Wordle Difficulty Calculator

Table of Contents

Introduction & Motivation

Overview of the Scoring System

Feature Breakdown (Non-Technical)

Technical Feature Breakdown

Scoring Formula and Weighting

Technical Implementation Overview

Data Sources & Preprocessing

Final Output and Use Cases

1. Introduction & Motivation

The game Wordle challenges players to guess a hidden five-letter word in six tries. While simple on the surface, some words are inherently harder to guess due to factors such as letter rarity, structure ambiguity, or lack of common patterns.

This project aims to develop a comprehensive, data-driven scoring system to quantify the difficulty of any 5-letter Wordle word using multiple linguistic and statistical features. The goal is to answer:

> “How hard is this word, and why?”

By breaking down a word into interpretable features and using machine learning to estimate structure-based difficulty, we can produce a reliable expected guess count and a transparent difficulty report.

2. Overview of the Scoring System

Each word is scored using six primary factors, each capturing a distinct dimension of difficulty. These are:

- LFS — How rare the letters are overall
- PLFS — How rare the letters are in their positions
- RLP — Whether the word repeats letters
- HLT — How hard its bigram transitions are
- WSA — How ambiguous its structure is when guessed
- OS — How obscure the word is in everyday usage

These values are combined into an Overall Difficulty Score (ODS) from 0 to 1, which is then converted to an Expected Guesses count (2.5–6.0 scale).

This multi-factor system ensures we don't just score a word for being obscure (like "BLAUD"), but also penalize those that mislead structure or hide rare transitions (like "CRYPT").

3. Feature Breakdown (Non-Technical)

1. Letter Frequency Score (LFS)

🟢 **What it means:**

How common the letters in a word are across all 5-letter English words.

🛠️ **How it's calculated:**

We take the average frequency of each of the 5 letters in the word, based on how often those letters appear in a large corpus of valid Wordle words.

📈 **Why it matters:**

Words made of rare letters (like Z, Q, or X) are harder to guess. More common letters make a word easier to hit through trial and error.

2. Positional Letter Frequency Score (PLFS)

🟢 **What it means:**

How likely each letter is to appear in the specific position it's in (first, second, etc.).

🛠️ **How it's calculated:**

We look at how frequently each letter appears in each of the 5 positions across all known words and average them.

📈 **Why it matters:**

Some letters are very common at the start or end of words (like S or E), while others are rare in certain positions. Words with letters in unexpected positions are harder to guess.

➖ 3. Repeated Letter Penalty (RLP)

🟢 **What it means:**

Whether the word has repeated letters — and if so, how many.

🛠️ **How it's calculated:**

It's a value from 0 to 1:

- **0** means all letters are unique
- **1** means several letters repeat

📈 **Why it matters:**

Repeated letters are very hard to guess in Wordle because once a letter is revealed, players don't usually guess it again right away — leading to extra turns.

➖ 4. Hard Letter Transitions (HLT)

🟢 **What it means:**

Whether the combinations of letters in the word are common or rare.

🛠️ **How it's calculated:**

We break the word into bigrams (e.g., CR, RY, YP, PT for "CRYPT") and score how rare these two-letter combos are.

📈 **Why it matters:**

Even if a word contains common letters, if they're arranged in rare patterns (like "PT" or "XZ"), it's tougher to guess.

➖ 5. Word Structure Ambiguity (WSA)

🟢 What it means:

How structurally similar the word is to other words, in terms of game feedback.

🛠️ How it's calculated:

We simulate using the word as a starting guess and measure:

- How many potential solutions are eliminated
- The average "informativeness" of the feedback (green/yellow/gray letters)

Then, we train a model to predict a WSA score from similar features.

📈 Why it matters:

Words that don't help narrow down possibilities (like "MAMBO") are harder to play with. Words that eliminate many options are easier to reason about and solve.

➖ 6. Obscurity Score (OS)

🟢 What it means:

How rare or unfamiliar the word is in common English usage.

🛠️ How it's calculated:

We use a large English frequency list (like from subtitles, books, etc.) and map each word to a score from 0 to 1, where:

0 = very common
1 = extremely rare

We use log-scaling and percentile smoothing to avoid big jumps in distribution.

📈 Why it matters:

If you've never heard of a word, you're unlikely to guess it. Even perfect feedback doesn't help if the solution is something you don't know (e.g., "ZEBEC").

4. Technical Breakdown of Each Feature

🔹 1. Letter Frequency Score (LFS)

File: features/letter_features.py

Function: compute_lfs(word: str)

Logic:

- Load the full list of 5-letter words from words.txt
- Count how often each letter (A-Z) appears across all words
- Normalize counts to a 0–1 scale
- For a given word, take the average score of its 5 letters

We used Norvig's Word Corpora

This gives us values from 0 to 0.8 (because the word needs to be "ZZZZZ" to have a value of 1)

To fix this, we distribute the values from 0 - 0.4 to 0 -0.5 and from 0.4 - 0.8 to 0.5 -1

This gives a more accurate distribution from 0 - 1

🔹 2. Positional Letter Frequency Score (PLFS)

File: features/letter_features.py

Function: compute_plfs(word: str)

Logic:

- Calculate frequency of each letter in each position (e.g., how often "A" appears in position 1)
- Store frequency for positions 0–4 as a dictionary
- For a word, average the frequencies of its letters in those position

🔹 3. Repeated Letter Penalty (RLP)

File: features/letter_features.py

Function: compute_rlp(word: str)

Logic:

- Checks whether the word consists only of unique letters
- Uses a binary approach to calculation
- The RLP value is 1 if the word contains repeated letters, and 0 if it doesn't

🔹 4. Hard Letter Transitions (HLT)

File: features/bigram_features.py

Function: compute_hlt(word: str)

Logic:

- Use a bigram_log_rarity_scores.csv file containing log rarity scores for all 2-letter combinations
- For each word, generate bigrams like "ST", "TA", "AR", "RE"
- Look up each bigram's rarity, average across the word

We used a percentile based rank system to get a smooth normalization from 0 - 1

🔹 5. Word Structure Ambiguity (WSA)

WSA is the only feature derived from a trained machine learning model, as it represents gameplay behavior rather than raw letter statistics.

Concept

WSA measures how useful a word is when used as a guess. Some words, like "SLATE", greatly reduce the remaining solution pool after one guess. Others, like "MAMBO", eliminate few candidates and are poor strategic openers.

This intuition is modeled by simulating feedback ("green/yellow/gray") across all Wordle solutions and recording:

- Average number of words eliminated
- Frequency of green/yellow/gray feedback
- Total match reduction per guess

Training the Model

A custom script generates wsa_training_data.csv, with each word's:

- Basic features (LFS, PLFS, RLP, HLT)
- Structural features (match reduction, feedback patterns)
- **Target value:** match reduction (to be inverted — low = hard)

A RandomForestRegressor is trained on this data.

Predictions are inverted and normalized:

- WSA = 1 → very unhelpful guess
- WSA = 0 → extremely helpful guess

Normalization range (wsa_range.txt) is saved to map predictions to 0–1 consistently during inference.

This model allows the system to simulate gameplay-based structure quality without requiring full Wordle simulation for each word at runtime.

🔸 Training Data

File: train/generate_training_data.py

Generates: data/wsa_training_data.csv

Each row includes:

- Word
- LFS, PLFS, RLP, HLT
- **Shape features:**
  - MatchReduction
  - AvgElimination
  - FeedbackGreen, FeedbackYellow, FeedbackGray

🔸 Model Training

File: train/train_wsa_model.py

Trains: models/wsa_regressor.pkl

model = RandomForestRegressor()

model.fit(X, inverted_match_reduction)

Also saves the normalization range in wsa_range.txt.

🔸 Inference

File: app/calculator.py

Function: calculate_difficulty(word: str)

Computes same shape features

Passes them into the trained model

Gets predicted WSA score and normalizes it

raw_wsa = model.predict(model_input)[0]

wsa_normalized = normalize(raw_wsa, wsa_min, wsa_max)

🔹 6. Obscurity Score (OS)

File: features/obscurity_score.py

Function: compute_os(word: str)

Logic:

- Load word_frequencies.csv with total frequencies for all English words
- For each word in words.txt, get its frequency
- Since the word frequency values are very haphazardly distributed, we segregate OS values directly by their raw frequency values

5. Scoring Formula and Weighting

Each of the six features contributes to the word's overall difficulty. The Overall Difficulty Score (ODS) is calculated as a weighted combination of all normalized factors:

ODS = 0.25 * LFS

+ 0.15 * PLFS

+ 0.15 * RLP

+ 0.15 * HLT

+ 0.15 * WSA

+ 0.15 * OS

Each feature is scaled from 0 (easy) to 1 (hard), so the final ODS is also between 0 and 1. This value is then transformed into a realistic guess count using:

ExpectedGuesses = 2.5 + ODS * 3.5

This mapping is based on the Wordle range of 6 guesses, with 2.5 being the baseline for trivial words and 6.0 being the upper cap for extremely hard ones.

LFS jumps out as the most influential factor while determining the difficulty of the word, hence its large weightage. The rest of the factors all play a relatively equal part in intuiting the word's difficulty.

6. Technical Implementation Overview

The entire pipeline is implemented in modular Python using:

- scikit-learn for machine learning (WSA)
- pandas for data manipulation
- joblib for model persistence
- Custom scripts and CLI utilities for testing and exporting results

Key Components

- **app/calculator.py:** Central logic for computing difficulty scores
- **features/:** Folder containing functions to compute LFS, PLFS, RLP, HLT, OS, and WSA features
- **models/:** Stores the trained WSA model (wsa_regressor.pkl) and metadata like wsa_range.txt
- **train/:** Scripts to generate training data and train the WSA model
- **data/:** Contains all necessary datasets like words.txt, bigram_log_rarity_scores.csv, word_frequencies.csv, etc.

7 . Limitations & Future Work

While the current model is a strong first implementation, some limitations remain:

Known Limitations

- **WSA Model Generalization:** The WSA regressor is only as good as its training simulation. Some rare edge-case words might have misleading scores.
- **OS Accuracy:** OS depends on external frequency data that may not reflect actual Wordle usage patterns (e.g. pop culture words, abbreviations).
- **Static Weights:** ODS weights are currently fixed. They could benefit from empirical optimization using user gameplay data.

Potential Improvements

- Train WSA model using real guess distributions from player data
- Add support for theme-based obscurity (e.g., science words, slang)
- Introduce dynamic weighting based on context or word length