# Decision Trees and Random Forests

Monica Dasi
*High Integrity Systems (M.Sc)*
*Frankfurt University of Applied Sciences*
Nibelungenplatz 1, 60318 Frankfurt am Main
Email: monica.dasi@stud.fra-uas.de

***Abstract*—Trees are one of the prominent data structures in visualizing any form of data. A tree represents the given data in a hierarchical structure; for instance, in machine learning, a tree refers to a decision tree, a supervised machine learning technique to predict the classification or regression function by learning the decision rules derived from a specific set of features. Although decision trees are easy to construct and interpret and do not require data normalization, unlike other supervised machine learning methods, they are susceptible to over-fitting, which means even the slightest change in the data can lead to high variance. Furthermore, random forests belong to the ensemble learning category as it integrates the outcome of various sub-decision trees with a single result. Each sub-tree is trained on multiple data samples and overcomes the classic issue of over-fitting seen in decision trees. This paper provides a research analysis on various aspects of decision trees and random forests. Besides, this paper also provides a practical approach to implementing a decision tree using the sklearn library in python, which employs a Classification and Regression Tree algorithm.**

***Index Terms*—classification, regression, algorithms, overfitting, decision trees, random forests**

## I. Introduction

An abundant amount of data is available in today's world, helping humanity prepare for natural calamities, starting with predicting pandemics to tsunamis. However, the more significant challenge is to employ efficient methods to analyze and understand this enormous data. Hence, it is essential to know how one can classify this data for predicting the required target. Machine learning offers a plethora of supervised learning algorithms, which are efficient and accurate in solving a classification problem. One such machine learning technique is a decision tree. There is rapid growth in the research of decision tree algorithms on classification and regression problems. The main reason is lucid interpretation and the ability to separate the data that is not linearly separable. In addition, decision trees deliver better prediction accuracy, faster computation, and wider availability of software [1].
Decision trees inherently incorporate numeric and categorical predictor variables and missing values and are invariant under the changes of independent variables. As a result, scaling and variable changes are easier to handle. Moreover, they are immune to the effects of predictor outliers. Ultimately, they perform internal feature selection as an integral part of the procedure, which is why they are immune to the inclusion of numerous unrelated predictor variables. These decision tree properties are the primary reason they have become the most prevalent learning method for machine learning [2].

## II. Decision Trees

A decision tree is a flowchart-type structure indicating a clear trail to a decision, allowing different groups across an institution to understand the outcome of a decision lucidly. They break down the complex data into small chunks in a graphical form. It is an algorithm that categorizes data with conditional 'control' statements. The decision tree starts at a single 'node,' which branches into two or more paths. Each branch delivers a different possible outcome, combining multiple decisions and possible events until there is an outcome. Henceforth, the applications are predominant in prediction analysis, data classification, and regression [3].

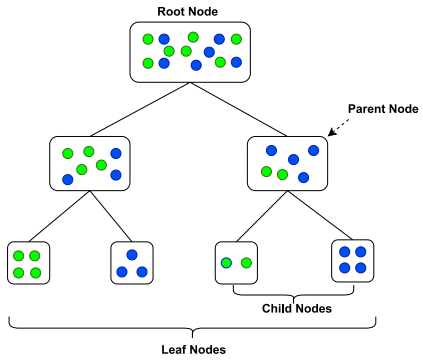


Fig. 1. Taxonomy of Decision Tree [4]

### A. Terminology

Decision trees can handle complex data, which makes them practical. All decision trees consist of essential parts called "nodes". It branches into the internal or decision nodes, starting with a top or root node. These nodes perform evaluations based on specific features and threshold values to form homogeneous or pure nodes called 'leaf nodes'. These leaf nodes, often termed terminal nodes, illustrate the possible outcomes from a dataset. Finally, tree branches are formed when edges are connected to nodes. Nodes and branches can often be used in various combinations to construct trees with different complexity. Fig1: depicts a root node, decision node, and leaf node. Besides these key terms, there are other important keywords to understand decision trees.

**Splitting:** When any node splits into two or more nodes, it is called Splitting or branching. Nodes after the split can be

internal or classified nodes, termed leaf or terminal nodes.

**Pruning:** Many times, based on the data, decision trees can evolve fairly complicated. In such cases, they are biased towards irrelevant data. To avoid this, we can remove specific nodes using a process called 'pruning', where the irrelevant branches of the tree are removed [5].

*a) Induction and Deduction:* An integral aspect of supervised machine learning is a classification problem that aims to construct class distribution models by evaluating a set of predictive features. Such models are useful for labeling the new unknown classes. Induction and deduction are the main steps performed by a classification algorithm [6].

The induction step uses the training set to render a model using abstract knowledge depiction, which is, in turn, employed for classifying samples whose information is unknown; this is done as part of the deduction step. Decision trees, with their ability to represent the data visually, are often preferred as a classification model. Moreover, one can express them in natural language as IF-THEN clauses [7].

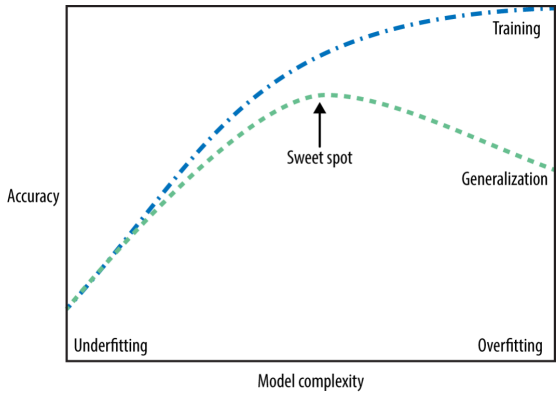


Fig. 2. Tradeoff between train and validation accuracy [8].

If the model can predict the unknown data accurately, then the model can generalize the data well. Therefore, the created model generally has good accuracy while predicting the train data. Furthermore, if the train and test data share the most common characteristics, then the model performs well on validation or test data. Nevertheless, there could be instances where this prediction can go wrong if the built model is complex.

The only way to test whether the algorithm is accurate, is to evaluate this on new test data. Intuitively, one expects simple models to generalize well on unknown data. Hence, the aim is to build a simple model. If the built models are too simpler, it is impossible to capture all variations in the data, which might lead to "**Underfitting**". On the other hand, "**overfitting**" occurs; if the model is too complex, it will perform well on the train data but cannot generalize the validation data; hence the accuracy will be poor for such complex models [8].

Hence, a trade-off between overfitting and underfitting will yield the best generalization performance which is illustrated in Fig.2

### *B. Attribute Selection Methods*

Attribute selection methods are a set of splitting rules to decide how the features will split. In other words, attribute selection methods best describe the splitting criteria for achieving the best optimal solution to the given classification problem. Furthermore, all the available decision tree algorithms require a splitting criterion for splitting a node. In most scenarios, distinct uni-variate functions are used as splitting criteria, which branches the internal node based on a single attribute value. The algorithm aims to select the best split point that minimizes a node's impurity after the split. Below are the selection measures that aid in splitting the node to make it more homogeneous as the splits go on [6] [5].

- Gini Index
- Entropy
- Information gain
- Gain Ratio
- Reduction in Variance
- Chi-Square

**Gini :** Gini talks about node impurity in other words, it is a measure of the probability of an attribute misclassification. The lower the values, the more homogeneous the node is [5] [9] .

$$Gini = 1 - \sum_{i=1}^{c} P_i^2 \tag{1}$$

where
$c$ : number of classes
$P_i$ : Probability associated with $i^{th}$ class.

**Entropy :** Entropy is an impurity criterion that measures the randomness in the data points [10].

$$Entropy = -\sum_{i=1}^{c} P_i log_2(P_i) \tag{2}$$

where
$P_i$: Proportion of samples that belongs to class "c" for a particular node.

**Information Gain :** The entropy impurity criterion is used as a measure to calculate the information gain of the node. The lesser the entropy, the higher the information gain [6].

$$IG(D_p, f) = I(D_p) - \sum_{i=1}^{m} \frac{N_i}{N} I(D_i) \tag{3}$$

can also be formulated as

$$InformationGain = Entropy(before) - \sum_{i=1}^{m} Entropy(i, after)$$

where,
$f$: feature split on
$D_p$: Dataset of parent node
$D_i$: Dataset of the $i_th$ child node
$I$: Impurity criterion( Gini, Entropy ...)
$N$: Total number of samples

$N_i$: Number of samples $i_th$ child node.

**Gain Ratio [6]:** Is given by,

$$GainRatio = \frac{InformationGain}{SplitInfo} \tag{4}$$

$$= \frac{Entropy(before) - \sum_{i=1}^{k} Entropy(i, after)}{\sum_{i=1}^{k} P_i log_2(P_i)}$$

where
$before$: before the split of dataset
$k$: Number of subsets generated by split
$i, after$: subset after the node split

### C. Review of Related Work

*a) Fifty Years of Classification and Regression Trees:* In this paper Loh et al. [11] shares insights about various decision tree algorithms which use different splitting criteria to build a decision tree, such as Classification and Regression Tree (CART) [5], CHi-Squared Automatic Integration Detector (CHAID) [12], Iterative Ditchomiser(ID3) and C4.5 [6], Fast and Accurate Classification Tree (FACT), Quick Unbiased and Efficient Statistical Tree (QUEST) [13], Classification Rule with Unbiased Interaction Selection and Estimation (CRUISE), Generalized, Unbiased, Interaction Detection and Estimation (GUIDE) [14].

- ID3 and C4.5: Quinlan et al., ID3 uses a greedy strategy to select the best attribute that yields maximum gain eq[3] or least entropy eq[2]. Information gain is biased towards picking the features which produce the larger number of node splits. C4.5, an improvement of ID3, which under the hood uses the gain ratio eq[4] to reduce the selection bias, a transformed formula of the information gain eq[3]. The gain ratio surpasses the problem by considering the resultant number of branches before the split. Pruning is done based on an empirical formula instead of a cross-validation score. The resultant models are computationally efficient with good prediction accuracy. However, the generated size of the tree is usually considerably larger than those using other methods.
- CHi-Squared Automatic Integration Detector (CHAID): It employs a stepwise regression method for splitting a node. It handles both categorical and ordered variables along with their missing values. A node split on ordered variables produces ten child whereas a categorical split produces one node corresponding to each category. This method uses the Bonferroni test for merging child node pairs.
- FACT: Uses recursive LDA to generate linear splits. It uses ANOVA F-test to choose split variables and univariate LDA to obtain univariate splits to find the split points. FACT is biased towards categorical variables as it converts these variables to ordered by using LDA before performing F-test; however, FACT is unbiased towards ordered variables.

*b) Classifiability-based omnivariate decision trees:* Yildiz et al. [15] have proposed an omnivariate hybrid tree that incorporates nodes features in univariate (only one feature value is considered for axis splits), linear multivariate (node splits the input space into two), and nonlinear multivariate(node splits the input space arbitrarily). In this paper, the author Kothari et al. [16] propose a classification approach for model selection in building omnivariate decision trees. Results show that this approach achieved better classification accuracy, as the underlying approach takes Bayes error and boundary complexity into account along with data density which enables the classification algorithm to perform well, especially at the leaf nodes, thereby generating a supermodel at the end. Furthermore, the authors have experimented on 26 datasets, and metrics show a significant improvement in the learning speed compared with other available approaches.

*c) Split Selection Methods For Classification Trees:* Decision trees use a rigorous approach to classify the nodes, which tend to be biased towards prioritizing in choosing the variables that can offer more significant node splits. In this paper, Loh et al. [17] propose a new algorithm, QUEST, which substantially reduces the selection bias. This algorithm is proven to have comparable accuracy with the thorough approach and is significantly faster. Bias is removed by using F -tests and contingency chi-squared tests on ordered and categorical variables. As a result, a comprehensive search obtains the split point if the chosen variable is ordered. Furthermore, if a variable is categorical, it is transformed into the largest linear discriminant coordinated, giving QUEST the computational advantage and is faster than CART. Nonetheless, the pruning strategy is similar to that of CART [11].

### D. Decision Tree Algorithms

General steps involved in generating a decision tree are [5], feature selection for splitting anode. the decision of whether the node is a leaf node or parent node, assigning the correct label to the leaf node. The aim of classification problems is to predict the label of a class or feature from the predefined set of possibilities. For example, suppose a model predicts precisely two classes. In that case, it is a binary classification, and in all other cases, it is a multi-class classification. Binary classification generally deals with a Yes/No or True/False. One of the classic examples is categorizing spam emails. In this case, the classification model would ask, "Is the email belongs to the spam class ?" In this section list of decision tree algorithms is listed. However, in this paper, we extensively discuss classification problems [8].

## III. Methods/procedures

### A. Gini Index Computation

Before starting with the practical dataset example, let us understand the calculation of the Gini index using a simple dataset from [6].

Let X denote a random variable, P(X=c) is the probability that X = c, where c denotes a class of an attribute.

TABLE I
SAMPLE DATASET

| No. | Attributes | | | | |
|---|---|---|---|---|---|
| | ***Outlook*** | ***Temperature*** | ***Humidity*** | ***Windy*** | ***CanPlay?*** |
| 1 | sunny | hot | high | no | no |
| 2 | overcast | hot | high | no | yes |
| 3 | sunny | mild | normal | yes | yes |
| 4 | overcast | mild | high | yes | yes |
| 5 | rain | mild | high | yes | no |
| 6 | rain | cool | normal | yes | no |
| 7 | rain | mild | high | no | yes |
| 8 | sunny | hot | high | yes | no |
| 9 | overcast | hot | normal | no | yes |
| 10 | rain | mild | high | yes | no |

Gini Impurity Calculation for Humidity Attribute:

$$P_{Humidity}(high) = \frac{7}{10}\ ,\ P_{Humidity}(normal) = \frac{3}{10}$$

$$P(Humidity = high \text{ and } CanPlay = Yes) = \frac{3}{7}$$

$$P(Humidity = high \text{ and } CanPlay = No) = \frac{4}{7}$$

$$Gini_{high} = 1 - ((\frac{3}{7})^2 + (\frac{4}{7})^2) = 0.4898$$

$$P(Humidity = normal \text{ and } CanPlay = Yes) = \frac{2}{3}$$

$$P(Humidity = normal \text{ and } CanPlay = No) = \frac{1}{3}$$

$$Gini_{normal} = 1 - ((\frac{2}{3})^2 + (\frac{1}{3})^2) = 0.44$$

Weighted Gini average of the humidity,

$$Gini_{Humidity} = \frac{7}{10} * 0.4898 + \frac{3}{10} * 0.4444 = 0.4762$$

With a similar analysis,

Gini Impurity Calculation for Weather Attribute:

$$Gini_{weather} = \frac{3}{10} * 0.4444 + \frac{3}{10} * \mathbf{0} + \frac{4}{10} * 0.375 = \mathbf{0.2833}$$

Gini Impurity Calculation for Temperature Attribute:

$$Gini_{Temperature} = \frac{4}{10} * 0.5 + \frac{5}{10} * 0.480 + \frac{1}{10} * 0 = 0.44$$

Gini Impurity Calculation for Wind Attribute:

$$Gini_{Wind} = \frac{6}{10} * 0.5556 + \frac{4}{10} * 0.375 = 0.4836$$

After calculating the Gini impurity of every attribute present in the dataset, we now compare them and choose the one having the minimum Gini value, which forms the base for the first split. Hence, the split will happen on the weather attribute as its Gini index is the smallest. After that, the weather category will branch out into sunny, overcast, and cloudy. On every child node, check if the node is pure; otherwise, continue splitting until the node contains minimum impurities. For instance, Overcast is fully classified with the first split as the Gini index also shows zero impurities; on the other hand, the split corresponding to sunny and rain is not pure. Recurse the steps and compute the Gini index on each impure node until the data is classified. Eventually, after splitting on all the features, the final decision tree would look like Fig: 3. With this understanding, we now move on to the next section III-B where we discuss exclusively on node splitting [18].

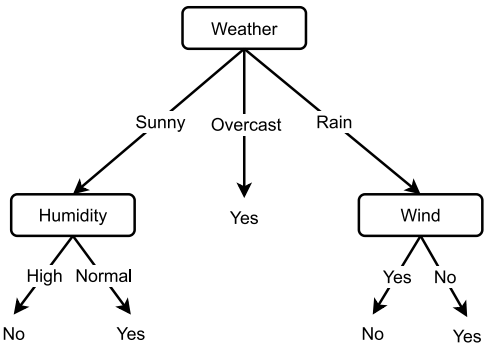


Fig. 3. Decision Tree from the Weather Dataset [6]

### B. Understanding Node Splitting

Several exhaustive algorithms are available for selecting the optimal split point for node splits [17]. However, we will only focus on understanding how decision boundary changes with node split in the CART algorithm with the help of the most popular iris dataset. Iris dataset has three species, 'Versicolor,' 'Setosa,' and 'Virginia,' with several attributes such as sepal and petal length. E.g., if we want to classify a flower whose petal length of 4.5 cm. The question that the model would ask is "whether the length of petal $\leq$ 2.45". The length is greater here (4.5 cm), so the answer would be False. Then the classification model would pick the next optimal split point or threshold value which checks whether petal length (cm) $\leq$ 4.95". As this evaluation returns True, flower species is predicted as Versicolor.

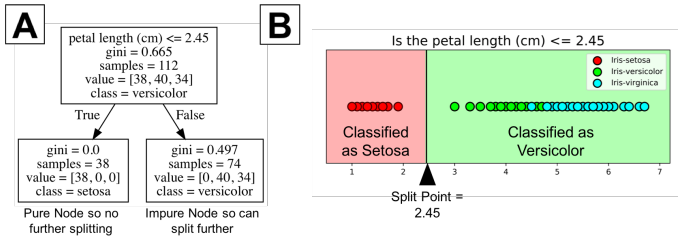


Fig. 4. Node Splitting based on feature and threshold - Step1 [19]

Furthermore, to understand how a split decision, let us look at the partial decision tree in Fig: 4. The conditional statement "petal length (cm)$\leq$ 2.45" splits the data into two based on a value (which is 2.45 in this scenario). An optimal split point value results in the highest information gain eq[3], which leads to more homogeneous nodes. From Fig:4we can infer all the points to the right of the split point are classified as Versicolor, whereas points to the left are labeled as Setosa [19].

From Fig:5, the tree has a maximum depth of two. The depth of the tree influences the number of possible splits

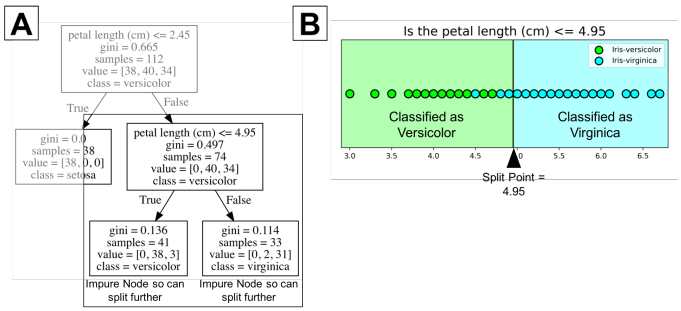


Fig. 5. Node Splitting based on feature and threshold - Step2 [19]

before making a prediction. Then, it is repeated until the tree becomes homogeneous. These recursive splits can make the tree complex and deep with many nodes, often overfitting the training dataset.

### C. Decision Tree CART Algorithm

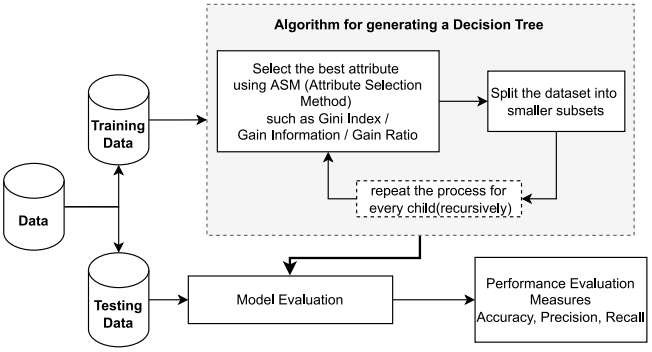


Fig. 6. Decision Tree Algorithm [5]

Fig: 6 illustrate the graphical representation of steps involved in splitting the nodes and how a decision tree is built. This involves the splitting of features inside the dataset into train and validation(test) data. After that, training data is utilized for building the model where attributes are picked based on feature importance and selection measures. Once the model is trained completely, the built model is then evaluated on the validation data for making predictions on the new unknown date, which determines the model's accuracy.

Classification and Regression Tree (CART): Pursues the same greedy approach as Automatic Interaction Detection(AID) and Theta AID(THAID) while adding significant improvements. In this method, trees are pruned instead of halting the tree growth with some rules. Pruning is done by cutting the weakest node link by utilizing the cross-validation score. Although it solves the over and under-fitting problems of AID and THAID, it comes at the cost of computation. This method uses gini metric eq[1] and relies on surrogate splits where the missing values are also handled. CART is biased towards selection, as an ordered K variable with 'i' unique values yields '(i-1)' splits, whereas a categorical variable with j unique values yields $(2^{j-1})-1$ splits [11]. For detailed CART algorithm [20] steps refer: Algorithm [1].

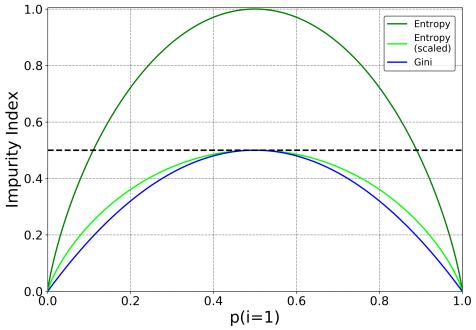


Fig. 7. Gini Vs Scaled Entropy [18]

Fig:7 on the Y-axis, we have an impurity measure, and on the X-axis, we have the probability of classifying a random sample. Entropy is 0 if the sample node is pure, whose probability is either 0 or 1, and entropy is 0.5 if the sample has an even distribution across the classes. In practice, the Gini eq[1] index and entropy eq[2] are comparable impurity criteria and yield similar results in CART algorithm computation [18].

**Algorithm 1** CART - Algorithm

1: Data at node s be represented by $Q_s$ with $n_s$ samples; ▷ start from an empty tree
2: **for** each candidate split $\theta = (k, t_s)$ consisting of a feature k and threshold $t_s$ partition the data into $Q_s^{left}(\theta)$ and $Q_s^{right}(\theta)$ subsets **do**

$$Q_s^{left}(\theta) = \{(u, v)|u_k \leq t_s\}$$
$$Q_s^{right}(\theta) = Q_s \setminus Q_s^{left}(\theta)$$

3: Partition the features based on the next best feature and expand tree for each split
4: Split of node m uses impurity or loss function to pick the best candidate.

$$G(Q_s, \theta) = \frac{n_s^{left}}{n_s} H(Q_s^{left}(\theta)) + \frac{n_s^{right}}{n_s} H(Q_s^{right}(\theta))) \quad (5)$$

5: Choose the attributes that minimises the impurity $\theta^* = argmin_\theta G(Q_s, \theta)$
6: **end for**
7: **Recurse** for subsets $Q_s^{left}(\theta^*)$ and $Q_s^{right}(\theta^*)$until $depth = max\_depth$

### D. Implementation on Red Wine Dataset

We now inspect the red wine dataset [21] downloaded from the [22].

*a) General information on the dataset:* The classes are unbalanced as there are fewer good wines than normal wine samples.

- Number of Red Wine Samples: 1599
- Number of Attributes: 11 (Input) + 1 (Output) attribute
- Missing Attribute Values: None
- Input variables:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   fixed acidity         1599 non-null   float64
 1   volatile acidity      1599 non-null   float64
 2   citric acid           1599 non-null   float64
 3   residual sugar        1599 non-null   float64
 4   chlorides             1599 non-null   float64
 5   free sulfur dioxide   1599 non-null   float64
 6   total sulfur dioxide  1599 non-null   float64
 7   density               1599 non-null   float64
 8   pH                    1599 non-null   float64
 9   sulphates             1599 non-null   float64
 10  alcohol               1599 non-null   float64
 11  quality               1599 non-null   int64
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

Fig. 8. Attributes inside red wine dataset [23]

*b) Data Exploration:* After a quick inspection of Fig: 9, it was found that correlation exists inside the data; we can infer that alcohol is strongly correlated with wine quality, and the correlation is positive.


Fig. 9. Correlation Heat Map[Own Creation]

*c) Data Wrangling:* Check for missing values using the command $df.isnull().sum().sum()$ which outputs **0**, indicating that there are no missing values inside the dataset. From the Fig:10 bar graph, one can infer that the low-quality wine contains less percentage of alcohol and good ones have more. Furthermore, we can inspect the top features which affect the wine quality, as they primary influence in classifying the wine into "good" or "regular" ones.

**Data preparation for Model Classification :** The ultimate goal is to classify the wine to be either a good one or not. We see the data range is not uniform. Hence the data needs to be preprocessed before creating the classification model.

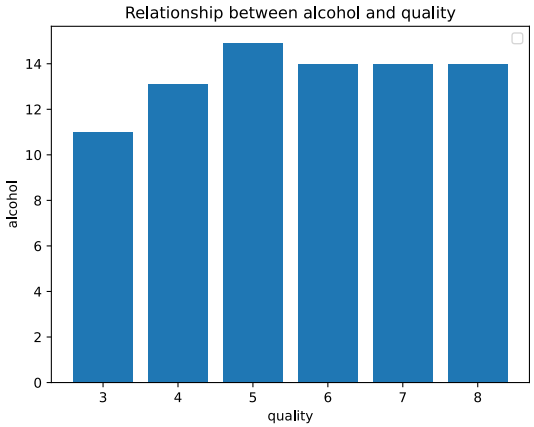


Fig. 10. Bargraph: Alcohol Level vs. Wine Quality[Own Creation]

Attributes such as volatile acidity have ranged between 0 and 1, whereas sulfur dioxide has an uneven distribution of values, some closer to 100 and others under 10. To overcome this issue, data is normalized between 0 to 1. Fig: 11 shows the data after normalization.

```
   fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0       0.247788          0.397260         0.00        0.068493   0.106845
1       0.283186          0.520548         0.00        0.116438   0.143573
2       0.283186          0.438356         0.04        0.095890   0.133556
3       0.584071          0.109589         0.56        0.068493   0.105175
4       0.247788          0.397260         0.00        0.068493   0.106845

   free sulfur dioxide  total sulfur dioxide   density        pH  sulphates  \
0             0.140845              0.098940  0.567548  0.606299   0.137725
1             0.338028              0.215548  0.494126  0.362205   0.209581
2             0.197183              0.169611  0.508811  0.409449   0.191617
3             0.225352              0.190813  0.582232  0.330709   0.149701
4             0.140845              0.098940  0.567548  0.606299   0.137725

    alcohol  quality
0  0.153846      0.4
1  0.215385      0.4
2  0.215385      0.4
3  0.215385      0.6
4  0.153846      0.4
```

Fig. 11. Data Normalization [23]

As the next step of model preparation, we now separate the target variable from the dataset and copy the "quality" attribute into the 'Y' variable and the rest of the attributes into X, which are used for predicting our target variable's wine quality. The model classifies the wine as good if the quality is seven or above and 'no' otherwise. We will now skim the "quality" attribute to see if enough good wine exists.

**Output:** The outcome Fig:12 looks bit unbalanced, at least we have 217 good wine samples.

**Baseline Classification using DummyClassifier**: The next step is to partition the dataset into train and test (validation) data where X variable contains all predictors, and the target variable is Y. Calculate the baseline accuracy using the most frequent strategy to classify a label with DummyClassifier and resulted accuracy: = **86.45%**

**Modeling of a decision tree using skearn python library:**

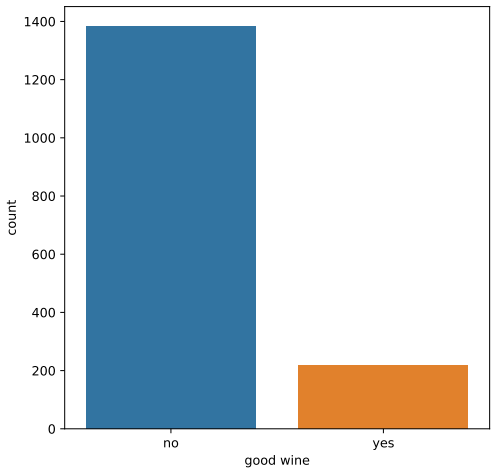


Fig. 12. Good Wine Count[Own Creation]

Listing 1. DecisionTreeClassifier

```
DecisionTreeClassifier(random_state=10)
dt.fit(X_train, y_train)
y_pred = dt.predict(X_test)
tree.plot_tree(dt, feature_names=fn,
               class_names=cn, filled=False)
```

As discussed in section III-C, once the model is built, it is used for making predictions on the validation data.

## IV. Discussion of Results

Sklearn python library offers accuracy metrics that can be used for checking the model accuracy. This metric is used to check the accuracy of the model in predicting the quality of the red wine dataset. The created model is then used to perform the predictions on the validation data, where the output of the model accuracy is: **88.12%**, which is better than the baseline accuracy. However, after examining the tree, the structure turned out to be quite complex (due to the huge size of the tree, the image is not referred here, but it is part of jupyter notebook).

Instead, the tree parameters are utilized to give a brief outline of the outcome, **node values: 201** and **depth: 14** which indicates that the tree is overfitting on the train data. Fig: 13 shows how the accuracy of the tree is affected by the increase in the number of nodes; this happens mainly due to overfitting [18]. Fig.14 depicts the confusion matrix, we can see that 388 samples are labeled as true negatives, which refers to the normal wine quality, and 35 samples are classified as good wine. If we look in detail, 27 regular wine samples are classified as 'Good wine'; in reality, they are 'regular wine' (false positives: Type 1 error), and 30 'Good Wine' samples are predicted as 'Regular Wine', but in reality, they belong to 'good wine' samples (false negatives: Type 2 error).

We now inspect the cross-validation score: 80.5%, which

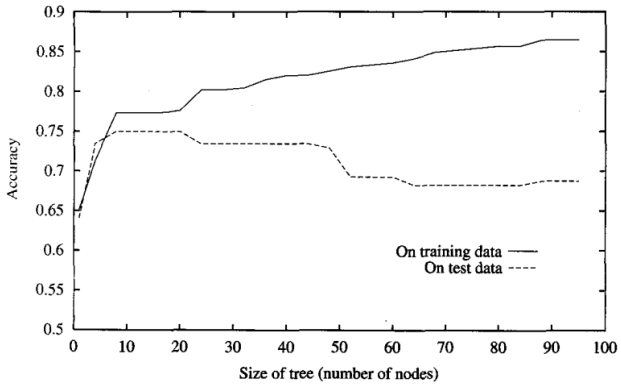


Fig. 13. Accuracy of decision tree based on number of nodes [18]

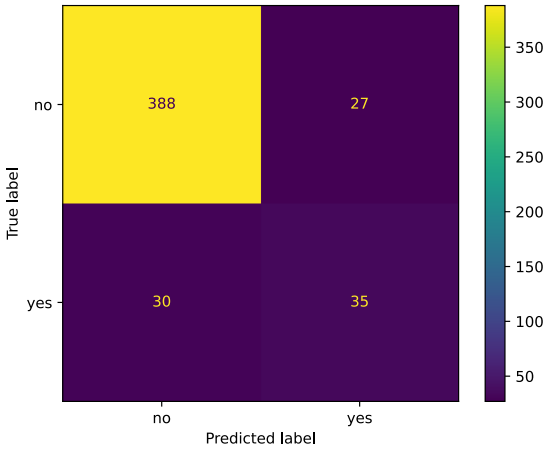


Fig. 14. Confusion Matrix [Own Creation]

is less than the baseline accuracy, while the training accuracy is 100%, which perfectly predicts the train data. In the next step, we will tune the model hyperparameters to overcome the overfitting issue we have seen in the above discussion. GridSearchCV library offers a novel approach to tuning the decision tree parameters to find optimal parameters for the given data. **Result :** 'criterion': 'gini', 'max_depth': 2, 'max_features': 8. We will now use the above result parameters to model the new decision tree classifier. With this approach, results indicate that the model's accuracy has improved.

Metrics of the new approach, model accuracy is **88.54%** and the new cross-validation score: 87.86%. Fig:15 illustrates the new decision tree after tuning the hyperparameters, resolving the overfitting issue observed to a certain extent. New tree depth: 2 and nodes: 7, tree accuracy has not improved significantly, but there is a decent improvement in the cross-validation score, and the complexity of the tree has been reduced. The same red wine dataset has been used to create a random forest model. The model accuracy, in this case, is **91.99%**, which is reasonably higher than the decision tree

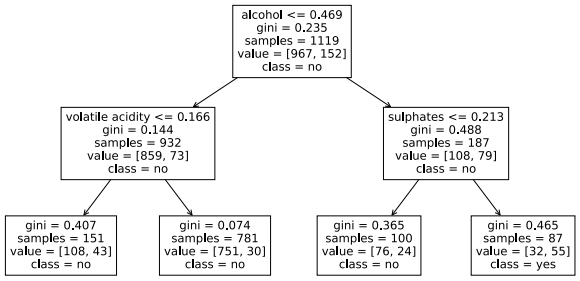


Fig. 15. Decision Tree for Red Wine Dataset [Own Creation]

model.

## V. Conclusion

In this paper, various aspects of the decision trees are discussed, starting with impurity criteria which is the basis for decision tree node splitting; the discussion also covers various tree algorithms along with a review of related papers and a practical implementation of the decision tree with red wine dataset using the scikit library sklearn in python.

Decision trees are clear to understand and construct, and they also mimic human thinking of answering a 'yes' or 'no' question. Nonetheless, one of the significant drawbacks is overfitting which impacts the model accuracy. There are other practical issues, such as choosing suitable attribute selection measures and handling of missing and continuous variables that might present unknown data. On the other hand, random forest overcomes the overfitting problem by combing the outputs of the different decision tree sub-units to make a final prediction.

To conclude, the random forest ensemble method has become a de facto technique in supervised learning due to its good accuracy, but this comes at an increased computation cost. Decision trees often encounter the overfitting issue, which can be solved by pruning or tuning the hyperparameters of the tree. Nevertheless, for someone who wants to understand the underlying concepts of random forests and supervised learning with trees, the decision tree will always lay the foundation for that.

## Acknowledgment

The author would like to thank Prof. Dr. Martin Simon for his guidance and constructive suggestions for this research work. This paper is the final report for the course 'Learning from Data', coursework in the Master's program High Integrity Systems(HIS).

## References

[1] L. Rokach and O. Maimon, *Data Mining and Knowledge Discovery Handbook*. Boston, MA: Springer US, 2005, pp. 165–192. [Online]. Available: $https://doi.org/10.1007/0-387-25465-X_9$

[2] T. Hastie, R. Tibshirani, J. H. Friedman, and J. H. Friedman, *The elements of statistical learning: data mining, inference, and prediction*. Springer, 2009, vol. 2.

[3] N. S. Chauhan, "Decision Tree Algorithm, Explained - KDnuggets," https://www.kdnuggets.com/2020/01/decision-tree-algorithm-explained.html, [Online; accessed 2022-09-23].

[4] A. Sharma, "Decision tree split methods," https://www.analyticsvidhya.com/blog/2020/06/4-ways-split-decision-tree/, Jun. 2020, accessed: 2022-9-24.

[5] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Chapman and Hall/CRC, 1983.

[6] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, pp. 81–106, 2004.

[7] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining (2nd Edition)*. Pearson.

[8] A. Müller and S. Guido, "Introduction to machine learning with python: A guide for data scientists," 2016.

[9] Richard J. Light and Barry H. Margolin, "An analysis of variance for categorical data," *Journal of the American Statistical Association*, vol. 66, no. 335, pp. 534–544, 1971. [Online]. Available: https://www.tandfonline.com/doi/abs/10.1080/01621459.1971.10482297

[10] R. Messenger and L. Mandell, "A modal search technique for predictive nominal scale multivariate analysis," *Journal of the American Statistical Association*, vol. 67, no. 340, pp. 768–772, 1972. [Online]. Available: https://doi.org/10.1080/01621459.1972.10481290

[11] W.-Y. Loh, "Fifty years of classification and regression trees," *International Statistical Review*, vol. 82, 06 2014.

[12] G. V. Kass, "An exploratory technique for investigating large quantities of categorical data," *Journal of The Royal Statistical Society Series C-applied Statistics*, vol. 29, pp. 119–127, 1980.

[13] H. Kim, W.-Y. Loh, Y.-S. Shih, and P. Chaudhuri, "Visualizable and interpretable regression models with good prediction power," *Iie Transactions*, vol. 39, pp. 565–579, 03 2007.

[14] W.-Y. Loh, "Improving the precision of classification trees," *Annals of Applied Statistics*, vol. 3, 11 2010.

[15] O. T. Yildiz and E. Alpaydin, "Omnivariate decision trees," *IEEE transactions on neural networks*, vol. 12 6, pp. 1539–46, 2001.

[16] Y. Li, M. Dong, and R. Kothari, "Classifiability-based omnivariate decision trees," *IEEE Transactions on Neural Networks*, vol. 16, pp. 1547–1560, 2005.

[17] W.-Y. Loh and Y.-s. Shih, "Split selection methods for classification trees," *Stat Sinica*, vol. 7, 07 1999.

[18] T. M. Mitchell and T. M. Mitchell, *Machine learning*. McGraw-hill New York, 1997, vol. 1, no. 9.

[19] M. Galarnyk, "Understanding decision trees for classification in python." [Online]. Available: https://www.kdnuggets.com/2019/08/understanding-decision-trees-classification-python.html

[20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[21] P. Cortez, A. Cerdeira, F. Almeida, T. Matos, and J. Reis, "Modeling wine preferences by data mining from physicochemical properties," *Decision Support Systems*, vol. 47, no. 4, pp. 547–553, 2009, smart Business Networks: Concepts and Empirical Evidence. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167923609001377

[22] "UCI machine learning repository: Wine quality data set," https://archive.ics.uci.edu/ml/datasets/wine+quality, accessed: 2022-9-25.

[23] J. Fang, "Predict red wine quality with SVC, decision tree and random forest," https://allysonf.medium.com/predict-red-wine-quality-with-svc-decision-tree-and-random-forest-24f83b5f3408, May 2021, accessed: 2022-9-25.