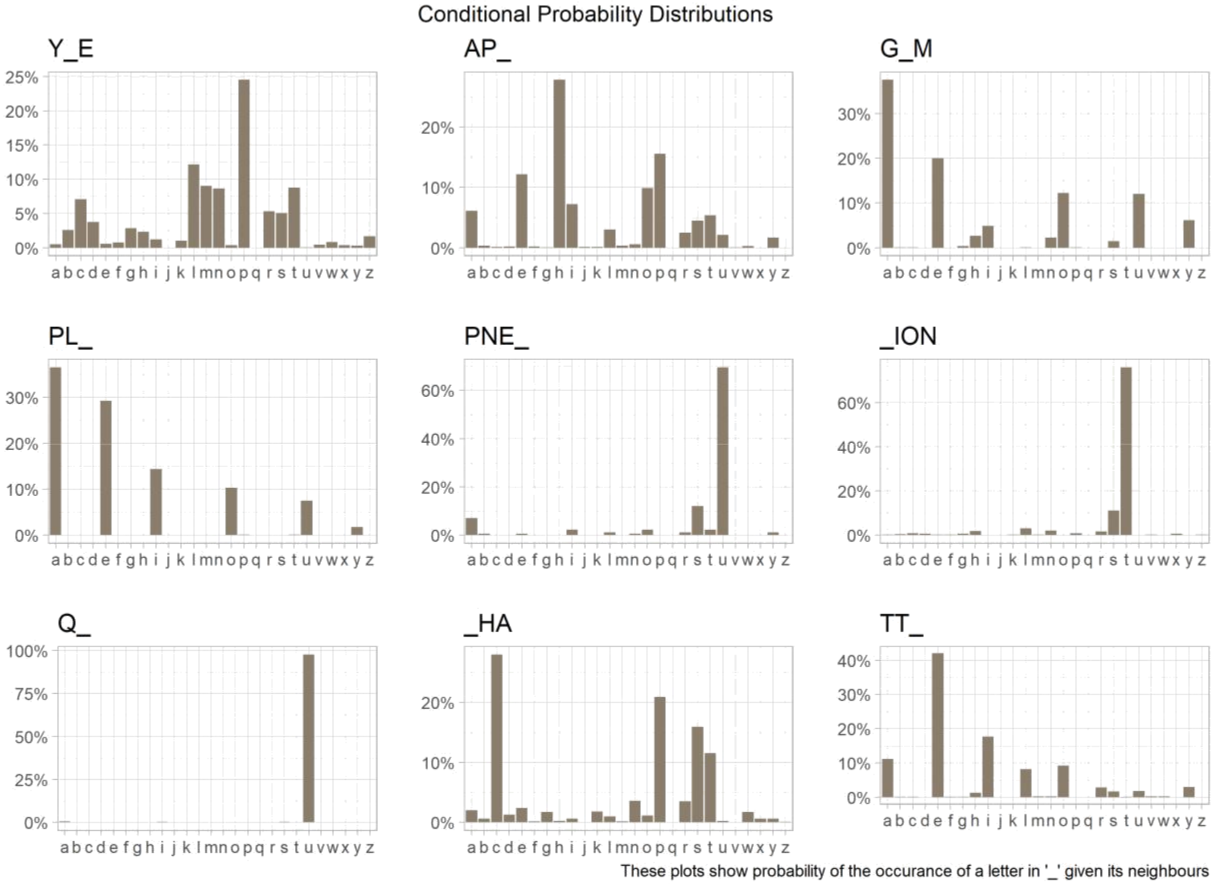


Figure 3: Examples of Conditional Probability distributions for some patterns

Table 3: Functions used in the implementaion

| Function | Description |
|---|---|
| guess(word) | This is the main function which combines the probabilities and returns the most likely letter |
| model(word, l) | Breaks the word into patterns of length l and returns the sum of the probabilities for each letter in the alphabet |
| Conditional_Probability(p) | Thisfunction takes a pattern p as input containing exactly one blank space and calculates the probability mass function of the missing letter given other letters in the pattern p |
| normalise_prob_array(x) | This function takes as input raw frequencies of each letter and returns the corresponding probability |
| build_freq_tables() | This function is run only once and it calculates the frequencies of every pattern in our dictionary which allows us to calculate $N(x)$ in constant time |

## Cross Validation

Before submitting the final result testing the algorithm's accuracy was crucial. Let $X$ denote the accuracy of the algorithm on a set of 1000 games. In order to calculate the distribution of $X$ we play 1000 sets of 1000 games to generate samples from the distribution of $X$ as $\{X_1, X_2 \dots X_{1000}\}$.

Firstly, 1000 words are removed from the dictionary at random for testing and the algorithm is allowed to train on the rest of the words. This process is repeated 1000 times to get the samples. Secondly, on each sample we calculate the accuracy which is denoted by $X_i$ . We can use these samples to calculate the point estimates for $X$ as well as upper one-sided confidence interval. Figure 4 shows the distribution of $X$

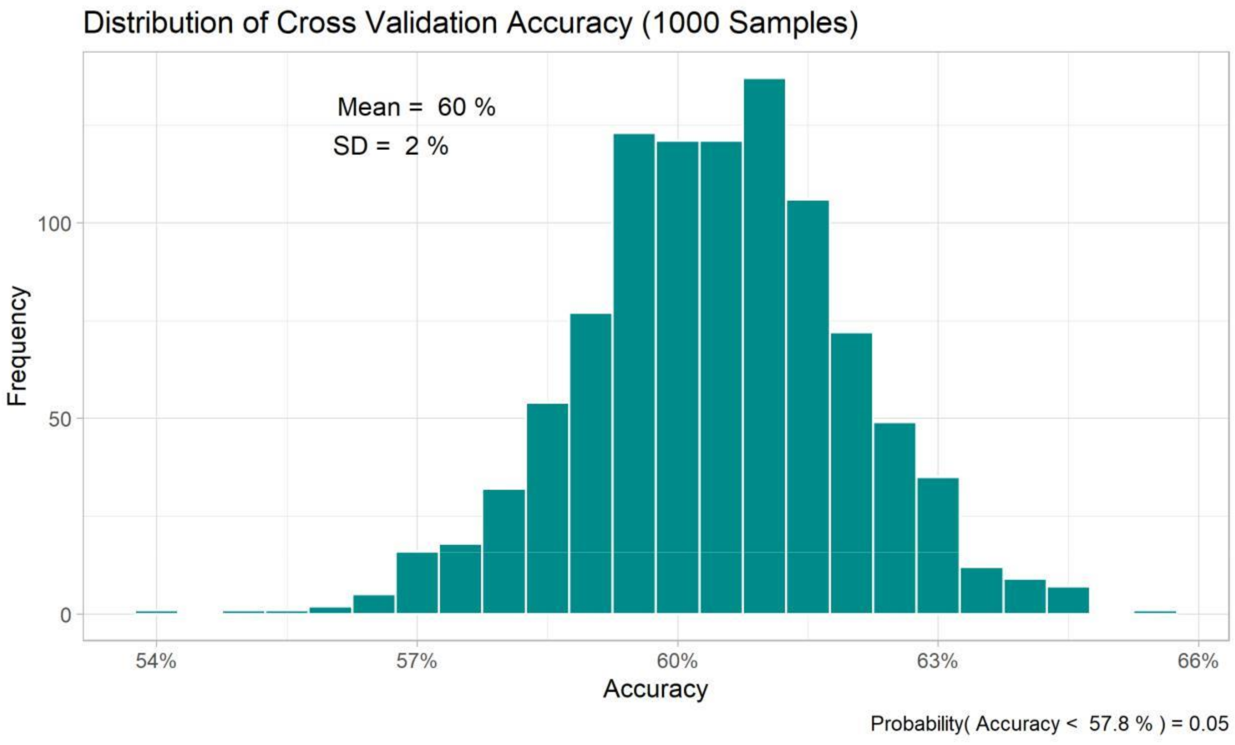


Figure 4: Distribution of accuracy on a set of 1000 games

Lastly we use these samples to find where does our algorithm fails. Our initial hypothesis was that it will fail for smaller length word. We find the probability of successfully guessing the word of a given length using the above sample games. We can observe from Figure 5 that if a word contains 5 letters we have only $\frac{1}{4}$ chances of guessing it correctly where as we can almost always guess the words which have a length larger than 15.

$$P\,robability(\,Success \mid W\,ord\ Length = l\,) = \frac{\text{number of instances we succeeded on a word of length}\ l}{\text{number of words of length}\ l\ \text{in our sample}}$$

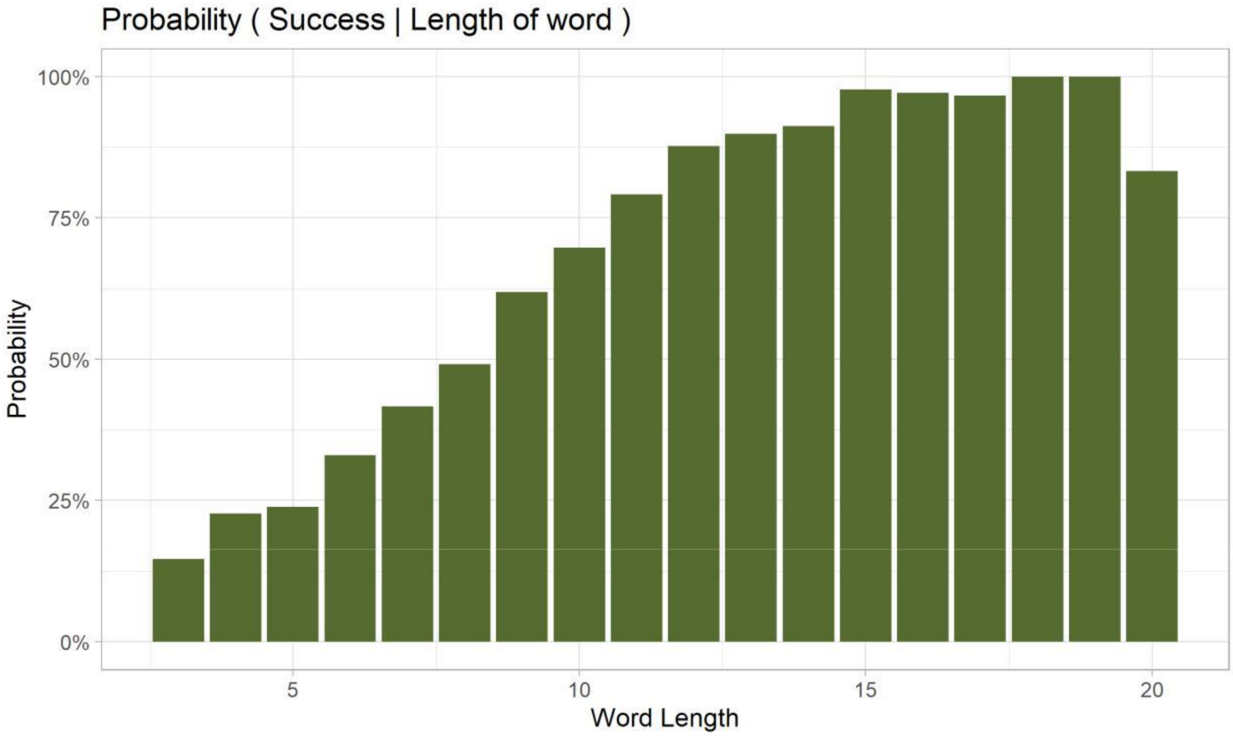


Figure 5: Probabilities of successfully guessing the the word of a given length

## Conclusion

In conclusion, this statistical algorithm for the Hangman game is a significant enhancement over the base model. Leveraging conditional probabilities, linguistic patterns, and statistical analysis, I achieved an accu-racy rate of 60%, surpassing the 50% target. Rigorous cross-validation and confidence interval estimation further validate the algorithm’s effectiveness.