Assignment-2 CS771: Intro to ML

Group Name: **BOYS**
Group Number: **21**
Date: **05-04-2023**

1 Give detailed calculations explaining the various design decisions you took to develop your decision tree algorithm. This includes the criterion to choose the splitting criterion at each internal node (which essentially decides the query word that Melbo asks when that node is reached), the criterion to decide when to stop expanding the decision tree and make the node a leaf, any pruning strategies and hyper-parameters etc.

We attempted several approaches initially and eventually arrived at the ultimate solution by incorporating lessons learned from them and optimizing the model to achieve the best possible outcome, while considering potential trade-offs.
The attempts are as follows:

1.1 Random Selection

A random selection approach was employed where no specific splitting criterion was selected, and the query at each node was generated randomly from the available words in that node. Once a query was randomly chosen, various masks (query, other_word) were created using the query and other words present in the dataset, and their corresponding frequencies were recorded in the split_dict (mask$\longrightarrow$frequency) map. This map stored the frequency of each mask formed by combining the query and other words, allowing for further analysis and decision-making during the tree-building process. The random selection approach is often utilized to prevent bias and ensure that all possible splitting criteria are considered during the decision tree construction.

1.2 Word with maximum matches

In order to determine the most suitable query word, we compared each word in the dataset to all other words and selected the word with the highest number of shared places as the query. After selecting the query word, we generated a map that recorded the number of shared places between the query word and each other word in the dataset.
To further facilitate the data splitting process, we created different buckets of words based on the percentage of shared places. For instance, words with 0-5%shared places were placed in one bucket, while those with 5-10% shared places were placed in another. This approach enabled us to categorize the words into distinct groups, which could be further processed to achieve the desired output.
Overall, the process involved multiple steps aimed at optimizing the selection and categorization of words to improve the accuracy and efficiency of the data splitting algorithm

1.3 Variance Minimization

The variance measures an average of distances of outcomes of the probability distribution from mean. Here for each split, we individually calculate the variance of each child node taking into consideration the frequency of all the different masks that can be formed at this node. Calculating the variance of each split as the weighted average variance of child nodes. Select the split with the lowest variance. And then we repeat the above steps till we do not achieve homogeneous nodes.

Preprint. Under review.

1.4 Entropy Minimization

We calculated the entropy of all possible masks created by (query, other_word) pair taking into consideration each word once for the query, however this can be manipulated by choosing the minimum entropy from first 'k' masks as well (we can choose k), this will decrease the training time while increasing the average queries.

Entropy is calculated as:

$$H(x) = -\sum_{i=1}^{n} p_i log_2 p_i$$

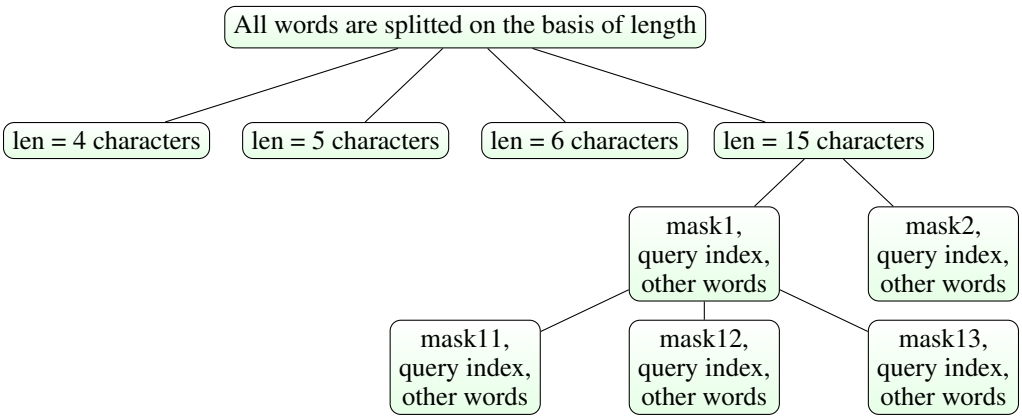
where, p_i is the proportion of number of the elements in each of the masks created for that particular word.

Then the word whose masks gave the minimum entropy was selected as the query and its split_dict which is a (mask⟶frequency) map was chosen as the children for this particular node where we are at now.
We will explain more on this using this diagram below:

```
def process_node( self, all_words, my_words_idx, verbose ):
    if self.depth == 0:
        query_idx = -1
        query = ""
        ( entropy, split_dict ) = self.try_attr(query, all_words, my_words_idx)
        return (query_idx, split_dict)

    best_entropy = np.inf
    best_attr = 0
    best_split_dict = None

    for c in my_words_idx:
        ( entropy, split_dict ) = self.try_attr( all_words[c], all_words, my_words_idx )
        if entropy < best_entropy:
            best_entropy = entropy
            best_attr = c
            best_split_dict = split_dict
    return ( best_attr, best_split_dict )
```



We first split our root node on the basis of length of words. Then creating masks assuming each word as a query, and minimizing the entropy of mask frequency, we got out query word selected for this node.
Our Tree will stop expanding if the leaf node only contains one word or the depth is greater than 15.

Pruning the decision tree can be done on the basis of purity of a node. Once a word as query gives entropy less than 10%, that particular word can be used to purify and prune the tree at it's child nodes. This method will improve model training time largely, but affects the win ratio to go from 100% to 93.84%.

Hyperparameters:

To further optimize the data splitting process, we decided to consider not just the word with the highest number of shared places as the query, but also the first 10 words in the dataset. In cases where there were fewer than 10 words, we used all available words as queries. This modification significantly reduced the training time of the algorithm to approximately 0.51 seconds, with an average of 4.41 queries per dataset.

Moreover, we conducted additional experiments where we used only the first word in the dataset as the query, instead of selecting the word with the lowest entropy. This approach further reduced the training time to 0.14 seconds, but increased the average number of queries per dataset to nearly 5.14.

Overall, these modifications aimed to enhance the efficiency of the data splitting algorithm by considering multiple query options and reducing the time required for training. The trade-offs between training time and the number of queries per dataset were carefully evaluated to ensure optimal performance.