University of Côte d'Azur

University of L'Aquila

Double-Degree Master's Programme - MathMods
Mathematical Modelling in Engineering: Theory, Numerics, Applications

Master
Mathematics and Applications

UNIVERSITY OF CÔTE D'AZUR (UCA)

Laurea Magistrale
Mathematical Modelling

UNIVERSITY OF L'AQUILA (UAQ)

Master's Thesis

DECISION TREES (CART) & BOOSTING METHODS

Supervisor
Prof. Sylvain Rubenthaler

[illegible]

Candidate
Bhargav Ramudu Manam

[illegible]

Student ID (UAQ): 274435
Student ID (UCA): 22109155

Academic Year 2021/2022

Université Côte d'Azur

Master 2 Thesis

DECISION TREES (CART) & BOOSTING METHODS

Author:
Bhargav Ramudu Manam

Supervisor:
Prof. Sylvain Rubenthaler

A thesis submitted in fulfilment of the requirements for the degree of Master 2 MPA

in the

Laboratoire J.A. Dieudonné

Nice, 02/09/2022

Declaration of Authorship

I, BHARGAV RAMUDU MANAM, declare that this thesis titled, '*DECISION TREES (CART) & BOOSTING METHODS*' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: ____

Date: ____

Contents

List of Figures

Abstract

Tree based methods are becoming increasingly attractive for data scientists, as a favoured algorithm for supervised learning models. This is mainly because of their simpler construction, lower computation times and easy interpretation. In this thesis, Classification and Regression Trees (CART) are discussed in general context. A comprehensive methodology for tree building and pruning are detailed. Two powerful ensemble methods, Adaptive Boosting and Gradient Boosting are discussed in detail with respective algorithms.

Various tree based models are explored for an multi-class classification problem with 5 classes. The models are first tuned with respect to (w.r.t) the tree parameters, maximum tree depth and maximum node size, and then solved for the hyper-parameters using regularization techniques like shrinkage and sub-sampling. A comparison has been made among trees, boosting methods and aggregate ensemble methods like , bagging and random forests.

Keywords: Decision Trees, CART, Classification, Regression, Machine Learning, Data Science, Cross-Validation, Cost-complexity Pruning, Weakest Link Cutting, Hyper-Parameter Tuning, Ensemble Methods, Boosting, Bagging, Random Forests, Adaptive Boosting, Stochastic Gradient Boosting, Shrinkage, Sub-Sampling

Acknowledgements

At First, I would like to offer my sincere thanks and gratitude to my supervisor, Prof. Sylvain Rubenthaler, for the guidance and support throughout the course of my thesis. The technical inputs and the discussions we had, helped to clarify my apprehensions and grasp the topics quickly. I highly regard, your time and report review (even during your summer break), as very critical in successful completion of the thesis.

Secondly, I like to convey my admiration and indebtedness to my program coordinator, Prof. François Delarue, for all the guidance and support you have shown throughout the year. Also, your lectures are very pleasant, interesting and motivated my approach on problem conceptualization. Along with Prof. Indira Chatterji and Ms. Fatima Zarour, I thank you for the assistance with administrative procedures that made my integration and student experience very enjoyable in Nice.

Additionally, I thank Prof. Etienne Tanré, for the passionate lectures and extraordinary support during and after lessons. Your attention to the finite details and graphical simulation helped a lot in assimilation of the ideas. I render thanks to Prof. Damien Garreau, for the eloquent teachings. Your preciseness and amicable atmosphere immensely improved my subject comprehension. A big thanks to Prof. Rémi Catellier, without the knowledge from your course I would not have proceeded this far. Your rigorous exercise lessons and incredible patience are to be credited for my better understanding of Stochastic Theory.

Special thanks to Athanasios Vasileiadis, who has made an pivotal contribution in shaping my thesis work. Meetings with you gave me a clear idea in organizing my thesis. It has been a pleasure to work with you and hope to catch up with your brains in future.

Most importantly, I express my utmost gratefulness to Prof. Bruno Rubino and the entire Mathmods team (academic and staff), for giving me the opportunity to participate in this program. I also appreciate the great hospitality and care taken by Prof. Ingenuin Gasser, Ms. Astrid Benz and her team, Henda and Helen, during my stay at Universität Hamburg for second semester. Studying in three great European countries gave me a chance to explore the traditions and culture of these nations and they have clearly expanded my horizons and perspectives.

Last but not the least, I thank my parents and family members beyond words for the love, trust and belief in my abilities. I acknowledge the immeasurable personal contributions of all my wonderful friends and especially of those I have met during these last two years.

Chapter 1

Introduction

1.1 Objectives:

The main objectives of this thesis are:

- To introduce Decision Trees (CART) and explain their systematic construction.
- To briefly present the Ensemble Methods and discuss the Boosting Methodology with Decision Trees as base estimator.
- To implement various Tree based algorithms to [Arbres-urbains] Data set and understand the feature importance.

The basic theory of the classifiers, regressors is provided in the appendix (.2) of this thesis report. It can be referenced for the background knowledge behind predictors, accuracy estimating methods and scaling of the errors. Chapter (2) introduces the trees, their structure and addresses the key points one should take into account before getting started with trees.

Chapter (3) details about the methodology followed for optimal tree selection through the concepts of Pruning and Splitting Criteria. Weakest link cutting as a means to prune the tree and robust criteria for splitting the nodes are discussed in this regard.

Chapter (4) is about the Boosting methods that follow Forward Stagewise Additive Modelling (FSAM). In Chapter (5), several tree algorithm models are built to analyze the [Arbres-urbains] Data set are the results are presented. The Python code used for the building the models and analysis can be found in the appendix (.1).

Remark 1.1. *After a thorough Literature Review, I have sourced all the the Definitions, Propositions, Theorems (including proofs) and Figures, in Chapters (2), (3) and (.2), from a foundational book in decision trees by Breiman et al.. Even though, most of them* *are simple and easy-to-follow, their usage meant for a better consistency and structure of the thesis.*

Remark 1.2. *For convenience, several topics are explained either in the context of classification (mostly) or regression, the application of the same criteria to the other kind of problem can be assumed in similarity.*

Remark 1.3. *The framework of the code used in Python is mostly from scikit-learn [Pedregosa et al., 2011] module and I extensively used the coding logic from the lecture notes [Rubenthaler, 2021] and the sk-learn MOOC course, [by Inria Learning Lab], which I have successfully participated during this thesis.*

Chapter 2

Decision Trees

2.1 What are Decision Trees?

Decision Trees are a Supervised Machine Learning technique used in making predictions for both Classification and Regression problems. They are flow chart like models which perform cut-based analysis on feature space or decision attributes using simple decision rules for analyzing and stratifying or segmenting the data into set of partitions (rectangles) and then fitting the model by using a constant value, either mean for regression problems or mode for classification problems.

There are various Tree algorithms in existence like $ID3$ (Iterative Dichotomiser 3) invented by Ross Quinlan Quinlan [b], which was extended to $C4.5$ Quinlan [a] and further to $C5.0$ (under a proprietary license). These algorithms differ from each other majorly based upon how the trees are subjected to one or more of the following: Pruning, Splitting rules and Stopping criteria, all of which will be discussed later part of this section. But in our study, we based our results and discussion by working with Classification and Regression Trees (CART) Algorithm, which was introduced and developed by Friedman of [Breiman et al., 1984]. $C4.5$ is one of the major competitors to CART Algorithm but technically they are very similar in construction.

As the name suggests, CART can be used for prediction of both Regression and Classification tasks and has its wide applications in the areas of Botany, High Energy Physics, Medical Diagnosis and Optical Character Recognition (OCR) of handwritten text etc.

2.1.1 Schematic Representation and Terminology of CART:

As illustrated in the figure (2.1), the CART algorithm grows the decision tree down the page, resembling a tree structure which is upside down. Even though it may sound trivial, the following terminology of trees are first defined by Breiman and is consistently in use since then.

So, a typical tree starts with the root node at the top, where the entire data set related to predictor space is passed through it. Then the tree is propagated down by recursively splitting at each intermediate point (on either sides of the root node) called internal nodes, where a decision is made to further split the data set until a final stopping criterion is reached or the node is pure. These last nodes are called as terminal nodes or leaves. Each leaf is assigned with a class label or constant value in case of regression trees. Those segments, generated on either side of the nodes are referred to as branches.

Any tree that can be formed by just cutting out a portion of the original parent tree at any node is called a subtree. Rooted subtree is the subtree which has the same root node as of its parent tree.

2.1.2 Flow of Decision Tree Output:

The decision tree output for a particular instance is determined by how its features behave in the tree:

(i) The first decision criterion is applied at the root node for a predetermined parameter from the feature space.

(ii) Instance is passed down the tree through one of its two branches (binary splits) depending on whether it succeeds or not the decision test.

(iii) Again a new decision criterion is applied at this new internal node and again the instance will go either left or right based upon the test result.

(iv) Repeat step (*iii*) until the instance ends up in a leaf/ terminal node.

(v) The decision tree output for the instance is the value associated with this leaf.

2.2 Tree Structured Classifiers:

Now that we have known the structure and functioning of the decision tree. In this section, we will discuss about the tree structured classifiers and list out the important factors involved in their construction.

In figure (2.1), a hypothetical structured classifier is constructed for a set X consisting of six classes, let us say, $\{1, 2, 3, 4, 5, 6\}$. Starting with X, the data points within X, are separated through binary splits, by some means or measure(will be discussed in the

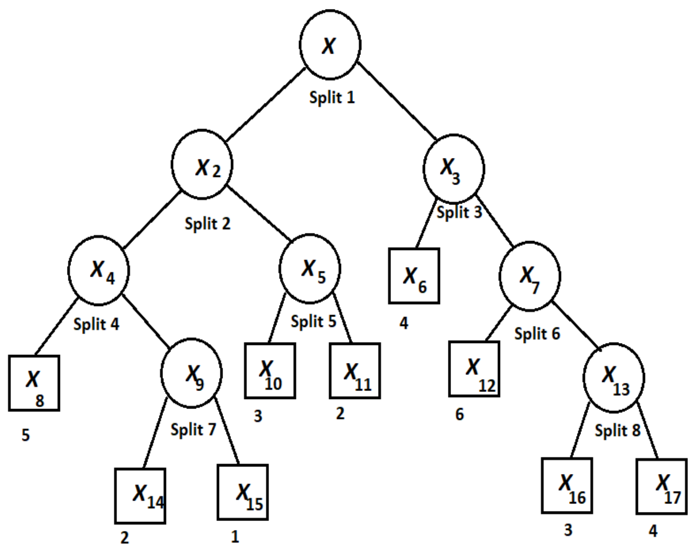


FIGURE 2.1: Schematic representation of a tree structured classifier

following section), into several subsets(mutually exclusive) by continuously segregating them into finer subsets until no more separation (based on some measure) is possible.

Suppose, if you consider the initial Split 1, X_2 and X_3 are disjoint : $X = X_1 \cup X_2$, likewise, $X_2 = X_4 \cup X_5$, $X_3 = X_6 \cup X_7$ and so on. All those subsets that cannot be split anymore $(X_6, X_8, X_{10}, X_{11}, X_{12}, X_{14}, X_{15}, X_{16}$ and $X_{17})$, are called terminal subsets (rectangular boxes in figure (2.1)) and all other subsets are non-terminal subsets (indicated by circles).

Each of the terminal subsets are designated by respective class label, thus forming the partition of the root subset X. They are given as Class $1 = X_{15}$, Class $2 = X_{11} \cup X_{14}$, Class $3 = X_{10} \cup X_{16}$, Class $4 = X_6 \cup X_{17}$, Class $5 = X_8$, and Class $6 = X_{12}$.

From the previous example discussed above, we have the basic understanding of the tree classifier. Now we focus on the following three key elements involved in tree construction, which are addressed in the next section; initial tree growing methodology, they revolve around using the data from the learning sample $\mathcal{L}$ to formulate rules that:

(i) decide on the selection of the splits.

(ii) decide when to declare a node as a terminal node or to continue splitting it.

(iii) decide on how to assign each terminal node to a particular class.

2.2.1 Initial Tree Growing Methodology:

Let us consider a learning sample $\mathcal{L}$ as in definition (.8) (consisting of J classes) of size N. N_j be the number of cases in the j^{th} class. The prior probabilities are given by:

$$\pi(j) = \frac{N_j}{N} \qquad \forall j \in 1, 2, \ldots, J. \tag{2.1}$$

Consider any node t, and denote $N(t)$ to be the total number of cases at t and $N_j(t)$ to be the total cases at node t in class j.
Then we have the following:

(i) The estimated probability of a case belonging both to j^{th} class and node t is:

$$p(j,t) = \pi(j)N_j(t)/N_j \tag{2.2}$$

(ii) The estimated probability of any case falling into node t is:

$$p(t) = \sum_j p(j,t) \tag{2.3}$$

(iii) The estimated probability of case belonging to j^{th} class given that the case is already at node t is:

$$p(j|t) = p(j,t)/p(t) \qquad \textit{such that} \sum_j p(j|t) = 1. \tag{2.4}$$

The four elements needed in the initial tree growing procedure were:

(1) A standard set of questions to make the splits at every node.

(2) A goodness of split criterion that can be evaluated at every node.

(3) A stop-splitting rule.

(4) A rule for assigning every terminal node to a class j, where $j \in C$

These four elements are detailed in the following subsections.

2.2.1.1 The Standard Set of Questions:

Let the measurement vector x, of M- dimension relating to the data be:

$$x = (x_1, x_2, \ldots, x_M)$$

The variables $x_1, x_2, \ldots, x_M$. be either ordered or categorical type. Then the standard set of questions, denoted by Q, is defined as follows:

(i) At each node t, the split s depends on the value of only a single variable.

(ii) If any variable x_m is of ordered type, then Q includes all questions of the form $(Is\ x_m \leq c?) \forall c \in \mathbb{R}$.

(iii) If any variable x_m is of categorical type, taking values, let us say, in $b_1, b_2, \ldots, b_L$, then Q includes all questions of the form $(Is\ x_m \in s?)$, where s is the set consisting of all possible subsets of $b_1, b_2, \ldots, b_L$.

It might appear there might be infinite number of distinct splits possible since "c" above varies over a real line, $\mathbb{R}$, but in fact there are only finite number of splits possible both for ordered and categorical variable types. But they increase at a quick rate depending on the size (N) of the data.

For any ordered variable x_m, at most N different splits are generated by Q (Is $x_m \leq c$?). These are given by (Is $x_m \leq c_n$?), $\forall n = 1, \ldots, N' \leq N$, here the c_n's are taken halfway between consecutive distinct data values of x_m.

For any categorical variable x_m, the questions $(x_m \in s)$ and $(x_m \notin s)$ generate the same split with t_L and t_R reversed, if x_m has L distinct values, then $2^{L-1} - 1$ splits are defined on the values of x_m.

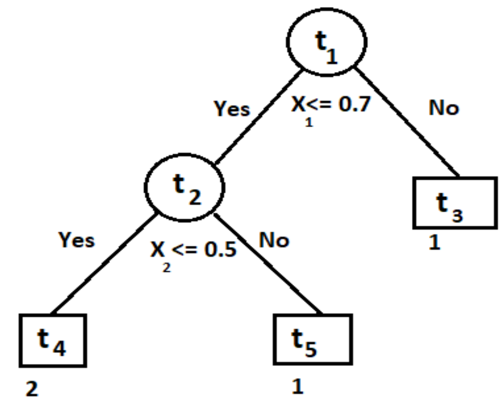
At each node, the CART algorithm incorporates these set of questions, for every variable, beginning with x_1 and continuing up to x_M. For each variable it finds the best split. Then it compares these M best splits (one for every variable) and selects the best among the best splits as the criteria to split the node.

Remark 2.1. *Above we mentioned best split but did not specify the measure to determine the best split, which will be discussed in later sections.*

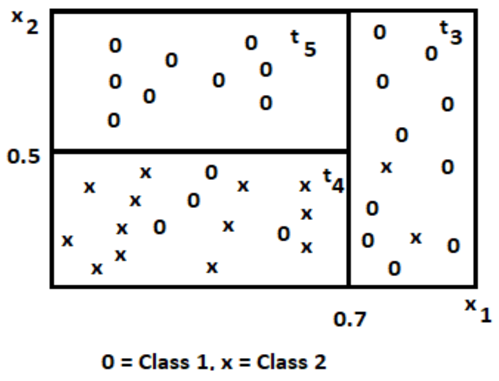
Special case (with only ordered variables):

Suppose all the independent variables are of ordered type, which is the case in this project as scikit-learn[Pedregosa et al., 2011] implementation of CART algorithm in Python does not support categorical variables (but they can be encoded into ordinal numbers), then the tree structured procedure results in recursive partitioning of the measurement space into rectangular boxes whose edges form the decision surfaces.

Consider a simple two-class problem with two ordered variables x_1 and x_2 with $0 \leq x_i \leq 1, i = 1, 2$. Let a hypothetical tree diagram for this problem look like in the figure (2.2a).



(A) Splits in case of only ordered variables



(B) Decision boundaries for two simple splits with only ordered variables

FIGURE 2.2: Splitting for numerical variables (2-dimension)

Another way of looking at this tree is that it divides the unit square formed by x_1 and x_2 into boxes as shown in figure (2.2b). The tree procedure aims to create a sequence of rectangles by recursively partitioning X such that the populations within each rectangle become more and more class homogeneous.

2.2.1.2 The Splitting and Stop-Splitting Rule:

First, we need to define impurity function and impurity measure in order to evaluate the goodness of split criterion at each node.

Definition 2.2. *An impurity function is a function ϕ defined on the set of all J-tuples of numbers $(p_1, \dots, p_J)$ satisfying $0 \le p_j \le 1, j = 1, \dots, J$, such that $\sum_j p_j = 1$ with the properties:*

(i) *ϕ is a maximum only at the point $(1/j, 1/j, \dots 1/j)$,*

(ii) *achieves its minimum only at the points $(1, 0, \dots, 0), (0, 1, 0, \dots, 0), \dots, (0, 0, \dots, 0, 1)$,*

(iii) *ϕ is a symmetric function of $p_1, \dots, p_j$.*

Definition 2.3. *Given an impurity function ϕ, define the impurity measure $i(t)$ of any node t as:*

$$i(t) = \phi(p(1|t), p(2|t), \dots, p(J|t)) \tag{2.5}$$

If a split s at node t sends a proportion p_R of the cases in t to t_R and the proportion p_L to t_L,

Definition 2.4. *Then the goodness of split $\phi(s,t)$, is defined as the decrease in impurity $\Delta i(s,t)$ given by:*

$$\phi(s,t) = \Delta i(s,t) = i(t) – p_R i(t_R) – p_L i(t_L) \tag{2.6}$$

Suppose that we have proceeded with growing the tree by randomly splitting among the variables and say, we have reached a current set of terminal nodes, denoted by $\widetilde{T}$. Then the resultant binary tree T, is defined by the set of splits performed and the order of those splits.

Definition 2.5. *The tree impurity, denoted by $I(T)$, with set of terminal nodes $\widetilde{T}$ is given by:*

$$I(T) = \sum_{t \in \widetilde{T}} I(t) = \sum_{t \in \widetilde{T}} i(t)p(t) \tag{2.7}$$

The term, $I(t) = i(t)p(t)$, for all $t \in \widetilde{T}$ is the proportion of total tree impurity at each of these terminal nodes. Further, if you take any node $t \in \widetilde{T}$ and split it into t_L and t_R using a new split s. The new tree $T^{'}$ has the impurity:

$$I(T^{'}) = \sum_{\widetilde{T}-\{t\}} I(t) + I(t_L) + I(t_R) \tag{2.8}$$

The decrease in the tree impurity is as follows:

$$I(T) - I(T^{'}) = I(t) - I(t_L) - I(t_R).$$

The above expression only depends on the node t and split s. Hence, the decrease in the tree impurity as a function of t and s is given by:

$$\Delta I(s,t) = I(t) - I(t_L) - I(t_R) \tag{2.9}$$

The proportions of node t population that go to t_L and t_R are given by p_L and p_R respectively, evaluated as:

$$p_L = p(t_L)/p(t)$$

$$p_R = p(t_R)/p(t)$$

$$then, p_L + p_R = 1.$$

And, equation (2.9) can be written as:

$$\Delta I(s,t) = [i(t) - p_L i(t_L) - p_R i(t_R)]p(t) = \Delta i(s,t)p(t) \tag{2.10}$$

As $\Delta I(s,t)$ and $\Delta i(s,t)$ are differed only by the factor $p(t)$, the same split s^* (maximizer over all s) will maximize both expressions. This repeated attempt to minimize overall tree impurity $I(T)$ is called as the split selection procedure. Selecting the splits that maximize $\Delta i(s,t)$ is same as selecting those splits that minimize the overall tree impurity $I(T)$.

The preliminary stop-splitting rule was simple, it involves setting some threshold $\beta > 0$ and declaring a node terminal if the decrease in the tree impurity is less than this β. But this rule produced unsatisfactory results, more about this in next chapter.

Preliminary condition for declaring a node as a terminal node:

$$\max_{s \in S} \Delta I(s,t) < \beta. \tag{2.11}$$

2.2.1.3 The Class Assignment Rule and Resubstitution Estimates:

After successfully constructing the tree T having terminal nodes $\widetilde{T}$. We need an established framework that help us to assign a class $j \in C$ to each of these nodes. This is achieved by *The Class Assignment Rule.*

Definition 2.6. *A class assignment rule assigns a class $j \in 1, \ldots, J$ to every terminal node $t \in \widetilde{T}$. The class assigned to node $t \in \widetilde{T}$ is denoted by $j(t)$.*

For any set of priors $\{\pi(j)\}$ and class assignment rule $j(t)$, the resubstitution estimate $r(t)$ of the probability of misclassification given that a case falls into node t is given by:

$$\sum_{j \neq j(t)} p(j|t) \tag{2.12}$$

Definition 2.7. *The class assignment rule $j^*(t)$ is the rule that minimizes the $r(t)$ in equation (2.12) and is given by:*

$$If\ p(j|t) = \max_i p(i|t),\ then\ j^*(t) = j$$

Remark 2.8. *If the maximum is achieved for two or more different classes, assign $j^*(t)$ arbitrarily as any one of the maximizing classes.*

Using the definition (2.7), we can alternatively define the resubstitution estimate $r(t)$ as:

Definition 2.9. *The resubstitution estimate $r(t)$ of the probability of misclassification, given that a case falls into node t, is:*

$$r(t) = 1 - \max_j p(j|t) \tag{2.13}$$

Denote,

$$R(t) = r(t)p(t) \tag{2.14}$$

Then the resubstitution estimate for the overall misclassification rate $R^*(T)$ of the tree T is:

$$R(T) = \sum_{t \in \widetilde{T}} R(t) \tag{2.15}$$

When we misclassify a particular case belonging to class j as class i, there is a cost or loss associated with this error and often this error is not same for all classes. For example, if a person gets tested for a virus, then it is more dangerous if the test result is found out to be a false negative rather than false positive. This depends on the misclassification cost function $c(i|j)$, where

Definition 2.10. *$c(i|j)$ is the cost of misclassifying a class j object as a class i object and satisfies:*

(i) $c(i|j) \geq 0, i \neq j,$

(ii) $c(i|j) = 0, i = j.$

At node t, if a random case of unknown class is classified as class i object, then the expected misclassification cost at node t is given by:

$$\sum_j c(i|j)p(j|t).$$

It makes sense to assign a class i to the object that minimizes the above expression. Therefore,

Definition 2.11. *Put $j^*(t) = i_0$ if i_0 minimizes:*

$$\sum_j c(i|j)p(j|t), \tag{2.16}$$

define the resubstitution estimate $r(t)$ of the expected misclassification cost, given the node t, by:

$$r(t) = \min_i \sum_j c(i|j)p(j|t), \tag{2.17}$$

and define the resubstitution estimate $R(T)$ of the misclassification cost of the tree T by:

$$R(T) = \sum_{t \in \widetilde{T}} r(t)p(t) = \sum_{t \in \widetilde{T}} R(t), \tag{2.18}$$

Where, $R(t) = r(t)p(t)$ is the node misclassification cost at node t.

Remark 2.12. *If the cost of misclassification is same for all the classes and is equal to unity i.e., $c(i|j) = 1, i \neq j, \forall i$ this is referred to as unit cost case. Then, the minimum cost rule reduces to the rule given in definition (2.11):*

$$\sum_j c(i|j)p(j|t) = 1 - p(i|t), \tag{2.19}$$

Hence, we can take $j^*(t)$ in the definition (2.11) as the class assignment rule without any restrictions.

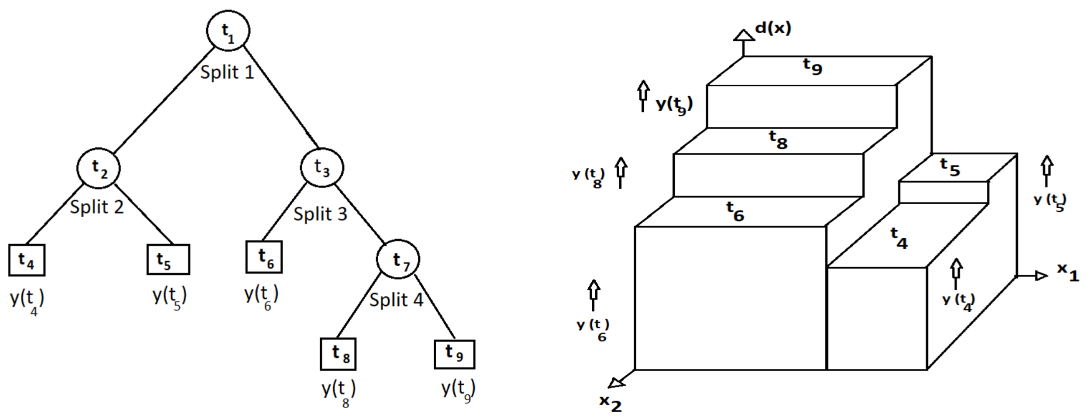
Proposition 2.13. *(From [Breiman et al., 1984]), For any split of a node t into t_L and t_R,*

$$R(t) \geq R(t_L) + R(t_R). \tag{2.20}$$

Proof. The proof is simple but for consistent reading we defer it to section (3.2.1) (Proposition 3.20) of next chapter. □

2.3 Tree Structured Regression:

A tree structured predictor is similar in construction to a tree structured classifier(as in figure (2.1)). Recursive binary splits are made to partition the measurement space $\mathcal{X}$ into set of rectangular boxes, representing the terminal nodes, as shown in the figure (2.3a). Only difference from classifier is that, instead of predicting the class response,



(A) Splitting a regression tree (B) Regression (decision) surfaces

FIGURE 2.3: Tree structured regression (2-dimension)

a constant value $y(t)$ is predicted by $d(x)$ at each of the terminal nodes and hence, tree structured regression can be seen as histogram estimation (see figure (2.3b)) of the regression surfaces.

Like in classification, the key elements in determining a tree predictor revolve around using the data from the learning sample $\mathcal{L}$ to formulate rules that:

(i) decide on the selection of the splits at each of the intermediate nodes.

(ii) decide when to declare a node as a terminal node or to continue splitting it.

(iii) decide on how to assign a value $y(t)$ at each terminal node.

The node assignment rule is fairly easy to resolve. We evaluate the resubstitution estimate for $R^*(d)$, i.e.,

$$R(d) = \frac{1}{N} \sum_n (y_n - d(x_n))^2.$$

Proposition 2.14. *(From [Breiman et al., 1984]), The value of $y(t)$ that minimizes $R(d)$ is the average of y_n for all cases (x_n, y_n) falling into t; that is, the minimizing $y(t)$ is:*

$$\bar{y}(t) = \frac{1}{N(t)} \sum_{x_n \in t} y_n, \tag{2.21}$$

where the sum is over all y_n such that $x_n \in t$ and $N(t)$ is the total number of cases in t.

Proof. The proof is based on the fact that the quantity "a" that minimizes $\sum_{n=1}^{N}(y_n - a)^2$ is:

$$a = \frac{1}{N} \sum_{n=1}^{N} y_n$$

□

Remark 2.15. *Similarly, for any other subset $y_{n'}$ of y_n, the average of the $y_{n'}$ minimizes the expression $\sum_{n'=1}(y_{n'} - a)^2$.*

Hence, following the above proposition we take $\bar{y}(t)$ as the predicted value for a given node t. Then, by using the notation $R(T)$ to replace $R(d)$ we have:

$$R(T) = \frac{1}{N} \sum_{t \in \widetilde{T}} \sum_{x_n \in t} (y_n - \bar{y}(t))^2 \tag{2.22}$$

The resubstitution estimate in (2.22) can be expressed as the sum of resubstitution estimates of terminal nodes as:

$$R(T) = \sum_{t \in \widetilde{T}} R(t) \tag{2.23}$$

Where,

$$R(t) = \frac{1}{N} \sum_{x_n \in t} (y_n - \bar{y}(t))^2$$

For a given set of splits S at a terminal node $t \in \widetilde{T}$.

Definition 2.16. *The best split s^* of t is that split in S which most decreases $R(T)$*

The change in resubstitution estimate for a split s of t into t_L and t_R is given as:

$$\Delta R(s,t) = R(t) - R(t_L) - R(t_R)$$

Then, the best split s^* solves:

$$\Delta R(s^*, t) = max_{s \in S}\, \Delta R(s, t)$$

Therefore, a regression tree is built by splitting nodes iteratively by maximizing the decrease in $R(T)$ (by choosing the best splits). Using this criteria for regression trees is found to be reliable unlike classification trees, which is discussed in (3.2) of next chapter.

Chapter 3

Methodological Development of Decision Trees

In this chapter, we will list out some of the major deficiencies in the initial tree growing methodology and propose some measures to make tree structures more flexible and accurate.

3.1 Growing Right Sized Trees via Pruning: (A Primary Issue)

The trees grown only using the initial methodology were found to give dishonest results. For instance, if the stop-splitting rule proposed as per equation (2.11) is used, Breiman et al. in section 3.1 showed that this is a very unstable rule.

He found out that for very small values of β, the tree sizes are very large and the estimated misclassification rate $R(T)$ (whose value decreases as the number of terminal nodes increases) is found to be 0, since the $p(j|t)$ values are all either 0 or 1. But this $R(T)$ nature does not reflect the same when the trees are subjected to an independent test sample and the estimates $R^{ts}(T)$ first decreased with increasing tree size and reach a minimum after which it increases again with tree size.

This is due to over-fitting, which makes you think that you do better with more splitting as the decision boundaries gets squeezed to accommodate the learning set. Alternatively, for higher values of β, the splitting is stopped prematurely. Other variants of stopping rules were tested but there is no significant improvement in overall misclassification rate.

Breiman concludes that looking for stopping rule is the wrong approach and proposes the following two key steps for satisfactory results:

(i) Minimal cost complexity pruning: Instead of applying stop-splitting rule to get right set of terminal nodes, apply pruning i.e., grow a very large tree and selectively recombine upward in the right way.

(ii) Honest estimation: More accurate estimates of the true misclassification rate of the tree $R^*(T)$ like test sample estimates, $R^{ts}(T)$, or cross validation estimates $R^{CV}(T)$ are used to pick out the right (best) sized tree from among the sequence of pruned subtrees.

When the tree is pruned upward, the estimated misclassification rate initially decreases slowly, reaches a gradual minimum, and then as the size of the tree further decreases it increases rapidly. This behaviour is due to a trade-off between bias and variance.

3.1.1 Pruning:

Before we start discussing pruning, make a note of $R(T)$ and $R(t)$ from equation (2.18) as the resubstitution estimates of misclassification cost for the tree T and terminal node t respectively.

The first step in pruning involves growing a very large tree T_{max} by continued splitting until all the terminal nodes become either pure or containing less than a small set of cases N_{min}. Generally, N_{min} lies between 1 and 5. The exact size of T_{max} is not very critical as long as T_{max} is sufficiently large. This is because taking the largest possible tree $T^{'}_{max}$ and pruning it produces the same set subtrees that can also be obtained by pruning T_{max}.

Definition 3.1. *The tree T is said to be trivial if any of the following equivalent conditions is satisfied:*

$$|T| = 1; |\widetilde{T}| = 1; T = root(T); T–\widetilde{T} \textit{ is empty.}$$

Else, T is said to be nontrivial.

Remark 3.2. *Here, $|T|$ is the number of elements (sum of all the root, non-terminal and terminal nodes) in the tree T. It can be noticed that for binary split trees, $|T| = 2|\widetilde{T}| - 1$.*

Definition 3.3. *Consider any non-terminal node t in the tree, then, any other node $t^{'}$, situated lower down the node t, is called the descendant of t if there exists a connected path that leads from node t to $t^{'}$. Consequently, node t is the ancestor of node $t^{'}$.*

In the figure (3.1), nodes t_6 and t_7 are the descendants of node t_3 but not t_4 and t_5.

Definition 3.4. *A branch T_t of tree T with root at node t, $t \in T$ consists of the node t and all descendants of t in T.*

Refer to the figure (3.1) for the branch T_{t_2}.

Definition 3.5. *Pruning a branch T_t from a tree T consists of deleting all descendants of t from T i.e., cutting off all of T_t except its root node t. The pruned tree is denoted by $T - T_t$.*

Refer to figure (3.1) for the pruned tree T_{t_2}.

Definition 3.6. *If $T^{'}$ is gotten from T by successively pruning off branches, then $T^{'}$ is called a pruned subtree of T and denoted by $T^{'} < T$. $T^{'}$ and T have the same root node i.e., all the pruned subtrees are also rooted subtrees.*

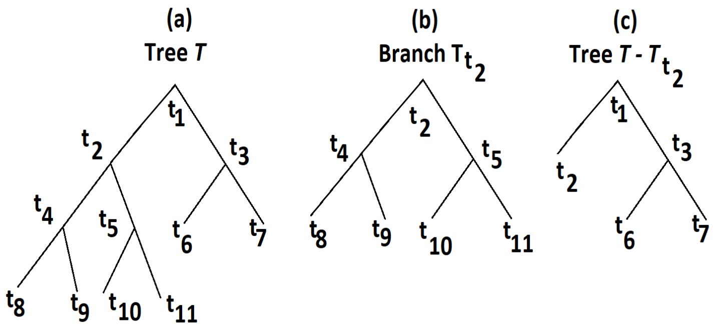


FIGURE 3.1: Terminology: Tree pruning

Intuitively, from the definition of pruned subtree we can have the following definition for trees T_1 and T_2 respectively.

Definition 3.7. *Let T_1 and T_2 be pruned subtrees of T. Then T_2 is a pruned subtree of T_1 if and only if, every nonterminal node of T_2 is also a nonterminal node of T_1.*

The number of pruned subtrees possible for a tree T is given by:

(i) If T is trivial, then $\#(T) = 1$.

(ii)
$$Else, \quad \#(T) = \#(T_L)\#(T_R) + 1 \tag{3.1}$$

Where $\#(T)$ denote number of pruned subtrees of tree T and $\#(T_L)$, $\#(T_R)$ for those of left and right primary branches of T.

3.1.2 Minimal Cost-Complexity Pruning:

Definition 3.8. *For any subtree $T \leq T_{max}$, define its complexity as $|\widetilde{T}|$, the number of terminal nodes in T. Let $\alpha \geq 0$ be a real number called the complexity parameter and define the cost-complexity measure $R_\alpha(T)$ as:*

$$R_\alpha(T) = R(T) + \alpha|\widetilde{T}| \tag{3.2}$$

i.e., the cost-complexity measure $R_\alpha(T)$ is a linear combination of two parameters (i) misclassification of the tree $R(T)$ and the (ii) complexity measure α. Here α can be seen as the complexity cost per unit terminal node. The term $\alpha|\widetilde{T}|$ penalizes growing huge trees.

The expression in (3.2) can also be written as, $R_\alpha(T) = R(T) + \alpha|\widetilde{T}| = \sum_{\widetilde{T}} R_\alpha(T)$ and $R_\alpha(T) = R(t) + \alpha$, where t is any node of tree T, $\forall t \in T$.

If in case the tree T, is a trivial tree with root t_1, then $R(T) = R(t_1)$ and $R_\alpha(T) = R_\alpha(t_1)$.

Definition 3.9. *A pruned subtree T_1 of T is said to be an optimally pruned subtree of T (for a given α) if:*

$$R_\alpha(T_1) = \min_{T' \leq T} R_\alpha(T').$$

As we know from (3.1) the total number of pruned subtrees can only be finite (though this quantity increases exponentially with increasing size of tree), it is possible to find the optimal one (i.e., existence) but will there be a unique one that solves the optimization problem in definition (3.9) The theorem (3.11) and proposition (3.14) addresses the uniqueness and existence of $T(\alpha)$, respectively.

We call this unique optimally pruned subtree as the smallest optimally (minimizing) pruned subtree of T. It is denoted by $T(\alpha)$.

Definition 3.10. *The smallest minimizing subtree $T(\alpha)$ for complexity parameter α is defined by the conditions:*

(i)

$$R_\alpha(T(\alpha)) = \min_{T \leq T_{max}} R_\alpha(T)$$

(ii)

$$If \ \ R_\alpha(T) = R_\alpha(T(\alpha)), \ \ then \ \ T(\alpha) \leq T$$

$T(\alpha)$ can also be interpretated as an optimal subtree T_1 which acts as the one and only smallest optimally pruned subtree of T for every $T' \geq T_1$, where T' is also an optimally pruned subtree of T.

For this T', let T'_L and T'_R be its two primary branches. Then:

$$R_\alpha(T') = R_\alpha(T'_L) + R_\alpha(T'_R)$$

Now, easily by induction the uniqueness of $T(\alpha)$ can be stated by the following theorem.

Theorem 3.11 (Uniqueness). *(From [Breiman et al., 1984]), Every tree T has a unique smallest optimally pruned subtree $T(\alpha)$. Let T be a nontrivial tree having root t_1 and* *primary branches T_L and T_R. Then*

$$R_\alpha(T(\alpha)) = \min[R_\alpha(t_1), R_\alpha(T_L(\alpha)) + R\alpha(T_R(\alpha))].$$

$If\ \ R_\alpha(t_1) \leq R_\alpha(T_L(\alpha)) + R_\alpha(T_R(\alpha))$, then $T(\alpha) = t_1$; otherwise, $T(\alpha) = t_1 \cup T_L(\alpha) \cup T_R(\alpha)$.

Remark 3.12. *$T_L(\alpha)$ and $T_R(\alpha)$ denote the smallest optimally pruned subtrees of T_L and T_R, respectively.*

Immediately, from the transitivity of $\leq$, we have

Theorem 3.13. *(From [Breiman et al., 1984]), If $T(\alpha) \leq T' \leq T$, then $T(\alpha) = T'(\alpha)$.*

Proposition 3.14 (Existence)**.** *(From [Breiman et al., 1984]), For every value of α, there exists a smallest minimizing subtree as defined by definition (3.10).*

Proof. We omit the proof but if interested kindly refer to ([Breiman et al., 1984]), Chapter 10, Theorem 10.9 for the explanation of the above proposition. □

Theorem 3.15. *(From [Breiman et al., 1984]),*

(i) If $\alpha_2 \geq \alpha_1$, then $T(\alpha_2) \leq T(\alpha_1)$.

(ii) If $\alpha_2 > \alpha_1$, and $T(\alpha_2) < T(\alpha_1)$., then

$$\alpha_1 < \frac{R(T(\alpha_2)) - R(T(\alpha_1))}{|\widetilde{T}(\alpha_1)| - |\widetilde{T}(\alpha_2)|} \leq \alpha_2.$$

Proof. We omit the proof but if interested kindly refer to ([Breiman et al., 1984]), Chapter 10, Theorem 10.9 for the explanation of the above theorem. □

Remark 3.16. *Statement (i) indicates that increasing α leads to significant penalty for having bigger trees and the smallest minimizing subtree will be the same or smaller than the initial $T(\alpha)$.*

Minimal cost-complexity pruning involves, for each α, finding that smallest optimally pruned subtree $T(\alpha) \leq T_{max}$ that minimizes the quantity in equation (3.2) i.e.,

$$R_\alpha(T(\alpha)) = \min_{T \leq T_{max}} R_\alpha(T).$$

And then increasing the value of α to α' and repeating the same procedure again to find $R_{\alpha'}(T(\alpha'))$ and so on until a global minimizer for the cost-complexity measure is found.

For small values of α, since the penalty per terminal node is less, the resultant $T(\alpha)$ will be large and vice-versa. If $\alpha = 0$, then $T(\alpha)$ is the initial starting tree T itself and sufficiently large values of α results in completely pruned tree containing only the root node.

Although α varies over the positive real line $\mathbb{R}^+$, the number of subtrees of T_{max} generated during the pruning process are finite. These finite sequence of subtrees $T_1, T_2, \ldots$ formed contains fewer and fewer terminal nodes with each iteration of α.

Now that we outlined minimal cost-complexity pruning, the problem remains finding the minimizer $T(\alpha)$ by searching through all those possible subtrees which is computationally intense. To solve this, we use weakest-link cutting which produces the smallest decreasing sub-sequence of trees containing $T(\alpha)$.

3.1.3 Weakest link cutting:

In pruning, we start with $\alpha = 0$ and the initial starting tree is the minimizing tree itself since no pruning is possible in the first step i.e., $T_1 = T(\alpha = 0) = T(0)$. We do not need to start the biggest possible tree T_{max} as the initial starting tree as this leads to more intermediate steps.

We can obtain T_1 from T_{max} by defining T_1 as the smallest subtree of T_{max} satisfying:

$$R(T_1) = R(T_{max})$$

To get this subtree T_1, starting with T_{max}, we consider the set of pairs of terminal nodes t_L and t_R resulting from any ancestor node t. Using the inequality, $R(t) \geq R(t_L) + R(t_R)$ from proposition (2.13) we will prune the terminal nodes t_L and t_R if $R(t) = R(t_L) + R(t_R)$. T_1 is the final tree achieved by continuing this process iteratively until no more pruning is possible.

For T_t any branch of T_1, define $R(T_t)$ by:

$$R(T_t) = \sum_{t' \in \widetilde{T}_t} R(t')$$

where $t' \in \widetilde{T}_t$ are the terminal nodes of T_t.

Proposition 3.17. *(From [Breiman et al., 1984]), For t any nonterminal node of T_1,*

$$R(t) > R(T_t)$$

.

Proof. We omit the proof but if interested kindly refer to [Breiman et al., 1984] Section 10.2, Theorem 10.11 for the explanation the above proposition. □

Beginning with T_1, the key methodology of minimal cost-complexity pruning is attributed to understanding that it works by weakest link cutting. As we search through possible values of α that corresponds to a new optimal subtree, the weakest link cutting actually finds that optimal subtree.

Now, let us consider a sub-branch of T_t with single node t denoted by $\{t\}$, for any node t belonging to T_1. Let us take the cost complexity measure of this subbranch with single node as:

$$R_\alpha(\{t\}) = R(t) + \alpha.$$

Similarly, for the branch T_t:

$$R_\alpha(T_t) = R(T_t) + \alpha|\widetilde{T}_t|.$$

Observing the figure below, initially when the value of α is 0 or very small, the misclassification rate term dominates over the complexity term and the strict inequality from proposition (3.17) also holds true for the cost complexity measure i.e.,

$$R_\alpha(T_t) < R_\alpha(\{t\})$$

As the value of α increases, the term $R_\alpha(T_t)$ rises at a faster rate than that of $R_\alpha(\{t\})$, this is because the complexity measure takes precedence, and the branch T_t has several terminal nodes compared to single node subbranch $\{t\}$. At a critical point of α, let us say α_1, both values $R_\alpha(T_t)$ and $R_\alpha(\{t\})$, becomes equal and further increase in α reverses the inequality to $R_\alpha(T_t) > R_\alpha(\{t\})$. This essentially means that even after splitting the

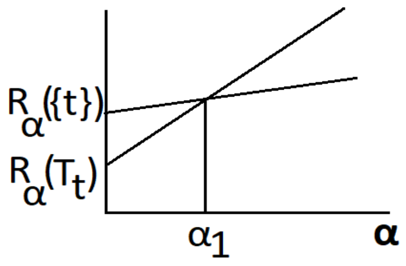


FIGURE 3.2: Minimal cost-complexity pruning: Trade-off between misclassification rate and tree complexity

node t into a subtree T_t it performs worse than the that of single node $\{t\}$ in terms of cost complexity and hence we can avoid this split i.e., we can prune subbranch T_t into a single node $\{t\}$. This node t at which the above-described phenomenon occurs is called as the weakest link and the process of pruning the subbranch T_t at that node t is known as *weakest link cutting.*

The critical value α can be evaluated by solving the inequality:

$$R_\alpha(T_t) < R_\alpha(\{t\}),$$

Which results in,

$$\alpha < \frac{R(t) - R(T_t)}{|\widetilde{T_t}| - 1} \tag{3.3}$$

By proposition (3.17), the value on the right side of (3.3) is positive.

Weakest link cutting procedure:

First, let us define a function $g_1(t)$, $\forall t \in T_1$, as:

$$g_1(t) = \begin{cases} \frac{R(t)-R(T_t)}{|\widetilde{T_t}|-1}, & \text{t} \notin \widetilde{T_1} \\ +\infty, & \text{t} \in \widetilde{T_1} \end{cases}$$

Then the weakest link t_1 dash in T_1 is defined as that node in T_1 such that:

$$g_1(\bar{t_1}) = \min_{t \in T_1} g_1(t)$$

and the corresponding value of α at which node t_1 dash becomes weakest link is taken as:

$$\alpha_2 = g_1(\bar{t_1})$$

Then the resulting new tree $T_2 < T_1$ obtained by pruning away the weakest link branch $T_{\bar{t_1}}$ dash is given by:

$$T_2 = T_1 - T_{\bar{t_1}}$$

Remark 3.18. *If the minimum for the function $g(t)$ occurs at multiple nodes then the new tree is formed by cutting away multiple weakest links at all such nodes.*

If all the above steps are repeated, then the weakest link cutting procedure produces a decreasing sequence of subtrees such as:

$$T_1 > T_2 > T_3 > \ldots > t_1,$$

where $\{t_1\}$ is the root node of the initial tree T_1. The weakest link cutting procedure can be summarized into Algorithm 2.

The connection to minimal cost-complexity pruning is given by:

Theorem 3.19. *(From [Breiman et al., 1984]), The $\{\alpha_k\}$ are an increasing sequence obtained by the algorithm 2, i.e., $\{\alpha_k\} < \{\alpha_k\} + 1, k \geq 1$, where $\{\alpha_1\} = 0$. For $k \geq 1$, $\{\alpha_k\} \leq \alpha < \{\alpha_k\} + 1$,*

$$T(\alpha) = T(\alpha_k) = T_k.$$

Proof. We omit the proof but if interested kindly refer to Breiman et al. Section 10.2 (more specifically Theorem 10.11) for the explanation of the above theorem. □

```
Algorithm 1 Weakest link cutting
Step 1 : Initialize $i = 1$, $\alpha_{i=1} = 0$ and consider the starting tree as $T_{i=1} = T_1$
Step 2 : Continue the algorithm while ($T_i \neq \{t1\}$)
#Where $t_1$ is the root node
    (i) For all the nodes $t$ of the tree $T_i$, Evaluate $g_i(t)$ as:
        $$g_i(t) = \begin{cases} \frac{R(t)-R(T_it)}{|\widetilde{T_it}|-1}, & \text{t} \notin \widetilde{T_i} \\ +\infty, & \text{t} \in \widetilde{T_i} \end{cases}$$
    (ii) Then evaluate:
        $$g_i(\bar{t_i}) = \min_{t \in T_i} g_i(t)$$
    (iii) Take:
        $$\alpha_{i+1} = g_i(\bar{t_i})$$
        .
    (iv) Check for multiplicity of the weakest links i.e., if $g_i(\bar{t_i})$ has multiple minimizers. If
         so, then denote them as:
        $$\bar{t_i}^1, \bar{t_i}^2, \bar{t_i}^3, \ldots$$
    (v) If there is no multiplicity of the weakest links, then:
        $$T_{i+1} = T_i - T_{\bar{t_i}}$$
        Else,
        $$T_{i+1} = T_i - T_{\bar{t_i}^1} - T_{\bar{t_i}^2} - T_{\bar{t_i}^3} - \ldots$$
Step 3 : Stop the Algorithm if ($T_i = \{t1\}$)
```

From this theorem we can conclude that, minimal cost-complexity pruning starts with $\alpha = \alpha_1 = 0$ and the initial the initial tree T_1, for which the weakest link branch $T_{\bar{t_1}}$ is identified and this $T_{\bar{t_1}}$ is pruned to get T_2 when α reaches α_2. When α reaches α_2, the weakest link branch $T_{\bar{t_2}}$ of T_2 is identified and pruned to get T_3. This process continues until the final tree that minimizes the (3.2) is found.

The entire pruning process is computationally fast and the time required is only a small portion of total time that needed for constructing the tree. Initially pruned subbranches are very large in size but decreases as the consequent trees get smaller.

3.2 Splitting Rules:

Given the set of questions Q or set of splits S for every node, the splitting rules play an important role in growing the tree and obtaining T_{max} before the start of pruning process. Despite of this, the splitting rules formulated around reducing misclassification cost for each split are found to have no significant effect in reducing the overall misclassification rate $R(T)$ of the tree, as already mentioned in the beginning of previous section (3.1).

They are defined in conjunction with the goodness of the split criterion $\phi(s,t)$, as in (2.4), at every node t and its corresponding split $s \in S$. The rule is chosen such that the split s^* maximizes the $\phi(s,t)$ in the equation (2.6).

At this stage, reducing the misclassification cost might seem like a natural goodness of split criterion but this has some serious shortcomings, which are discussed in next section 3.2.1. We propose a class of splitting criteria (for two-class problem, unit cost case) to address these shortcomings in section 3.2.2. A generalized criterion is given in case of multi-class problem (unit cost case) in Section 3.2.3.

3.2.1 Reducing Misclassification Cost:

In Section (2.2.1.2), the framework for splitting rules was already given. In brief, it involves defining an impurity function $\phi(p_1, ..., p_J)$ in line with definition (2.2). At every node t, its impurity function $i(t)$ is defined as $\phi(p(1|t), ..., p(1|t))$. For $I(t) = i(t)p(t)$, the tree impurity $I(T)$ is given by:

$$I(T) = \sum_{t \in \widetilde{T}} I(t)$$

At any current terminal node t, we choose the split s^* that maximizes the goodness of split criteria given by:

$$\Delta I(s,t) = I(t) - I(t_L) - I(t_R)$$

or

$$\Delta i(s,t) = i(t) - p_{L_i}(t_L) - p_{R_i}(t_R).$$

At this point, it might be logical and natural to consider $R(T)$ as the tree impurity $I(T)$. This is also equivalent to defining $i(t)$ as $r(t)$, given by:

$$r(t) = \min_i \sum_j c(i|j)p(j|t) = 1 - \max_j p(j|t) \quad (\textit{unit cost case}).$$

Then s^* is the split that most reduces the estimated misclassification rate and maximizes the following expression:

$$r(t) - p_L r(t_L) - p_R r(t_R) \tag{3.4}$$

or

$$R(t) - R(t_L) - R(t_R) \tag{3.5}$$

The respective node impurity function (unit cost case) is:

$$\phi(p_1, \ldots, p_J) = 1 - \max_j p_j. \tag{3.6}$$

The above function satisfies all the properties in definition (2.2).

Despite of this favorability of $R(T)$ as a candidate for splitting criterion, selecting the splits that focus on reducing the $R(t)$ has two serious deficiencies.

First deficiency is that the criterion in (3.4) or (3.5) may be zero for all possible splits s belonging to S.

Proposition 3.20. *(From [Breiman et al., 1984]), For any split of a node t into t_L and t_R,*

$$R(t) \geq R(t_L) + R(t_R). \tag{3.7}$$

with equality if $j^*(t) = j^*(t_L) = j^*(t_R)$.

Proof. It is to be noted that:

$$R(t) = \sum_j c(j^*(t)|j)p(j,t) = \sum_j c(j^*(t)|j)[p(j,t_L) + p(j,t_R)]$$

or

$$\begin{aligned} R(t) - R(t_L) - R(t_R) = & \sum_j c(j^*(t)|j)p(j,t_L) - \min_i \sum_j c(i|j)p(j,t_L) \\ & + \sum_j c(j^*(t)|j)p(j,t_R) - \min_i \sum_j c(i|j)p(j,t_R) \end{aligned}$$

The r.h.s is almost surely non-negative and is equal to zero under the conditions $j^*(t) = j^*(t_L) = j^*(t_R)$. $\square$

Now let us consider that we are dealing with a two-class problem with equal priors and having a huge portion of class 1 cases at some node t. It can be seen that every split t produces branches, with nodes t_L and t_R, with class 1 majorities. In this scenario, $R(t) - R(t_L) - R(t_R) = 0$ for all splits s belonging to S with no single or some number of best splits.

The second deficiency is rather difficult to explain quantitatively but explained using a suitable example below. As shown in fig (3.3), we again consider a two-class problem with equal priors and a learning sample $\mathcal{L}$ of size 800. Now, at the root node we perform two kinds of splits, Split 1 and Split 2, respectively.

Split 1 leads to a tree (left in fig (3.3)) where 200 cases are misclassified i.e., $R(T) = 200/800 = 0.25$. Similarly, Split 2 also resulted in a tree with 200 misclassifications i.e.,

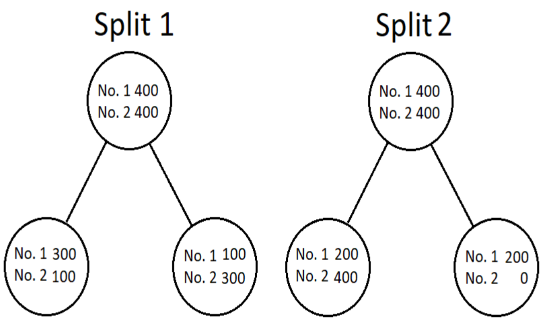


FIGURE 3.3: Two-class problem: Misclassification rate deficiencies

$R(T) = 0.25$. Even though both splits have the same $R(T)$, the Split 2 is more desirable in terms of future growth of the tree. This is because both terminal nodes in the tree due to Split 1 are impure and further splitting is needed for better accuracy. Whereas in the tree due to Split 2, the terminal node on the right is already pure and splitting is required only on the other terminal node.

In addition to this degeneracy problem, it seems that $R(T)$ criterion does not reward for producing the splits that are more beneficial for the continued growth of the tree. This result is due to the fact that tree structures are based on one-step optimization and does not incorporate strategic approach keeping in mind the implications of current actions on the future development of the tree. They only look to obtain a best possible tree at a given step. This is referred to as *top-down greedy approach.*

3.2.2 Splitting Rules for Two-Class Problem: (Unit Costs)

3.2.2.1 A Class of Splitting Criteria for the Two-Class Problem:

For a two-class problem with Class 1 and Class 2 and assume that the cost of misclassifying one class as another is same for both classes and is equal to unity. Then the node impurity function w.r.t the misclassification rate/error is given by:

$$\phi(p_1, p_2) = 1 - \max(p_1, p_2) = \min(p_1, p_2) = \min(p_1, 1 - p_1)$$

Its graph as a function of p_1 is shown in the following figure: As you can see the impurity function in terms of misclassification cost $\min(p_1, 1-p_1)$ satisfies all the three properties as in the definition (2.2). But the example in fig (3.3) of previous section shows that the $\phi(p_1) = \min(p_1, 1 - p_1)$, failed in rewarding the purer nodes.

When $p_1 > 1/2$, $\phi(p_1) = 1 - p_1$, as node gets purer in Class 1, i.e., p_1 increases this function $\phi(p_1)$ decreases only linearly in terms of p_1.

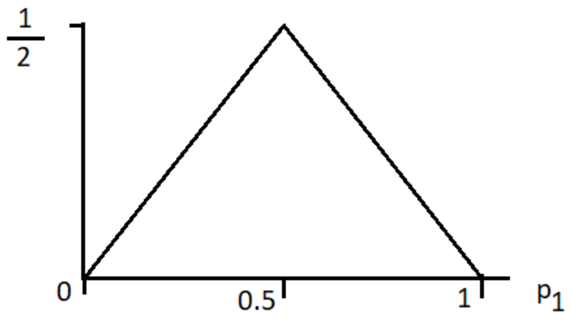
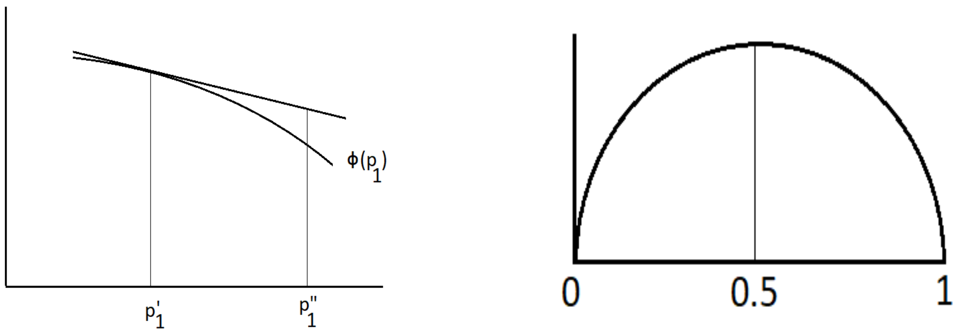


FIGURE 3.4: Impurity function with misclassification error rate

In order to be able to construct a class of criteria that can successfully select the Split 2 over Split 1 of the example in figure (3.3), we will need that $\phi(p_1)$ decreases at much faster rate than linearly as p_1 increases. This is formulated by forcing that if $p_1^{''} > p_1^{'}$ then $\phi(p_1^{''})$ is lower than the respective point on the tangent drawn at $p_1^{'}$ (refer to the figure below). This can only be possible if ϕ is strictly concave. We need strict concavity in order to avoid degeneracy. Assuming ϕ has a continuous second derivative on $[0, 1]$, then by strict concavity we have $\phi^{''}(p_1) < 0, 0 < p_1 < 1$. Hence, let us define the class



(A) Desirable impurity Function (Concavity) (B) Class of Functions of F

FIGURE 3.5: Requirements: Impurity Function

of impurity functions that can be used in this context as $\phi(p_1)$, belonging to the space of functions F, satisfying the three conditions stated below:

$$\begin{aligned} &(i)\ \ \phi(0) = \phi(1) = 0 \\ &(ii)\ \ \phi(p_1) = \phi(1 - p_1) \\ &(iii)\ \ \phi^{''}(p_1) < 0, 0 < p_1 < 1. \end{aligned} \tag{3.8}$$

Remark 3.21. *Here the function $\phi(p_1)$, for $p_1 \in [0, 1]$ must have continuous second order derivatives over its domain $p_1 \in [0, 1]$*

The general graph of the class of functions F is shown in the figure (3.5b) above.

Now for a multi-class problem with J classes and corresponding portion of cases, $p_1, p_2, \ldots, p_J.$, at node t. We show that all of the functions contained in space F exhibit the common property mentioned in the following proposition:

Proposition 3.22. *(From [Breiman et al., 1984]), Let $\phi(p_1, p_2, \ldots, p_J)$ be a strictly concave function on $0 \leq p_j \leq 1$, for $j = 1, \ldots, J$, $\sum_j p_j = 1$. Then for $i(t) = \phi(p(1|t), \ldots, p(J|t)), \phi \in F$ and any split s,*

$$\Delta i(s,t) \geq 0$$

with equality if, and only if, $p(j|t_L) = p(j|t_R) = p(j|t),\ for\ j = 1, \ldots, J$.

Proof. By the strict concavity of ϕ,

$$\begin{aligned} i(t_L)p_L + i(t_R)p_R &= \phi(p(1|t_L), \ldots, p(J|t_L))p_L + \phi(p(1|t_R), \ldots, p(J|t_R))p_R \\ &\leq \phi(p_L p(1|t_L) + p_R p(1|t_R), \ldots, p_L p(J|t_L) + p_R p(J|t_R)) \end{aligned}$$

The equality will only hold if, and only if, $p(j|t_L) = p(j|t_R)$, for $j = 1, \ldots, J$.

Now,

$$p_L p(j|t_L) + p_R p(j|t_R) = [p(j,t_L) + p(j,t_R)]/p(t) = p(j|t)$$

This implies that => $i(t_L)p_L + i(t_R)p_R \leq i(t)$,

which further implies => $\Delta i(s,t) \geq 0$.

With equality if, and only if, $p(j|t_L) = p(j|t_R),\ for\ j = 1, \ldots, J$. □

The proposition (3.22) essentially says that you can never increase the impurity by splitting. Generally, the chances of resulting in the equality condition is very difficult to obtain than that of corresponding condition for $R(t)$ and strict concavity requirement removes the degeneracy.

3.2.2.2 The Criteria Class and Categorical Variables:

In this section, we will discuss the splitting procedure in case of a categorical variable and give a theorem result that establishes the existence of an optimal split.

For a two-class problem, consider a measurement variable x, which is a categorical variable having the set of values $\{b_1, \ldots, b_L\}$.

During the construction of the tree at some node t, when splitting w.r.t to x we look for all the possible subsets of x and finally chooses the subset B^* of the form,

$$B^* = \{b_{i_1}, \ldots\} \subset \{b_1, \ldots, b_L\}$$

that produces a split s^* which maximizes the splitting criteria. There can be multiple maximizers B^*.

Now, $N_{j,l}(t)$ be the number of cases that belongs to class j, j belonging to $\{1,2\}$, at node t such that $x = b_l$ and define:

$$p(j|x = b_l) = \pi(j)N_{j,l}(t) / \sum_{j=1,2} \pi(j)N_{j,l}(t)$$

i.e., $p(j|x = b_l)$ can be seen as the estimated probability of an object at node t and belonging to class j, j belonging to $\{1,2\}$, such that the value of categorical variable x category is b_l. Under this setting, the result of the following theorem holds.

Theorem 3.23. *(From [Breiman et al., 1984]), Order the $p(1|x = b_l)$, i.e.,*

$$p(1|x = b_{l_1}) \leq p(1|x = b_{l_2}) \leq \ldots \leq p(1|x = b_{l_L}).$$

If ϕ is in the class F, then one of the L subsets:

$$\{b_{l_1}, \ldots, b_{l_h}\}, \quad h = 1, \ldots, L.$$

is maximizing.

Proof. We omit the proof but if interested kindly refer to [Breiman et al., 1984] section 9.4 for the explanation of the above theorem. □

Remark 3.24. *For large values of L, the computational efficiency is greatly enhanced since the total possible subsets reduce from 2^{L-1} to just L.*

3.2.2.3 Selection of a Single Criterion:

Since all the possible functions of ϕ belonging to F, needs to have continuous second order derivatives, the simplest polynomial that can belong to space of functions F is a quadratic of the form:

$$\phi(x) = a + bx + cx^2.$$

Condition (3.8)(i) results in $a = 0, b + c = 0$, so

$$\phi(x) = b(x - x^2),$$

and (3.8)(ii) implies that $b > 0$.

Without loss of generality, we take $b = 1$, giving the criterion:

$$i(t) = p(1|t)p(2|t). \tag{3.9}$$

The criterion,

$$i(t) = -p(1|t)\log p(1|t) - p(2|t)\log p(2|t) \tag{3.10}$$

also belongs to F.

The criterion in (3.9) is called as Gini criterion or Gini index or Gini measure. It is simple and can be computed very quickly. It can also be interpreted as a measure of total variance across the J classes (for multi-class problem).

The criterion in (3.10) is called Cross-entropy or simply Entropy or Deviance. Here, $0log0 = 0$.

Both Gini criterion and Cross-entropy behave and produce, similar numerical results (evident from the figure below) and also used as node purity measure.

The figure below shows the Misclassification error, Gini index and Entropy, as a function of proportion p of class 2. (Cross-entropy has been scaled to fit in $(0.5, 0.5)$).

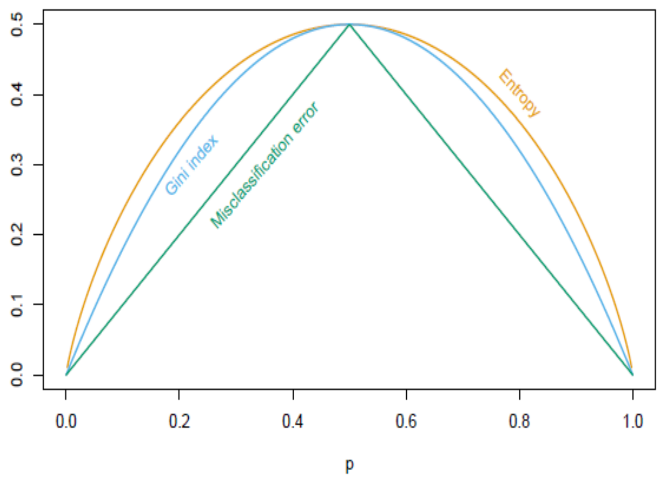


FIGURE 3.6: Misclassification error, Gini index and Entropy, as a function of proportion p of class 2. (Figure sourced from [Hastie et al., 2009], fig. (9.3))

3.2.3 Splitting Rules for Multi-Class Problem: (Unit Costs)

In this section, we will generalize the two-class criterion discussed in previous section to the multi-class problem with unit misclassification costs. Particularly, Gini criterion is discussed. It should be possible to derive similar results for the cross-entropy criterion as well.

Node impurity $i(t)$, at a node t, with estimated class probabilities $p(j|t)$, for $j = 1, \ldots, J$, is given by:

$$i(t) = \phi(p(1|t..., p(J|t)).$$

As discussed in previous section, we can extend the Cross-entropy criterion that most reduces the goodness of split criterion to multi-class problem with J classes, as:

$$\phi(p_1,\dots,p_J) = -\sum_j p_j \log p_j. \tag{3.11}$$

Similarly, the Gini criterion is:

$$i(t) = \sum_{j\neq i} p(j|t)p(i|t) \tag{3.12}$$

This can also be re-written as:

$$i(t) = (\sum_j p(j|t))^2 - \sum_j p^2(j|t) = 1 - \sum_j p^2(j|t). \tag{3.13}$$

The Gini index has two interesting interpretations. First, as already discussed in previous section, is in terms of variance. At a node t, assign value 1 to all the class j objects and 0 to all other objects not belonging to j^{th} class. Then, these values have the sample variance equal to $p(j|t)(1-p(j|t))$. By repeating this procedure for all J classes and adding them up, we get the same result as in (3.13):

$$\sum_j p(j|t)(1-p(j|t)) = 1 - \sum_j p^2(j|t).$$

The second interpretation is w.r.t (3.12), if we use the rule, that classifies an object to class i, at random with probability $p(i|t)$ instead of the plurality rule (i.e., using priors $\{\pi(i)\}$, for all $i \in J$, to classify an object as that class i for which $\pi(i)$ is largest.). Then, $p(j|t)$ is the estimated probability of that object is actually in class j. Hence, the Gini index is the estimated probability of misclassification at node t, equal to:

$$\sum_{j\neq i} p(j|t)p(i|t)$$

Gini index taken as a function $\phi(p_1,\dots,p_J)$ of the proportions of classes, $p_1,\dots,p_J$, is a quadratic polynomial containing non-negative coefficients. Therefore, it is also concave in the sense that for $r+s=1, r\geq 0, s\geq 0$,

$$\phi(rp_1 + sp_1^{'}, rp_2 + sp_2^{'},\dots, rp_J + sp_J^{'} \geq r\phi(p_1,\dots,p_J) + s\phi(p_1^{'},\dots,p_J^{'}).$$

Hence, from proposition (3.22), for any split s,

$$\Delta i(s,t) \geq 0.$$

In fact, the concavity is strict and consequently $\Delta i(s,t) = 0$ if $p(j|t_R) = p(j|t_L) = p(j|t)$, for all $j = 1, ..., J$.

Chapter 4

Boosting in Decision Trees

Decision Trees, built using the methodology described in previous chapters, possess lot of advantages over conventional regression and classification techniques. They are easy to explain and visualized. Given the tree is not too large, they are also very easy to interpret even by a non-sophisticated person. The data pre-processing phase for trees is relatively simple unlike other methods which may require data normalization, imputation or use of dummy variables, etc.

Despite of this, trees are prone to issues like over-fitting (unstable or high variant trees) and under-fitting (biased trees). The accuracy of the trees is generally inferior to those of alternative techniques. In addition, the problem of finding an optimal tree is NP-complete and hence, the splitting rules are based on heuristics like greedy algorithm which favors in finding locally optimal solutions (splits) at each node t. This will not result in globally optimal tree as already seen in previous chapter.

However, the accuracy and robustness of the trees are increased substantially through ensemble methods. Ensemble methods involves combining several base predictors (in our case we use trees) to produce a very high level predictor that is more accurate and stable.

These methods are broadly classified into two categories. First, averaging/ aggregating methods, in this several base trees are grown independently and the final model predicts either by majority voting or averaging the individual predictors. Bagging trees (Bootstrap aggregating) and Random forests are grown using this method. They usually result in variance reduction. Second category belongs to boosting methods, in which several weak learners are combined sequentially to produce a strong learner. A weak learner is an estimator whose predictions are only slightly better than random guessing. These methods generally reduce both bias and variance of the model. Boosting methods via. Forward Stagewise Additive Modelling (FSAM), like Gradient boosting and Adaptive boosting are going to be the focus of this chapter. We will discuss FSAM and its connection to Boosting as established by Hastie et al. in chapter 10 of their book.

4.1 Forward Stagewise Additive Modelling: (FSAM)

The sequential learning in boosting is achieved through FSAM. The final predictor is in the form of an additive expansion as:

$$f(x) = \sum_{m=1}^{\mathcal{M}} \beta_m b(x; \gamma_m), \tag{4.1}$$

where $\beta_m, m = 1, 2, \ldots, \mathcal{M}$ are the expansion coefficients and $b(x; \gamma_m)$ are the elementary base predictors (trees) of the multivariate argument x and set of parameters γ. γ characterizes the base trees that are characterized by several factors like splitting variables, split points and predictions made by the terminal nodes.

Evaluate the average loss $\boldsymbol{L}$ of the model, given by the loss function $\boldsymbol{L}(y, f(x))$, in estimating y by the predictor $f(x)$ over the training data $\mathcal{L}_1$. The total loss is given by:

$$\boldsymbol{L}(f) = \boldsymbol{L}(y, f(x)) = \sum_{i=1}^{N_1} \boldsymbol{L}(y_i, f(x_i)) = \sum_{i=1}^{N_1} \boldsymbol{L}(y_i, \sum_{m=1}^{\mathcal{M}} \beta_m b(x_i; \gamma_m)) \tag{4.2}$$

Then our aim is to find the solution to the following minimizing problem over β_m, γ_m:

$$\min_{\beta_m, \gamma_m} \sum_{i=1}^{N_1} \boldsymbol{L}(y_i, \sum_{m=1}^{\mathcal{M}} \beta_m b(x_i; \gamma_m)) \tag{4.3}$$

Finding the global optimizer for the above optimization problem can be computationally intensive. FSAM approximates the solution to (4.3) by solving the sub-problem at each iteration m and sequentially adding new predictors to the expansion without altering the previous predictor parameters. The FSAM is given by the Algorithm 3. In each

Algorithm 2 FSAM

```
Step 1 : Initialize $f_0(x) = 0$
Step 2 : For $m = 1$ to $\mathcal{M}$;
    (i) Compute:
        $(\beta_m, \gamma_m) = arg \min_{\beta,\gamma} \sum_{i=1}^{N_1} \boldsymbol{L}(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma))$.
    (ii) Set:
        $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$
```

iteration m, the optimal base predictor $b(x; \gamma_m)$ and corresponding β_m is evaluated and added to the existing $f_{m-1}(x)$ expansion term, resulting in $f_m(x)$ and the procedure is repeated(without modifying already added terms).

4.1.1 Gradient Boosting:

From equation (4.2), the loss function (should be differentiable) for a given training data $\mathcal{L}_1$ of size N_1 is:

$$\boldsymbol{L}(f) = \sum_{i=1}^{N_1} \boldsymbol{L}(y_i, f(x_i)) \tag{4.4}$$

For now, ignore the constraint that $f(x)$ is the sum of trees on the training data, then the optimization problem in (4.3) can be re-written in terms of f as a function approximation:

$$\hat{f} = arg \min_{f} \boldsymbol{L}(f), \tag{4.5}$$

The predictor functions, $f \in \mathbb{R}^{N_1}$, are approximated as $f(x_i)$ at every data point $x_i \forall i = 1, 2, \ldots, N_1$. Numerical optimization procedures like "Steepest Descent" solve (4.5) as the sum of component vectors:

$$f_{\mathcal{M}} = \sum_{m=0}^{\mathcal{M}} h_m \;,\; h_m \in \mathbb{R}^{N_1}, \tag{4.6}$$

the initial guess is $f_0 = h_0$ and just like additive expansion, each f_m is formed by consecutive induction with previous f_{m-1}. The $h_m{}'$s are given by $h_m = -\rho_m g_m$, where ρ_m is some scalar that acts as the step length or learning rate and $g_m \in \mathbb{R}^{N_1}$ is the gradient of the loss function in (4.4) at $f = f_{m-1}$. This method can also be seen as a greedy strategy as the gradient is chosen such that the loss function $\boldsymbol{L}(f)$is rapidly decreasing at $f = f_{m-1}$

The individual components of the gradient g_m or residual r_m, ($r_m = -g_m$), are given as:

$$r_{im} = -g_{im} = -\left[\frac{\partial \boldsymbol{L}(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x_i)=f_{m-1}(x_i)} \tag{4.7}$$

It can be noticed that the gradients are defined only at the data points x_i, $\forall i = 1, 2, \ldots, N_1$ but our goal is to generalize $f_{\mathcal{M}}(x)$ for any data x that is not present in the training data. This can be achieved by inducing the base tree by the parameters such that its predictions are close to the negative gradient i.e., our base predictor should be parallel (correlated) to the gradients. This can be equivalently seen as the following solution:

$$\gamma_m = arg \min_{\gamma, \beta} \sum_{i=1}^{N_1} [-g_m(x_i) - \beta b(x_i; \gamma)]^2 \tag{4.8}$$

Now, we will use this generalized constrained gradient over the data distribution, $b(x; \gamma_m)$, to replace the $g_{im}{}'$s (since they are data specific) in the Steepest descent procedure.

The step length ρ_m is obtained by solving through "line search":

$$\rho_m = arg \min_{\rho} \sum_{i=1}^{N_1} \boldsymbol{L}(y_i, f_{m-1}(x_i) + \rho b(x_i; \gamma_m)) \tag{4.9}$$

Then the resultant f is updated as:

$$f_m = f_{m-1} + \rho_m b(x; \gamma_m) \tag{4.10}$$

and the procedure is repeated during the next iteration and so on.

The base predictor tree is detailed as follows with $|\tilde{T}|$ terminal nodes represented by the regions R_t and corresponding value b_t, $\forall t = 1, 2, \ldots, |\tilde{T}|$:

$$b(x; \{b_t, R_t\}_1^{|\tilde{T}|}) = \sum_{t=1}^{|\tilde{T}|} b_t \mathbb{1}(x \in R_t) \tag{4.11}$$

For this regression tree, equation (4.10) can be written as:

$$f_m(x) = f_{m-1}(x) + \rho_m \sum_{t=1}^{|\tilde{T}_m|} b_{tm} \mathbb{1}(x \in R_{tm}) \tag{4.12}$$

The regions $\{R_{tm}\}_1^{|\tilde{T}_m|}$ are formed by gradient descent as in accordance with equations (4.7) and (4.8) and $\{b_{tm}\}$ are the least squares coefficients given by:

$$b_{tm} = \; ave_{x_i \in R_{tm}} r_i$$

Equation (4.12) can also be written as:

$$f_m(x) = f_{m-1}(x) + \sum_{t=1}^{|\tilde{T}_m|} \lambda_{tm} \mathbb{1}(x \in R_{tm}) \tag{4.13}$$

with $\lambda_{tm} = \rho_m b_{tm}$. Equation (4.13) can be seen as adding $|\tilde{T}_m|$ separate basis functions at each iteration m, $\{\mathbb{1}(x \in R_{tm})\}_1^{|\tilde{T}_m|}$, unlike equation (4.12), where a single term is added at each iteration m. Thus, the quality of the fit can be improved by these optimal coefficients for each of these separate basis functions (4.13). These optimal coefficients are obtained by solving:

$$\{\lambda_{tm}\}_1^{|\tilde{T}_m|} = arg \min_{\{\lambda_t\}_1^{|\tilde{T}_m|}} \sum_{i=1}^{N} \boldsymbol{L}\left(y_i, f_{m-1}(x_i) + \sum_{t=1}^{|\tilde{T}_m|} \lambda_t \mathbb{1}(x \in R_{tm})\right).$$

Since the regions of the regression trees are disjoint, this reduces to:

$$\lambda_{tm} = arg \min_{\lambda} \sum_{x_i \in R_{tm}} \boldsymbol{L}(y_i, f_{m-1}(x_i) + \lambda) \tag{4.14}$$

The commonly used loss functions and their gradients are given in the table (4.1).

Algorithm 3 GRADIENT BOOSTING

```
**Step** 1 : Initialize $f_0(x) = arg\min_\lambda \sum_{i=1}^{N_1} \boldsymbol{L}(y_i, \lambda)$
**Step** 2 : For $m = 1$ to $\mathcal{M}$;
    (i) For $i = 1, 2, \ldots, N_1$ compute
            $r_{im} = -\left[\frac{\partial \boldsymbol{L}(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}$
    (ii) Fit a regression tree to the targets $r_{im}$ obtaining terminal regions $R_{tm}$, $t = 1, 2, \ldots, |\tilde{T}_m|$
    (iii) For $t = 1, 2, \ldots, |\tilde{T}_m|$ compute
            $\lambda_{tm} = arg\min_\lambda \sum_{x_i \in R_{tm}} \boldsymbol{L}(y_i, f_{m-1}(x_i) + \lambda)$
    (iv) Update $f_m(x) = f_{m-1}(x) + \sum_{t=1}^{|\tilde{T}_m|} \lambda_{tm} \mathbb{1}(x \in R_{tm})$
**Step** 3 : Output $\hat{f}(x) = f_{\mathcal{M}}(x)$
```

Setting	**Loss Function**	$-\partial \boldsymbol{L}(y_i, f(x_i))/\partial f(x_i)$
Regression	$\frac{1}{2}[y_i - f(x_i)]^2$	$y_i - f(x_i)$
Regression	$\|y_i - f(x_i)\|$	$sign[y_i - f(x_i)]$
Regression	Huber	$y_i - f(x_i)$ for $\|y_i - f(x_i)\| \leq \delta_m$, $\delta_m sign[y_i - f(x_i)]$ for $\|y_i - f(x_i)\| > \delta_m$; Where $\delta_m = \alpha th - quantile\{\|y_i - f(x_i)\|\}$
Classification	Deviance	jth component: $\mathbb{1}(y_i = j) - p_j(x_i)$

Table 4.1: Gradients for commonly used loss functions.

For the classification task of J classes, we have the class probabilities $p_j(x)$, $j = 1, 2, \ldots, J$ and the generalized logistic model for J classes is:

$$p_j(x) = \frac{e^{f_j(x)}}{\sum_{l=1}^{J} e^{f_l(x)}}, \ \forall j = 1, 2, \ldots, J. \tag{4.15}$$

This ensures that $0 \leq p_j(x) \leq 1$, $\sum_{j=1}^{J} = 1$ and $f_j(x)'$ s are the respective predictors for each class. A symmetrical constraint $\sum_{j=1}^{J} f_j(x) = 0$, is used on the predictors in order for them to be estimable, since these predictors are redundant i.e., adding some arbitrary $h(x)$ to each of the $f_j(x)$ does not change the loss of the model given by (4.17).

The J-class multinomial deviance loss function is given as:

$$\boldsymbol{L}(y, p(x)) = -\sum_{j=1}^{J} \mathbb{1}(y = j) \log p_j(x) \tag{4.16}$$

Using equation (4.15), we can rewrite the loss function as:

$$\boldsymbol{L}(y, f(x)) = -\sum_{j=1}^{J} \mathbb{1}(y=j) f_j(x) + \log\left(\sum_{l=1}^{J} e^{f_l(x)}\right) \tag{4.17}$$

Here, the loss function penalizes the predictors for incorrect classifications linearly in their degree of incorrectness.

The gradient boosting algorithm for classification is also similar. At each iteration m, the step $2(i)-(iv)$ are repeated J times, once for each class the tree T_{jm} is fitted closest to its corresponding negative gradient vector g_{jm} given by:

$$-g_{ijm} = \left[\frac{\partial \boldsymbol{L}(y_i, f_1(x_i), \ldots, f_J(x_i))}{\partial f_j(x_i)}\right]_{f(x_i)=f_{m-1}(x_i)} = \mathbb{1}(y_i = j) - p_j(x_i) \tag{4.18}$$

At the Step 3, there will be J separate tree expansions $f_{j\mathcal{M}}$, $j = 1, 2, \ldots, J$ and the resultant class is selected for which the class probability is maximum i.e.,

$$j:\ j = arg\ max_l\ p_l(x),\ \forall l = 1, 2, \ldots, J$$

4.1.2 Adaptive Boosting:(Multi-class)

Now, we will examine the multi-class adaptive boosting algorithm called *SAMME*- Stagewise Additive Modelling using Exponential loss function, proposed by Zhu et al.. As the name suggests, this method is also based on FSAM and uses exponential loss function ($\boldsymbol{L}(y, f) = \exp^{-yf(x)}$), this algorithm is actually modified version (for multi-class) of the original AdaBoost by Freund and Schapire(can be obtained from SAMME by using $J = 2$) algorithm for two classes.

The theoretical justification that the SAMME ADABOOST algorithm forms a FSAM with exponential loss and obtains a minimizer are proven in section 2.3 of [Chu et al., 2020] for binary classifier and in section 2 of [Zhu et al., 2006] for multi-class classifier.

Similarly, Adaptive boosting can be extended to regression problems using squared error as loss function, refer to [Drucker, 1997] for the algorithm and procedure.

4.2 Boosting Parameters:

4.2.1 Right Sized Trees:

Here the total number of terminal nodes, $|\tilde{T}_m|$, is a model hyper-parameter. One approach can be, at each iteration m, we can grow very large trees and use cost-complexity pruning to estimate $|\tilde{T}_m|$. But [Hastie et al., 2009] found this approach is found to

```
Algorithm 4 SAMME ADABOOST
Step 1 : Initialize equal weights $w_i$ to all the data in the training sample $\mathcal{L}_1$, i.e.,
    $w_i = \frac{1}{N},\ \forall i = 1, 2, \ldots, N_1.$
Step 2 : For $m = 1$ to $\mathcal{M}$;
    (i) Fit the tree classifier $T^{(m)}(x_i)$ to the training data $x_i$ in proportion to their respective weights $w_i$, $\forall i = 1, 2, \ldots, N_1$.
    (ii) Evaluate:
        $err^{(m)} = \frac{\sum_{i=1}^{N_1} w_i \mathbb{1}(c_i \neq T^{(m)}(x_i))}{\sum_{i=1}^{N_1} w_i}$
    (iii) Evaluate:
        $\alpha^{(m)} = \log \frac{1 - err^{(m)}}{err^{(m)}} + \log(J - 1)$
    (iv) Set:
        $w_i \leftarrow w_i . \exp(\alpha^{(m)} . \mathbb{1}(c_i \neq T^{(m)}(x_i))),\ \forall i = 1, 2, \ldots, N_1.$
    (v) Re-normalize $w_i$.
Step 3 : Output
    $C(\mathbf{x}) = arg \max_j \sum_{m=1}^{\mathcal{M}} \alpha^{(m)} . \mathbb{1}(T^{(m)}(\mathbf{x}) = j)$
```

be not rewarding as typically the size of trees are large and this effects the accuracy and increases computation. Alternatively, they proposal is to fix the size of the tree $|\tilde{T}_m| = |\tilde{T}|$, $\forall m$ and the size of the tree is estimated to maximize the accuracy over training data. After several experiments, they found that the boosting with decision stumps ($|\tilde{T}| = 2$) performs the best but this is not sufficiently enough in all the applications and typically they are chosen in the range $4 \leq |\tilde{T}| \leq 8$.

4.2.2 Regularization:

Another important hyper-parameter for boosting is the total number of iterations, $\mathcal{M}$. As $\mathcal{M}$ increases the total number of trees involved in the boosting procedure also increases and this leads to better accuracy. However, this often leads to over-fitting, sometimes with even zero error on the training data. So, there is an optimal number $\mathcal{M}^*$ that minimizes the error on future data and this can be estimated by minimizing the error on test(unseen) data.

4.2.2.1 Shrinkage Method:

In shrinkage method, the contribution by the tree predictor in the additive modelling at every iteration is scaled by a factor $0 < \nu < 1$, ν acts as a parameter controlling the learning rate, i.e., Step $2(iv)$ in Algorithm 4 is replaced by:

$$f_m(x) = f_{m-1}(x) + \nu. \sum_{t=1}^{|\tilde{T}_m|} \lambda_{tm} \mathbb{1}(x \in R_{tm}) \tag{4.19}$$

Both ν and $\mathcal{M}$ control the error on training data. These are the tuning parameters that are dependent on each other. For a given $\mathcal{M}$, more shrinkage i.e., smaller values of ν results in higher training error. Whereas, for a given error, smaller values of ν results in bigger values of $\mathcal{M}$. Hence, there is a trade off between ν and $\mathcal{M}$.

[Friedman, 2000], empirically found that high shrinkage (small ν) favours better test error which in turn mean larger values of $\mathcal{M}$. Having $\mathcal{M}$ large increases the computational cost. So one can choose a very small ν, typically $\nu < 0.1$, and then choose $\mathcal{M}$ by early stopping. This method leads to significant improvements of accuracy both for regression and classification tasks.

4.2.2.2 Sub-sampling:

Sub-sampling involves using a portion of the training data(sub-sample) instead of whole data. This is similar to the bagging procedure (without replacement). Stochasticity is induced by randomizing the sub-samples at each iteration.

[Friedman, 2002], details the Stochastic gradient boosting, where a fraction η is sampled at each iteration m for growing the trees. The rest of the procedure is similar to Algorithm 4. Typically η is chosen as $\frac{1}{2}$ but if the size of the training data N_1 is large than η can be smaller than $\frac{1}{2}$. The sub-sampling reduces the computation by the factor η and is also found to produce more accurate predictors due to the randomness in the training data.

Therefore, for boosting we have the following tuning parameters: $\tilde{T}, \mathcal{M}, \nu$ and η. Typically, we choose suitable values for $\tilde{T}, \nu$ and η based on empirical explorations and solve for the primary parameter $\mathcal{M}$.

4.3 Interpretation: Relative Importance of Predictor Variables

A single decision tree is very easy to interpret, as the model can be presented as a graphic that is fairly simple to visualize. But in boosting, the final result is due to combination of several trees and here our interest is to find those variables having prominent effect relative to others in predicting the response variable.

The square importance measure (squared relevance or relative importance)of a predictor variable, x_l, is given by $\mathcal{I}_l^2$, and for a given tree T, it is as follows:

$$\mathcal{I}_l^2(T) = \sum_{t=1}^{|T|-|\tilde{T}|-1} i_t^2 \mathbb{1}(x(t) = x_l) \tag{4.20}$$

The sum is over the internal nodes for which split across x_l gives the maximum improvement i_t^2 in squared error estimate.

For an additive tree expansion, like in boosting, this measure over all the trees can be generalized as:

$$\mathcal{I}_l^2 = \frac{1}{\mathcal{M}} \sum_{m=1}^{\mathcal{M}} \mathcal{I}_l^2(T_m) \tag{4.21}$$

As the measure for each predictor is calculated relative to other predictor variables, the predictor with highest value is assigned 100 and others are scaled down accordingly. The relative measure helps in identifying the masked variables.

For $J-$class classification, J separate models, consisting of a sum of trees, for each class are induced i.e., $f_j(x_l)$, $j = 1, 2, \ldots, J$

$$f_j(x) = \sum_{m=1}^{\mathcal{M}} T_{jm}(x) \tag{4.22}$$

The equation (4.21) generalizes to:

$$\mathcal{I}_{lj}^2 = \frac{1}{\mathcal{M}} \sum_{m=1}^{\mathcal{M}} \mathcal{I}_l^2(T_{jm}) \tag{4.23}$$

Where $\mathcal{I}_{lj}^2$ is the relevance of predictor variable x_l in separating the class j objects from the remaining classes. The overall relevance of x_l about all the classes is:

$$\mathcal{I}_l^2 = \frac{1}{J} \sum_{j=1}^{J} \mathcal{I}_{lj}^2 \tag{4.24}$$

Chapter 5

Analysis of Arbres-Urbains Data

5.1 Data Overview:

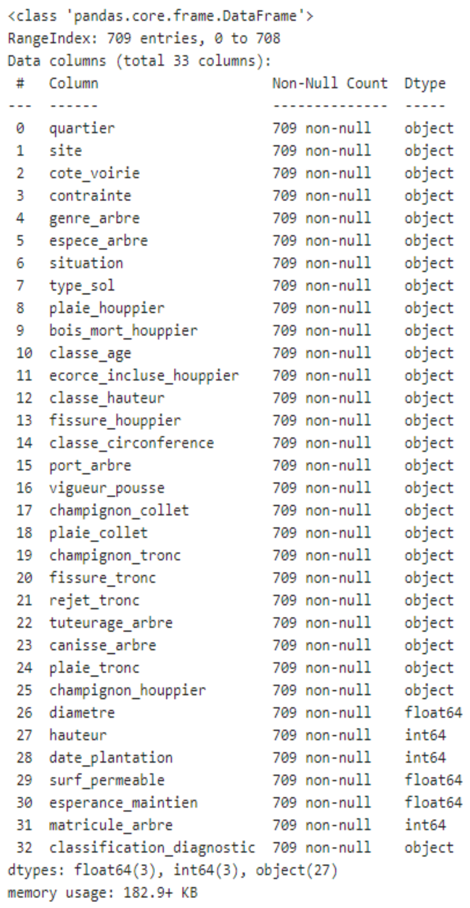
After pre-processing the Data for missing values using the "mode" for categorical variables and "median" for numerical variables, the resultant Data set can be viewed in figure (5.1a)below.

The Arbres-Urbains Data set contains a total of 32 features out of which only 6 are numerical features. Without the loss of the generality, Ordinal Encoding is used to convert remaining 26 categorical features into numeric data, so that we can build mathematical models with the data. The target or dependent variable is "classification_diagnostic" with 5 classes, namely, $C1, C2, C3, C4, C5$. The size of the entire data set is 709. Since the sample size is not very huge, we utilize 90% of it for training the model and the remaining 10% as test sample for validating the model using Cross-validation Algorithm (5) in appendix. The size of training sample is 638 and that of test sample is 71.

To estimate the model validity, "Learning Curve" w.r.t a base decision trees is fitted with varying the size of training set. From, figure 5.1b it is clear that the accuracy of the decision trees increased linearly with increasing training sizes but the the model is severely over-fitted.

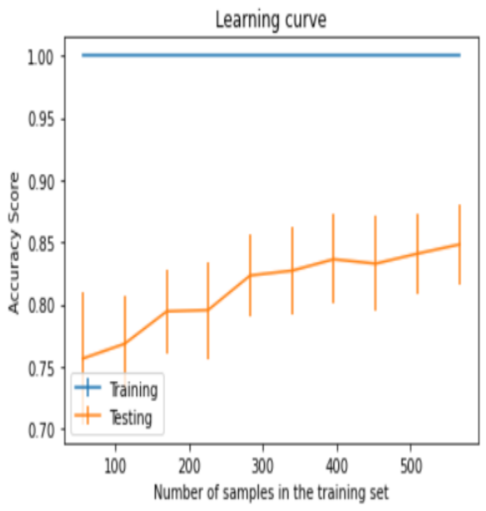
Remark 5.1. *The class labels of the target variable $C1, C2, C3, C4, C5$, are used interchangeably with $0, 1, 2, 3, 4$, respectively because of Label encoding in Sk-learn module.*

The histogram distribution of the data is shown in figure (5.2). It can seen from histogram that the data is mostly very unbalanced and skewed. The target variable, classification_diagnostic's, distribution is as follows: $C1 = 237$, $C2 = 408$, $C3 = 43$, $C4 = 11$, $C5 = 10$.



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 709 entries, 0 to 708
Data columns (total 33 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   quartier                    709 non-null    object
 1   site                        709 non-null    object
 2   cote_voirie                 709 non-null    object
 3   contrainte                  709 non-null    object
 4   genre_arbre                 709 non-null    object
 5   espece_arbre                709 non-null    object
 6   situation                   709 non-null    object
 7   type_sol                    709 non-null    object
 8   plaie_houppier              709 non-null    object
 9   bois_mort_houppier          709 non-null    object
 10  classe_age                  709 non-null    object
 11  ecorce_incluse_houppier     709 non-null    object
 12  classe_hauteur              709 non-null    object
 13  fissure_houppier            709 non-null    object
 14  classe_circonference        709 non-null    object
 15  port_arbre                  709 non-null    object
 16  vigueur_pousse              709 non-null    object
 17  champignon_collet           709 non-null    object
 18  plaie_collet                709 non-null    object
 19  champignon_tronc            709 non-null    object
 20  fissure_tronc               709 non-null    object
 21  rejet_tronc                 709 non-null    object
 22  tuteurage_arbre             709 non-null    object
 23  canisse_arbre               709 non-null    object
 24  plaie_tronc                 709 non-null    object
 25  champignon_houppier         709 non-null    object
 26  diametre                    709 non-null    float64
 27  hauteur                     709 non-null    int64
 28  date_plantation             709 non-null    int64
 29  surf_permeable              709 non-null    float64
 30  esperance_maintien          709 non-null    float64
 31  matricule_arbre             709 non-null    int64
 32  classification_diagnostic   709 non-null    object
dtypes: float64(3), int64(3), object(27)
memory usage: 182.9+ KB
```

(A) Data Overview



(B) Learning Curve

Figure 5.1: Data Validation

Remark 5.2. *In this chapter, only the results are presented. The detailed Python Code using Sk-learn module is given in the appendix (.1). The node size discussion in this chapter refer to maximum leaf nodes (terminal) $|\tilde{T}|$.*

5.2 Building Decision Tree Model:

Initially, simply a Decision Tree Model is built by fitting the training data and accuracy over test data with 5-fold cross validation is found to be 83.1% with approximately 3% standard deviation. The total time taken is 0.55 seconds. Very large trees are allowed to grow without any restriction, and tree sizes during validation usually ranged between 160 to 200 nodes and the maximum depth of the trees is around 13.

5.2.1 Pruning:

Validation curves with varying maximum depth of the tree are shown in figure (5.3), first (5.3a) by Early Stopping at the particular depth and then (5.3b, 5.3c) by pruning

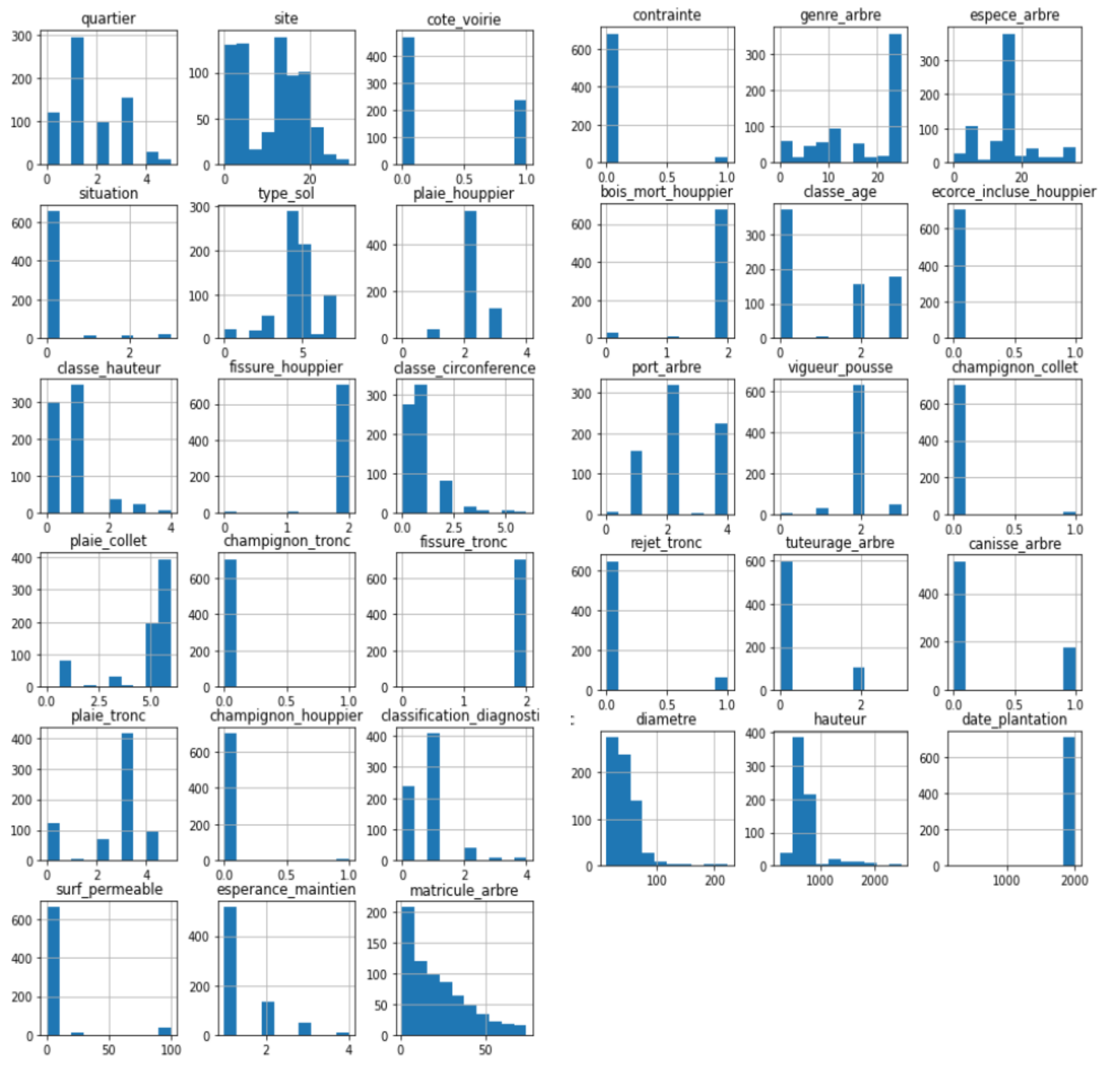


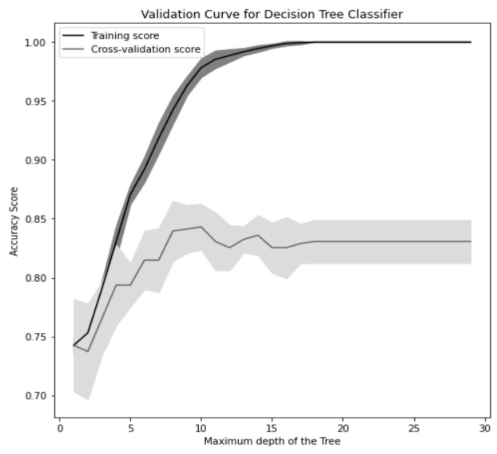
FIGURE 5.2: Data Histogram

a T_{max} by minimal cost-complexity pruning and the α values are calculated by weakest link cutting.

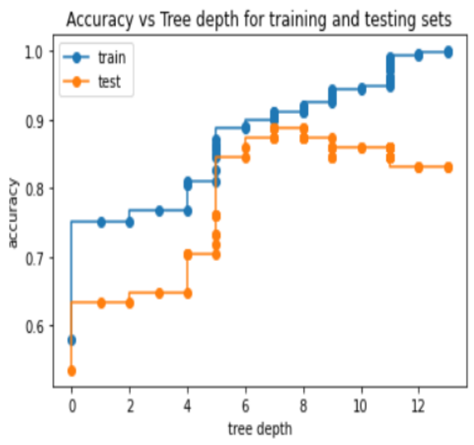
Figure (5.3d) shows the values of $\alpha^{'}$s as the pruning progresses. One can infer from the figures that for the higher sizes of the trees, the model over-fits and the model generalizes better with pruning, as test accuracy increases, characterized by parallel decrease in training error but after a critical value α the model starts to get too simple and under-fits. In the next section, we will try to estimate this critical α and respective tree size parameters.

5.2.2 Tree Parameter Tuning:

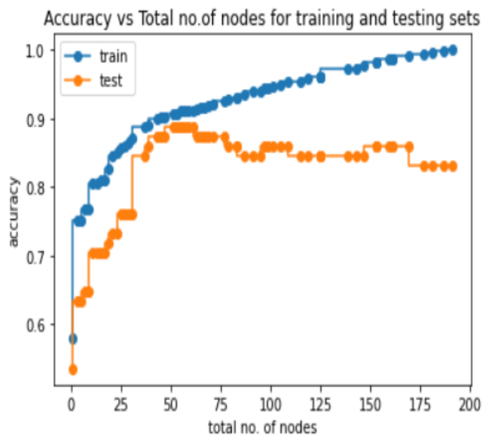
From the first four figures of (5.3) it can be inferred that the critical value of complexity measure, α is less than 0.025 and optimum values of for tree depth and node sizes should in the range $6-10$ and $35-75$, respectively. For finding the best model(hyper) parameter



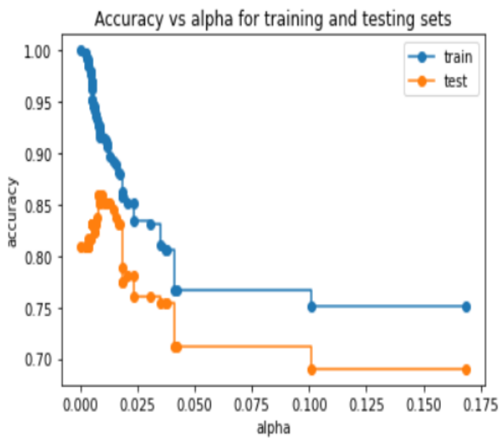
(A) Accuracy vs. Maximum depth of the tree by Early Stopping



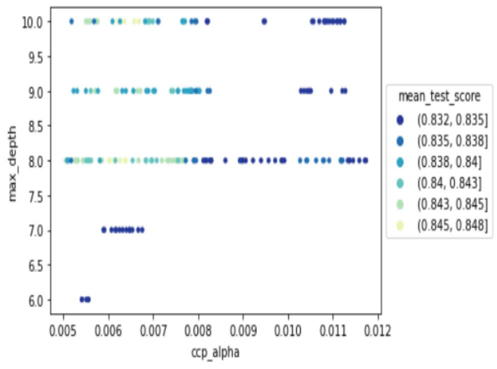
(B) Accuracy vs Depth during Minimal cost-complexity pruning



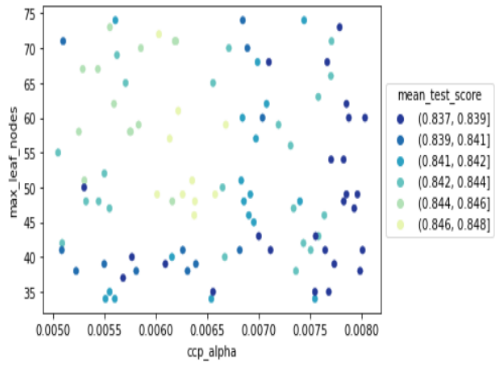
(C) Accuracy vs Total node size during Minimal cost-complexity pruning



(D) Pruning Path by weakest link cutting



(E) Top 10% results for α and depth combination



(F) Top 10% results for α and node size combination

FIGURE 5.3: Validation results w.r.t size of the tree, Pruning Path and Parameter Tuning

α and the tree parameters depth and node size, I randomly tested out 2000 different tree models (Random search tuning) (cross-validated) in the approximate ranges of all these parameters (simultaneously) and analysed the top 10% results with highest accuracy (lowest error). These results can be seen in figure (5.3e and 5.3f). The best parameters (approximately preferring the smaller trees) are found out to be $\alpha = 0.00625$, maximum tree depth= 8 and maximum leaf size= 50. The Accuracy of this model is 84.8%, with standard deviation of approximately 3% and took only 0.09 seconds. Thus, tree pruning not only produced a better model but also a faster one.

5.3 Boosting Models:

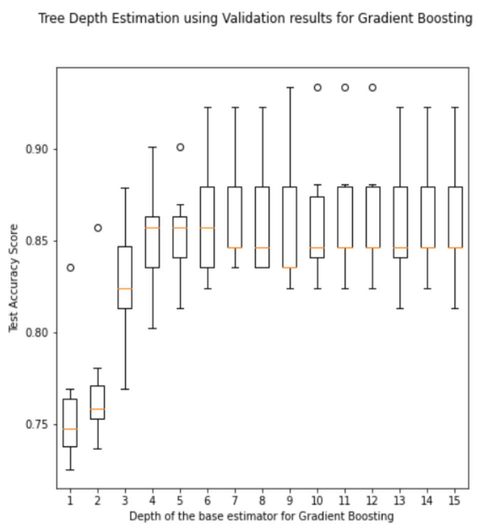
For boosting models, growing the base predictor trees of sizes similar to individual decision tree models or applying pruning is not required as boosting requires the base predictors only to be better than random guessing. In fact the size of trees used in boosting is far less than that of optimal tree found through pruning. However, the size of the estimators in each iteration is a tuning parameter that depends on the feature dependencies, no. of classes in the target variable and also the quality of the training data.

The model (hyper) parameters controls the learning process and assists for better generalisation of the model for the new (unseen) data, i.e., reduced variance (due to over-fitting) and reduced bias (under-fitting). Although, there are several hyper-parameters to tune for, we choose the most prominent ones. For Gradient Boosting, we have Learning rate (ν), Sub-sample ratio(η) and total no. of iterations or (no. of estimators), $\mathcal{M}$ as hyper-parameters. AdaBoost only have Learning rate and no. of estimators as hyper-parameters.

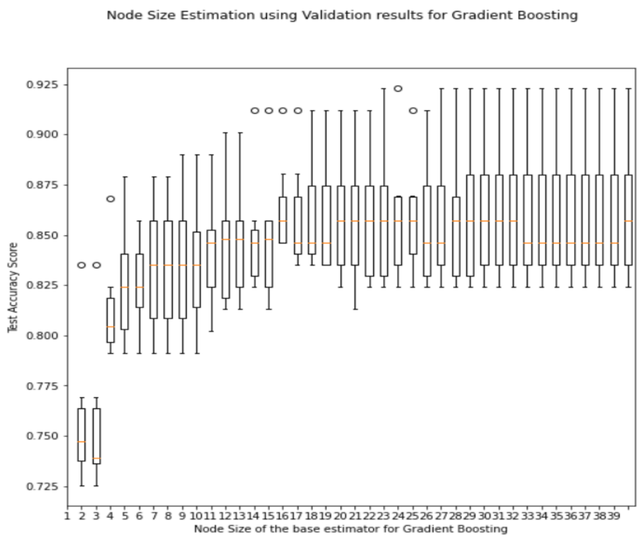
To estimate the tree parameters, validated results are compared for various levels of tree depth and node size, using 500 and 1000 estimators (iterations) for Gradient Boosting and AdaBoost, refer to figures (5.4 and 5.5)respectively. I used less estimators for Gradient Boosting because of its faster convergence rates. Optimum tree depth of 6 and node size of 16 is found for Gradient Boosting, 4 and 11 in case of AdaBoost. These sizes of the weak classifiers are very less when compared to those of 8 and 50 found for a pruned tree.

With these tree sizes, validation curves (refer figures in (5.6) w.r.t no. of estimators and learning rate (with 500 iterations for Gradient Boosting and 1000 for AdaBoost) are constructed to view the model generalisation. It is observed that from figures (5.6a and 5.6c), that both models achieve peak accuracy for a particular value of no. of iterations, after which the accuracy drops, also Gradient Boosting converges faster than AdaBoost and also over-fits very quickly. This is explained by the higher rates of shrinkage required for Gradient Boosting than AdaBoost (refer. 5.6b and 5.6d).

Similar to as in case of decision tree models, various parameter combinations (2000) are generated via Random Search tuning Algorithm (refer to appendix (.1) and validated.

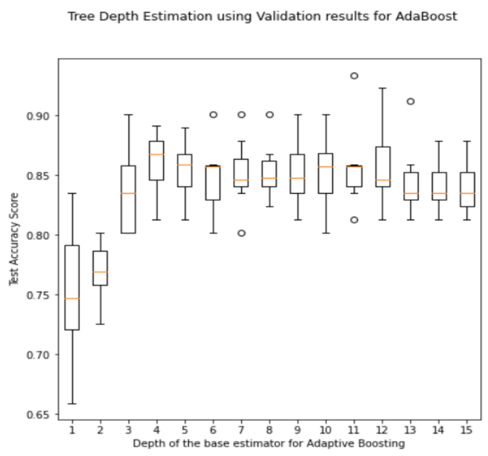


(A) Tree Depth tuning

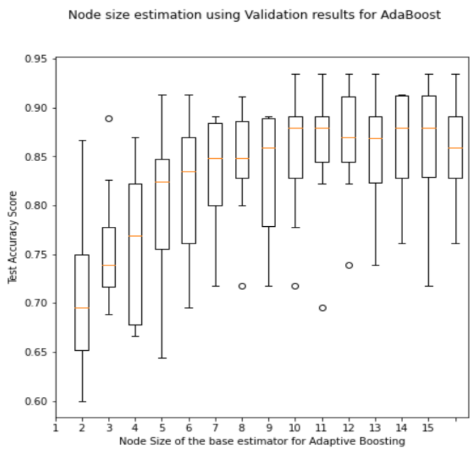


(B) Node size tuning

Figure 5.4: Tree parameters tuning for Gradient Boosting: Box Plot Validation



(A) Tuning for tree depth



(B) Tuning for node size

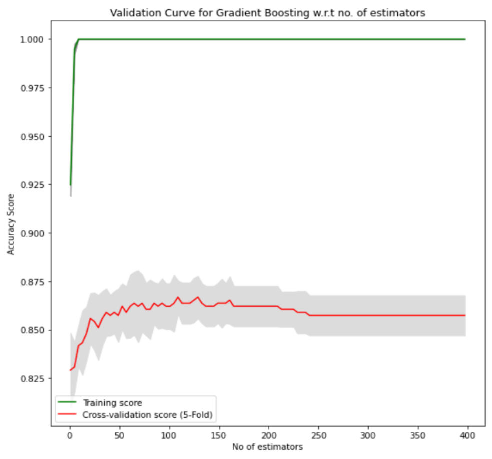
Figure 5.5: Tree parameters tuning for AdaBoost: Validation results (Box Plot)

The top 10% results are compared (refer to figures in 5.7) to find the following hyper-parameters:

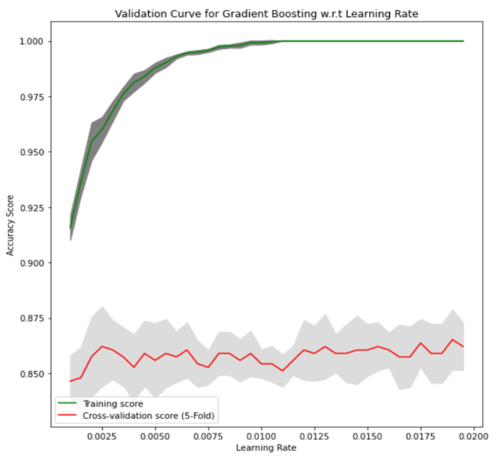
(i) Gradient Boosting: $\nu = 0.01$, $\eta = 0.7$, $\mathcal{M} = 350$

(ii) AdaBoost: $\nu = 0.4$, $\mathcal{M} = 350$

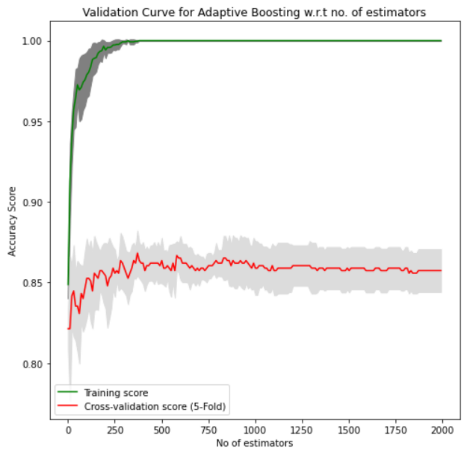
It is interesting to see that even with very high Shrinkage of 0.01, Gradient Boosting still requires iterations than that of AdaBoost. This is mainly because of the combination of robust log-loss function and sub-sampling, whose benefits can be seen in figures (5.8c and 5.8d). The loss function for the model without any regularization increases rapidly, whereas that of regularization converges towards a minimum with Stochastic Gradient Boosting performing the best.



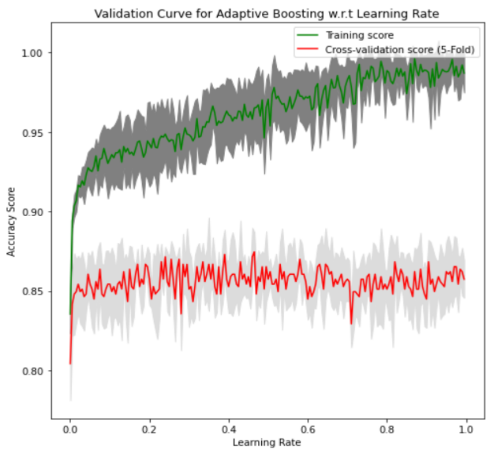
(A) Gradient Boosting: w.r.t no. of estimators



(B) Gradient Boosting: w.r.t Learning Rate



(C) AdaBoost: w.r.t no. of estimators



(D) AdaBoost: w.r.t learning rate

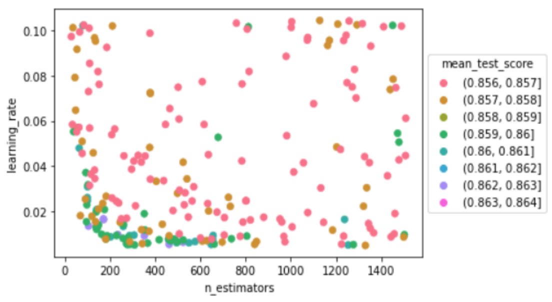
FIGURE 5.6: Validation Curve with tuned tree parameters

The Gradient Boosting Algorithm with these parameters has an accuracy of 86.05% and standard deviation of 1% the computing time is 13.1 seconds. The base estimator accuracy is 83%.

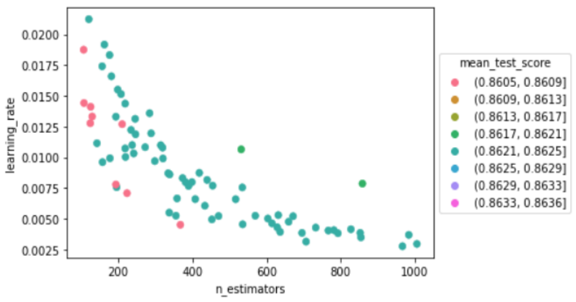
The AdaBoost Algorithm with these parameters has an accuracy of 85.27% and standard deviation of 1.58% the computing time is 3.69 seconds. The base estimator accuracy is 81%.

5.4 Concluding Remarks:

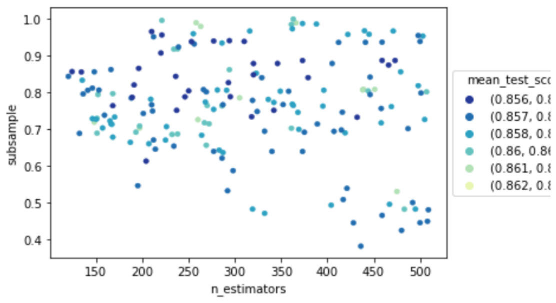
Overall tree methods are found to be very fast computationally and the same time making decent predictions. Coupled that with high interpretability makes the case for them to be included in every machine learning engineer tool-kit. AdaBoost computing times are also very less to that of Gradient Boosting and exhibit similar accuracy. Even after, hyper-parameter tuning the boosting models performance was restricted to a marginal



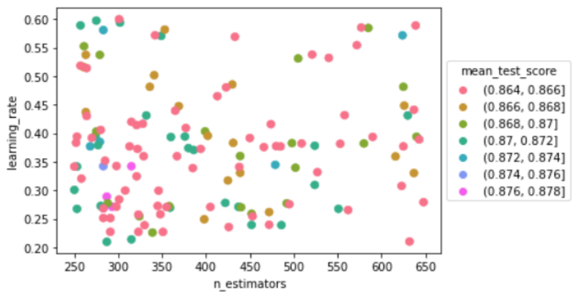
(A) Gradient Boosting: for learning rate (1^{st} iteration



(B) Gradient Boosting: for learning rate (2^{nd} iteration



(C) Gradient Boosting: for sub-sample size



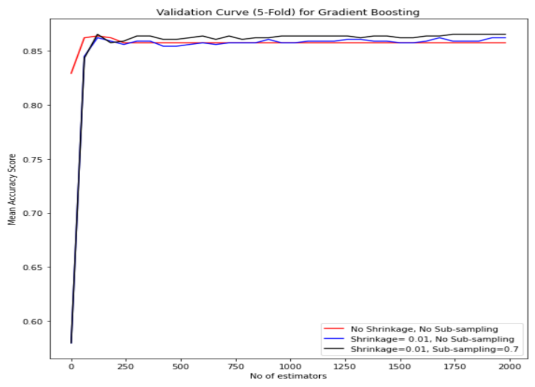
(D) AdaBoost

FIGURE 5.7: Hyper-parameter estimation for Boosting by Random Search

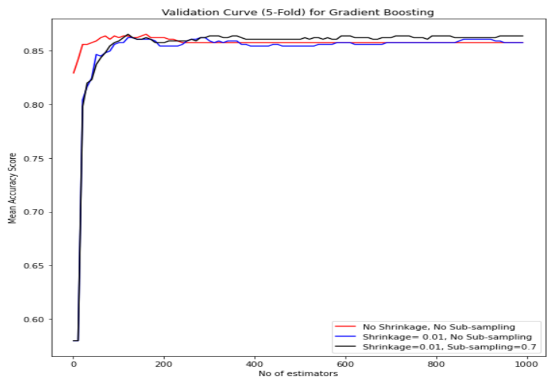
increase to that of a base tree (weak classifier). This suggests that Arbres-Urbains data has good separability and simple tree structures can easily identify the decomposition rules for splitting them into their respective classes.

The alternative models like, logistic regression had an accuracy of only 80.7% and takes a lot more time than trees. But other ensemble methods, Random forests and Bagging models performance is similar to both AdaBoost and Gradient Boosting but they are incredibly fast. But boosting methods are found to be more robust than that of Random forests and Bagging, when other criteria like log-loss or specificity and sensitivity analysis is carried out.

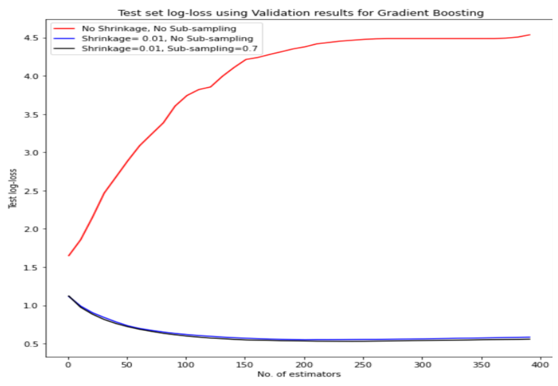
Comparing and generalising based upon analysis of a single data set is difficult and more research is to be made by selecting and testing variety of models with a varied approach. My interest in future to compare these other model evaluation metrics to make a better intuition and judgement on model behaviour and develop the ability to choose the apt direction for making best predictions. However, the goal in this thesis is to prove that with simple cut-based analysis, state-of-art performance can be achieved.



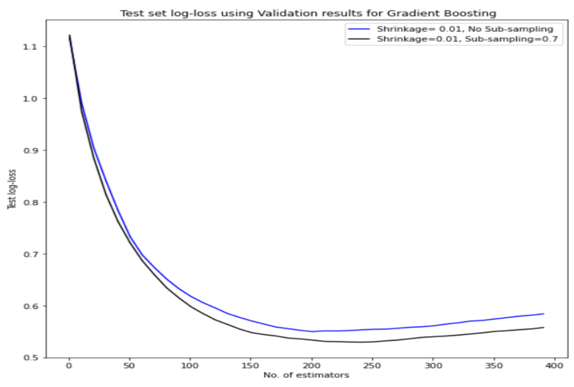
(A) No. of estimators $\mathcal{M}$ range until 2000



(B) Closer view with $\mathcal{M}$ range until 1000

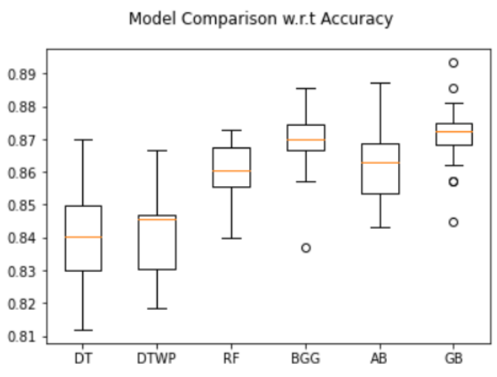


(C) Log-loss comparison with and without regularization

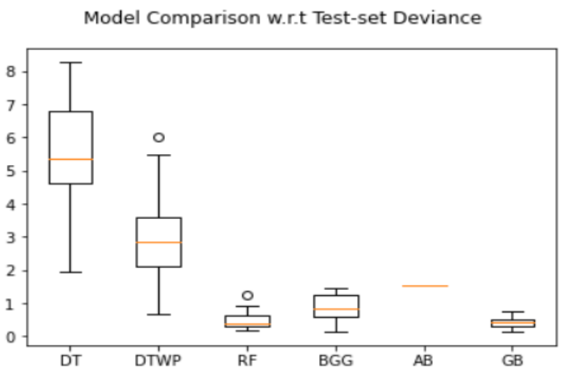


(D) Log-loss comparison for the regularized models

FIGURE 5.8: Meta-parameter estimation for Gradient Boosting through Validation curves



(A) Accuracy



(B) Loss (Test Set Deviance)

FIGURE 5.9: Model Comparison- LR: Logistic Regression, DT: Decision Trees, DTWP: Decision Trees with Pruning, RF: Random Forests, BGG: Bagging, AB: AdaBoost, GB: Gradient Boosting

Bibliography

Arbres-urbains. Municipality of saint-germain-en-laye. URL https://www.data.gouv.fr/fr/datasets/arbres-urbains/#description.

L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth International Group, 1984.

F. M. by Inria Learning Lab. Machine learning in python with scikit-learn. URL https://www.fun-mooc.fr/en/courses/machine-learning-python-scikit-learn/.

J. Chu, T.-H. Lee, A. Ullah, and R. Wang. *Boosting*, pages 431–463. 01 2020. ISBN 978-3-030-31149-0. doi: 10.1007/978-3-030-31150-6_14.

H. Drucker. Improving regressors using boosting techniques. *Proceedings of the 14th International Conference on Machine Learning*, 08 1997.

Y. Freund and R. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55:119–, 08 1997.

J. Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29, 11 2000. doi: 10.1214/aos/1013203451.

J. Friedman. Stochastic gradient boosting. *Computational Statistics and Data Analysis*, 38:367–378, 02 2002. doi: 10.1016/S0167-9473(01)00065-2.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer series in statistics. Springer, 2009. ISBN 9780387848846. URL https://books.google.fr/books?id=eBSgoAEACAAJ.

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 Édouard Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(85):2825–2830, 2011. URL http://jmlr.org/papers/v12/pedregosa11a.html.

R. Quinlan. C4.5 algorithm, a. URL https://en.wikipedia.org/wiki/C4.5_algorithm#Improvements_in_C5.0/See5_algorithm.

R. Quinlan. Id3 algorithm, b. URL https://en.wikipedia.org/wiki/ID3_algorithm.

S. Rubenthaler. *An Introduction to Machine Learning with Probabilities, in R (2021-2022).* 2021. URL https://math.unice.fr/~rubentha/enseignement/poly-machine-learning.pdf.

J. Zhu, S. Rosset, H. Zou, and T. Hastie. Multi-class adaboost. *Statistics and its interface*, 2, 02 2006. doi: 10.4310/SII.2009.v2.n3.a8.

.1 Python Codes:

.1.1 Decision Trees:

```
## PreProcessing
### Importing the libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import time
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder, LabelEncoder
from sklearn.model_selection import train_test_split, GridSearchCV,
RepeatedStratifiedKFold
from sklearn.linear_model import LogisticRegression

### Importing the Data set, Cleaning Data and Handling Missing Values

df = pd.read_csv('sgl-arbres-urbains-wgs84.csv')
df= df.drop(["ID_ARBRE", "commune", "controle", "insecte_collet",
"insecte_tronc", "insecte_houppier","circonference (en cm)",
"observation_collet", "observation_tronc", "observation_houppier"],
axis=1 )
df.info() #For Data Overview
data= df

#Code for Making model pipeline

from sklearn.tree import DecisionTreeClassifier
categorical_columns_selector = selector(dtype_include=object)
categorical_columns = categorical_columns_selector(data)

categorical_preprocessor = OrdinalEncoder(handle_unknown="use_encoded_value",
                                          unknown_value=-1)
preprocessor = ColumnTransformer([
    ('cat_preprocessor', categorical_preprocessor, categorical_columns)],
    remainder='passthrough', sparse_threshold=0)
```

```
#Decision Tree Model (Without Pruning)
model_dt = make_pipeline(preprocessor, DecisionTreeClassifier(random_state=0,
criterion='entropy'))
#Splitting Data set
X_train, X_test, y_train, y_test = train_test_split(
    data, target, random_state=0, test_size=0.1)
_ = df.hist(figsize=(20, 14)) #For Histogram
from sklearn import set_config
set_config(display='diagram')
model_dt

#Code for Cross-Validation results
start = time.process_time()
cv_results_dtmnp = cross_validate(decision_tree_model_no_pruning, X_train, y_train, cv=5)
print("Test Scores for the Decision Tree Model with 5-Fold Cross-Validation:")
print(cv_results_dtmnp["test_score"])
scores = cv_results_dtmnp["test_score"]
print("The Mean Cross-Validation Accuracy is: "f"{scores.mean():.3f} with Standard
Deviation of +/- {scores.std():.3f}")
print("Total time taken: {:}".format(time.process_time() - start))

#Pruning decision trees with cost complexity pruning and analysing results
classifier = DecisionTreeClassifier(random_state=0, criterion= "entropy")
classifier.fit(X_train, y_train)
path = classifier.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas, impurities = path.ccp_alphas, path.impurities
fig, ax = plt.subplots()
ax.plot(ccp_alphas[:], impurities[:], marker="o", drawstyle="steps-post")
ax.set_xlabel("effective alpha")
ax.set_ylabel("total impurity of leaves")
ax.set_title("Total Impurity vs effective alpha for training set")
classifiers = []
for ccp_alpha in ccp_alphas:
    classifier = DecisionTreeClassifier(random_state=0, criterion="entropy",
    ccp_alpha=ccp_alpha)
    classifier.fit(X_train, y_train)
    classifiers.append(classifier)
print(
    "Number of nodes in the last tree is: {} with ccp_alpha: {}".format(
        classifiers[-1].tree_.node_count, ccp_alphas[-1]
    )
)
node_counts = [classifier.tree_.node_count for classifier in classifiers]
depth = [classifier.tree_.max_depth for classifier in classifiers]
train_scores = [classifier.score(X_train, y_train) for classifier in classifiers]
test_scores = [classifier.score(X_test, y_test) for classifier in classifiers]

fig, ax = plt.subplots()
ax.set_xlabel("alpha")
ax.set_ylabel("accuracy")
ax.set_title("Accuracy vs alpha for training and testing sets")
ax.plot(ccp_alphas, train_scores, marker="o", label="train", drawstyle="steps-post")
ax.plot(ccp_alphas, test_scores, marker="o", label="test", drawstyle="steps-post")
ax.legend()
plt.show()
train_scores = [classifier.score(X_train, y_train) for classifier in classifiers]
test_scores = [classifier.score(X_test, y_test) for classifier in classifiers]
node_counts = [classifier.tree_.node_count for classifier in classifiers]
fig, ax = plt.subplots()
ax.set_xlabel("total no. of nodes")
ax.set_ylabel("accuracy")
ax.set_title("Accuracy vs Total no.of nodes for training and testing sets")
ax.plot(node_counts, train_scores, marker="o", label="train", drawstyle="steps-post")
ax.plot(node_counts, test_scores, marker="o", label="test", drawstyle="steps-post")
ax.legend()
plt.show()
train_scores = [classifier.score(X_train, y_train) for classifier in classifiers]
test_scores = [classifier.score(X_test, y_test) for classifier in classifiers]
depth = [classifier.tree_.max_depth for classifier in classifiers]
fig, ax = plt.subplots()
ax.set_xlabel("tree depth")
ax.set_ylabel("accuracy")
ax.set_title("Accuracy vs Tree depth for training and testing sets")
ax.plot(depth, train_scores, marker="o", label="train", drawstyle="steps-post")
ax.plot(depth, test_scores, marker="o", label="test", drawstyle="steps-post")
ax.legend()
plt.show()

#Validation Curve
# Create range of values for the parameter
param_range= np.arange(1, 30, 1)
from sklearn.model_selection import validation_curve
```

```
train_scores, test_scores = validation_curve(classifier,
                                   X_train, y_train, param_name="max_depth",
                                   param_range=param_range,
                                   cv=7, scoring="accuracy", n_jobs=-1)

#Learning Curve
train_sizes = np.linspace(0.1, 1.0, num=10, endpoint=True)
train_sizes
cv = ShuffleSplit(n_splits=20, test_size=0.2, random_state= 0)
from sklearn.model_selection import learning_curve
results = learning_curve(
    classifier, X, y, train_sizes=train_sizes, cv=cv,
    scoring="accuracy", n_jobs=-1)
train_size, train_scores, test_scores = results[:3]
# Convert the scores into errors
train_errors, test_errors = train_scores, test_scores

#Validation Curve with multiple plots like shown for Gradient Boosting
n_est=[]
results_gbll_1=[]
results_gbll_2=[]
results_gbll_3=[]
for i in range (1, 400, 10):
    log_j_1 =[]
    log_j_2 =[]
    log_j_3 =[]
    for j in range (1, 4, 1):
        classifier_1 = GradientBoostingClassifier(loss='deviance', learning_rate=1,
        subsample=1, n_estimators=i, max_depth=6, max_leaf_nodes=16, random_state=j,)
        model_gdbst_1 = Pipeline([("preprocessor", preprocessor),
        ("classifier", classifier_1),])
        model_gdbst_1.fit(X_train, y_train)
        log_ii_1 = log_loss(y_test, model_gdbst_1.predict_proba(X_test))
        log_j_1.append(log_ii_1)

        classifier_2 = GradientBoostingClassifier(loss='deviance',
        learning_rate=0.01,subsample=1, n_estimators=i, max_depth=6, max_leaf_nodes=16,
        random_state=j,)
        model_gdbst_2 = Pipeline([("preprocessor", preprocessor),
        ("classifier", classifier_2),])
        model_gdbst_2.fit(X_train, y_train)
        log_ii_2 = log_loss(y_test, model_gdbst_2.predict_proba(X_test))
        log_j_2.append(log_ii_2)

        classifier_3 = GradientBoostingClassifier(loss='deviance',
        learning_rate=0.01,subsample=0.7, n_estimators=i, max_depth=6,
        max_leaf_nodes=16,random_state=j,)
        model_gdbst_3 = Pipeline([("preprocessor", preprocessor),
        ("classifier", classifier_3),])
        model_gdbst_3.fit(X_train, y_train)
        log_ii_3 = log_loss(y_test, model_gdbst_3.predict_proba(X_test))
        log_j_3.append(log_ii_3)
    n_est.append(i)
    results_gbll_1.append(log_j_1)
    results_gbll_2.append(log_j_2)
    results_gbll_3.append(log_j_3)
plt.subplots(1, figsize=(8,8))
plt.plot(n_est, np.mean(results_gbll_1, axis=1), label="No Shrinkage,
No Sub-sampling", color="red")
plt.plot(n_est, np.mean(results_gbll_2, axis=1), label="Shrinkage= 0.01,
No Sub-sampling", color="blue")
plt.plot(n_est, np.mean(results_gbll_3, axis=1), label="Shrinkage=0.01,
Sub-sampling=0.7", color="black")
plt.title(' Test set log-loss using Validation results for Gradient Boosting')
plt.xlabel("No. of estimators")
plt.ylabel("Test log-loss")
plt.tight_layout()
plt.legend(loc="best")
plt.show()

#Random Search Tuning Algorithm Example code
from sklearn.pipeline import Pipeline
classifier = GradientBoostingClassifier(loss='deviance',
max_depth=6, max_leaf_nodes=16, random_state=0,)

model_gdbst = Pipeline([
    ("preprocessor", preprocessor),
    ("classifier", classifier),
])
from sklearn.model_selection import RandomizedSearchCV

param_distributions = {
    'classifier__n_estimators': uniform_int(10, 2000),
    'classifier__learning_rate': uniform(0.0001, 0.1),
}

model_random_search_gdbst = RandomizedSearchCV(
    model_adabst, param_distributions=param_distributions, n_iter=1000,
    cv=5, verbose=1, random_state =0, n_jobs= 12,
)
model_random_search_gdbst.fit(X_train, y_train)
import seaborn as sns

df_tuning = pd.DataFrame(
    {
        "n_estimators": cv_results["n_estimators"],
        "learning_rate": cv_results["learning_rate"],
        "score_bin": pd.cut(
            cv_results["mean_test_score"], bins=np.linspace(0.854245, 0.87, 9)
        ),
    }
)
sns.set_palette("YlGnBu_r")
ax = sns.scatterplot(
    data=df_tuning,
    x="n_estimators",
    y="learning_rate",
    hue="score_bin",
    s=40,
    color="k",
    edgecolor=None,
)
#ax.set_xscale("log")
#ax.set_yscale("log")

_ = ax.legend(title="mean_test_score", loc="center left", bbox_to_anchor=(1, 0.5))
```

.2 Basic Theory of Classifiers and Regressors:

.2.1 Classifiers:

The main aim of a classification study is to produce an accurate machine learning model (classifier) using the learning sample that classifies the future observations with very high accuracy or to detect the predictive structure of the data presented.

Let us consider we are dealing with a problem of assigning some observations containing certain attributes or features, say $x_1, x_2, \ldots, x_M$ that characterize them into J distinct classes or categories. Let C be the set containing all the classes, i.e., $C = \{1, 2, \ldots, J\}$: $J \in \mathbb{N}$.

Definition .3. *A measurement space $\mathcal{X}$ of dimension M, i.e., $\mathcal{X} \in \mathbb{R}^M$ is a collection of all possible measurement vectors, where a measurement vector x is an ordered vector of features containing information of a particular case in the form of $x_1, x_2, \ldots, x_M$.*

Therefore, a measurement vector $x \in \mathcal{X}$ is given by:

$$x = (x_1, x_2, \ldots, x_M)$$

Remark .4. *Any variable, $x_i \, \forall i \in 1, 2, \ldots, M$, is called ordered or numerical if its measured values are in $\in \mathbb{R}$ and categorical if it takes values in a finite set not having any natural ordering.*

Definition .5. *A classifier or classification rule is a function $d(x)$ defined on $\mathcal{X}$ so that for every x, $d(x)$ is equal to one of the numbers $1, 2, \ldots, J$.*

$$d(x) := j, \forall x \in \mathcal{X} \; \& \; j \in C.$$

A classifier or a classification rule is a systematic rule of predicting what class the case or instance object lies in. Then set A_j is defined as:

$$A_j := x : d(x) = j, \forall x \in \mathcal{X} \; \& \; j \in C.$$

Remark .6. *The sets $A_1, A_2, \ldots, A_J$ are disjoint and $\mathcal{X} = \bigcup_j A_j, \forall j \in C$*

Definition .7. *A classifier is a partition of $\mathcal{X}$ into J disjoint subsets $A_1, A_2, \ldots, A_J$, i.e., $\mathcal{X} = \bigcup_j A_j, \forall j \in C$. Such that for every $x \in A_j$ the predicted class is j.*

Classifiers are constructed by using past observations by means of recorded data referred as learning sample.

Definition .8. *A learning sample consists of data* $(x_1, j_1), \ldots, (x_N, j_N)$ *on* N *cases where* x_n *are the measurement* $x_n \in \mathcal{X}$ *and* $j_n \in C, n = 1, \ldots, N$*. The learning sample is denoted by* $\mathcal{L}$*, and the set* $\mathcal{L}$ *is given by:*

$$\mathcal{L} = (x_1, j_1), \ldots, (x_N, j_N)$$

Remark .9. *If all measurement vectors* x_n *are of fixed dimensionality, say* M*, we say that the data have standard structure.*

Classifier Construction and Bayes Rule:

Let us define $\mathcal{X} \times C$ as the space containing all the possible values of the couple (X, Y) where $X \in \mathcal{X}$ and Y is a class label: $Y \in C$.

Let $P(A, j)$ be a probability on $\mathcal{X} \times C$, $A \subset \mathcal{X}, j \in C$ (for simplicity of discussion Borel measurability will be ignored).

$P(A, j)$ can be interpreted as probability distribution of chance of occurrence of a case drawn at random from the relevant population, such that its measurement vector X is in A and its class Y is equal to j. This notion is formally defined in the following definition.

Definition .10. *Take* $(X, Y), X \in \mathcal{X}, Y \in C$*, to be a new sample drawn randomly from the probability distribution* $P(A, j)$*; i.e.,*

(i) $P(X \in A, Y = j) = P(A, j)$

(ii) (X, Y) *is independent of* $\mathcal{L}$*.*

Assuming the learning sample $\mathcal{L}$ as in definition (.8) consists of N cases $(x_1, j_1), \ldots, (x_N, j_N)$ which can be seen as the N different cases drawn independently at random from the distribution $P(A, j)$. Now, construct the classifier $d(X)$ using $\mathcal{L}$.

Definition .11. *Given a classifier* $d(X)$ *defined on* $\mathcal{X}$ *and taking values in* C*, then its true misclassification rate is defined as the probability of the classifier misclassifying the new sample drawn from the same distribution as* $\mathcal{L}$*. It is denoted by* $R^*(d)$*.*

$$R^*(d) = P(d(X) \neq Y/\mathcal{L}), \tag{1}$$

for a given sample $\mathcal{L}$*.*

Now, we will discuss the concept of Bayes rule which is being used as the guide for constructing of classifiers. Let us say a new sample is drawn in accordance with the definition (.10), then Bayes rule is given as a classifier that has the least misclassification rate. In other words, Bayes rule is the classifier that produces the best predictor i.e., no other classifier can do better than Bayes rule. Therefore, any given classifier's effectiveness can be gauged by comparing it with Bayes rule. It is denoted by $d_B(x)$.

Definition .12. *$d_B(x)$ is a Bayes rule if for any other classifier $d(x)$,*

$$P(d_B(X) \neq Y) \leq P(d(X) \neq Y) \tag{2}$$

Then the Bayes misclassification rate is:

$$R_B = P(d_B(X) \neq Y) \tag{3}$$

Let us formulate the $d_B(x)$ from $P(A,j)$ under a special case.

Definition .13. *Define the prior class probabilities $\pi(j), j = 1, ..., J$, as:*

$$\pi(j) = \mathcal{P}(Y = j) \tag{4}$$

and the probability distribution of the j^{th} class measurement vectors by:

$$P(A|j) = P(A,j)/\pi(j) \tag{5}$$

Remark .14. *We use a different notation $\mathcal{P}$ for the prior probabilities in equation (4) as they are estimated rather than drawn from a random distribution.*

Assumption .15. *$\mathcal{X}$ is M-dimensional euclidean space and for every $j, j = 1, ..., J, P(A|j)$ has the probability density $f_j(x)$; i.e., for sets $A \subset \mathcal{X}$,*

$$P(A|j) = \int_A f_j(x)dx \tag{6}$$

Under the assumption (.15), Bayes rule and Bayes misclassification rate are given by the following theorem,

Theorem .16. *(From [Breiman et al., 1984]), Under Assumption (.15), the Bayes rule is defined by:*

$$d_B(x) = j \text{ on } A_j = \{x; f_j(x)\pi(j) = \max_i f_i(x)\pi(i)\} \tag{7}$$

and the Bayes misclassification rate is:

$$R_B = 1 - \int \max_j [f_j(x)\pi(j)]dx \tag{8}$$

Even though d_B is the Bayes rule, it can also be recognized as a maximum likelihood rule i.e., Classify x as that j for which $f_j(x)\pi(j)$ is a maximum. Also, the solution to

(7) is not unique. Therefore, for those points of x, which has multiple maximizing $j^{'}$s any one of those $j^{'}$s can be chosen at random.

Practically, information regarding Bayes rule i.e., neither $\pi(j)$ nor $f - j(x)$ are known as we do not know anything about the population distribution to which the data belongs. But they can be estimated from the learning sample L, Breiman et al. [1984] discusses methods to estimate Bayes rule in the section 1.5 of Breiman et al. [1984].

Accuracy Estimation:

In practice, we only have fixed data set of features and classes in means of a learning sample $\mathcal{L}$. Hence, we use $\mathcal{L}$ to construct both $d(x)$ and to estimate the $R^{*}(d)$. Such estimates of $R^{*}(d)$ are referred to as internal estimates.

Breiman et al. proposes three kinds of internal estimates. We will discuss them now.

The first one is called *resubstitution estimate* and it is evaluated using the entire learning sample $\mathcal{L}$ that is available for analysis. At first, the classifier d is built and the cases from $\mathcal{L}$ are run through the classifier.

Definition .17. *Then the resubstitution estimate is defined as the proportion of the misclassified cases in the $\mathcal{L}$. It is denoted by $R(d)$, given by:*

$$R(d) := \frac{1}{N} \sum_{1}^{N} \mathbb{1}_{(d(x_n) \neq j_n)} \tag{9}$$

It is to be noted that the data or sample used to estimate resubstitution estimate is same as that used in the construction of the classifier d. This result in very high accuracy of the model (sometimes with zero error!) as almost all the major machine learning algorithms try to minimize $R(d)$. This gives the impression that the model is very accurate but when tested on a new random data this is not the case. This phenomenon is referred as *over-fitting*.

The second one is called *test sample* estimate. For this estimation, the learning sample $\mathcal{L}$ of size N is split into two parts $\mathcal{L}_1$ and $\mathcal{L}_2$ with sample size of N_1 and N_2 respectively. The classifier d is built using the data only from $\mathcal{L}_1$ sample and $R^{*}(d)$ is estimated using the cases in $\mathcal{L}_2$ sample.

Definition .18. *Then the test sample estimate, denoted by $R^{ts}(d)$, is given by:*

$$R^{ts}(d) := \frac{1}{N_2} \sum_{(x_n, j_n) \in \mathcal{L}_2} \mathbb{1}_{(d(x_n) \neq j_n)} \tag{10}$$

This is the most common procedure followed by majority of the machine learning algorithms to get a basic error estimate. The sample $\mathcal{L}_1$ is called *training* sample and the sample $\mathcal{L}_2$ is called the *test* sample. Care must be taken to ensure that $\mathcal{L}_1$ and $\mathcal{L}_2$ are taken at random in order to have an honest and effective estimation. Usually, the ratio of training to test sample ranges from 90 : 10 to 60 : 40 depending on the total sample size N of learning sample $\mathcal{L}$. 90 : 10 is preferred if the N is very less. Higher the sample size, better the estimate will be.

Generally, this method gives better estimates than resubstitution estimate. Still, both methods discussed above suffers from high *variance* and *bias* in estimation due to *over-fitting* or *under-fitting* depending on model complexity.

The third method is called V- *fold cross-validation* and this method is particularly preferred in case of smaller sample sizes. The procedure for V- fold cross-validation is summarized in the algorithm below: It is to be noted that during step 3 of the algorithm,

Algorithm 5 V- fold cross-validation

```
Step 1: Choose the number of folds of cross-validation, V.
Step 2: From the learning sample $\mathcal{L}$ containing $N$ data points, make $V$ distinct subsets by splitting them almost equally in size and let these subsets be denoted as $\mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_V$.
Step 3: For each $v$, $v$ belongs to $1, 2, \ldots, V$
    (i) Consider the new learning sample, $\mathcal{L} - \mathcal{L}_v$ i.e., excluding those cases belonging to the $\mathcal{L}_v$ set.
    (ii) Construct the classifier $d^{(v)}(x)$.
    (iii) Evaluate the test sample estimate using data from $\mathcal{L}_v$ as the test sample, $R^{ts}(d^{(v)})$ for $R^*(d^{(v)})$:
        $$R^{ts}(d^{(v)}) = \frac{1}{N_v} \sum_{(x_n, j_n) \in \mathcal{L}_v} \mathbb{1}_{(d^{(v)}(x_n) \neq j_n)} \quad Where, N_v \approxeq N/V \qquad (11)$$
Step 4: Evaluate the $V$- fold cross-validation, denoted by $R^{CV}(d)$
    $$R^{CV}(d) = \frac{1}{V} \sum_{v=1}^{V} R^{ts}(d^{(v)}) \qquad (12)$$
```

each of the $d(v)$, $v = 1, 2, \ldots, V$, classifiers are constructed with a learning sample of size $N(1 - \frac{1}{V})$. The stability of the cross-validation gets better as the total number of folds V increases, since all the classifiers are using almost all the cases in the original sample $\mathcal{L}$ and have misclassification rates $R^*(d^{(v)})$ very close to $R^*(d)$.

Another method called *bootstrap* can also be used to estimate the $R^*(d)$ but cross validation performs better in tree structures. (see section 11.7 of [Breiman et al., 1984] for details)

.2.2 Regressors:

Regression contains a case of the form (x, y) where x belongs to $\mathcal{X}$ and y is a real-valued number i.e., $y \in \mathbb{R}$. x is the measurement vector and $\mathcal{X}$ is the measurement space as in the definition (.3), respectively. The variables $x_1, x_2, \ldots, x_M$ are referred to as the *independent* or the *predictor* variables. Whereas the variable y whose value is to be predicted is called as the *dependent* or the *response* variable. Like *classification*, the two main aims of regression are to successfully evaluate (with *high* accuracy) the response variable corresponding to the future set of values of independent variables and to understand the structural relationships among dependent and independent variables.

Definition .19. *A regressor or regression rule is a function $d(x)$ defined on measurement space $\mathcal{X}$ taking real values, i.e., $d(x)$ is a real valued function on $\mathcal{X}$.*

$$d(x) := y : \quad y \in \mathbb{R} \tag{13}$$

Remark .20. *The names regressor and predictor are used interchangeably to represent $d(x)$ since they form the prediction (regression) rule to evaluate the dependent variable y.*

Regression analysis involves the construction of predictor or regressor $d(x)$ from the learning sample $\mathcal{L}$.

Suppose the learning sample $\mathcal{L}$, consisting of N cases $(x_1, y_1), \ldots, (x_N, y_N)$, was used to construct a predictor $d(x)$. Now let us assume that we have a test sample of size N_2, given by $(x_1^{'}, y_1^{'}), \ldots, (x_{N_2}^{'}, y_{N_2}^{'})$. Then the accuracy of the predictor $d(x)$ is given by the following two common measures.

Definition .21. *(i) The average error of the predictor $d(x)$ as in $d(x_n^{'})$ as a predictor of $y_n^{'}$, where $n = 1, 2, \ldots, N_2$*

$$\frac{1}{N_2} \sum_{n=1}^{N_2} |y_n^{'} - d(x_n^{'})| \tag{14}$$

Remark .22. *This measure leads to least absolute deviation regression.*

(ii) The average squared error of the predictor $d(x)$ as in $d(x_n^{'})$ as a predictor of $y_n^{'}$, where $n = 1, 2, \ldots, N_2$

$$\frac{1}{N_2} \sum_{n=1}^{N_2} (y_n^{'} - d(x_n^{'}))^2 \tag{15}$$

Remark .23. *This measure leads to least squares regression and is the most classically used one in regression.*

Now, we will proceed in a more theoretical framework which is required in the next section for the error measures and care is taken (for consistency) to use similar notations like $R^*(d)$ (as in classification) for representing the measure of accuracy of a predictor.

Let us say that the vector (X, Y) and the learning sample $\mathcal{L}$ are drawn randomly and independently from the same underlying distribution (similar to as in the definition (.10) for classifiers). Here X denotes the $\mathcal{X}$-valued random variable with some probability distribution and Y denotes the dependent or response variable.

Definition .24. *The mean squared error of the predictor d, denoted by $R^*(d)$, is defined as:*

$$R^*(d) := \mathbb{E}[(Y - d(X))^2] \tag{16}$$

i.e., $R^*(d)$ is the expected squared error using $d(X)$ as a predictor of Y, where the expectation is taken holding $\mathcal{L}$ fixed.

Using the preceding definition (.12) of Bayes rule for classifiers, the Bayes or the optimal predictor has a simple form and is given in the following proposition:

Proposition .25. *(From [Breiman et al., 1984]), The Bayes rule or the predictor d_B which minimizes $R^*(d)$ is:*

$$d_B(x) = \mathbb{E}[(Y|X = x)] \tag{17}$$

Essentially, $d_B(x)$ is the conditional expectation of the dependent variable for a given measurement vector x. The regression surface of Y on X is given by $y = d_B(x)$.

Remark .26. *Also, notice that for any rule d,*

$$R^*(d) = R^*(d_B) + \mathbb{E}[(d_B(X) - d(X))^2]. \tag{18}$$

Accuracy Estimation:

Using the exact procedure as detailed in section (.2.1) for classifiers, for a given learning sample $\mathcal{L}$ of size N, one can use $\mathcal{L}$ to both construct the predictor $d(x)$ and also estimate its error $R^*(d)$. We can give the internal estimates used to estimate R^* for regressors as following:

(i) Resubstitution estimate:

$$R(d) = \frac{1}{N}\sum_n (y_n - d(x_n))^2 \tag{19}$$

Here, the entire sample $\mathcal{L}$ of size N is used in constructing the predictor $d(x_n)$ and estimating the error, $R(d)$.

(ii) Test sample estimate:

$$R^{ts}(d) = \frac{1}{N_2} \sum_{(x_n,y_n)\in\mathcal{L}_2} (y_n - d(x_n))^2 \tag{20}$$

Here, the learning sample is $\mathcal{L}$ is randomly divided into sets of $\mathcal{L}_1$ (training sample) and $\mathcal{L}_2$ (test sample) of sizes N_1 and $N_2 : N = N_1 + N_2$. $\mathcal{L}_1$ is used to construct the predictor $d(x_n) : x_n$ belongs to $\mathcal{L}_1$ and $\mathcal{L}_2$ is used to evaluate the test sample estimate.

(iii) V-fold cross validation estimate:
Here, the learning sample $\mathcal{L}$ containing N data points, is split into V distinct subsets of almost equally in size and let these subsets be denoted as $\mathcal{L}_1$, $\mathcal{L}_2$, ..., $\mathcal{L}_\mathcal{V}$. Entire procedure is exactly the same as in Algorithm (5).
The test sample estimate in the step 3 of Algorithm (5) is given by:

$$R^{ts}(d^{(v)}) = \frac{1}{N_v} \sum_{(x_n,y_n)\in\mathcal{L}_v} (y_n - d(x_n))^2 \quad Where, N_v \approxeq N/V \tag{21}$$

The V- fold cross-validation estimate is given by:

$$R^{CV}(d) = \frac{1}{V} \sum_{v=1}^{V} R^{ts}(d^{(v)}) \tag{22}$$

Normalization and scaling of the error:

Unlike the misclassification rate in classification, the mean squared error lacks the intuitive meaning and interpretation as the value of $R^*(d)$ depends on the scale of the dependent variable.
To avoid this scale dependency, a normalised measure of accuracy is often used.
Let $\mu = \mathbb{E}(Y)$.
Then $R^*(\mu) = \mathbb{E}(Y - \mu)^2$ is the mean squared error using the μ (a constant) as a predictor of Y. This error is also the variance of Y.

Definition .27. *The relative mean squared error $RE^*(d)$ in $d(x)$ as a predictor of Y is:*

$$RE^*(d) = \frac{R^*(d)}{R^*(\mu)} \tag{23}$$

The idea is that μ is the baseline predictor for Y if there is no information available about X. Then the performance of any predictor d based on X is judged by comparing its mean squared error $R^*(d)$ to that of μ, $(R^*(\mu))$.

The relative mean squared error is always non-negative. It has a value that is usually (but not always) less than 1. For good predictors $d(x)'s$ are more accurate than μ, and hence $RE^*(d) < 1$. But occasionally, we have some poor predictors $d(x)'s$ whose performance is worse than the baseline predictor μ and they have $RE^*(d) \geq 1$.

Let,

$$\bar{y} = \frac{1}{N} \sum_n y_n$$

and,

$$R(\bar{y}) = \frac{1}{N} \sum_n (y_n - \bar{y})^2 \tag{24}$$

Then the relative mean squared error for the internal estimates are given as:

(i) Resubstitution estimate:

$$RE(d) = \frac{R(d)}{R(\bar{y})} \tag{25}$$

(ii) Test sample estimate:

$$RE^{ts}(d) = \frac{R^{ts}(d)}{R^{ts}(\bar{y})} \tag{26}$$

(iii) V-fold cross validation estimate:

$$RE^{CV}(d) = \frac{R^{CV}(d)}{R(\bar{y})} \tag{27}$$