CS771 : Assignment 2

1. Rahul Rustagi (200756)
Department of Aerospace Engineering
rrustagi20@iitk.ac.in

2. Arnav Shendurnikar (200929)
Department of Chemical Engineering
sarnav20@iitk.ac.in

3. Rachit Khamsera (200747)
Department of Economics Engineering
rachit20@iitk.ac.in

4. Arihant Jha (200181)
Department of Aerospace Engineering
arihantj20@iitk.ac.in

5. P Madhav (200657)
Department of Chemical Engineering
pmadhav20@iitk.ac.in

6. Siddesh Bharat Hazare (200976)
Department of Aeropsace Engineering
siddeshbh20@iitk.ac.in

Team Name : **Melbo's Guys**

This Document contains exhaustive explanation of the functions added in submit.py file by us. We have included performance parameters and demonstrated an example alongside to depict the working of our code.

INDEX

Preprint. Under review.

Solution

We are given a dictionary to guess the word from.
The secret word is also from the dictionary.
We follow the below pipeline for training and testing hangman model.

- Initially we are provided the number of blanks in the secret word.
- We do not divide the dictionary. The idea we chose was to create and maintain a new array containing of the indices from 0 to N-1, each linking to corresponding word in the dictionary.
- As the game is started, we know about the number of letters in the secret word.
- The indices_array is divided into 2 arrays, one containing all the indices (in order) containing of all the words of length equal to the number of letters in secret word. The other array contains of rest of the indices.
- Suppose the length of the secret word is n. Now we know the indices of all the n-letter words that are present in the dictionary.
- We make the best guess out of the new truncated dictionary.
- Now there will be some letters that match and some that do not
- We update the indices array to a new array that contain words from that satisfy any subset of the current constraint condition. NOTE: Current index_words array contain words from the dictionary that satisfy all the conditions that were imposed by the all the nodes before the current node (root to the current node) by the Melbot during execution of the game.
- This process is repeated until a leaf node is reached. NOTE: It is clear that we reach nearer to answer when more letters of the secret word are revealed in single guess. However it should be noted that even when none of the letter is matched in a guess that is also qualifies our guess to be quite good. As now we have less words to guess from. Look in terms of information gain.

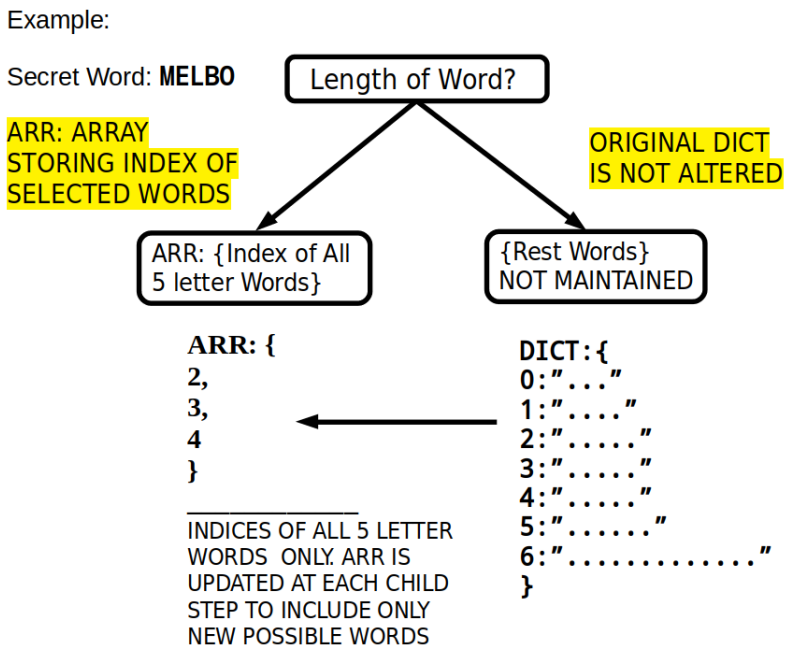

Figure 1: Initial Breakdown of Tree Before the First Query

Algorithm for Best Guess

1. Maintain a dummy variable best_word_index that stores the index of the word with the minimum entropy which will give maximum information gain.
2. The minimum entropy is calculated using

$$S = \sum_{i}^{N_c} p_i \log p_i$$

   Here N_c are the number of classes into which a word is divided for calculating its entropy.
3. We pick a word from the current dictionary and then check for all possible matching patterns that the word can generate in the given dictionary if it is used as our guess. Let the number of words having a common pattern be N_{ci}. Let N be the total number of words at the current node. Then p_i is calculated as follows,

$$p_i = \frac{N_{ci}}{N}$$

4. We calculate the entropy for that word by calculating p_i for all possible matching patterns

$$S = \sum_{i}^{N_c} p_i \log p_i$$

   Example: Melbot initially provides us with the number of letters in the secret word. Let the word we choose be 'ugly' then we calculate the set of words having all patterns in the dictionary: 'u_ _ _', 'ug__','ugl_','ugly','_g__'...etc We then calculate the number of words having the above patterns and divide it by total number of words in the current dictionary to get p_i
5. We repeat the above process and calculate entropy for all the words present in the dictionary at the current node. We then select the word having maximum entropy out of the current dictionary.

```
# Function get_entropy takes in the parameter 'counts' and calculates the
    entropy of the corpus 'counts'
def get_entropy(self,counts):

   # Calculate the total number of elements in 'counts'
   num_elements = counts.sum()

   if num_elements <=1:
      # print(f"warning")
      return 0

   # Calculate the proportion of each element in 'counts'
   proportions = counts / num_elements

   # Calculate the entropy of the distribution
   return np.sum( proportions * np.log2( counts ) )
```

6. Guess the word and let MelBot spit out the letters that match.

Splitting Method:

1. After obtaining the best possible query word, we query it and Melbot returns all the common letters (if there are in the correct position) between the guess word and query
2. We include only those indices from the current dictionary for which the condition(words corresponding to the index have same common letters in the same position as the secret word revealed after the query was passed) thereby splitting the dictionary in smaller one

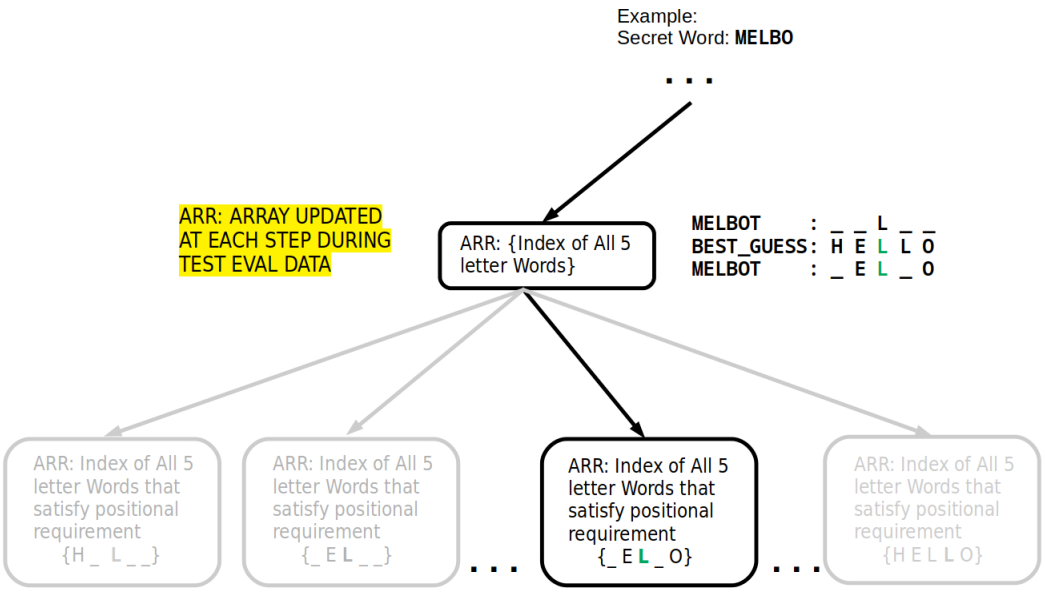

Figure 2: Progression of Decision Tree

3. We name the node as a leaf when the size of this list containing the indices to the dictionary is of length of 1, which means that only one word satisfying all the condition could reach that node
4. Hence, that must be the secret word!

Find the implemented code below :

```
# Function best_word_index that takes in the parameters words and index and outputs the best guessed word's index
def best_word_index(self, words, ind):

   # Initialize the best_entropy variable to infinity
   best_entropy = np.inf

   # Initialize an empty dictionary called best_split_dict
   best_split_dict = {}

   # Initialize the best_guess_ind variable to zero
   best_guess_ind = 0

   # Initialize an empty string called best_guess
   best_guess = ''

   # Iterate through each index i in the ind list
   for i in ind:

      # Call the partition function to split the words into two groups based on the word at index i
      entropy, split_dict = self.partition(words, ind, words[i])

      # If the entropy of the split is lower than the best entropy seen so far
      if entropy < best_entropy:

         # Update the best_entropy, best_split_dict, best_guess, and best_guess_ind variables
         best_entropy = entropy
         best_split_dict = split_dict
         best_guess = words[i]
         best_guess_ind = i

   # Return the index of the best guess word
   return best_guess_ind
```

Performance Parameters

- **Average training time** : 12.24 secs
- **Model size** : 1184075 bytes
- **Win percentage** : 100
- **Average number of queries asked** : 5.1449

Other Possible solution:

1. We pick a word from the current dictionary
2. We then calculate the probability of each of its letter to occur at that position by calculating the number of times the letter occurs in that position in all the words in the dictionary and then dividing it by number of words in the current dictionary
3. We then calculate the probability of that word by the following formula, since every index is independent Example: if the word chosen is 'ability', then

$$P[x_1 =' a', x_2 =' b', x_3 =' i', x_4 =' l', x_5 =' i', x_6 =' t', x_7 =' y'] = \prod P[x_i = ch_i]$$

4. We repeat the above process for all the words in the current dictionary and calculate the probability of each word
5. We select the word having maximum probability to be our guess word for this node/round.