Using Information Theory to Optimize Wordle Solving

Adam Boswell

I. Background on Information Theory and Wordle

Wordle is a captivating game where players guess a five-letter secret word, using feedback to systematically narrow down potential solutions. As explored in Problem Set 5, Wordle's mechanics highlight a strong connection to information theory, offering a structured approach to problem-solving through iterative refinement. Inspired by P-Set 5, this project involves scaling up that problem and fully realizing a Wordle Solver that can be applied to a real game.

At the core of solving Wordle lies the concept of entropy, which measures uncertainty in a system. In the context of Wordle, entropy quantifies how much information a guess provides about the secret word. A high-entropy guess is one that maximally partitions the remaining pool of possible solutions into distinct subsets based on the feedback.

The entropy $H(X)$ of a discrete random variable X is defined as:

$$H(X) = -\sum_{i=1}^{n} P(x_i) \log_2 P(x_i)$$

Here:

- X represents the set of possible feedback patterns for a given guess.
- $P(x_i)$ is the probability of receiving feedback pattern x_i.

In Wordle, a guess's feedback depends on:

1. **Letter correctness:** Matches the secret word.
2. **Position correctness:** Matches the correct position.
3. **Exclusion:** Does not exist in the secret word.

For each potential guess, the entropy calculation involves:

1. Simulating feedback for that guess against all possible secret words.
2. Calculating probabilities for each type of feedback based on how often it occurs in the remaining word pool.
3. Summing the weighted information contributions of each feedback type.

By prioritizing guesses with the highest entropy, the solver ensures maximum information gain, which accelerates the elimination of improbable words.

Inference

Inference in Wordle involves updating the remaining solution space based on the feedback from each guess. After receiving feedback:

1. **Green (correct letter and position):** The solver keeps only words with the specified letter in the specified position.

2. **Yellow (correct letter, wrong position):** The solver retains words containing the specified letter but excludes it from the guessed position.

3. **Gray (letter not in the word):** The solver eliminates all words containing the specified letter.

To initialize the system, I used data collected from the Google Books N-Gram frequency lists for all five-letter words. This dataset provides historical frequency information for 12,972 words. To generate priors for these words, I averaged the last 10 recorded frequencies for each word, treating these averages as the baseline probability distribution. While the exact temporal ordering of the data storage was not fully understood, this method ensured a reasonable prior probability for each word based on historical trends.

As the solution pool was narrowed down with each guess, these priors were dynamically updated to reflect the constraints imposed by feedback. For example:

- Words that no longer fit the feedback (e.g., containing gray letters or lacking required green/yellow letters) were removed from the pool.
- The remaining words were reweighted according to their adjusted probabilities, ensuring that the inference process reflected both the constraints and the historical frequency data.

This adaptive use of priors provided a solid foundation for probabilistic reasoning, enabling the solver to leverage real-world frequency distributions while systematically refining the search space with each iteration.

Thompson Sampling and Beta Distributions

Inference in Wordle involves updating the remaining solution space based on the feedback from each guess. After receiving feedback:

1. A Beta distribution $\text{Beta}(\alpha, \beta)$ is initialized for each word in the guess pool and updated based on observed feedback.
2. Sampling from this distribution provides a probabilistic estimate of a word's likelihood, balancing exploration (testing less certain words) and exploitation (focusing on high-probability candidates).

The Thompson sampling process dynamically adjusts α and β after every guess:

$$\alpha = \alpha + \text{matches}, \quad \beta = \beta + \text{non-matches}$$

At the beginning of the game, all words in the guess pool were initialized with a uniform prior distribution $\text{Beta}(1, 1)$, reflecting equal likelihood across all options. As the game progresses and feedback is incorporated, the parameters α and β for each word are updated based on its alignment with the observed feedback.

The goal of this approach is that over multiple rounds of play, the Beta distribution for each word evolves to better reflect its actual likelihood of being a viable guess, informed by both prior knowledge and real-time updates. This dynamic adjustment ensures that the solver continually refines its probabilities, improving its decision-making as it gains more information.

Marrying Entropy, Inference, and Thompson Sampling

This project presents an opportunity to integrate entropy calculations, inference, and Thompson sampling into a unified model for Wordle solving. The system operates as follows:

1. **Entropy Maximization:** Scores each guess based on its potential to partition the solution space.
2. **Inference:** Refines the remaining word pool based on feedback constraints.
3. **Thompson Sampling:** Incorporates probabilistic modeling to prioritize words dynamically.

The combined model ensures:

- High information gain from each guess.

- Efficient narrowing of the solution space.
- Adaptability to feedback uncertainty through Bayesian updates.

Mathematically, the scoring function used to determine the best word g for narrowing down the solution pool was defined as:

$$S(g) = H(g) + 0.5 \cdot (T(g) + P(g))$$

Where:

- $H(g)$ is the entropy of the guess, reflecting the potential information gain.
- $T(g)$ is the Thompson-sampled probability of the word g being the secret word, based on its Beta distribution.
- $P(g)$ is the prior probability of the word g, derived from the Google Books N-Gram dataset.
- The factor 0.5 balances the contribution of Thompson sampling and the prior probability in the overall score.

The inclusion of both $T(g)$ and $P(g)$ ensures that the model accounts for real-time feedback while grounding decisions in historical word frequency data. By combining these elements, the scoring function effectively integrates uncertainty reduction (via entropy), probabilistic reasoning (via Thompson sampling), and historical trends (via priors).

By iterating this process, the solver systematically converges on the secret word with both efficiency and precision. This hybrid approach demonstrates how computational techniques can blend seamlessly to tackle real-world combinatorial challenges like Wordle.

Notes on Implementation and Performance

This project culminated in a fully functional Wordle solver web application, featuring a Flask-based backend for handling probability calculations and a React-based frontend for creating an interactive user interface. The frontend displays the Wordle game board, feedback indicators, and detailed calculations for entropy, prior probabilities, and Thompson sampling results. A particularly noteworthy feature of the frontend is the ability to dynamically click on the Thompson sampling result for a guessed word and visualize the Beta distribution that was sampled. This visualization provides an intuitive understanding of how Bayesian inference guides the solver's decision-making process.

Implementation Journey

When starting this project, I had no prior experience with backend development. Learning to set up Flask API endpoints, connect them to the React frontend, and manage stateful interactions was both challenging and rewarding. I relied on ChatGPT to understand best practices and troubleshoot issues during development. Despite these challenges, integrating the probabilistic backend with a dynamic frontend was a valuable exercise in bridging theoretical concepts with practical application.

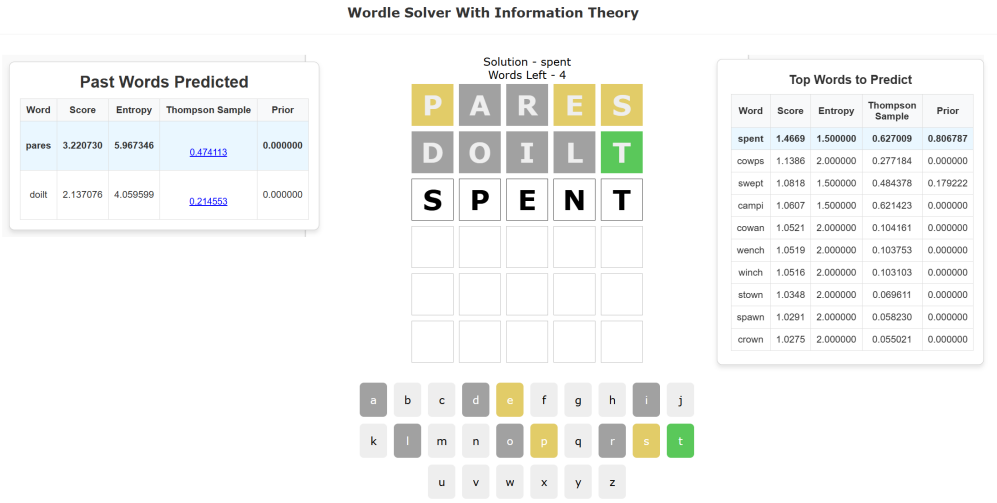
*Attached at the end is an example image of the application interface :)

Performance and Limitations

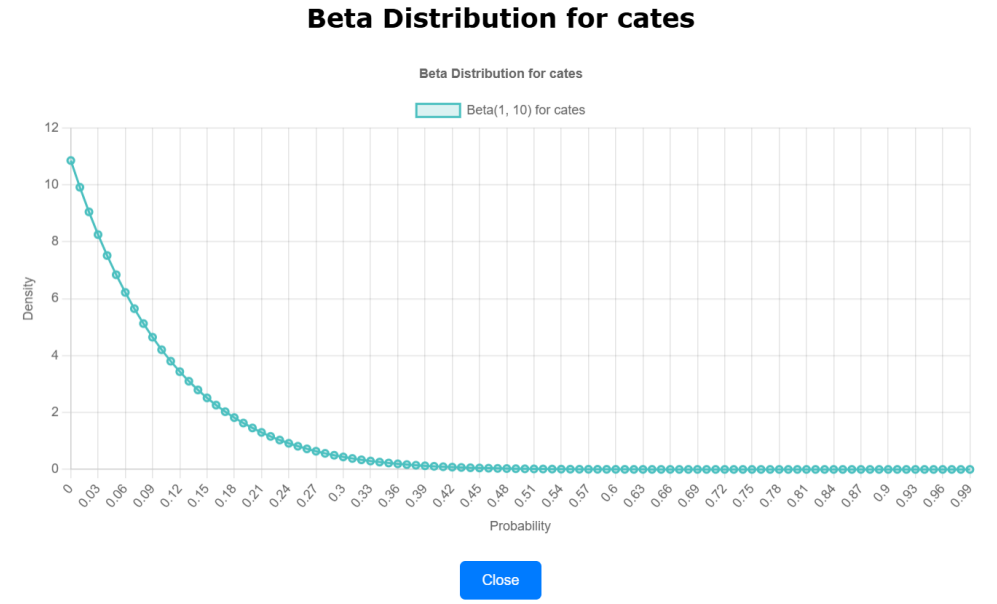
One limitation of the current implementation is the computational overhead of the recommendation algorithm. To determine the best next guess, the solver iterates through all viable words in the solution pool and calculates a score for each. This exhaustive approach, while effective for small-scale testing, results in longer processing times for larger word pools. Consequently, I was unable to conduct large-scale testing with extensive datasets.

Despite this limitation, observational results suggest that the solver effectively narrows down Wordle puzzles in an average of 3 to 4 guesses. This performance is consistent with the principles of information

theory and Bayesian inference, which the solver was designed to leverage. Future iterations could improve efficiency by optimizing the scoring algorithm or parallelizing computations.



User Interface for Wordle



This is the result of clicking on blue hyperlink underneath the Thompson Sample Column. It displays the Beta distribution that the thompson sample was sambled from.

** For a more robust code review check out my repository! - https://github.com/abboswell/wordle **