Hangman Report

Arnav Singla, IIT Kanpur
Statistics and Data Science
arnavs21@iitk.ac.in

September 15, 2024

Contents

1 Introduction

Hangman stands as a timeless game wherein players endeavor to unveil a concealed word, depicted by blank spaces, by guessing one letter at a time, all while aiming to circumvent six incorrect guesses. During my application for a remote internship at TrexQuant, I dedicated efforts to enhancing a computerized version of Hangman. The primary objective was to devise a more efficient algorithm, achieving an accuracy surpassing 50%
.

This report elucidates the process behind the development of my Hangman-playing program. I delved into English word patterns and statistical analyses to construct an intelligent model. The overarching goal was to heighten the likelihood of correctly guessing letters and subsequently augmenting the program's success rate.

2 Intuition

1. Guessing words containing only 2 or 3 letters poses a formidable challenge for both humans and computers alike. Therefore, prompting the computer to decipher these short words presents a particularly demanding task.
2. Approximately 75% of the words in the training corpus are longer than seven. In machine learning, it is common practice to assume that the distribution of our test set is comparable to that of the training data.
3. My first goal was to target lengthier words because we can more readily guess them by using common patterns seen in the English language.
4. Longer words are frequently made up of a short word plus "ing" or "ness".

3 Methodology

I investigated deep learning and statistical techniques;Before moving on to the methodology, I will give a brief overview of what **n-gram** models are.

3.1 N-gram Models

I N-gram models are widely used in natural language processing for language modeling and prediction tasks. These models analyze sequences of n consecutive items such as words or characters in a text to predict the likelihood of the next item based on the context provided by the preceding $n-1$ items.

II Bigram models (where $n = 2$) focus on pairs of consecutive items. For example, in the sentence "The cat sat," a bigram model would analyze pairs like ("The", "cat") and ("cat", "sat"). The model estimates the probability of a word occurring based on the previous word. If you encounter the word "cat," the bigram model would predict the likelihood of the next word being "sat" based on the frequency of the pair ("cat", "sat") in the training data.

III Trigram models (where $n = 3$) extend this idea to sequences of three items. For instance, in the same sentence "The cat sat," a trigram model would examine triples like ("The", "cat", "sat"). This model predicts the probability of a word occurring based on the two preceding words. So, given the sequence "The cat," the trigram model would predict the next word by considering the frequency of the triple ("The", "cat", "sat") and similar trigrams in the training data.

IV By capturing these patterns and dependencies, n-gram models provide valuable insights into the structure of language and are used in various applications such as text generation, speech recognition, and machine translation. The choice of n impacts the model's ability to capture context and the computational resources required, with larger n -grams generally providing more accurate predictions at the cost of increased complexity.

3.2 Rough sketch of the pipeline

1. First we add a start(<) and an end(>) token to the given word

   A _ _ L E $\rightarrow$ < A _ _ L E >

   Now instead of 5 characters we will be working with 7 characters. This step is done for all the words given in the training dictionary.

2. Now we employ the n-gram models one by one, iterating over the entire word to get all the **patterns** formed under that n-gram model. I have used **unigram, bi-gram, up to six-gram models** for calculations.

   Let's look at the tri-gram model to see what kind of **patterns** will be formed. We start from the first character,using a sliding window of size 3, moving until our window reaches the last character.

   < A _ $\rightarrow$ 1 blank
   A _ _ $\rightarrow$ 2 blanks
   _ _ L $\rightarrow$ 2 blanks
   _ L E $\rightarrow$ 1 blank
   L E > $\rightarrow$ 0 blanks

   We apply this process to all n-gram models from 1 to 6. For each model, we analyze all the words in the training dictionary to identify the patterns being formed and record the frequency count of each pattern. These frequency counts are then used to calculate the probability scores in the following steps.

3. Once we have the patterns, we start calculating a probability distribution over all the alphabets for just that pattern, for that particular n-gram model. A given pattern can have varying number of blanks. If it has no blanks, no calculations are necessary. **I am only using patterns that contain 1, 2 or 3 blanks**; rest are ignored Now let's examine how probabilities are calculated for different number of blanks.

   I. **1 Blank** : Say our pattern is $\boxed{L_1\ L_2\ \_}$, Here L_1 and L_2 can be any two letters. Let's calculate the probability distribution according to this pattern.

   Letters which already have been guessed are given a probability of zero. For all letters $l_3 \in L$ (Here L includes all the letters which have not been guessed till now) :

$$P(l_3 \in \boxed{L_1\ L_2\ \_}) = P(\boxed{L_1\ L_2\ l_3} \mid \boxed{L_1\ L_2\ \_}) = \frac{N(\boxed{L_1\ L_2\ l_3})}{N(\boxed{L_1\ L_2\ \_})}$$

$$N(\boxed{L_1\ L_2\ \_}) = \sum_{l \in L} N(\boxed{L_1\ L_2\ l})$$

   Here $N(.)$ denotes the count of the given pattern, calculated using the given training words.

   II. **2 Blanks** : Say our pattern is $\boxed{L_1\ \_\ \_}$, Here L_1 can be any letter. Let's calculate the probability distribution according to this pattern.

   Letters which already have been guessed are given a probability of zero. For all the letters $l_2 \in L$ (Here L includes all the letters which have not been guessed till now) :

$$
\begin{aligned}
P(l_2 \in \boxed{L_1 \; \_ \; \_}) &= P(\boxed{L_1 \; l_2 \; \_} \cup \boxed{L_1 \; \_ \; l_2} \mid \boxed{L_1 \; \_ \; \_}) \\
&= P(\boxed{L_1 \; l_2 \; \_} \mid \boxed{L_1 \; \_ \; \_}) + P(\boxed{L_1 \; \_ \; l_2} \mid \boxed{L_1 \; \_ \; \_}) - P(\boxed{L_1 \; l_2 \; \_} \cap \boxed{L_1 \; \_ \; l_2} \mid \boxed{L_1 \; \_ \; \_}) \\
&= P(\boxed{L_1 \; l_2 \; \_} \mid \boxed{L_1 \; \_ \; \_}) + P(\boxed{L_1 \; \_ \; l_2} \mid \boxed{L_1 \; \_ \; \_}) - P(\boxed{L_1 \; l_2 \; l_2} \mid \boxed{L_1 \; \_ \; \_}) \\
&= \frac{N(\boxed{L_1 \; l_2 \; \_}) + N(\boxed{L_1 \; \_ \; l_2}) - N(\boxed{L_1 \; l_2 \; l_2})}{N(\boxed{L_1 \; \_ \; \_})}
\end{aligned}
$$

$$
\begin{aligned}
N(\boxed{L_1 \; l_2 \; \_}) &= \sum_{l_3 \in L} N(\boxed{L_1 \; l_2 \; l_3}) \\
N(\boxed{L_1 \; \_ \; \_}) &= \sum_{l_2 \in L} \sum_{l_3 \in L} N(\boxed{L_1 \; l_2 \; l_3})
\end{aligned}
$$

Here $N(.)$ denotes the count of the given pattern, calculated using the given training words.

III. **3 Blanks** : Say our pattern is $\boxed{L_1 \; \_ \; \_ \; \_}$. Here L_1 can be any letter. Let's calculate the probability distribution according to this pattern.

Letters which already have been guessed are given a probability of zero. For all the letters $l_2 \in L$ (Here L includes all the letters which have not been guessed till now) :

$$
P(l_2 \in \boxed{L_1 \; \_ \; \_ \; \_}) = P(\boxed{L_1 \; l_2 \; \_ \; \_} \cup \boxed{L_1 \; \_ \; l_2 \; \_} \cup \boxed{L_1 \; \_ \; \_ \; l_2} \mid \boxed{L_1 \; \_ \; \_ \; \_})
$$

$$
= \frac{N(\boxed{L_1 \; l_2 \; \_ \; \_}) + N(\boxed{L_1 \; \_ \; l_2 \; \_}) + N(\boxed{L_1 \; \_ \; \_ \; l_2}) - N(\boxed{L_1 \; l_2 \; l_2 \; \_}) - N(\boxed{L_1 \; l_2 \; \_ \; l_2}) - N(\boxed{L_1 \; \_ \; l_2 \; l_2}) + N(\boxed{L_1 \; l_2 \; l_2 \; l_2})}{N\boxed{L_1 \; \_ \; \_ \; \_}}
$$

$$
N(\boxed{L_1 \; \_ \; \_ \; \_}) = \sum_{l_2 \in L} \sum_{l_3 \in L} \sum_{l_4 \in L} N(\boxed{L_1 \; l_2 \; l_3 \; l_4})
$$

Here $N(.)$ denotes the count of the given pattern, calculated using the given training words.

All of these calculations are performed in the same way for each n-gram model. For a given n-gram model and its associated pattern, we now have the probability distribution over all the letters.

4. Next, we sum the probability distributions obtained from all the patterns within a given n-gram model to compute the final scores for all the letters in that model. Now that we have the scores for each letter across all the n-gram models, we compute a weighted sum of these scores to obtain the final scores, which will be used for prediction.

   For $l \in L$ (Here, L is the set of all letters) :

$$
f(l) = \sum_{i=1}^{6} i^2 \cdot f_i(l),
$$

   Here $f_i(l)$ is the score for letter l obtained under the i-gram model.

5. To obtain the guess letter we simply take an argument maximum over f

$$
\text{guess\_letter} = \arg\max_{l \in L} f(l)
$$

4 Final Result

I employed the method described above for the final evaluation. Consequently, I achieved an accuracy of **70.6%**, surpassing the 50% target.