PROJECT TITLE

Top-Down Induction of Decision Tree (TDIDT) algorithm implementation with three different attribute splitting techniques (information gain, Gini index, and Gain ratio) from scratch and a comparative discussion on predictive accuracy.

PROJECT OVERVIEW

The main objective of the project is to establish decision trees using the TDIDT algorithm, which uses different attribute splitting techniques for generating decision rules. In this project, three different attribute splitting techniques—information gain, Gini index, and Gain ratio—were used. All three of these splitting techniques were built from scratch. After that, the main TDIDT algorithm was established. Using the splitting techniques, the TDIDT algorithm selects the attribute for generating decision rules to build the decision tree. With these decision rules, the decision tree will predict the target class attributes for test data. With three different attribute splitting techniques, three different decision trees will be generated. Each of the decision trees will be trained on the same training data and evaluated with test data. The predictive accuracy will then be analyzed with a confusion matrix. At the end, the evaluation of the decision tree will be demonstrated with different visualization techniques.

PROJECT SOLUTION

The project solution will be based on the following steps –

1. **Dataset selection-** A dataset with several attributes and a class attribute (target attribute) must be chosen before the model is built. As a requirement, the dataset should contain at least 5000 instances, 2 attributes and 1 class attribute.

2. **Pre-processing -** Before the implementation begins, the dataset needs to be pre-processed and cleaned. This project required categorical dataset and before using the dataset it was preprocessed according to the need of the project.

3. **Building Attribute Splitting Techniques –** As per the requirement of the project three different attribute splitting techniques to be used:

   a. ***Entropy Information Gain***- Entropy is a measure of uncertainty or impurity in a dataset. Information Gain is a concept used in Decision Trees to determine the best attribute to split the data at each node. It measures the reduction in entropy achieved by splitting the data on a particular attribute.

$$\boldsymbol{Entropy}(\boldsymbol{S}) = -\sum_{i=1}^{n} p_i \, log_2(p_i)$$

$$\boldsymbol{Information\ Gain}(\boldsymbol{S},\boldsymbol{A}) = Entropy(S) - \sum_{v=V} \frac{|S_v|}{|S|} * Entropy(S_v)$$

b. **Gini Index-** Gini Index is another measure of impurity used in Decision Trees. It quantifies the probability of a random sample being misclassified if it were randomly classified according to the distribution of the target classes in the set.

$$\boldsymbol{Gini\ Index(S)} = 1 - \sum_{i=1}^{n} (p_i)^2$$

c. **Gain Ratio-** Gain Ratio is a modification of Information Gain to account for the bias towards attributes with many distinct values (high cardinality). It helps in handling attributes with many unique values by normalizing the Information Gain.

$$\boldsymbol{Splitting\ Information(S, A)} = -\sum_{v \equiv k} \frac{|s_v|}{|s|} * log_2\left(\frac{|s_v|}{|s|}\right)$$

$$\boldsymbol{Gain\ Ratio}(\mathbf{S}, \mathbf{A}) = \frac{Information\ Gain}{Splitting\ Information}$$

4. **TDIDT Algorithm Implementation -** The TDIDT (Top-Down Induction of Decision Trees) algorithm is a recursive approach to building decision trees. It starts with the entire dataset as the root node and recursively splits the data into subsets based on the best attribute at each node. Here is a step-by-step description of the TDIDT algorithm:

a. **Input-** The training dataset with the features (attributes) and the target class. A list of attribute names that will be used for splitting the dataset. The name of the target class column.

b. **Terminating Criteria-**

i. If all instances in the subset belong to the same class, create a leaf node with that class label.

ii. If there are no more attributes left to split on, create a leaf node with the majority class label in the subset.

iii. Other stopping criteria, such as a predefined depth limit or a minimum number of instances per leaf node, can also be used to avoid overfitting.

c. **Attribute Selection-** Select any one of the three techniques of the splitting attribute function and select the appropriate value (decision rules) for the current node.

d. **Split the dataset-** Split the dataset into subsets based on the selected attribute's values. Create child nodes for each unique value of the attribute.

e. **Output:** The algorithm will eventually create a decision tree, which can be used for predictions on new data.

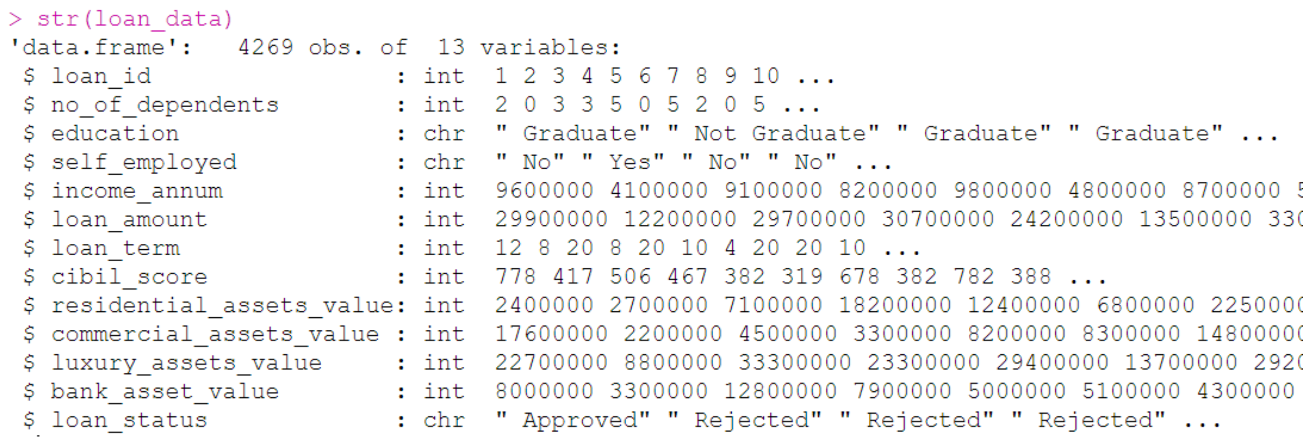
5. **Generating Decision Trees-** With three attribute splitting techniques, three different decision tree models will be generated. Using these three models rest of the testing, validation and evaluation will be made.

6. **Result Analysis-** After acquiring all the results for each decision tree for the attribute splitting techniques, the results will be shown in a confusion matrix. This will help the project to identify which model and which splitting techniques, giving better results.

7. **Further Improvement and Discussion-** The last step of the project is to verify the results, finding limitations and discussing the advantages and disadvantages of using different attribute splitting techniques for the TDIDT algorithm.

PROJECT DATASET

The dataset that has been chosen for this project is "Loan Approval Data" dataset. It was collected from *Kaggle* under the name of "*loan_approval_data.csv*." This dataset contains a total of 13 variables (columns) and 4269 observations (rows). Below are the dataset details—



```
> str(loan_data)
'data.frame':   4269 obs. of  13 variables:
 $ loan_id                 : int  1 2 3 4 5 6 7 8 9 10 ...
 $ no_of_dependents        : int  2 0 3 3 5 0 5 2 0 5 ...
 $ education               : chr  " Graduate" " Not Graduate" " Graduate" " Graduate" ...
 $ self_employed           : chr  " No" " Yes" " No" " No" ...
 $ income_annum            : int  9600000 4100000 9100000 8200000 9800000 4800000 8700000 5
 $ loan_amount             : int  29900000 12200000 29700000 30700000 24200000 13500000 33
 $ loan_term               : int  12 8 20 8 20 10 4 20 20 10 ...
 $ cibil_score             : int  778 417 506 467 382 319 678 382 782 388 ...
 $ residential_assets_value: int  2400000 2700000 7100000 18200000 12400000 6800000 225000
 $ commercial_assets_value : int  17600000 2200000 4500000 3300000 8200000 8300000 148000
 $ luxury_assets_value     : int  22700000 8800000 33300000 23300000 29400000 13700000 292
 $ bank_asset_value        : int  8000000 3300000 12800000 7900000 5000000 5100000 4300000
 $ loan_status             : chr  " Approved" " Rejected" " Rejected" " Rejected" ...
```

This project will only focus on classifying the 'loan_status' variable. This class attribute contains the categorical value as "Approved" and "Rejected" which indicates if an instance of the loan data will be given a loan or not. Generally, a loan status depends on many other criteria other than the mentioned attributes. But occasionally the decision is made considering attributes like number of dependents, education status, employment status, annual income, loan terms, histories, asset values, and etc. Therefore, the project will be focusing only on the six attributes – "*no_of_dependents, education,self_employed, income_annum, loan_amount, loan_term*" and the target class attribute *'loan_status'*. Other attributes will be omitted for the project as that will be not necessary for the project. For the necessity of the project, numerical data in the feature attributes will be also factorized into categorical data.

DATA PREPROCESSING

Before building the model, data pre-processing on the selected dataset is necessary. The pre-processing steps that have been used for this dataset is discussed below –

a. **Data Cleaning:** The whole dataset has been checked for any missing data and found no such data. Then it has been checked for missing data or zero values in the data points and found 179 data points with that. The project removed that data and further checked for mismatched variables and no such data found. There was noisy data found and no major irrelevancy discovered.

b. **Data Reduction:** The first data point attributes *“loan_id”* and the last five attributes - *"cibil_score", "residential_assets_value", "commercial_assets_value", "luxury_assets_value", "bank_asset_value” attributes* also been reduced from the original dataset. Finally, the six attributes - *"no_of_dependents", "education", "self_employed", "income_annum", "loan_amount”,* and “*loan_term”* were chosen for the final dataset along with the class attribute “*loan_status.”*

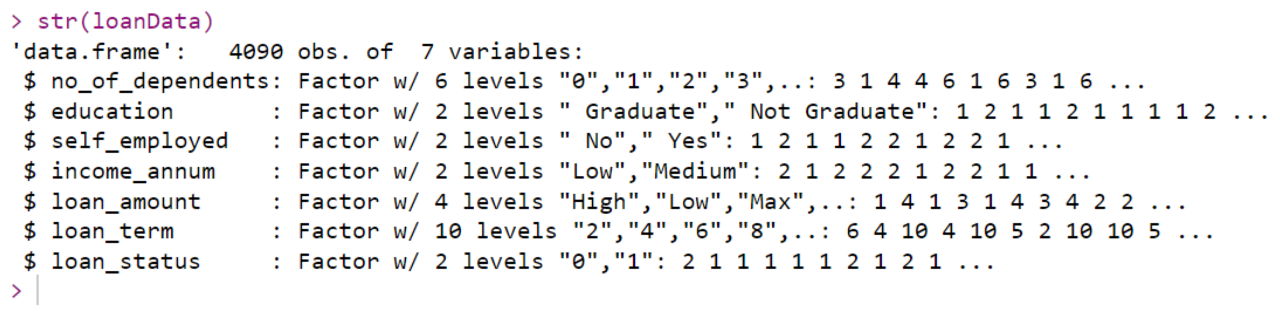
c. **Data Factorization:** Two of the attributes ‘*income_annum’* and *‘loan_amount’* were factorized to 4 types of categorical values. The ranges of each category were –

➢ 0 – 5000	: “Low”
➢ 5001 – 15000	: “Medium”
➢ 15001 – 30000	: “High”
➢ 30001 or more	: “Max”

d. **Feature Engineering:** The target class attribute ‘*loan_status*’ column was renamed as ‘*Class’* column and the values of “Approved” was replaced by 1 and “Rejected” was replaced by 0. The rest of the data was taken from the dataset for this project.

THE FINAL DATASET

The final dataset, after all the data preprocessing and data munging, contains 4090 instances and 7 columns – 6 for attributes and 1 column for the target attribute. Here are the details of the final dataset –

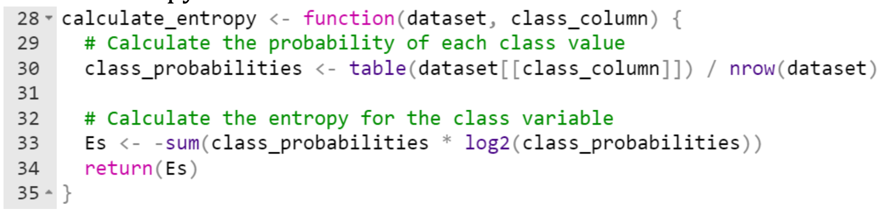


```
> str(loanData)
'data.frame':	4090 obs. of  7 variables:
 $ no_of_dependents: Factor w/ 6 levels "0","1","2","3",..: 3 1 4 4 6 1 6 3 1 6 ...
 $ education       : Factor w/ 2 levels " Graduate"," Not Graduate": 1 2 1 1 2 1 1 1 1 2 ...
 $ self_employed   : Factor w/ 2 levels " No"," Yes": 1 2 1 1 2 2 1 2 2 1 ...
 $ income_annum    : Factor w/ 2 levels "Low","Medium": 2 1 2 2 2 1 2 2 1 1 ...
 $ loan_amount     : Factor w/ 4 levels "High","Low","Max",..: 1 4 1 3 1 4 3 4 2 2 ...
 $ loan_term       : Factor w/ 10 levels "2","4","6","8",..: 6 4 10 4 10 5 2 10 10 5 ...
 $ loan_status     : Factor w/ 2 levels "0","1": 2 1 1 1 1 1 2 1 2 1 ...
> 
```

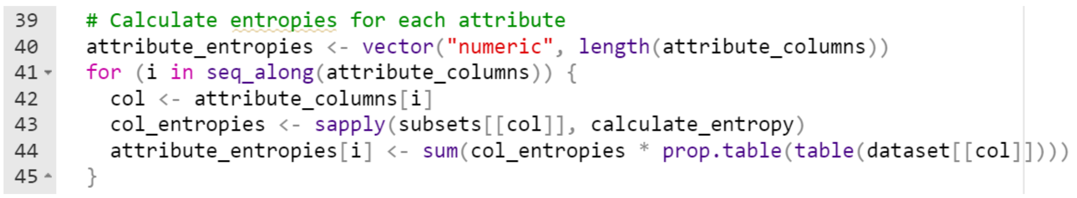
ATTRIBUTE SPLITTING METHODS

❖ **Entropy and Information Gain:** The code implementation of this attribute splitting technique was developed and divided into three steps –

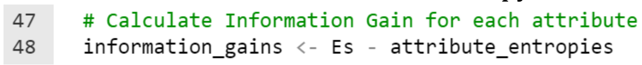
1. ***Entropy for Class Attribute***: First the entropy for class attribute (Es) will be calculated. Then, the entropy for the rest of the attributes will be calculated.



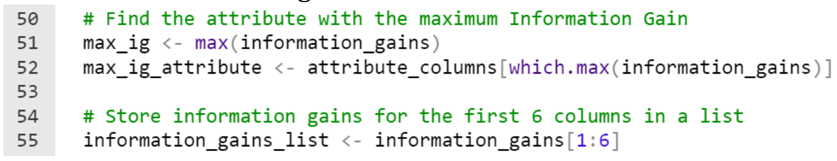
```
calculate_entropy <- function(dataset, class_column) {
  # Calculate the probability of each class value
  class_probabilities <- table(dataset[[class_column]]) / nrow(dataset)

  # Calculate the entropy for the class variable
  Es <- -sum(class_probabilities * log2(class_probabilities))
  return(Es)
}
```


```r
  # Calculate entropies for each attribute
  attribute_entropies <- vector("numeric", length(attribute_columns))
  for (i in seq_along(attribute_columns)) {
    col <- attribute_columns[i]
    col_entropies <- sapply(subsets[[col]], calculate_entropy)
    attribute_entropies[i] <- sum(col_entropies * prop.table(table(dataset[[col]])))
  }
```

2. ***Calculate the Information Gain**:* Next the information gain for each attribute will be calculated from the calculated entropy results.

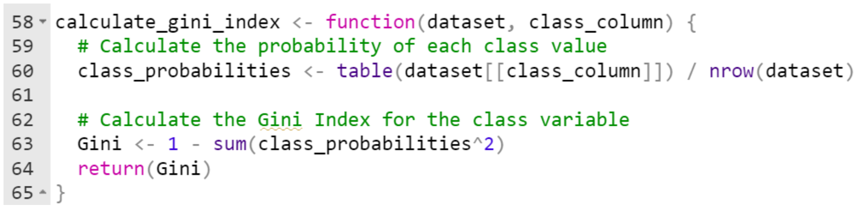

```r
  # Calculate Information Gain for each attribute
  information_gains <- Es - attribute_entropies
```

3. ***Storing Maximum Information Gain**:* The maximum information gain will be calculated and stored in descending order.

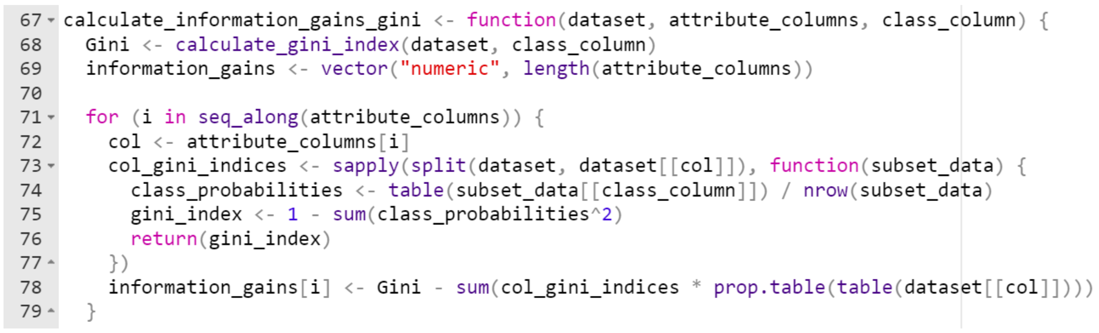

```r
  # Find the attribute with the maximum Information Gain
  max_ig <- max(information_gains)
  max_ig_attribute <- attribute_columns[which.max(information_gains)]

  # Store information gains for the first 6 columns in a list
  information_gains_list <- information_gains[1:6]
```

❖ **Gini Index:** For implementing the Gini Index, two parts were made –

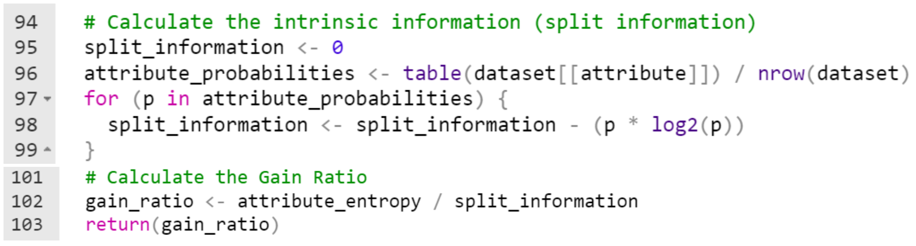
1. ***Calculate Gini Index for each Attribute:*** Using the equation for Gini index, class probabilities were calculated like this-


```r
calculate_gini_index <- function(dataset, class_column) {
  # Calculate the probability of each class value
  class_probabilities <- table(dataset[[class_column]]) / nrow(dataset)

  # Calculate the Gini Index for the class variable
  Gini <- 1 - sum(class_probabilities^2)
  return(Gini)
}
```

2. ***Calculate Gini Information Index:***

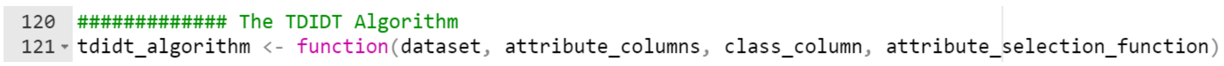

```r
calculate_information_gains_gini <- function(dataset, attribute_columns, class_column) {
  Gini <- calculate_gini_index(dataset, class_column)
  information_gains <- vector("numeric", length(attribute_columns))

  for (i in seq_along(attribute_columns)) {
    col <- attribute_columns[i]
    col_gini_indices <- sapply(split(dataset, dataset[[col]]), function(subset_data) {
      class_probabilities <- table(subset_data[[class_column]]) / nrow(subset_data)
      gini_index <- 1 - sum(class_probabilities^2)
      return(gini_index)
    })
    information_gains[i] <- Gini - sum(col_gini_indices * prop.table(table(dataset[[col]])))
  }
```

❖ **Gain Ratio:** Using the previously calculated entropy values, and information gain, the implementation of Gain ratio becomes quite easier. Just an additional calculation for the splitting information was calculated, shown below-


```r
  # Calculate the intrinsic information (split information)
  split_information <- 0
  attribute_probabilities <- table(dataset[[attribute]]) / nrow(dataset)
  for (p in attribute_probabilities) {
    split_information <- split_information - (p * log2(p))
  }
  # Calculate the Gain Ratio
  gain_ratio <- attribute_entropy / split_information
  return(gain_ratio)
```

TDIDT ALGORITHM IMPLEMETATION

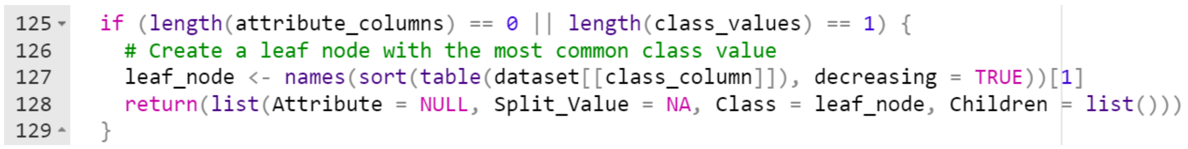
The TDIDT algorithm was implemented explicitly after finishing with the splitting techniques. This function of the algorithm takes 4 arguments – the dataset as data frame, attribute columns, class column, and the desired attribute selection function previously defined.



```r
############# The TDIDT Algorithm
tdidt_algorithm <- function(dataset, attribute_columns, class_column, attribute_selection_function)
```

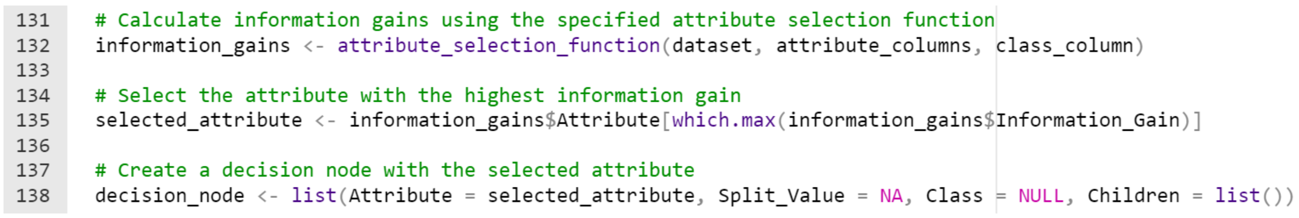
The rest of the implementation of the algorithm will be demonstrated below –

i. The condition of checking if the dataset has no attributes or has just one unique class values, then creates the leaf node and terminates it:



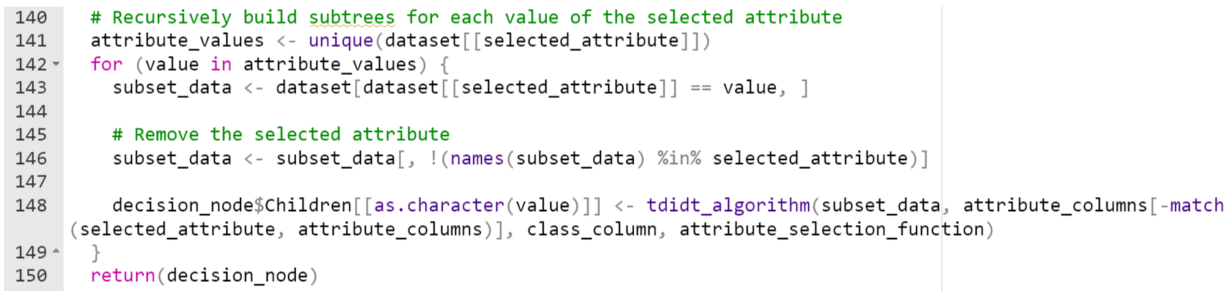
```r
  if (length(attribute_columns) == 0 || length(class_values) == 1) {
    # Create a leaf node with the most common class value
    leaf_node <- names(sort(table(dataset[[class_column]]), decreasing = TRUE))[1]
    return(list(Attribute = NULL, Split_Value = NA, Class = leaf_node, Children = list()))
  }
```

ii. Calculate the information gain value using the selected attribute splitting function and create a decision node accordingly:



```r
  # Calculate information gains using the specified attribute selection function
  information_gains <- attribute_selection_function(dataset, attribute_columns, class_column)

  # Select the attribute with the highest information gain
  selected_attribute <- information_gains$Attribute[which.max(information_gains$Information_Gain)]

  # Create a decision node with the selected attribute
  decision_node <- list(Attribute = selected_attribute, Split_Value = NA, Class = NULL, Children = list())
```

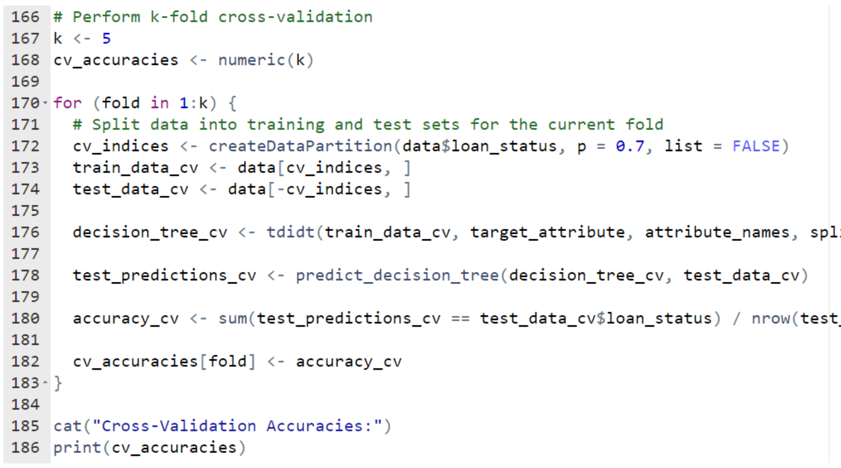
iii. Recursively build the sub-trees for each value of the selected attribute:



```r
  # Recursively build subtrees for each value of the selected attribute
  attribute_values <- unique(dataset[[selected_attribute]])
  for (value in attribute_values) {
    subset_data <- dataset[dataset[[selected_attribute]] == value, ]

    # Remove the selected attribute
    subset_data <- subset_data[, !(names(subset_data) %in% selected_attribute)]

    decision_node$Children[[as.character(value)]] <- tdidt_algorithm(subset_data, attribute_columns[-match(selected_attribute, attribute_columns)], class_column, attribute_selection_function)
  }
  return(decision_node)
```

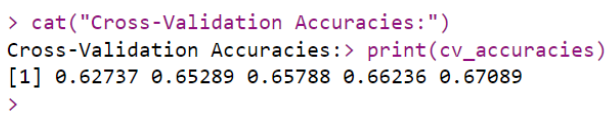
PREDICTIVE ACCURACY

K-fold Cross Validation: For the cross-validation process, the k-fold cross validation has been chosen. The value of k has been chosen 5 for this dataset. Here is the code snippet for 5-fold cross-validation:



```r
# Perform k-fold cross-validation
k <- 5
cv_accuracies <- numeric(k)

for (fold in 1:k) {
  # Split data into training and test sets for the current fold
  cv_indices <- createDataPartition(data$loan_status, p = 0.7, list = FALSE)
  train_data_cv <- data[cv_indices, ]
  test_data_cv <- data[-cv_indices, ]

  decision_tree_cv <- tdidt(train_data_cv, target_attribute, attribute_names, splitting_technique, max_depth = 5)

  test_predictions_cv <- predict_decision_tree(decision_tree_cv, test_data_cv)

  accuracy_cv <- sum(test_predictions_cv == test_data_cv$loan_status) / nrow(test_data_cv)

  cv_accuracies[fold] <- accuracy_cv
}

cat("Cross-Validation Accuracies:")
print(cv_accuracies)
```

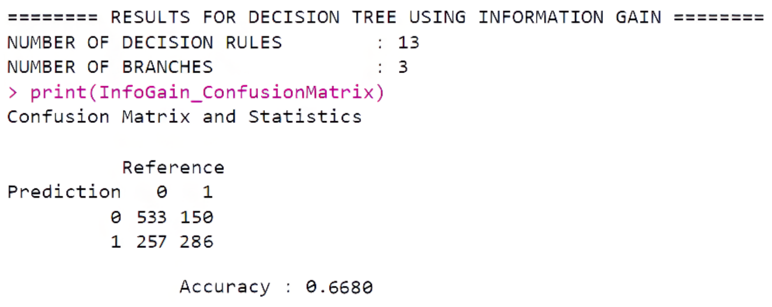
Output:



```
> cat("Cross-Validation Accuracies:")
Cross-Validation Accuracies:> print(cv_accuracies)
[1] 0.62737 0.65289 0.65788 0.66236 0.67089
>
```

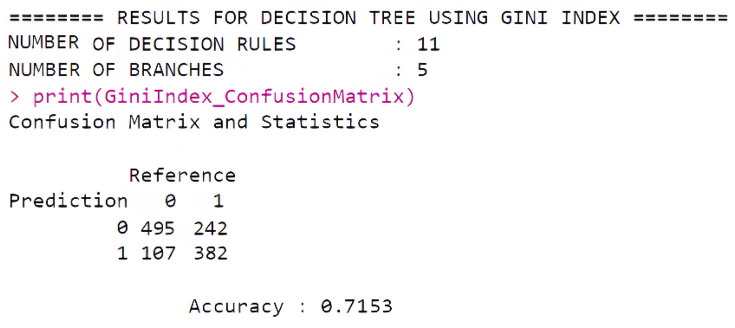
Testing:

Using the decision tree by ***Information Gain*** on **1226** test data:



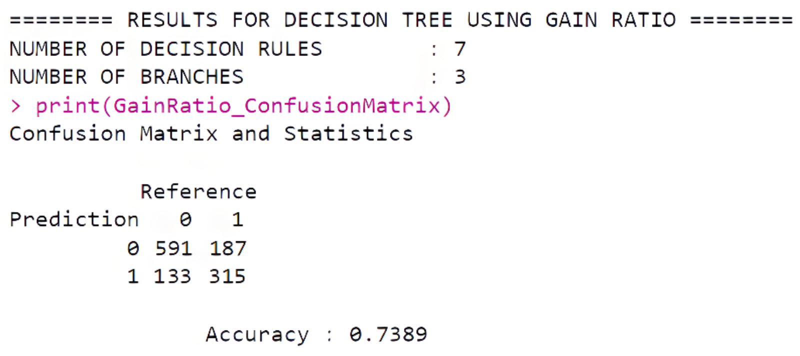
```
======== RESULTS FOR DECISION TREE USING INFORMATION GAIN ========
NUMBER OF DECISION RULES          : 13
NUMBER OF BRANCHES                : 3
> print(InfoGain_ConfusionMatrix)
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 533 150
         1 257 286

               Accuracy : 0.6680
```

Using the decision tree by ***Gini Index*** on **1226** test data:



```
======== RESULTS FOR DECISION TREE USING GINI INDEX ========
NUMBER OF DECISION RULES          : 11
NUMBER OF BRANCHES                : 5
> print(GiniIndex_ConfusionMatrix)
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 495  242
         1 107  382

               Accuracy : 0.7153
```

Using the decision tree by ***Gain Ratio*** on **1226** test data:



```
======== RESULTS FOR DECISION TREE USING GAIN RATIO ========
NUMBER OF DECISION RULES          : 7
NUMBER OF BRANCHES                : 3
> print(GainRatio_ConfusionMatrix)
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 591 187
         1 133 315

               Accuracy : 0.7389
```

RESULTS

Confusion Matrix: Information Gain

Information Gain	Reference 0	Reference 1
Predicted 0	533	150
Predicted 1	257	286

Confusion Matrix: Gini Index

Gini Index	Reference 0	Reference 1
Predicted 0	495	242
Predicted 1	107	382

Confusion Matrix: Gain Ratio

Information Gain	Reference 0	Reference 1
Predicted 0	591	187
Predicted 1	133	315

Top-Down Induction of Decision Tree, or TDIDT algorithm, is used with three different attribute splitting techniques like Information gain, Gini index, and Gain ratio for predictive accuracy. We got the lowest accuracy from using the Information gain technique, which is 66.80 percent. With the Gain Ratio technique, predictive accuracy was at its maximum, which is 73.89%. But using the Gini Index, we got 71.53 percent accuracy for our dataset.

Attribute Splitting Techniques	Branches	Decision Rules	Test Accuracy (in %)
Information Gain	3	13	66.80
Gini Index	5	11	71.53
Gain Ratio	3	7	73.89

DISCUSSION

The Top-Down Induction of Decision Tree (TDIDT) attribute splitting approach strikes a balance between interpretability and predictive power. Its advantages, including model transparency, automatic feature selection, and ability to manage non-linearity, align well with industries that prioritize understandable models. However, its vulnerabilities to overfitting, sensitivity to data variations, and limitations in handling complex relationships necessitate thoughtful mitigation strategies. In practice, striking the right balance between the model's complexity and its interpretability while addressing these challenges requires a careful evaluation of the dataset's characteristics, the desired level of transparency, and the available resources for model tuning and validation.

LIMITATIONS

The Top-Down Induction of Decision Tree (TDIDT) algorithm, while powerful and widely used, does have certain limitations:

1. *Overfitting:* TDIDT can easily overfit the training data, especially if the tree is grown too deep. This occurs when the algorithm creates splits to accommodate noise and outliers in the training data, resulting in poor generalization to new, unseen data.

2. *Instability:* TDIDT is sensitive to slight changes in the training data. Minor variations can lead to different attribute splits, causing instability in the resulting tree structure and potentially impacting predictions. This makes the model less robust and consistent.

3. *Bias Towards Dominant Classes:* In datasets with imbalanced class distributions, TDIDT tends to create decision trees that favor the majority class. As a result, the model might perform poorly in minority classes. Balancing techniques or ensemble methods are often required to address this bias.

4. *Inability to Capture Complex Relationships:* TDIDT constructs decision trees using a series of simple binary splits based on attribute values. This can make it challenging to capture intricate relationships that involve multiple attributes or interactions. More complex models, like random forests or gradient boosting, may be better suited for such scenarios.

5. *Greedy Approach:* TDIDT employs a greedy strategy by choosing the best attribute split at each step. While this approach simplifies the algorithm, it can lead to suboptimal solutions in certain cases. TDIDT might miss globally optimal splits because it does not consider the impact of future splits.

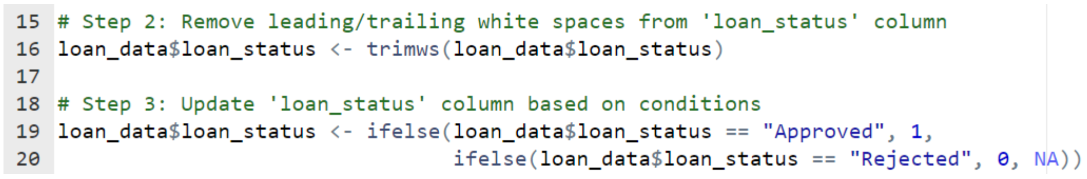
6. *Exponential Growth:* The depth of the decision tree can grow exponentially with the number of attributes and the complexity of the dataset. Deep trees can become hard to interpret, and their performance might degrade due to overfitting or increased complexity.

7. *Limited to Supervised Learning:* TDIDT is primarily designed for supervised learning tasks where the target variable is known during training. It might not be directly applicable to unsupervised learning problems or tasks involving sequential data, text, or images.

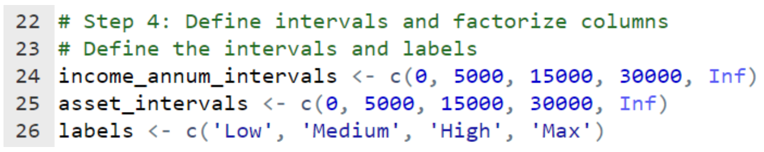
8. *Lack of Global Optimization:* The algorithm makes local decisions at each node based on the available data, without considering the global context of the entire tree. This can result in suboptimal decision paths that do not fully exploit the dataset's underlying patterns.

CODE SNIPPETS

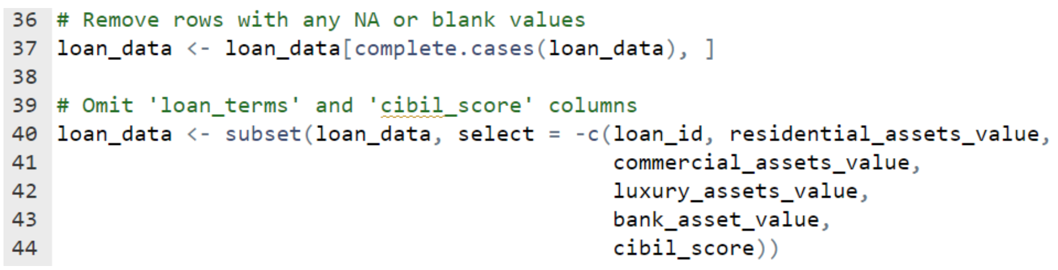
Pre-processing steps-



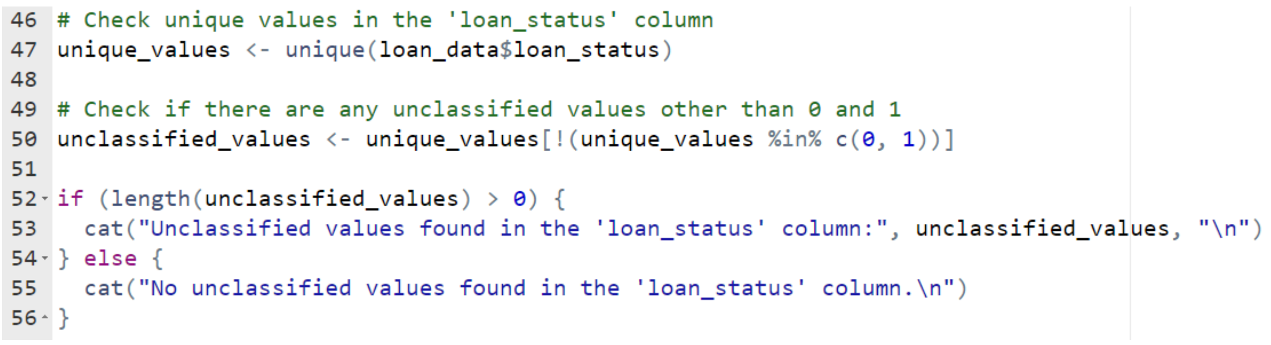
```r
# Step 2: Remove leading/trailing white spaces from 'loan_status' column
loan_data$loan_status <- trimws(loan_data$loan_status)

# Step 3: Update 'loan_status' column based on conditions
loan_data$loan_status <- ifelse(loan_data$loan_status == "Approved", 1,
                                ifelse(loan_data$loan_status == "Rejected", 0, NA))
```



```r
# Step 4: Define intervals and factorize columns
# Define the intervals and labels
income_annum_intervals <- c(0, 5000, 15000, 30000, Inf)
asset_intervals <- c(0, 5000, 15000, 30000, Inf)
labels <- c('Low', 'Medium', 'High', 'Max')
```



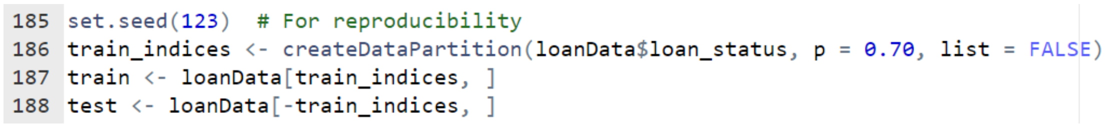
```r
# Remove rows with any NA or blank values
loan_data <- loan_data[complete.cases(loan_data), ]

# Omit 'loan_terms' and 'cibil_score' columns
loan_data <- subset(loan_data, select = -c(loan_id, residential_assets_value,
                                           commercial_assets_value,
                                           luxury_assets_value,
                                           bank_asset_value,
                                           cibil_score))
```



```r
# Check unique values in the 'loan_status' column
unique_values <- unique(loan_data$loan_status)

# Check if there are any unclassified values other than 0 and 1
unclassified_values <- unique_values[!(unique_values %in% c(0, 1))]

if (length(unclassified_values) > 0) {
  cat("Unclassified values found in the 'loan_status' column:", unclassified_values, "\n")
} else {
  cat("No unclassified values found in the 'loan_status' column.\n")
}
```

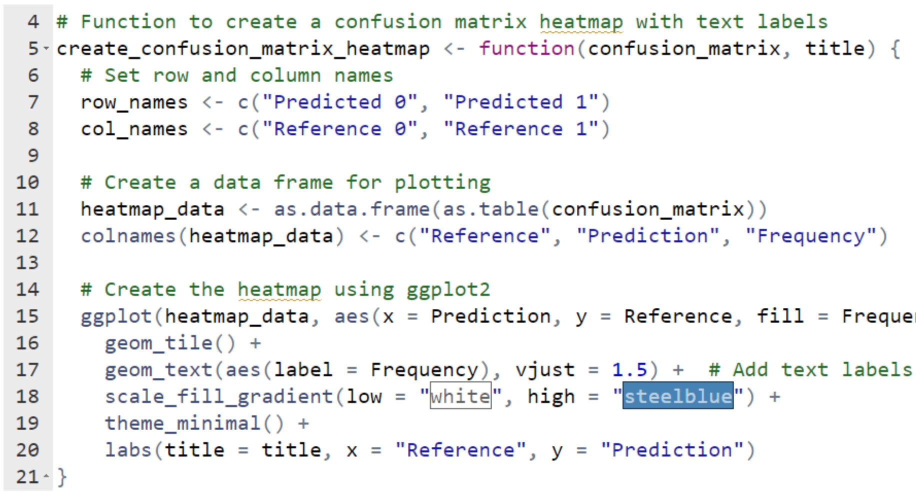
Output: `No unclassified values found in the 'loan_status' column.`

Data segmentation steps-

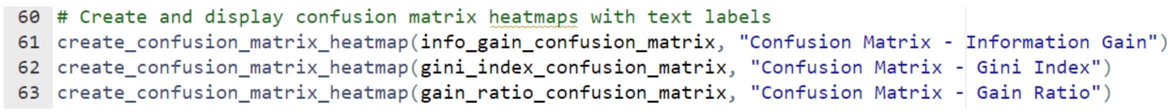


```r
set.seed(123)  # For reproducibility
train_indices <- createDataPartition(loanData$loan_status, p = 0.70, list = FALSE)
train <- loanData[train_indices, ]
test <- loanData[-train_indices, ]
```

Plotting Code-



```r
# Function to create a confusion matrix heatmap with text labels
create_confusion_matrix_heatmap <- function(confusion_matrix, title) {
  # Set row and column names
  row_names <- c("Predicted 0", "Predicted 1")
  col_names <- c("Reference 0", "Reference 1")

  # Create a data frame for plotting
  heatmap_data <- as.data.frame(as.table(confusion_matrix))
  colnames(heatmap_data) <- c("Reference", "Prediction", "Frequency")

  # Create the heatmap using ggplot2
  ggplot(heatmap_data, aes(x = Prediction, y = Reference, fill = Frequency)) +
    geom_tile() +
    geom_text(aes(label = Frequency), vjust = 1.5) +  # Add text labels
    scale_fill_gradient(low = "white", high = "steelblue") +
    theme_minimal() +
    labs(title = title, x = "Reference", y = "Prediction")
}
```



```r
# Create and display confusion matrix heatmaps with text labels
create_confusion_matrix_heatmap(info_gain_confusion_matrix, "Confusion Matrix - Information Gain")
create_confusion_matrix_heatmap(gini_index_confusion_matrix, "Confusion Matrix - Gini Index")
create_confusion_matrix_heatmap(gain_ratio_confusion_matrix, "Confusion Matrix - Gain Ratio")
```

VISUALIZATION

Confusion Matrix plots –

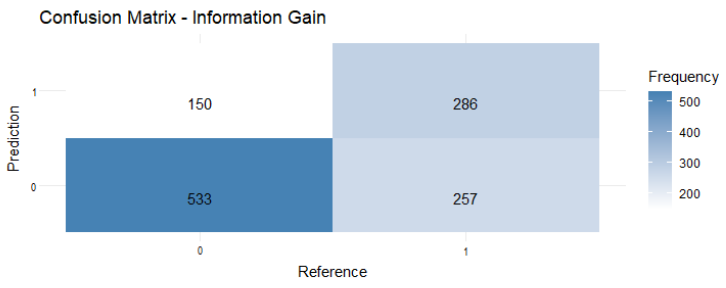


Fig 1: Confusion Matrix Plot for Information Gain

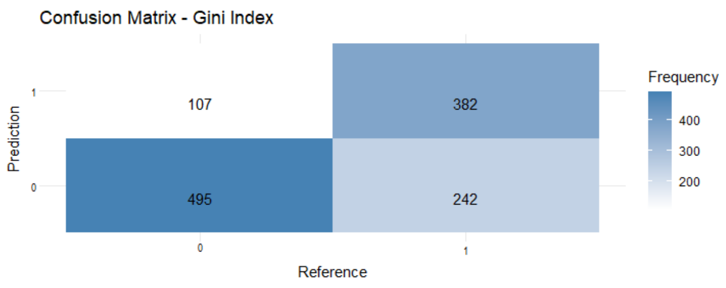


Fig 2: Confusion Matrix Plot for Gini Index

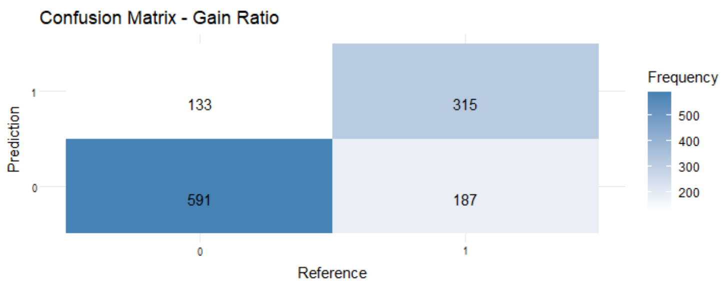


Fig 3: Confusion Matrix Plot for Gain Ratio