MACHINE LEARNING

Assignment - 1

Decision Tree

GROUP (GID: 53)
19CS10050 Rupinder Goyal
19CS30043 Shrinivas Khiste

Decision Tree

Decision Trees are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. A tree can be seen as a piecewise constant approximation. They are widely used for classification and regression problems.

Building a Decision Tree

We have built the decision tree using a top-down approach.

1. Start with the complete training data.
2. Iterate over all attributes and choose the best one (Let's say A) according to some criteria (discussed later). Split the dataset according to the different values of A.
3. Assign the node that particular attribute and also save its parent and whether it is terminal or not (used while printing the tree).
4. Iterate over all the split datasets and repeat the same steps for each.
5. This will go on recursively until either the max depth is reached or we have reached a terminal node such that all the data in it has the same class value.
6. Save all the children nodes of the parent too and also maintain the count of the tree nodes.

Best Attribute

We can decide the best attribute using two metrics

Information Gain

This is measured by the change in entropy upon splitting the data according to some attribute.

Entropy is a measure of uncertainty, purity and information content. The entropy of a subset of the dataset is defined as

$$\sum_{i=1}^{n} .p_i . \log_2(p_i)$$

Where n is the number of class values and pi is the proportion of examples of the ith class value.

We calculate this value before the split and after the split and take a weighted average of the entropy of the children as follows

$$\sum_{v \in Values(A)} .\frac{|S_v|}{|S|} .Entropy(S_v)$$

Where A is the attribute according to which we are splitting,|Sv| is the number of samples of that attribute value and |S| is the total number of samples before the split.

Information gain is defined as the difference of entropy of samples before the split and the above weighted average after the split.

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} .\frac{|S_v|}{|S|} .Entropy(S_v)$$

:

The values of this gain function vary between 0 and 1 and the higher the gain the better. 0 gain means positive and negative samples get split equally and a gain of 1 is a perfect split.

Gini Gain

This is another measure of impurity. It can be interpreted as the expected error rate. It is defined as :

$$GiniIndex = \sum_{i}^{n} (1 - p(i)^2)$$

Where pi is the proportion of examples of the ith class value and n is the number of class values.

The value of the Gini gain is the weighted sum of the Gini Index of datasets split according to the different values of the attribute value.

$$Gain(S, A) = \sum_{v \in Values(A)} . \frac{|S_v|}{|S|} . GiniIndex(S_v)$$

Here a lower gain is considered as better. The gain of 0 is for a perfect split and a gain of 1 is when the positive and negative classes get split equally.

Stepwise Explanation

Importing Libraries and loading data

We have used the following libraries in the project

1. **Pandas**: For handling the tabular data
2. **Numpy**: For handling data vectors
3. **Matplotlib**: for the graphs
4. **Sklearn train test split**: for splitting the data into train and test set
5. **Default Dict**: To create an empty dictionary with a default value of 0

To load the data we have first defined the column names as per the dataset and then loaded the data using the pandas read CSV function.

Gain Functions

In this section, we have defined the gain functions required to figure out the best attribute.

We have used the following functions:

Utility Functions

1. getClassProportions(dataset)

- Parameters:
    - Dataset: The data whose class proportions you want to calculate
- This function is used to generate an attribute of the node which stores how many samples are present for each class value in the dataset at that node.
- For this, we first get the unique values of the classes in the dataset and then iterate on these values and store their count in the form of a map
- Return value
    - The Map of class-wise proportions
    - Total number of samples

Gini Gain Functions

1. gini_index_for_child(dataset):
    - Parameters:
        - Dataset: The part of the data for which you want to calculate the Gini index value.
    - Here we iterate over all the unique class values and then calculate the proportion of sample of that class and calculate the Gini index according to the formula discussed before.
    - Returns
        - Calculated Gini Index
2. gini_index(dataset,column_name):
    - Parameters:
        - Database: The part of the data for which you want to calculate the Gini index value.
        - Column Name: The attribute according to which you want to calculate the Gini index
    - In this function, we calculate the Gini Index if the dataset is split according to the attribute column name.
    - We split the dataset according to different values of the attribute and calculate the Gini index and then take the weighted average of this.
    - Returns
        - Calculated Gini Index
        - Splits of the dataset according to the attribute value

3. bestSplitGiniIndex(dataset):
    - Parameters:
        - Datasets: The part of the data for which you want to calculate the best split according to the Gini Index.
    - Here we want to calculate the best split according to the Gini index.
    - We iterate over all the attributes, calculate the Gini index according to that attribute using the above function and then choose the attribute that has the least Gini Index.
    - We save the attribute, the best Gini index, best split, the class-wise composition and the total number of samples.
    - Returns
        - Decision Tree node having all the above attributes.

Information Gain Functions

1. entropy(dataset):
    - Parameters:
        - Dataset: The part of the data for which you want to calculate the entropy value.
    - Here we iterate over all the unique class values and then calculate the proportion of sample of that class and calculate the entropy according to the formula discussed before.
    - Returns
        - Calculated Entropy
2. information_gain(dataset,column_name):
    - Parameters:
        - Database: The part of the data for which you want to calculate the information gain
        - Column Name: The attribute according to which you want to calculate the information gain
    - In this function, we calculate the information gain if the dataset is split according to the attribute column name.
    - We start with the entropy of the complete dataset and subtract it with the weighted average of the entropy values of the split according to different attribute values.

- Returns
    - Calculated Info Gain
    - Splits of the dataset according to the attribute value

3. bestSplitInfoGain(dataset):
    - Parameters:
        - Datasets: The part of the data for which you want to calculate the best split according to the Information Gain.
    - Here we want to calculate the best split according to the information gain.
    - We iterate over all the attributes, calculate the information gain according to that attribute using the above function and then choose the attribute that has the most information gain.
    - We save the attribute, the best information gain, best split, the class-wise composition and the total number of samples.
    - Returns
        - Decision Tree node having all the above attributes.

Building the Tree

1. terminal_node(dataset,column value, parent)
    - Parameters
        - Dataset: The dataset left till that terminal node
        - ColumnValue: The column value of the attribute he was split according to
        - Parent: The parent of the node in the Decision Tree
    - Here we define a terminal node of the Decision Tree
    - It saves the column value, parent and class-wise composition and total samples of the dataset.
    - Its sub-tree size is defined as 1.
    - Its prediction is also stored, which is the class value with the maximum number of samples in this part of the dataset.
    - Returns
        - This terminal node
2. pure_leaf(data):
    - Parameters
        - Data: The part of the dataset of that node

- Checks whether the given node is a pure leaf. This is done by checking whether all the samples are of the same class value.
- Returns
    - Whether it is a pure leaf or not

3. split(data,depth, MaxDepth = None, criterion = "gini", columnValue = None, parent = None)
    - Parameters
        - Data: the part of the dataset that we have to split
        - Depth: current depth
        - Max Depth: nullable value that is till which depth should we limit the growth of the tree.
        - Criterion: Which criterion to use for choosing the best attribute. Default Gini.
        - Column Value: The value of the attribute according to which we have split
        - Parent: Parent of the current node.
    - This is a recursive function to build the decision tree using the top-down approach. First, we check whether it is a pure leaf or max depth is reached. Then we find the best split node and iterate over the best split and call split function on it. On returning, we maintain the tree size (number of nodes) and all the children nodes in the form of a map of attribute-value according to which it was split to the node.
    - Returns
        - Current node
        - Sub-Tree size
4. build_tree(root,MaxDepth, criterion)
    - Parameters
        - Data: The complete train data set
        - Max Depth: nullable max depth according to which we have to limit the tree.
        - Criterion: The criterion to use to select the best attribute
    - Here we call the split function on the complete dataset to generate the decision tree recursively using the top-down approach.
    - Returns

        - The root of the thus formed decision tree.

Print Functions

In this section, we have defined the function to print the decision tree

1. printTree(root, depth)
    - Parameters:
        - root -> root node of the subtree needs to be printed
        - depth -> stores the depth of the root node (default value 0)
    - This function prints the tree recursively.
    - printTree(root) is called from the main code to print the decision tree (default depth value 0)
    - It prints the information of the current node and calls the print function for the children (if there are any).
    - Depth is passed as a parameter to format the printed tree properly.

Prediction & Accuracy Functions

In this section, we have defined the functions to predict and find the accuracy.

1. predict(root, data):
    - Parameters:
        - root -> root of the decision tree
        - data -> data set for which prediction is required (dataframe)
    - For each data value, traverse the tree according to the attribute values for the data value, until the current node is leaf node or the required child is not there.
    - Class value having maximum frequency in the last visited node is assumed as the class value(output) for the data value.
    - Return Variables:
        - prediction -> Pandas Series, Same as the size of dataset (input), stores the predicted class value for each data value.

2. accuracy(root, data):
    - Parameters:
        - root-> root of the tree
        - data-> dataset on which accuracy needs to be calculated

- This function calls the ‘predict‘ function to find the predicted class values and then matches it with the actual output and finds the accuracy.
- Return Variables:
    - acc -> accuracy of the decision tree on the given dataset

Pruning Functions

1. getTerminalPars(root):
    - Parameter:
        - root -> root of the decision tree
    - This function traverses the root and stores the nodes whose children are terminal nodes.
    - Return Value:
        - Terminal parents: a list of the nodes stored.
2. pruneTreeStatisticalMethod (root):
    - Parameters:
        - root -> root of the tree to be pruned
    - This function follows the Chi-squared statistical method to prune the decision tree.
    - First, get the nodes whose children are terminal
    - Iterate on those nodes, calculate the chi-square value, compare it with the threshold and update the tree accordingly.
    - Repeat these steps again until no pruning occurs
    - Return Variable:
        - root -> root of the updated decision tree
3. getTerminals(root):
    - Parameters:
        - root -> root of the decision Tree
    - This function stores all the terminal nodes of the tree
    - Return:
        - terminals: a list storing all the terminal nodes of the tree
4. pruneTreeCV (root, data):
    - Parameters:
        - root -> root of the decision tree
        - data -> validation dataset

- This function performs the reduced error post pruning on the tree.
- Find the current accuracy.
- For each terminal node, Remove it, find the new accuracy and then connect it again
- Among all the trees, keep the one with the highest accuracy.
- Repeat until no pruning occurs
- Return Values:
    - root -> root of the updated tree
    - curr_acc -> accuracy of the current Tree

Question Wise Explanation

Question 2

We tested the performance across 10 random 80/20 splits and the results are as follows:

1. Gini Index

```
Iteration : 1 Accuracy: 0.9104046242774566
Iteration : 2 Accuracy: 0.9248554913294798
Iteration : 3 Accuracy: 0.9364161849710982
Iteration : 4 Accuracy: 0.9161849710982659
Iteration : 5 Accuracy: 0.9219653179190751
Iteration : 6 Accuracy: 0.9335260115606936
Iteration : 7 Accuracy: 0.9104046242774566
Iteration : 8 Accuracy: 0.9335260115606936
Iteration : 9 Accuracy: 0.8988439306358381
Iteration : 10 Accuracy: 0.9075144508670521

Average Accuracy: 0.919364161849711
Best Accuracy: 0.9364161849710982
```

   Here we can see that the best accuracy is 93.64% and the average accuracy is 91.94%.

2. Information Gain

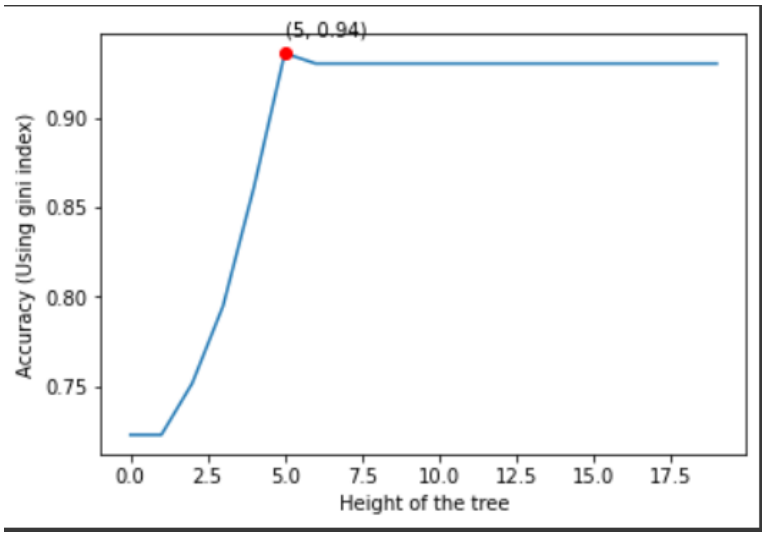
```
Iteration : 1 Accuracy: 0.9479768786127167
Iteration : 2 Accuracy: 0.9190751445086706
Iteration : 3 Accuracy: 0.9219653179190751
Iteration : 4 Accuracy: 0.8959537572254336
Iteration : 5 Accuracy: 0.9421965317919075
Iteration : 6 Accuracy: 0.9393063583815029
Iteration : 7 Accuracy: 0.9017341040462428
Iteration : 8 Accuracy: 0.9219653179190751
Iteration : 9 Accuracy: 0.9161849710982659
Iteration : 10 Accuracy: 0.9132947976878613

Average Accuracy: 0.921965317919075
Best Accuracy: 0.9479768786127167
```

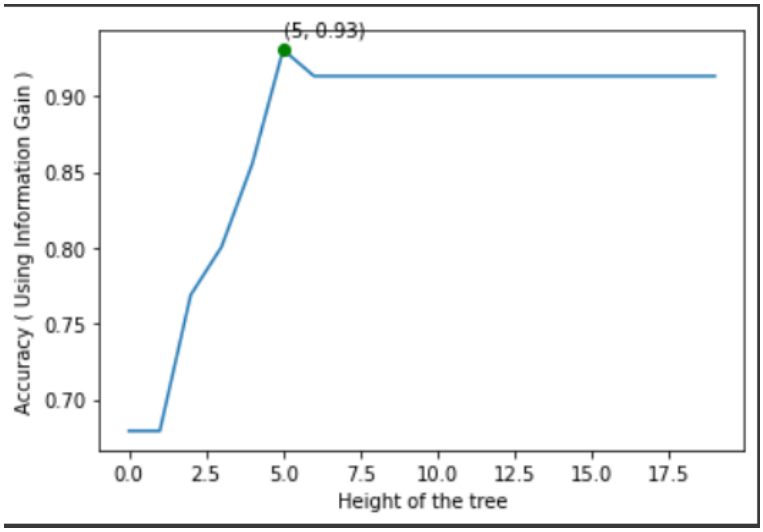
Here we can see that the best accuracy is 94.80% and the average accuracy is 92.20%.

We can see that although the results for Information Gain are slightly better, they are really very similar. It should be noted that Gini Index is faster to calculate so if speed is important then it should be preferred although the accuracy might be a little less.

Question 3

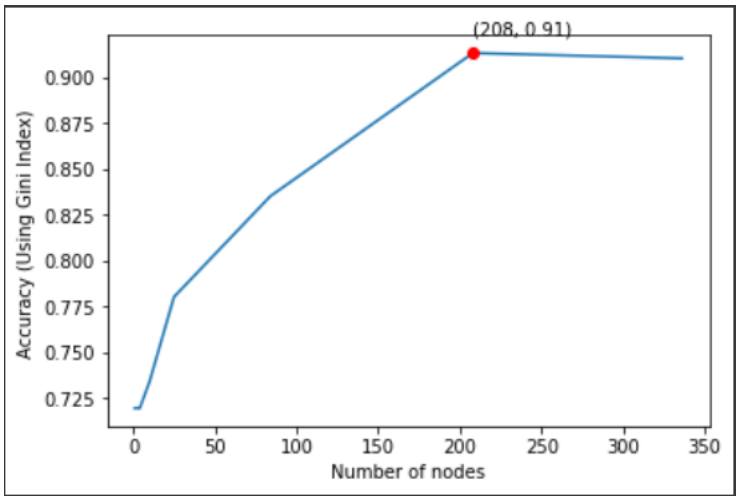


Plot: Accuracy (using Gini Index) Vs Height of the Tree

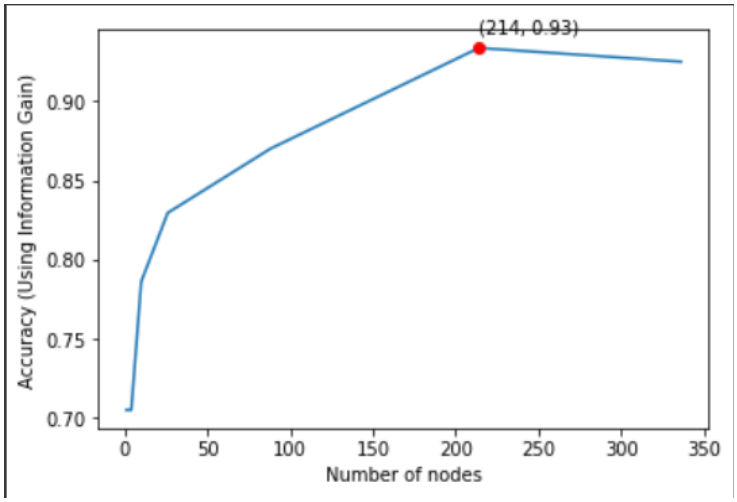


Plot: Accuracy (using Gini Index) Vs Height of the Tree

As the plots show, the best possible depth limit is 5.



Plot: Accuracy (using Gini Index) Vs Number of the nodes in the tree



Plot: Accuracy (using Information Gain) Vs Number of the nodes in the tree

As we can observe, the graphs plotted using gini index and information gain respectively are very similar.

Question 4

We have performed 2 types of pruning on our decision tree.

1) **Bottom up pruning with Chi- square as the statistical test.**

This is a statistical method to prune the decision tree. In this, we first calculate the Chi-square value for all the nodes whose children are terminals and then if the chi-square value is less than a threshold value , then children of the node are pruned and the node is marked as terminal. This process is repeated until no pruning occurs.

The threshold value depends on the number of the children and the total number of unique output values.

First we calculate the degree of freedom which is:

$$Df = (\text{number of children} - 1) * (\text{total output classes} - 1)$$

Using this degree of freedom and the confidence value, the threshold value is calculated.

Threshold Value is taken to be at 0.05 confidence level.

We can see that for the statistical method, there was a lot of pruning. The number of nodes decreased from 209 to 76 nodes.

After the pruning, the accuracy of the decision tree came out to be 90.17 % which is 3 % less than the best accuracy before the pruning.

2) **Reduced Error Post Pruning Method**

This method removes a terminal node and sees if the accuracy is increased or not. Node is removed which gives the maximum positive accuracy change.

On average, 2-3 nodes are pruned using this method and accuracy goes up by 1 - 2%.

Question 5

Column: safety , Gain: 0.38142868469429725 , Samples: 1382

safety == high:

    Column: persons , Gain: 0.43791646147529195 , Samples: 461

    persons == more:

        Column: buying , Gain: 0.5558199991178583 , Samples: 155

        buying == high:

            Column: maint , Gain: 0.0946305156831473 , Samples: 38

            maint == vhigh:

                Prediction: unacc

            maint == low:

                Prediction: acc

            maint == high:

                Prediction: acc

            maint == med:

                Prediction: acc

buying == med:

Column: maint , Gain: 0.3454545454545455 , Samples: 41

maint == med:

Prediction: vgood

maint == vhigh:

Prediction: acc

maint == high:

Prediction: acc

maint == low:

Prediction: vgood

buying == vhigh:

Column: maint , Gain: 0.09324009324009329 , Samples: 39

maint == high:

Prediction: unacc

maint == med:

Prediction: acc

maint == vhigh:

Prediction: unacc

maint == low:

Prediction: acc

buying == low:

Column: maint , Gain: 0.41966966966966973 , Samples: 37

maint == low:

Prediction: vgood

maint == med:

Prediction: vgood

maint == vhigh:

Prediction: acc

maint == high:

Prediction: vgood

persons == 2:

Prediction: unacc

persons == 4:

Column: buying , Gain: 0.522836884398973 , Samples: 159

buying == vhigh:

Column: maint , Gain: 0.0 , Samples: 37

maint == low:

Prediction: acc

maint == vhigh:

Prediction: unacc

maint == med:

Prediction: acc

maint == high:

Prediction: unacc

buying == low:

Column: maint , Gain: 0.37209302325581395 , Samples: 43

maint == vhigh:

Prediction: acc

maint == high:

Prediction: acc

maint == med:

Prediction: vgood

maint == low:

Prediction: good

buying == med:

Column: maint , Gain: 0.22857142857142854 , Samples: 42

maint == vhigh:

Prediction: acc

maint == high:

Prediction: acc

maint == med:

Prediction: vgood

maint == low:

Prediction: vgood

buying == high:

Column: maint , Gain: 0.0 , Samples: 37

maint == low:

Prediction: acc

maint == med:

Prediction: acc

maint == high:

Prediction: acc

maint == vhigh:

Prediction: unacc

safety == med:

Column: persons , Gain: 0.3909070082972797 , Samples: 452

persons == more:

    Column: luggage_boot , Gain: 0.4868428982331189 , Samples: 149

    luggage_boot == med:

        Column: maint , Gain: 0.4781827016520893 , Samples: 49

        maint == low:

            Prediction: acc

        maint == vhigh:

            Prediction: unacc

        maint == high:

            Prediction: acc

        maint == med:

            Prediction: acc

    luggage_boot == big:

        Column: maint , Gain: 0.3917948717948717 , Samples: 50

        maint == vhigh:

            Prediction: acc

        maint == high:

            Prediction: acc

        maint == med:

            Prediction: acc

        maint == low:

            Prediction: good

    luggage_boot == small:

        Prediction: unacc

persons == 2:

Prediction: unacc

persons == 4:

Column: buying , Gain: 0.4735462049513251 , Samples: 154

buying == low:

Column: maint , Gain: 0.3655603655603656 , Samples: 42

maint == low:

Prediction: acc

maint == high:

Prediction: acc

maint == vhigh:

Prediction: unacc

maint == med:

Prediction: good

buying == med:

Column: maint , Gain: 0.36808905380333956 , Samples: 35

maint == med:

Prediction: acc

maint == low:

Prediction: acc

maint == vhigh:

Prediction: unacc

maint == high:

Prediction: unacc

buying == vhigh:

Prediction: unacc

buying == high:

Column: luggage_boot , Gain: 0.3102240896358543 , Samples: 34

luggage_boot == small:

Prediction: unacc

luggage_boot == big:

Prediction: acc

luggage_boot == med:

Prediction: unacc

safety == low:

Prediction: unacc

How To Run the Code ?

You should run first run all the sections of the code (except the questions section) and then you can use the functions as per the requirements