HANGMAN GAME PREDICTION REPORT

Prepared By:
Aditya Mishra

PROJECT OVERVIEW:

This project aims to develop a Hangman game solver using Machine Learning (ML) techniques. Hangman is a word guessing game where one player thinks of a word, and the other player tries to guess it by suggesting letters within a certain number of guesses. ML models such as N-Gram Neural Networks can be employed to predict the most likely words based on the letters guessed so far, thus optimizing the guessing strategy and enhancing the solver's efficiency. By leveraging features such as word length, letter frequency, and known letters, the model can accurately predict the next optimal letter to guess, facilitating a successful solution to the Hangman game.

PROBLEM STATEMENT:

Hangman is a classic word-guessing game typically played between two players where one player thinks of a word and the other player tries to guess it by suggesting letters within a certain number of attempts. In this context, we'll focus on the computer generating a word for the player to guess.

The goal of this problem statement is to develop an algorithm/model capable of playing the Hangman game effectively with a good score. Your algorithm has to suggest the next letter to guess based on the current state of the partially revealed word. The implementation should be able to handle unseen words during testing and provide accurate suggestions to maximize the chances of winning the game.The dataset provided for this problem statement consists of 50,000 English words. It's important to note that the dataset is self-contained, with no external sources allowed during training.

The task for us is to implement the ‘suggest_next_letter_sol’ function in the provided ‘your_solution.py’ file, similar to that implemented in ‘guess.py’. This function should take into account the current state of the word (with known letters revealed and unknown letters represented by dashes) and suggest the next letter to guess. The function should be designed to make intelligent decisions based on the available information to maximize the chances of successfully guessing the word within the given number of attempts.

APPROACH & ALGORITHM :

The approach adopted for the Hangman game predictor revolves around the utilization of **n-gram models**, specifically focusing on unigram, bigram, trigram, four-gram, and five-gram models, to predict the next optimal letter to guess during the game.

An n-gram model is a probabilistic language model used in natural language processing (NLP) and machine learning. It predicts the occurrence of a word or letter based on the context provided by the preceding n -1 words or letters. In simple terms, it analyzes sequences of n items (words, letters, or tokens) to infer the likelihood of the next item in the sequence.
The first step involves preprocessing the training dataset, which consists of 50,000 words. Non-alphabetic characters are removed, and all letters are converted to lowercase to ensure consistency and uniformity in the dataset. After preprocessing, n-gram models are constructed using nested dictionaries to capture the patterns and relationships between letters within words. Each n-gram model (unigram, bigram, trigram, four-gram, and five-gram) considers sequences of consecutive letters within words to calculate probabilities of letter occurrences.
Probabilities are calculated based on the occurrences of letter sequences observed in the training dataset. The frequency of each letter sequence within words of varying lengths is recorded in the nested dictionaries associated with each n-gram model.To enhance the predictive accuracy, probabilities obtained from different n-gram models are interpolated. Interpolation allows for the adjustment of probabilities between different models, providing a more refined estimate of the likelihood of each letter appearing in a particular context within a word.

During the Hangman game, the constructed n-gram models are utilized to predict the next optimal letter to guess. The probabilities calculated by the models help in making informed guesses,aiming to minimize mistakes and increase the likelihood of correctly guessing the hidden word.Overall, the solver leverages n-gram models to analyze letter sequences within words and make informed predictions during the Hangman game, enhancing the player's gaming experience and performance.

DATA PRE-PROCESSING

- **DATA ANALYSIS :**
  The dataset comprises 50,000 words, which are preprocessed to remove non-alphabetic characters and converted to lowercase. Unique words are extracted, and letter frequencies are analyzed to identify patterns and relationships between letters within words.

- **TRAIN-TEST SPLITTING:**
  The train-test split is a fundamental step in machine learning model development that involves partitioning a dataset into two distinct subsets: the training set and the testing set. The primary purpose of this split is to evaluate how well a trained model generalizes to new, unseen data. The training set is used to train the model by exposing it to labeled examples, allowing it to learn the underlying patterns in the data. Meanwhile, the testing set, kept separate from the training process, serves as a benchmark

to assess the model's performance on data it has not encountered during training.
Striking the right balance between the training and testing sets is crucial for developing a model that can make accurate predictions on new, real-world data. This practice helps detect overfitting, where a model memorizes the training data but fails to generalize well to unseen examples. In summary, the train-test split is a vital practice in machine learning to ensure the robustness and reliability of a trained model when deployed in practical scenarios.
The dataset is thereby split into training and test sets, with 90% of the data used for training the model and the remaining 10% reserved for evaluating the model's performance. This ensures that the model is trained on a diverse range of words and validated on unseen data to assess its generalization ability.

MODEL & ITS TRAINING:

MODEL IMPLEMENTATION:

The Hangman game predictor utilizes n-gram models (specifically unigram, bigram, trigram, four-gram, and five-gram) to predict the next optimal letter to guess during the game. The data which was preprocessed into a word list is now fed into the n_gram model.

- **BUILDING N-GRAM MODEL:**
  The build_n_grams function constructs nested dictionaries to store occurrences of n-letter sequences for each word in the training set. For each word, the function iterates through its characters to count occurrences of n-grams up to five letters long. Unigrams capture occurrences of single letters within each word. Bigrams capture occurrences of letter pairs within each word. Trigrams capture occurrences of three-letter sequences within each word. Similarly, four-grams and five-grams capture occurrences of four-letter and five-letter sequences, respectively. These n-grams are stored in nested dictionaries for efficient access during prediction.

- **PROBABILISTIC PREDICTION:**
  Functions namely unigram_probs, bigram_probs, trigram_probs, fourgram_probs, and fivegram_probs calculate the probabilities of each letter being the next correct guess. These probabilities are interpolated between different n-grams to provide more accurate predictions. The model considers the context of the word (sequence of letters) and adjusts the probabilities accordingly.

- **PROBABILITY CALCULATION:**
  - The function traverses each character in the word and checks if it is a blank space ('_'), indicating an unguessed letter. For each blank space encountered, the function iterates over each letter in the alphabet.
  - If the letter has occurred in words of the given length in the training data (unigram[len(word)][letter] > 0) and it hasn't been guessed yet (letter not in

guessed_letters), its occurrence count is added to total_count and letter_count.

- After traversing the entire word, the probabilities of each letter are calculated by dividing its count by the total count of all letters observed in words of the given length.
- The probabilities are then added to the existing probabilities (probabilities) with a small weight (0.05) to interpolate between the unigram model and previously calculated probabilities. Finally, the probabilities are adjusted so that they sum up to one

Similar calculations are performed for other n-gram models The probabilities are thereby interpolated with the previous calculated probabilities and the necessary actions are taken in the model function.

- **MODEL USAGE:**
  The model utilizes the probabilities derived from the n-grams to suggest the next letter to be guessed in the Hangman game. The Hangman game solver makes informed guesses based on the probabilistic predictions generated by the n-gram model, aiming to minimize the number of mistakes made during gameplay.

- **GUESS:**

  The guess function invokes the model from my_model.py after each guess by passing the displayed word. Additionally, if we're running low on guesses (i.e., if the last guess was incorrect and the number of incorrect guesses

exceeds 3), we remove all words from the dictionary containing letters present in the incorrect guesses list. Then, we create a new dictionary using the remaining words and calculate probabilities based on this new dictionary. Finally, we call the model to make the next guess.

MODEL TRAINING:

- The training data is loaded from the processed word list extracted from the file training.txt. The data is then broken into two sets consisting of training and testing sets.
- The n-gram model is then built and trained upon the training sets.
- The hangman function is utilized for training and testing the model simultaneously.
- During training, the model uses the probabilities derived from the n-grams to make predictions for the next letter to be guessed in the Hangman game. The training process involves updating the model parameters based on the mistakes made during gameplay.
- Testing is performed by evaluating the model's performance on a separate test set, measuring the average number of incorrect guesses made by the model across multiple words.

  After training the model,the model is evaluated using test_guesser function.The average number of incorrect guesses made by the model is calculated, providing a metric to assess the model's effectiveness in solving the Hangman game.

RESULT AND DISCUSSIONS:

The trained Hangman game predictor is evaluated on the test set to assess its performance. The average number of mistakes made by the predictor during the Hangman game is calculated, indicating the effectiveness of the model in minimizing mistakes and predicting correct letters. The results demonstrate the capability of the 5-gram model in accurately predicting letter sequences and enhancing the player's performance in the Hangman game. It is concluded that the accuracy of the 5 gram model turns out to be approximately 66% which is much better than the statistical modeling approach which gives around 18% accuracy.

IMPROVEMENTS:

- Experiment with higher-order n-grams (e.g., six-gram or higher) to capture more intricate patterns within words.

- Implement more sophisticated smoothing techniques to handle unseen n-gram combinations more effectively, such as Good-Turing smoothing or Kneser-Ney smoothing.

- Fine-tune the interpolation weights between different n-gram models to optimize the balance between each model's contribution to the overall prediction.

- Experiment with different interpolation functions or approaches, such as adaptive or dynamic weighting based on the context of the word being guessed.

- Implement robust error-handling mechanisms to gracefully handle edge cases or unexpected scenarios encountered during gameplay

REFERENCES:

- https://www.kdnuggets.com/2022/06/ngram-language-modeling-natural-language-processing.html
- https://www.researchgate.net/publication/221489288_Growing_an_n-gram_language_model
- https://www.geeksforgeeks.org/n-gram-language-modelling-with-nltk/