CS771 Assignment 2
Team 82 - Syntax_Error

Team Members
Aryan Jain - 200201
Ritik Raj - 200803
Shivang Pandey - 200941
Varun Maghnani - 201092
Suhani - 201011
Kashishpreet Kaur - 200496

Details of the decision tree algorithm

We have solved the problem using two approaches. The general node and tree structure used is given below. Modifications will be made in these structures depending on the approach. These will be discussed in the later part.

Node structure

A node stores its own depth (root = depth 0), a link to its parent and a link to all the words as well as the words that reached that node A dictionary is used to store the children of a non-leaf node. Each child is paired with the response that selects that child.

Each node implements a **get_query** method that generates a query to ask when we reach that node.

For the root node, that query is simply an empty string. For every non-leaf node, **get_child** method is implemented that takes a response and hence, selects one of the children to be asked as a query. For leaf node, this query is usually the final answer.

We create a reveal function that compares Melbo's query with the secret word and returns the response by Melbot that reveals the common alphabets and has dashes at rest of the locations.

Tree structure

The tree structure stores the information about its root, words included, minimum possible leaf size and maximum depth. The root is trained with all the words and outputs sub-trees depending on just the number of characters.

1 APPROACH-1

1.1 Criterion to choose the splitting criterion at each internal node

At root node, we split it on the basis of number of characters in the words provided in dictionary.

For each subsequent node, we find the the word having most distinct characters and apply the **reveal function** for all the words in the node with that word. Every distinct mask obtained is then added to the **split_dict**. Each element contained in the **split_dict** is a child of our node.

For this, we add another information to the node structure which is **self.char_count** array which stores the no. of characters of each of the words in that node.

Preprint. Under review.

Function **count_uni_char** is defined that takes all words in the dictionary and a list of indexes of the words in the current node. It finds the no. of unique characters in each of the words in that node and appends that number to the **char_count** array of that node.

Function **chose_ind** is defined for each node that returns the index of the word in that node that has the maximum no. of unique characters.

1.2 Criterion to decide when to stop expanding the decision tree and make the node a leaf

When at the node, the number of words left(**my_words_index**) <=1(**min_leaf_size**) or the number of queries exceed 15 (self.depth >= max_depth), we make it a leaf node.

Case 1: If the number of words left = 1, then that is the final answer.

Case 2: if the number of queries exceed 15, then we return the word which has most unique characters among all.

1.3 Results on dummy dataset

Time taken = 0.11880620516000363
Size = 893423.0
Win accuracy = 1.0
No. of queries = 5.083607509192962

2 APPROACH-2 (The final approach)

2.1 Criterion to choose the splitting criterion at each internal node

We use a probabilistic approach in this method. The code submitted is of this approach.

At root node, we split it on the basis of number of characters in the words provided in dictionary.

For each subsequent node, we find the the word having the maximum occuring probability and apply the **reveal function** for all the words in the node with that word. Every distinct mask obtained is then added to the **split_dict**. Each element contained in the **split_dict** is a child of our node.

For this, we add another information to the node structure which is **self.word_prob** array which stores the no. of characters of each of the words in that node.

Function **find_prob** is defined that takes all words in the dictionary and a list of indexes of the words in the current node. It finds probability of each of the words in that node and appends that number to the **word_prob** array of that node.

Function **chose_prob_ind** is defined for each node that returns the index of the word in that node that has the maximum no. of unique characters.

2.2 Method of finding the probability

Do note that here by probability of a word we just mean it qualitatively and may not be between 0-1. Its just a number assigned to each word. Higher this number, higher is the probability (chances that that word is the secret word).

Steps to find the probability:

- At starting of code itsef, while defining class Tree, we run the function **find_prob** and calculate the probability of each word in the dictionary all at once and store it in a dictionary **dic**.
- Every unique alphabet is a key of the dictionary and each key corresponds to an array of size 15 (which is the maximum length a word can have).

$$Every\ element\ in\ this\ array = \frac{no.\ of\ words\ that\ have\ that\ character\ at\ that\ particular\ index}{no.\ of\ words}$$

Thus the dictionary is basically a matrix of size no. of unique alphabets * 15.

- Next, we calculate probability of each word using its characters.
  $Probability\ of\ a\ word = \sum_{i=0}^{n-1} word[i] * dic[word[i]][i]$ where
  n = length of word
  word[i] = alphabet at ith index of the word
  dic = The dictionary that was made earlier
- We call the function **chose_prob_ind** for each node to get the word with maximium probability.
- Send this word as the next query.

2.3 Criterion to decide when to stop expanding the decision tree and make the node a leaf

When at the node, the number of words left(**my_words_index**) <=1(**min_leaf_size**) or the number of queries exceed 15 (self.depth >= max_depth), we make it a leaf node.

Case 1: If the number of words left = 1, then that is the final answer.

Case 2: if the number of queries exceed 15, then we return the word which has the highest probability among all.

2.4 Results on dummy dataset

Time taken = 0.10334479763000673
Size = 878723.0
Win accuracy = 1.0
No. of queries = 4.461583123669429

References

[1] www.geeksforgeeks.org

[2] www.stackoverflow.com