Information Theory-based Wordle Game Word Difficulty Classification and Dynamic Planning Optimization Research

Xin Gui *, Chen Su, Keyu Pan

School of Civil Engineering, Shandong Jianzhu University, Jinan, China, 250101

* Corresponding Author Email: xgui73830@163.com

Abstract. Wordle has gained popularity as a word-guessing game. This study aims to describe the difficulty attributes of target words in Wordle and categorize them based on their difficulty. After two rounds of word guessing, this paper calculated the average mutual information for the third-round correct answers. This data was utilized to build a K-means classification model, categorizing words into three distinct groups: easy, normal, and hard. For instance, the word "EERIE" has an average mutual information of 8.997 bits, categorizing it as a word of 'hard' difficulty. Given the vast number of words to process, the computation time was extensive. To address this, this paper employed dynamic programming, leading to a significant reduction in operation time. Additionally, a Monte Carlo simulation model was established to simulate potential player guessing patterns, validating the classification model's robustness. The model developed in this research offers fresh perspectives on strategy selection in Wordle games and the difficulty assessment of target words. It serves as a dependable guide for game developers when classifying word difficulty.

Keywords: Assessment of Vocabulary Difficulty, Source Entropy, Dynamic Programming, Monte Carlo, K-means.

1. Introduction

Wordle, a word-guessing game, was ingeniously designed and launched by Jonah Lupton in 2021. As players begin, the system randomly selects a word as the riddle. Players are tasked to discern this word within a maximum of six attempts. With each guess, the game provides feedback: missing letters are colored gray, misplaced ones in yellow, and correctly placed letters in green. Given its expansive vocabulary and varying difficulty levels, Wordle appeals to a broad spectrum of players, irrespective of their age or proficiency in English. Its allure is evident from its widespread acclaim on social media platforms.

The game's meteoric rise in popularity spurred scholarly interest. A segment of researchers embarked on unveiling optimal word-guessing strategies. Martin B. [1], for instance, devised two word-guessing methodologies rooted in a greedy algorithm. Throughout the gameplay, players whittle down their choices until a sole word remains. Martin B.'s simulations divulged that an average of 3.7696 rounds suffices to ascertain the correct word. Successive research [2] introduced an exact dynamic programming method for word guessing, pinpointing "SALET" as the prime starting word, necessitating an average of 3.421 guesses. Literature [3] dissected word-guessing through an operations research lens, establishing a strategy averaging 3.67 guesses and ensuring game completion within six guesses 98% of the time. Information entropy theory's integration [4] enriched the game's analytical dimensions, inspiring subsequent research. In a similar vein, Nisansa de Silva [5] utilized character statistics to identify optimal starting words, settling on 'raise', 'clout', and 'nymph' due to their common usage. Studies [6][7], leveraging Monte Carlo simulations in tandem with word composition rules and convolutional neural network perspectives, further buttressed this field, setting a precedent for this paper.

Predominant research orbits around Wordle players' opening moves and in-game strategies. Our research pivots to a game developer's perspective. this paper aim to categorize potential target words by difficulty, aiding developers in crafting a nuanced and precise word difficulty module. Such refinements empower players to select challenges tailored to their capabilities, enriching the gaming experience. This study embarks on quantifying the uncertainty in discerning the correct word. After

two guessing rounds, this paper compute the average mutual information of possible outcomes and harness this metric to forge a K-means classification model. The words are segmented into three echelons: easy, normal, and hard. The computational heft of sifting through vast outcomes became an impediment. To surmount this, this paper integrated dynamic programming, markedly slashing computation durations. Moreover, this paper employed a Monte Carlo simulation to mirror actual player behavior, validating our model's congruence with genuine gameplay experiences. Wordle's difficulty nuances significantly influence player engagement. Equipping developers with tools to adeptly categorize word difficulty augments the game's allure and sustains player retention.

2. Materials and preparatory knowledge

2.1. Data Acquisition

Owing to Wordle's open-source nature, the entire set of words that can be employed as riddles is accessible. For this study, a collection of 2309 potential words was extracted from 3Blue1Brown's GitHub repository. This repository contains materials associated with their 2022 video series dedicated to Wordle. The link to the repository is https://github.com/3b1b/videos/tree/master/2022/wordle.

2.2. Source Entropy

The essence of information lies in resolving uncertainty; only messages that reduce uncertainty truly contain information. This study harnesses the subsequent concepts from information theory to quantify the uncertainty inherent in deducing a riddle's answer:

Self-informative: This measures the uncertainty surrounding a particular event or a symbol from a source.

$$I(x_i) = -\log\left(p(x_i)\right) \tag{1}$$

Source entropy: This denotes the average uncertainty of the source.

$$H(X) = E[I(X)] = -\sum_i \ p(x_i)\log\big(p(x_i)\big) \tag{2}$$

Conditional entropy: This expresses the average uncertainty of Y when X is recognized.

$$H(Y \mid X) = -\sum p\big(x_i, y_j\big)\log\Big(p\big(y_j \mid x_i\big)\Big) \tag{3}$$

Mutual information: If Y is known, it diminishes the uncertainty surrounding X. This quantifies the information Y provides about X.

$$l\big(x_i; y_j\big) = \log\left(\frac{p\big(x_i|y_j\big)}{p(x_i)}\right) = \log\left(\frac{p_{\text{post}}}{p_{\text{prior}}}\right) \tag{4}$$

Average Mutual Information: This gauges the mutual information in a generalized sense.

$$I_{avg} = \sum_i \ p\big(x_i, y_j\big)\log\left(\frac{p\big(x_i|y_j\big)}{p(x_i)}\right) \tag{5}$$

$$I_{avg} = H(X) - H(X|Y) \tag{6}$$

3. Model Establishment and Solution

3.1. Definition and Assumption of the Model

To facilitate the subsequent formula and algorithm expression, this paper stipulates the following definitions and symbol representations. All words are numbered as n ($n = 1,2,\ldots,2309$).

(1) Target Word a: The word that players need to submit to win a game. There is only one in each game.

(2) Target Word Space A: All words that can be used as target words. In this paper, it's a collection of 2309 possible target words.

(3) Guess Word g^i: The word players choose to submit for verification in the i-th round of guessing.

(4) Guess Word Space G_i: Before the start of the i-th guessing round, all words could possibly be guessed. At the start of the game, the guess word space is represented by all possible target g^i words as shown in equation (7).

$$G_1 = A \tag{7}$$

(5) Guessing Strategy: In the i-th round of guessing, the method of selecting the guess word g^i from the guessing word space G_i. In this paper, each word's probability of being guessed follows a uniform distribution $U(0,1)$. It is assumed that players haven't read any related strategies and don't guess words based on their familiarity with them.

(6) Clue C^i: After the i-th round of guessing, the system provides hints based on the relationship between the guess word g^i and each letter of the target word a. To simplify the representation of green, yellow, and gray hints, they are represented as 2,1,0 in this paper. For example, when a=671 (error) and g^i=677 (ethos), the clue $C^i$is as shown in equation (8). This clue is based on the previous guessing results of all possible five-letter words. Since each C^i is discrete and unrelated to the previous guessing results, this paper can regard C^i as a five-fold expansion of a discrete memoryless source.

$$C^i_{a,g^i} = 20020 \tag{8}$$

(7) Information Word Space I_i: After the i-th round of guessing, it's a collection of all words that comply with C^i and g^i.

(8) Contraction of the Guess Word Space: After obtaining the information of the guess word g^i and C^i, words that do not match the information are removed from the previous guess word space G_{i-1}. That is, G_{i-1} is the intersection of G_{i-1} and I_{i-1}.

$$G_i = G_{i-1} \cap I_{i-1}(g^i, C^i) \tag{9}$$

(9) Word Space Length S: Represents the mapping from word space to the number of words contained in the word space. For instance, the target word space has 2309 words, represented as equation (10):

$$S(A) = 2309 \tag{10}$$

(10) Termination Rounds r: If the player guesses the word correctly in the i-th round, then the termination rounds $= i$. If the word is not guessed correctly, the termination rounds $r = 6$.

3.2. Development of Evaluation Metrics

Almost all studies on Wordle game strategies [1][2][3][4] simulate an average termination rounds between 3-4. In a single game, the termination rounds is discrete. For instance, using a golf analogy, if four rounds are considered 'par', then three rounds would be a 'birdie'. Keeping the termination rounds within 3 is not only a matter of guess strategy research but also a key issue in classifying the difficulty level of target words. If a target word, after giving clues C^1 and C^2 in the first two rounds, has a significantly reduced uncertainty, indicating a high probability of guessing it right in the third round, then such a target word should be considered of low difficulty. Conversely, if the target word has high uncertainty in the third round, then it should be considered high difficulty.

In our research, this paper introduced a key metric, the average mutual information [8] I_{avg}, to quantify the reduction in uncertainty about the target word a after two rounds of guessing. This

method particularly considers the process of selecting a word g_q^2 from set G_1, and every choice of g_p^1 will produce a corresponding new set G_2, which contains all possible words g_q^2 related to the g_q^2 chosen in the first round.

Specifically, we first considered the total uncertainty about a in the absence of any additional information, i.e., the source entropy $H(a)$. Then, we introduced the conditional entropy $H\left(a \mid (g_p, g_q)\right)$, representing the remaining uncertainty about a after knowing g_p^1 and g_q^2. The expression for average mutual information is:

$$I_{avg} = H(a) - H\left(a \mid (g_p^1, g_q^2)\right) \tag{11}$$

This represents how much the prior uncertainty is reduced compared to the posterior uncertainty after giving conditions g_p and g_q. Further simplified as:

$$I_{avg} = \sum_{g_p^1 \in G_1} \sum_{g_q^2 \in G_2} p(g_p^1, g_q^2) \times \left(I(a) - I(a' \mid g_p^1, g_q^2)\right) \tag{12}$$

where the amount of mutual information I is denoted as:

$$I(a; g_p^1, g_q^2) = I(a) - I(a' \mid g_p^1, g_q^2) \tag{13}$$

At the start of the game, the source entropy of guessing the target correctly, $I(a)$, is expressed as:

$$I(a) = -\log_2 p(a) = -\log_2 \frac{1}{S(A)} = -\log_2 \frac{1}{2309} \tag{14}$$

The probability of selecting g_p^1 and g_q^2 as guesses for the two rounds, $p(g_p^1, g_q^2)$, is:

$$p(g_p^1, g_q^2) = p(g_p^1) p(g_p^1 | g_q^2) = \frac{1}{S(G_1)S(G_2)} \tag{15}$$

Under the condition of two rounds of guesses, the conditional entropy of guessing the target correctly, $I(a' \mid g_p^1, g_q^2)$, is:

$$I(a' \mid g_p^1, g_q^2) = -\log_2 \frac{1}{S(G_3)} \tag{16}$$

3.3. K-means Classification Modeling

This study adopted the K-means clustering algorithm to classify words based on the calculated I_{avg} as described in section 3.2. Specifically, this paper chose K-means++ as the initialization strategy. K-means++ aims to provide a more efficient and accurate initial set of cluster centers for the K-means algorithm, which accelerates the convergence speed and increases the likelihood of obtaining a globally optimum solution. The reason for choosing K-means++ lies in its unique initialization method: first, a data point is randomly selected as the first cluster center, and then when selecting the subsequent centers, each data point is chosen with a probability proportional to the square of its distance to the already selected centers. This method ensures that new cluster centers are as far away as possible from the already selected centers, resulting in a more even distribution of centers in the data space.

Regarding the specific parameter settings, this study employed the following configurations:

(1) Initialization method: K-means++

(2) Number of clusters: 3 (representing three difficulty levels: "easy," "normal," and "hard")

(3) Maximum number of iterations: 300

(4) Number of initial center points: 10

(5) Number of concurrent threads during model fitting: 8

(6) Random generator seed value: 42, to ensure the reproducibility of the experiments.

This configuration ensures the stability and accuracy of the clustering process while considering computational efficiency.

3.4. Model implementation and algorithm optimization

This study used the Python programming language in conjunction with data processing libraries such as Pandas and Numpy to implement the model proposed in section 3.2. The pseudocode for the developed algorithm is outlined below. This process resembles a Markov process [9], where the transition of the guess word space from the state of the current guess to the state of the next round depends only on the current state and the guess word of this round. The transition probability is determined by the guess word and does not depend on any prior states. The essence of the code is to design the transition method of the guess word space state after each round of guessing.

Main steps of the algorithm:

(1) Set all possible words (comprising 2309 five-letter words) as the target word space A. Choose the first word as the target word a.

(2) Set all possible words as the guess word space G_1 and note the length of G_1 as $S(G_1)$.

(3) The player chooses the first word from G_1 as the guess word $g^1_{p(p=1)}$ for the first round; the system checks g^1_1 against a and provides clue C^1.

(4) Find all words that display the same clues when checked against $g^1_{p(p=1)}$, forming the information word space I_1. The next round's guess word space G_2 is the intersection of G_1 and I_1. Note the length of G_2 as $S(G_2)$. One round of guessing ends at this point.

(5) Repeat steps (2), (3), and (4), choosing the first word from G_2 as the guess word $g^2_{q(q=1)}$ for the second round obtaining the length of G_3, $S(G_3)$. One experiment ends at this point. Calculate the mutual information $I(a; g^1_1, g^2_1)$ according to equation (13) at this point.

(6) Traverse all words in G_1, G_2, and calculate the average mutual information I_{avg} according to equation (11). This is the average mutual information when $a = 1$.

(7) Traverse all words in the base word space A, repeat the above 6 steps, and calculate the I_{avg} for each word.

The corresponding pseudo-code is shown in Table.1:

Table 1. Pseudo-code for calculating average mutual information

Algorithm 1: Calculation of Entropy
Input: Word List A
Output: Average Mutual Information I_{avg}
1. FUNCTION shrink_word_space(G, g, a):
2. Calculate the clue between word g and target word a
3. Filter words in G that have the same clue with g
4. RETURN filtered words minus g
5.
6. FOR each word a in A:
7. FOR each word g^1 in $G1$:
8. G_2= shrink_word_space(G_1, g^1, a)
9. FOR each word g^2 in G_2:
10. G_3= shrink_word_space(G_2, g^2, a)
11. I_{avg}+=-1/[len(G_1)*len(G_2)]*log_2(len(G_3))

Considering the number of words and the potential guesses in each round, traversing all possibilities requires a tremendously large computational cost. To improve computational efficiency, this study uses dynamic programming optimization, introducing a relationship matrix and modifying the code with recursive algorithms.

Relationship Matrix: This paper introduces a relationship matrix R. Its size is 2309×2309. The element in the u-th row and v-th column, $R_{u,v}$, represents the clue C_{a,g_i} when the answer word is a and the guessed word is g_i. For instance, when $a = 671$ (error) and $g_i = 677$ (ethos), Equation (7) can also be directly written as the form of Equation (17):

$$R_{671,677} = 20020 \tag{17}$$

Introducing the relationship matrix has the following advantages:

(1) Pre-computation Advantage: The relationship matrix provides pre-computed information about the relationship between all possible guessing words and target words. This means that during the actual word-guessing process, it's unnecessary to recompute the relationship between each pair of words. Instead, one can directly reference the relationship matrix, avoiding repetitive computations.

(2) Memory Efficiency: Even though the relationship matrix itself occupies a certain amount of storage space, it prevents the need to compute and store the relationship of every word with every other word during each guess. This approach conserves memory during the actual guessing process.

(3) Code Clarity & Readability: Using the relationship matrix allows representing the relationship between the guessing word and the target word as a simple lookup process, making the code cleaner, and more readable, and reducing the potential for errors.

(4) Scalability: If more words are added in the future or the relationships between words are changed, only a new relationship matrix needs to be generated, without the need to modify the main logic of the guessing process.

Figure 1 shows a small part of the relationship matrix.

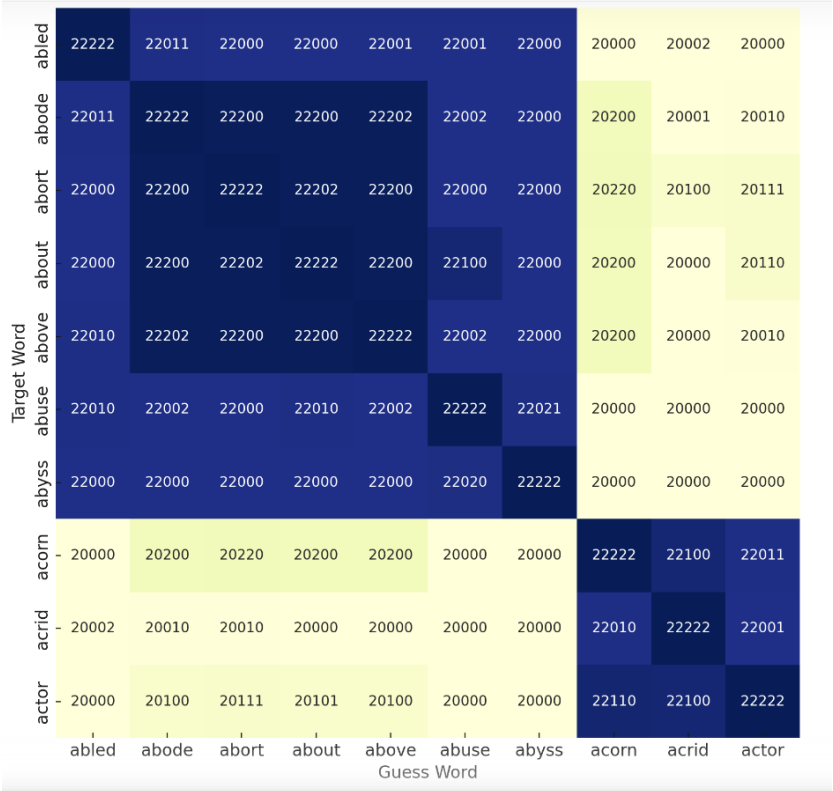


Figure 1. Relationship matrix (local)

In addition, this study also utilized the Parallel and delayed functions from the joblib library to parallelize the word guessing function, enabling multiprocessing and significantly improving the processing speed. In order to have an intuitive understanding of the entire traversal guessing process, the changes in the length of the guessing word space for each traversal are stored and displayed in the result matrix R_e. The transpose of the block R_e^a in the a-th column of R_e is shown as in Equation (18). Here, $G_i^k (i = 2,3)$ is the guessing word space of the i-th round after the k-th word of G_{i-1} is selected from the previous round. Then average mutual information was calculated Table.2 shows local results of the word space length changes for some words.

$$R_e^a = \left(G_1 \quad G_2^1 \quad G_3^1 \quad \cdots \quad G_1 \quad G_2^1 \quad G_3^{S(G_2)} \quad \cdots \quad G_1 \quad G_2^{S(G_1)} \quad G_3^{S\left(G_2^{S(G_1)}\right)} \right)^T \tag{18}$$

Table 2. Table of changes in the length of the word space (local)

aback	abase	abate	abbey	abbot	abhor	abide	abled
2309	2309	2309	2309	2309	2309	2309	2309
1	2	2	11	11	11	11	11
1	1	1	1	1	3	4	1
2309	2309	2309	2309	2309	2309	2309	2309
1	2	2	11	11	11	11	11
1	1	1	1	1	1	4	4
2309	2309	2309	2309	2309	2309	2309	2309
1	1	1	11	11	11	11	11
1	1	1	5	1	1	5	5
2309	2309	2309	2309	2309	2309	2309	2309
4	1	1	11	11	11	11	11
1	1	1	1	5	5	1	1

Table.3 below shows the pseudo-code of the optimized program:

Table 3. Pseudo-code of the optimized procedure for calculating I_{avg}

Algorithm 2: Optimized Calculation of Entropy

Input: Word List A, Relationship Matrix R
Output: Average Mutual Information I_{avg}
1. FUNCTION shrink_word_space(G, g, a, R):
2. clue = $R[a, g]$
3. FILTERED_WORDS = words in G where R[word, g] == clue
4. RETURN filtered words minus g
5.
6. FUNCTION recursive_guess(G, depth, counts, results, a, R):
7. IF depth == 0 OR G is empty:
8. RETURN counts
9. new_counts = EMPTY LIST
10. FOR each word g in G:
11. G_{new} = shrink_word_space(G, g, a, R)
12. new_counts APPEND recursive_guess(G_{new}, depth-1, updated_counts, a, R)
13. RETURN new_counts
14.
15. FUNCTION compute_results(R):
16. results = PARALLEL PROCESSING of recursive_guess for each a
17. FOR each result in results_list:
18. CONVERT result to {index: result} dictionary
19. FLATTEN and PAD each list in the results to make them of the same length
20. RETURN CONVERTED results
21. flattened_results = compute_results(R)
22. I_{avg}= COMPUTE Average Mutual Information for flattened_results

Finally, the K-means algorithm is run (specific parameter settings can be found in section 3.2) to classify the words into three difficulty levels (easy, normal, and hard). As a result, we obtain the difficulty label for each word. For instance, the word "EERIE" has an I_{avg} of approximately 8.997, which falls under the hard difficulty category. Table.4 provides an overview of the K-means classification results.

Table 4. Overview of K-means classification results

Difficulty level	Average Mutual Information Minimum/bit	Average Mutual Information Maximum/bit	Number of words
hard	7.438671	9.140118	791
normal	9.142559	9.576962	981
easy	9.577474	10.29857	537

3.5. Analyse of result

To more visually display the words of different difficulty levels, this study randomly selected 15 words from each difficulty level and charted them in a word cloud. In this word cloud shown in Figure 2, the levels of difficulty - easy, normal, and hard - are represented by blue, orange, and green colors, respectively. Moreover, the font size of the words corresponds proportionally to their average mutual information. This means words with higher average mutual information are displayed larger in the word cloud. Through this word cloud, readers can preliminarily understand which sample words are relatively hard in the game and which ones are comparatively easier.



Figure 2. Word cloud map (local)

The corresponding data for some of the words in the word cloud is shown in Table.5 below:

Table 5. Difficulty level of words in the word cloud

Words	Difficulty level	Average Mutual Information
flume	hard	9.04
wordy	hard	9.10
jumpy	hard	8.59
…	…	…
abbot	normal	9.23
natal	normal	9.15
qualm	normal	9.29
…	…	…
flora	easy	9.64
error	easy	9.68
raise	easy	9.84
…	…	…

4. Monte Carlo simulation validation and analysis

4.1. Establishing the Monte Carlo Simulation Model

Monte Carlo simulation is a method that uses random numbers to estimate numerical answers, especially suitable for solving problems that are hard to calculate directly. For the Wordle game, the Monte Carlo simulation provides us with a feasible means to verify and optimize the previously established classification model.

To verify the accuracy of our constructed classification model, we simulated a real word-guessing scenario and conducted a Monte Carlo simulation guessing experiment. The simulation continues until the computer correctly guesses the target word or exceeds 6 rounds to stop the simulation. The average round when the simulation stops is recorded, representing the average number of attempts required to guess the word. The specific implementation steps are as follows:

(1) Define the target word and guess word space:

This is the same as steps (1) and (2) in the 3.3 modeling process.

(2) Simulate the word guessing process:

For each target word a, 1000 simulations were conducted. In each simulation, a word is randomly selected from the current guess word space as the guess word. Based on the selected guess word and the actual target word, the corresponding clue is found in the relationship matrix. Based on the obtained clue, the guessing word space is updated, i.e., words that do not match the clue are excluded. Repeat the above steps until the player guesses the target word or exceeds 6 rounds, which counts as completing a simulation.

(3) Record the simulation results:

For each target word, the number of rounds required for the player to guess the target word in the 1000 simulations was recorded, referred to as the termination rounds, denoted as $r_1, r_2, \cdots r_{1000}$. Finally, the average termination rounds r_{avg} for each word is calculated using formula (19):

$$r_{avg} = \frac{\sum_i^{1000} r_i}{1000} \tag{19}$$

The pseudo-code representation of the algorithm is shown in Table.6 below:

Table 6. Pseudo-code for the Monte Carlo simulation algorithm

Algorithm 3: Monte Carlo Simulation
Input: Word List A, Relationship Matrix R
Output: Simulation Results
1. FUNCTION shrink_word_space(G, g, a, R):
2. clue = R[a, g]
3. FILTERED_WORDS = words in G where R[word, g] == clue
4. RETURN filtered words minus g
5.
6. FUNCTION recursive_guess(G, depth, counts, results, a, R):
7. IF depth == 0 OR G is empty:
8. RETURN counts
9. new_counts = EMPTY LIST
10. FOR each word g in G:
11. G_new = shrink_word_space(G, g, a, R)
12. new_counts APPEND recursive_guess(G_{new}, depth-1, updated_counts, a, R)
13. RETURN new_counts
14.
15. FUNCTION compute_results(R):
16. results = PARALLEL PROCESSING of recursive_guess for each a
17. FOR each result in results_list:

18. CONVERT result to {index: result} dictionary
19. FLATTEN and PAD each list in the results to make them of the same length
20. RETURN CONVERTED results
21. flattened_results = compute_results(R)
22. I_{avg}= COMPUTE Average Mutual Information for flattened_results

Running the code to get a table of the difficulty of the vocabulary game and the number of termination rounds shows that the word EERIE, which belongs to the difficulty level, has several termination rounds of $r(EERIE) = 3.502$, which is relatively high. Table.7, Table.8 show partially the difficulty level of words and their termination rounds:

Table 7. Difficulty scale (local)

words	Number of termination rounds	difficulty level
eerie	3.502	hard
exalt	3.245	normal
excel	3.571	hard
exert	3.614	hard
exile	3.982	hard
exist	3.238	normal
expel	3.724	hard
gloom	3.481	hard
glory	3.108	easy
gloss	3.515	normal
glove	3.667	hard

Table 8. Overview of the overall situation with words of different levels of difficulty

difficulty level	sample size	average value of r_{avg}	the standard deviation of r_{avg}	upper quartile of r_{avg}
hard	791	3.8475	0.298	3.796
normal	981	3.4028	0.1706	3.385
easy	537	3.0624	0.1336	3.071

4.2. Analysis of results

This section aims to explore the correlation between the average termination rounds of target words of different difficulty levels (easy, normal, hard) to verify the rationality and scientific nature of the classification.

This section uses the violin plot [10] (as shown in Figure 3) to display the overall distribution of average termination rounds at different difficulty levels. The violin plot is a visualization method used to display distribution and its probability density. This type of plot simultaneously shows the apparent median, quartiles, and outliers of the data, providing detailed information about the shape of the data distribution, which is very suitable for verifying whether there is a significant difference in the distribution of termination rounds among words of different difficulty levels.

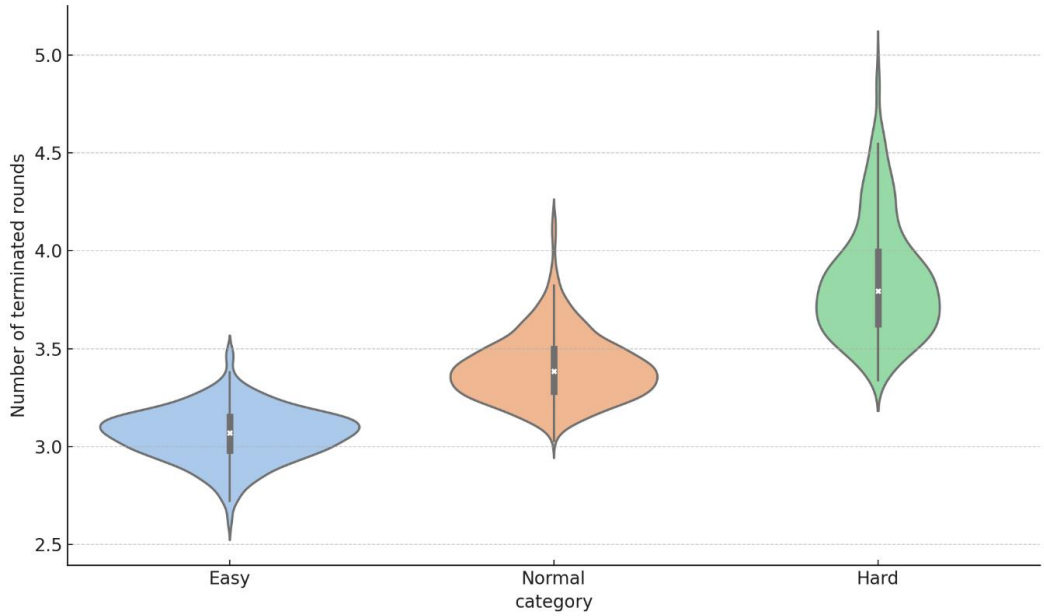


Figure 3. Violin plot

From the violin plot, we can observe that for "easy" difficulty, the density curve is relatively wide, indicating that the data points are more dispersed, but the vast majority of termination rounds are concentrated in a lower range. For "normal" difficulty, the density curve is relatively wide and the distribution is flatter. Although some termination rounds are smaller, there is also a significant proportion of higher termination rounds. For "hard" difficulty, the density curve is narrower and taller, indicating that most of the termination rounds are concentrated in a higher range. Overall, the violin plot intuitively shows the distribution difference of termination rounds under different difficulty levels. As the task difficulty increases, both the median and the quartiles of the termination rounds also gradually increase, reflecting the strong correlation between difficulty level and average termination rounds.

"In addition, this paper also conducted a numerical calculation of the correlation between difficulty levels and the number of termination rounds. The results of which are shown in Table.9, showing that the p-values are all less than 0.001, and there is a strong positive correlation between termination rounds and word difficulty. Therefore, we can reject the null hypothesis and conclude that there is a significant difference in the average termination rounds between the three difficulty levels (easy, normal, hard).

Table 9. Statistical verification of difficulty level and termination rounds

Statistical Measure	Value
Pearson Correlation Coefficient	0.806897
Pearson p-value	0
Spearman Rank Correlation Coefficient	0.862451
Spearman p-value	0
ANOVA F-statistic	2197.666
ANOVA p-value	0

To intuitively grasp the distribution of termination rounds and whether there are any outliers, this paper has drawn a histogram of the termination rounds distribution, as shown in Figure 4:

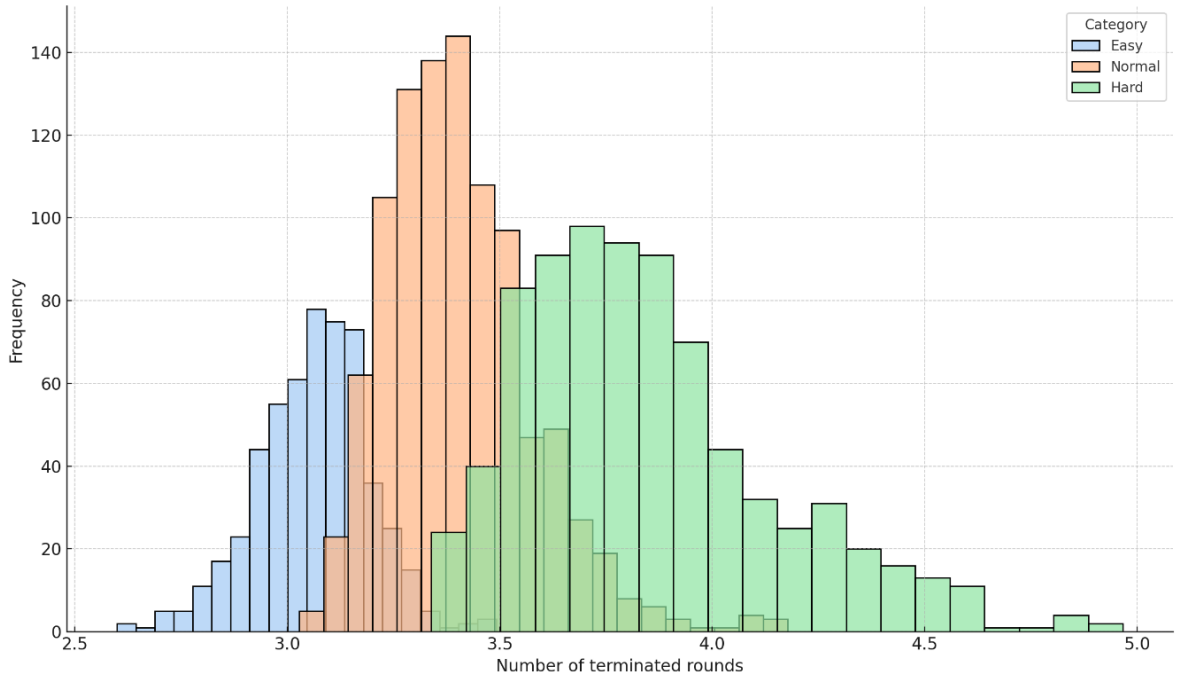


Figure 4. Histogram of the distribution of terminated rounds

Through the termination rounds distribution histogram, it can be seen that words of easy difficulty mostly terminate within fewer simulation rounds. Words of normal difficulty have a broader distribution of termination rounds, showing moderate variability. Hard difficulty: Most of the simulations terminate in a higher number of rounds. As the task difficulty increases, the required termination rounds also gradually increase. Overall, an average of 84.32% of the termination rounds are between 3-4, which verifies the fact that in word guessing strategy research [1][2][3][4], it's challenging to surpass 3 rounds.

5. Conclusion

This study delves deeply into the classification problem of Wordle target word difficulty and proposes a classification model based on average mutual information. We have proven that through optimization using dynamic programming techniques, the efficiency of this classification process can be effectively improved. Furthermore, the Monte Carlo simulation model provides us with a verification tool, ensuring that the classification model is consistent with the actual player's word-guessing experience. This classification method offers developers of Wordle and other similar word-guessing games a powerful tool, allowing them to provide players with more precise and diversified difficulty choices. Moreover, players can choose the appropriate difficulty based on their abilities and preferences, enhancing the gaming experience. Finally, this research not only provides new insights into the Wordle game but also offers references for other related field studies. We hope that future research can further explore other relevant factors of word difficulty in the game based on this foundation, providing game designers and developers with more in-depth insights.

References

[1] Li I. Analyzing difficulty of Wordle using linguistic characteristics to determine the average success of Twitter players [J]. 2022.

[2] Bertsimas D, Paskov A. An Exact and Interpretable Solution to Wordle[J]. Available at URL. (Accessed: 14 November 2022), 2022.

[3] Liu C L. Using wordle for learning to design and compare strategies[C]//2022 IEEE Conference on Games (CoG). IEEE, 2022: 465-472.

[4] Liu Wei,Zou Peng,Jiang Dingguo,Quan Xiufeng,Dai Huichao. Zoning of reservoir water temperature field based on K-means clustering algorithm[J]. Journal of Hydrology: Re-gional Studies,2022,44.

[5] de Silva N. Selecting seed words for wordle using character statistics[J]. arXiv preprint arXiv:2202.03457, 2022.

[6] Liu B, Zhang Y, Zhang S. Exploring the difficulty of words and its influential attributes based on the Wordle game[J]. arXiv preprint arXiv:2305.03502, 2023.

[7] Yang Q, Fang Y, Zheng Y. Word Data Research and Prediction Based on Wordle Game[J]. Academic Journal of Computing & Information Science, 6(4): 106-109.

[8] Fetterman D, Unemployed O. Solving N-dle using Information Entropy[J].

[9] Kundu A, He Y. On optimal order in modeling sequence of letters in words of common language as a Markov chain[J]. Pattern Recognition, 1991, 24(7): 603-608.

[10] Hullman J, Resnick P, Adar E. Hypothetical outcome plots outperform error bars and violin plots for inferences about reliability of variable ordering [J]. PloS one, 2015, 10(11): e0142444.