# Assignment 2

# Team Gryffindor

Yashika Malhotra (201152)
Utkarsh Kandi (201068)
Prem Milind Gujrathi (200393)
Piwal Abhishek Satish (200684)
Vedant Sanjay Gitte(201104)
Sayan Deori(200909)

## 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), criterion to decide when to stop expanding the decision tree and make the node a leaf, any pruning strategies and hyperparameters etc

**The Splitting Criterion** used at each node was based on picking such a query id from the *my_words_idx* that gives maximum number of unique interestions with the list *my_words_idx*. for that we iterate over complete list of *my_words_idx* , for each iteration of word in list *my_words_idx* we calculate the number of unique interestions for that word successively we kept track of the id of the word that gave maximum number of unique interestions till this iterations doing so at the last we got id (stored in *max_id* in ***get_optimized_query*** function word that gives maximum number of unique interestions. we select that word corresponding to the ***max_id*** as the query at this node,then selection of each child of that node was done on the basis of words in *my_words_idx* that has same interestion with query. doing in this way leads to choosing the query that maximizes the number of children at that depth (number of children at each node will be equal to unique interestions with selected query) and decision trees has more possibilities at each depth which leads to lesser queries to guess each word in *my_words_idx*.

**Criterion to decide when to make node a leaf:**

- If the *minimum_leaf_count* value of 1 is reached or the maximum depth of 15 is reached, the node is made a leaf.
- We kept track of the letters revealed till now, using the *get_intersection* function, which made use of the responses of the history. If at any point, all the letters of the secret word get revealed, the node is converted into a leaf, and the query corresponding to the secret word is returned.

Different pruning strategies and algorithms were experimented with to develop the decision tree model, which are as follows:

- **ID3 Algorithm**: This involved iterating through the words coming from *my_words_idx*, and choosing the query word that maximizes the information gain. For a particular query, splitting is performed by masking the query with words coming from *my_words_idx* and creating a child for every unique intersection. Information gain for a query is calculated in the following manner:

Let N be the length of *my_words_idx*, and $n_i$ be the number of words contained in the $i_{th}$ child, which is the length of the indices list in the $i_{th}$ item of split dictionary. Also, the relative frequency distribution for all the words was found to be exactly same, making them equally likely to occur

$$H(s) = \log_2 N$$
$$H(i) = \log_2 n_i$$
$$IG = H(s) - \sum_{i=1}^{n} \frac{n_i}{N} * H(i)$$
$$\Rightarrow IG = \log_2 N - \sum_{i=1}^{n} \frac{n_i}{N} * log_2 n_i$$

- **Maximising the number of children at each depth**: The splitting criterion involved choosing the query that maximizes the number of children at that depth. By increasing the number of children at each depth, the decision tree can explore more possibilities at each step. This can lead to a more efficient search for the target word and reduce the number of guesses required.

- **Sampling**: The splitting criterion involved was based on sampling the entire set of words using the np.random.choice() function, and choosing the optimized query from the randomly sampled words. The query chosen was the one that gave the maximum information gain. Training a model on all possible words was found to be computationally expensive and time-consuming. By sampling words, the training process can be made more efficient while still providing sufficient coverage of the search space.

- **C4.5 Algorithm**: This algorithm uses information gain ratio as the splitting criterion, which is the ratio of information gain to the intrinsic information of a split. The optimal query is the one that maximizes the information gain ratio, which is calculated as follows:

$$Information\_split = -\sum_{i=1}^{n} \frac{n_i}{N} * log_2 \frac{n_i}{N}$$
$$IG\_ratio = \frac{IG}{Information\_split}$$

- **Gini Index**: The splitting criterion involves choosing the query corresponding to the minimum Gini index. A low Gini index indicates that the set is relatively pure, while a high Gini index indicates that the set is relatively impure. Ginni index corresponding to a particular query can be calculated as follows:

$$Gini(n_i) = 1 - [(\frac{1}{n_i})^2 + (\frac{1}{n_i})^2 + ...]$$
$$\Rightarrow Gini(n_i) = 1 - \frac{1}{n_i}$$
$$Weighted\_Gini = \sum_{i=1}^{n} \frac{n_i}{N} * (1 - \frac{1}{n_i})$$

The table shows the training parameters obtained through varying algorithms and pruning strategies:

| Parameters | | | | |
|---|---|---|---|---|
| **Methods** | **t train** | **m size** | **win** | **query** |
| **Maximizing children at each depth** | 8.43629 | 894711.0 | 1.0000 | 4.087865 |
| **ID3 Algorithm** | 11.80181 | 1050969.0 | 1.0000 | 3.999226 |
| **Gini Index** | 12.42283 | 1047334.0 | 1.0000 | 4.110122 |
| **C4.5** | 20.40533 | 1144797.0 | 0.9984 | 5.520805 |
| **Sampling** | 30.95517 | 1096869.4 | 1.0000 | 4.404916 |

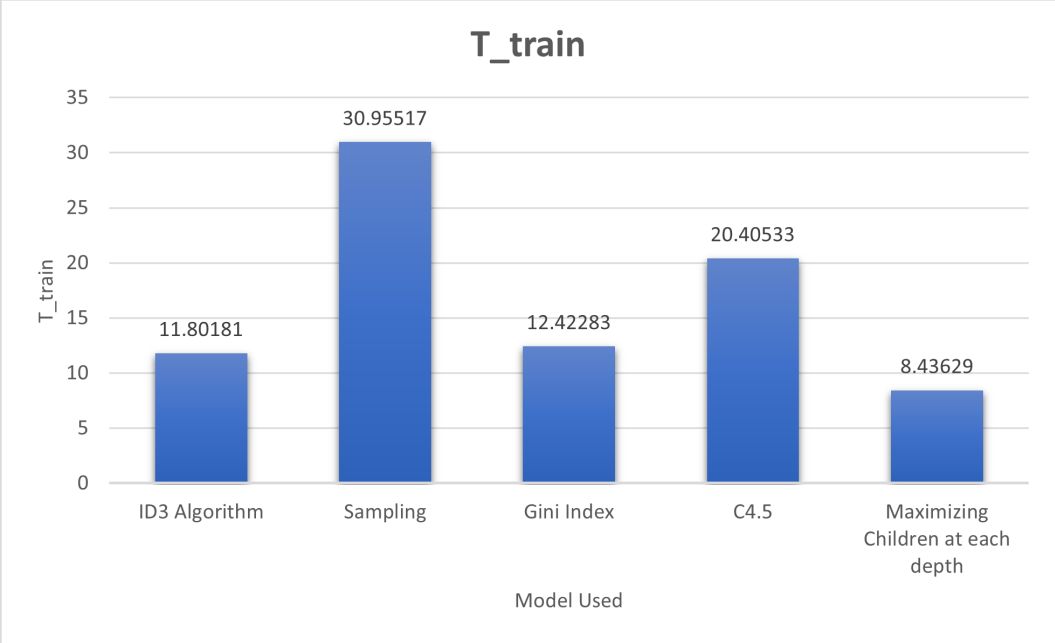


Figure 1: Training time of different models

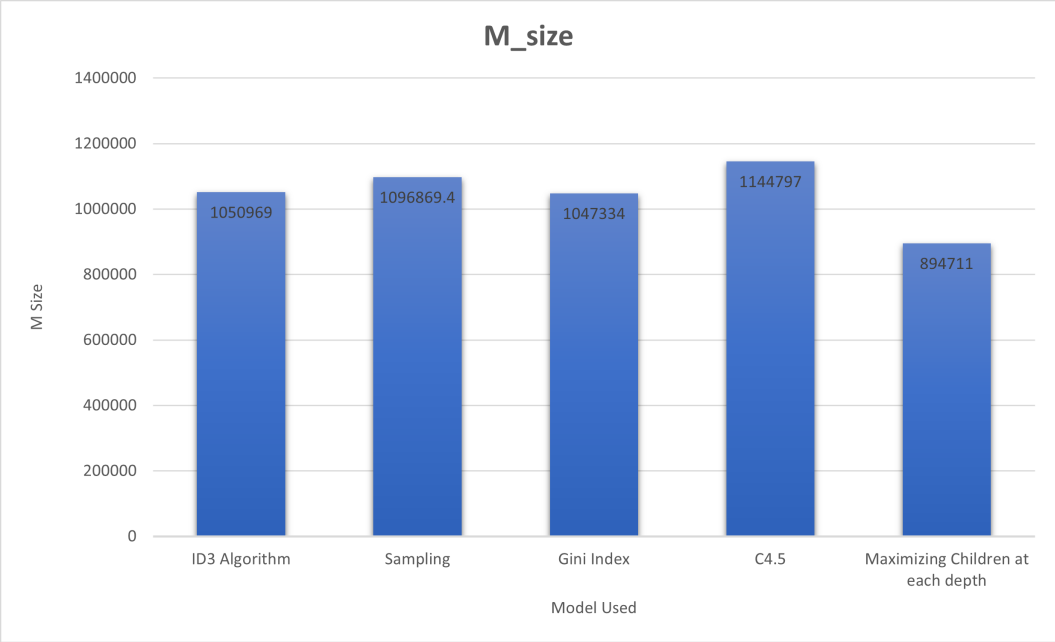


Figure 2: Model size information

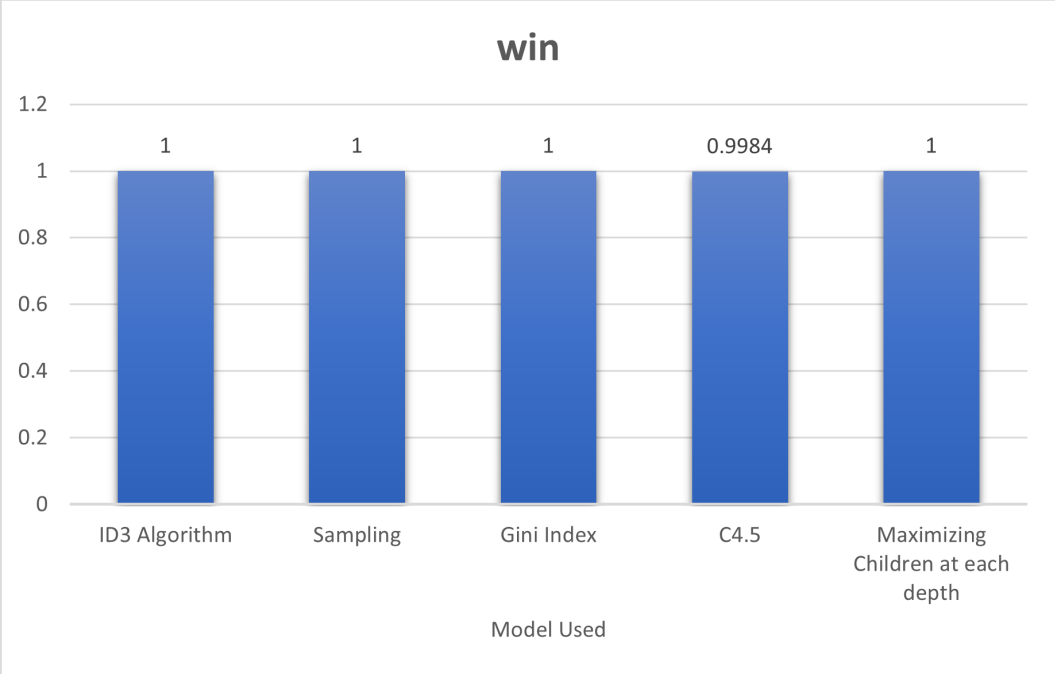


Figure 3: Accuracy of different models

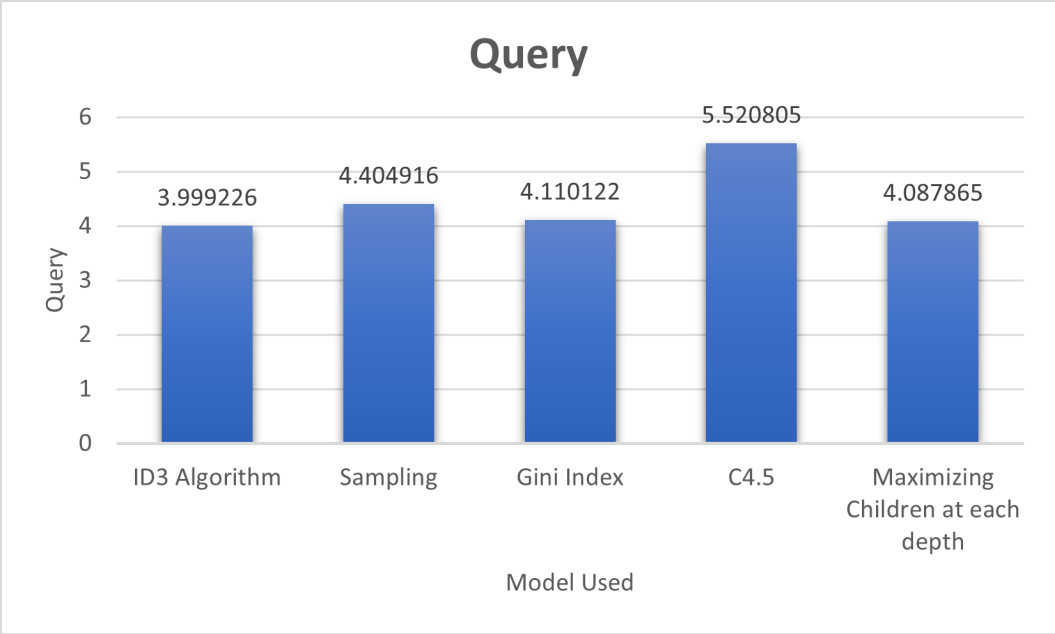


Figure 4: Average query count of different models

We tried solving the given problem by every method as mentioned above. We got the most optimized model by **Maximizing Children at each depth** as shown in the table. The time required to train the final model for the same model was **8.436289391s**. Model size was **894.711 kb**. Average number of queries was **4.087865299.**