CS771 : Introduction to Machine Learning
Assignment - 2
Team: Syntax Killer

Dhruv Varshney
220366
Dept. of Electrical Engineering
IIT Kanpur
vdhruv22@iitk.ac.in

Sarthak Paswan
220976
Dept. of Computer Science and Engineering
IIT Kanpur
sarthakp22@iitk.ac.in

Ansh Jain
220166
Dept. of Civil Engineering
IIT Kanpur
anshjain22@iitk.ac.in

Shyam Sundar
221049
Dept. of Mathematics and Scientific Computing
IIT Kanpur
shyams22@iitk.ac.in

Rudraksh Pal Singh
220920
Dept. of Mechanical Engineering
IIT Kanpur
rudrakshps22@iitk.ac.in

Daniel Siwakoti
231040127
Dept. of Electrical Engineering
IIT Kanpur
daniels23@iitk.ac.in

Abstract

This is our solution to the second Assignment of the course CS771. We are required to create a ML model to predict the words given its depulicated, lexicographically sorted first five bigrams. If we choose decision tree as our ML model then we have to show the detailed calculations explaining the various design decisions like splitting criteria, when to make a particular node a leaf node, pruning strategies, hyperparameters etc. to develop our ML algorithm.

1 Our ML Model

We approached this problem using a Decision Tree Classifier. We started with all the words given in the dictionary and their processed bigrams at the root node. We developed a node splitting criterion to split nodes based on a property that all the tuples of bigrams hold. Finally, to make a node a leaf, we applied certain conditions to that node.

1.1 Data Preprocessing

We extracted all the processed bigrams from the dictionary of 5167 words provided to us. Then, using those words as keys, we mapped all the corresponding processed tuples of bigrams and stored them in a dictionary. Then passed that dictionary to the root node of the decision tree.

1.2 Node Splitting Criteria

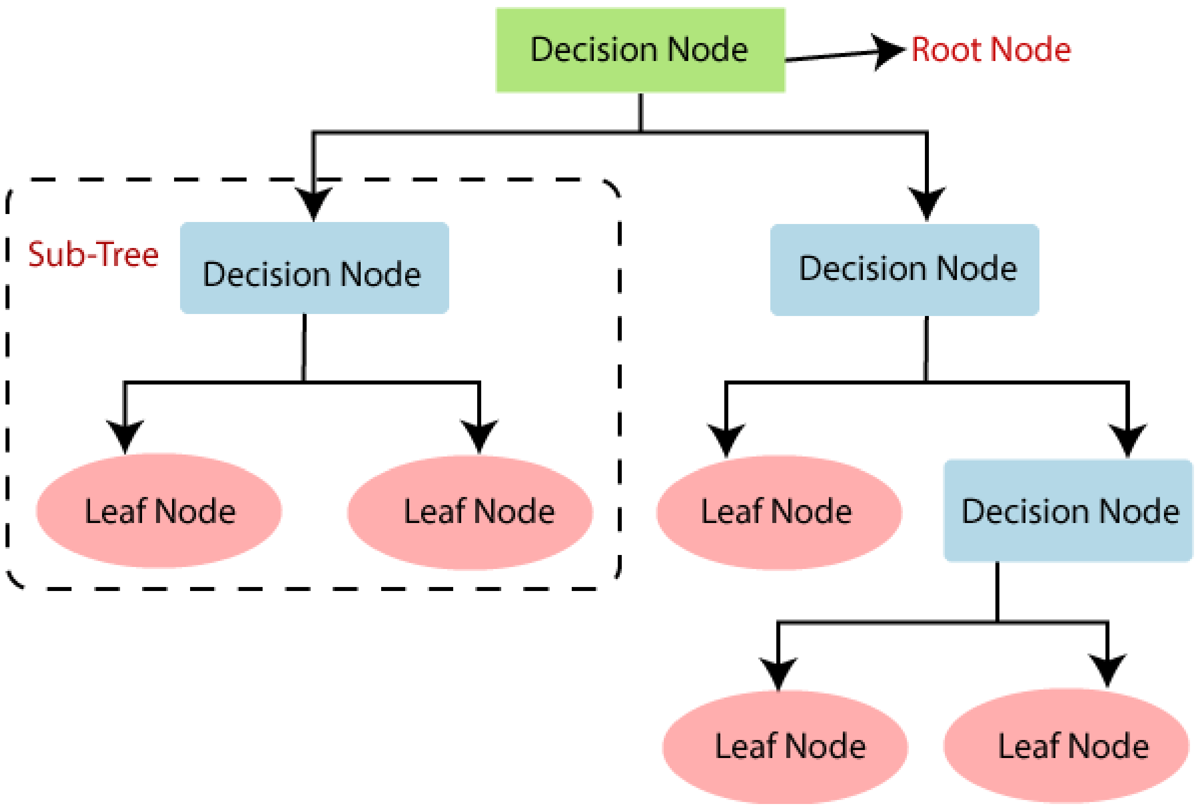

Figure 1: A simple Decision Tree

At any particular node, we first find the most frequent bigram that occurs in the words. Using that bigram as the decision criterion, we split the words: those containing the particular bigram go to the left subtree, and the remaining words go to the right subtree. Throughout this process, we also maintain a set containing all the bigrams we have used to split the nodes, ensuring that each bigram is used only once for node splitting.

In the left subtree, we recursively find the most frequent bigram that is not present in the used bigram set for the left subtree and build the tree recursively. The same process applies to the right subtree.

1.3 Criteria to make Node a Leaf Node

We pass a bigram dictionary to each node, which diminishes in size as we split. When its length reaches one, it signifies that further splitting isn't feasible, and the node becomes pure, hence making it a leaf node. Additionally, there's a hyperparameter criterion: if the maximum tree depth reaches a user-defined value, all nodes at that level are declared as leaf nodes.

2 Code for our Decision Tree Model

Code Submitted

3 Results

3.1 Training Set

The model is trained and tested on the public dictionary of words `'dict'` with different hyperparameters.

3.2 Model Performance Results

The performance of the implemented model was evaluated by the validation code on various metrics, including training time, model size, precision, and test time. The best results we obtained are as follows:

- **Training Time:** 0.2113 seconds
- **Model Size:** 324,504 bytes
- **Precision:** 97.50%
- **Test Time:** 0.0598 seconds

These results indicate that the model is not only efficient in terms of training and prediction times but also maintains high precision, making it suitable for the given task.

3.3 Hyperparameter Tuning

The primary hyperparameter for the decision tree model used in this project is the **maximum depth of the tree**. This hyperparameter controls the maximum number of levels that the tree can have, which directly impacts the model's complexity and performance.

During the hyperparameter tuning process, we experimented with various values for the maximum depth. We observed that reducing the maximum depth resulted in a significant drop in precision due to the formation of leaf nodes too early, which caused the model to underfit the data.

We made the observations mentioned below for different values of the hyperparameter `max_depth`:

Table 1: Max Depth Hyperparameter

Max_depth	**Precision(%)**
15	32.94
30	81.65
50	94.45
75	97.29
100	97.50

Through testing, we found that the best precision was achieved when the maximum depth was set to **84** or more. Beyond this point, increasing the depth further did not result in any improvement in precision, indicating that the model had reached its optimal performance level. As a result, the final model was configured with a maximum depth of more than 84 to balance complexity and performance.