Documentation for HangmanAI1 Code

Core Idea: Word Length Differentiation

The primary strategy is to use different approaches based on word length:

My strategy does the following differentiation:

- **Short Words (6 characters or less)**:
  - Relies on dictionary-based filtering
  - Utilizes vowel priors based on length of the word
  - Uses simple heuristics and frequency analysis
- **Long Words (more than 6 characters)**:
  - Leverages machine learning models
  - Improves predictions for the ML methods using all the above methods

Reason for doing the above differentiation: While using the umbrella pipeline (the pipeline described for long words) for all types of words, I observed poor performance of the ML models on the short words. Therefore, I switched to purely statistics and pattern matching based methods for the short words. The intuition behind this is that shorter words tend to follow common structural patterns or templates, such as CVC (Consonant-Vowel-Consonant), or certain n-grams, which are easier for statistics-based approaches to detect and predict.

Data Encoding and Preprocessing:

Before inputting the data to the ML models, the following preprocessing was performed:

1. **Variation Generation:**
   - **Generates multiple training samples from each word**
   - **Creates variations by systematically removing one letter**
   - **For example: Input word: "hello"**

     Variations: "h_llo" (missing 'e'), "he_lo" (missing 'l'), "hel_o" (missing 'l'), "hell_" (missing 'o')
2. **Alphabet Encoding:**
   - **Converts letters and underscore to numerical representation**
3. **One-hot encoding:**
   - **Fixed-length encoding (30 characters)**
   - **Pads shorter inputs with zeros**

Word Length Differentiation in detail:

Short Word Strategy

For words with 6 or fewer characters, the algorithm uses:

a. **Dictionary Filtering:**
- **Converts the partially guessed word into a regex pattern**
- **Filters the full dictionary to find words matching the current pattern**
- **Counts letter frequencies across all matching dictionary words**
- **Prioritizes letters that appear more frequently in potential words**

b. **Fallback mechanisms based on frequency of letters and vowel priors:**
- **Uses vowel prior probabilities (vowel probability distribution for different word lengths from the given dictionary)**
- **Falls back to most frequent letters if no vowels remain**

Long Word Strategy

For words with more than 6 characters, the algorithm uses:

a. **Machine Learning Ensemble:**
   - **Employs an ensemble of three ML models: CatBoost, XGBoost and LightGBM**
   - **Uses weighted averaging to combine predictions of the three models**

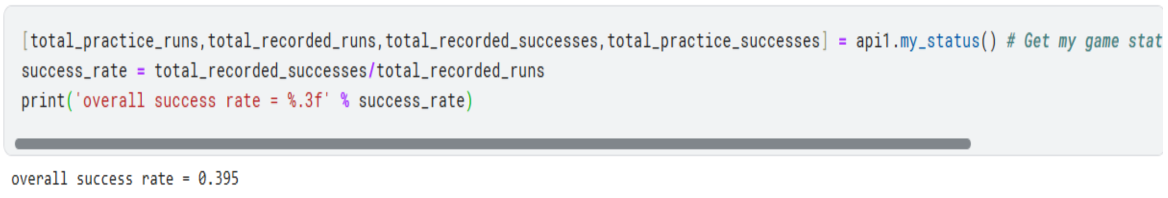
b. **Dictionary Filtering:**
   - **Constraints predictions using the matching procedure described in 1a.**

c. **Bias towards vowel selections:**
   - **Vowel Preference:**
     - **Maximum of 3 vowel guesses allowed**
     - **Activated dynamically during guessing**
     - **Integrated with ML predictions for long words**
     - **As a result, it increases vowel prediction probabilities and provides a subtle bias towards vowel guessing**

d. **Fallback mechanisms based on frequency of letters and vowel priors:**
   - **Uses vowel prior probabilities**
   - **Falls back to most frequent letters if no vowels remain**

Results:

Success rate: 0.395



```
[total_practice_runs,total_recorded_runs,total_recorded_successes,total_practice_successes] = api1.my_status() # Get my game stat
success_rate = total_recorded_successes/total_recorded_runs
print('overall success rate = %.3f' % success_rate)
```

```
overall success rate = 0.395
```