Classification and Regression Trees (CART)

### Student: Yucehan Kucukmotor

### Professor: Lawrence Tatum

### Class: CIS/STA3920[MNSA]

### Due: 12/20/17 by 6PM

Table of Contents

**1. Introduction to Classification and Regression Trees (CART)3**

1.1 Overview3

1.2 History3

1.3 Classification and Regression Trees Comparison3

2. **Classification and Regression Trees 4**

2.1 Growing the Tree and Split Selection5

Classification Tree Construction 5

2.2 Implementation in R using rpart and rpart.plot 6

Mammals Example on 4 variables using rpart8

2.3 Loss Functions 10

2.4 Tree Pruning 11

Cost-Complexity Pruning 12

**3. References 13**

##### 1. introduction to classification and regression trees

**1.1 Overview**

Classification and regression tree (CART) is one of the well-known machine-learning methodologies to construct prediction models from our dataset. Thus, decision trees are very commonly used in data mining for the objective of establishing a model that predicts the value of a target (or dependent variable) based on the values of several input (or independent variable).

**1.2 History**

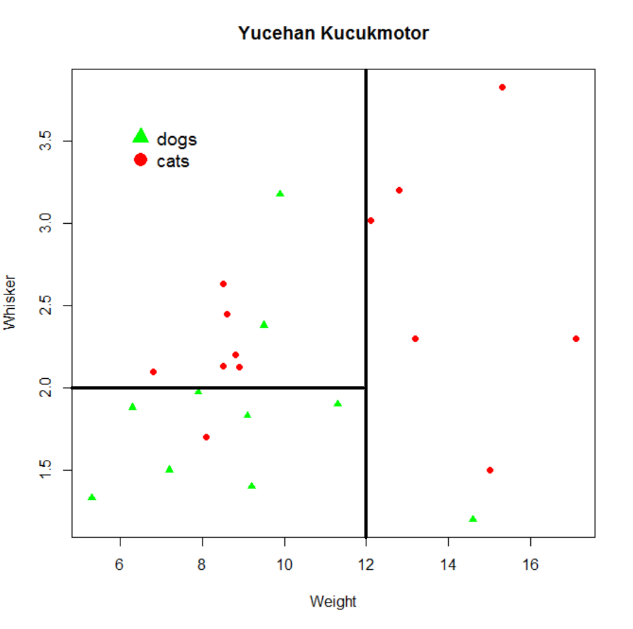
The CART methodology was first introduced in 1984 by Leo Breiman, Richard Olshen, Charles Stone, and Jerome Friedman as a term to refer to *Classification* trees as well as *Regression* trees.

**1.3 Classification vs. Regression trees**

To obtain our prediction models, we recursively (repeatedly) partition the plane in which our data is present and fit a prediction model within each divided space, with as little error while doing so as possible. As a result of this recursive process, the partitioning can be represented graphically as a decision tree. Both classification and regression trees are designed to take a dependent variable, however, there is a difference between the two. Classification trees, other than dependent variables, also take finite number of ordered values, with prediction error measured in terms of misclassification cost. On the other hand, regression trees are for, apart from dependent variables, continuous or ordered discrete values, with prediction error typically measured by the squared difference between the observed and predicted values. At this point, it would be critical to understand what is meant by measurement of prediction error. Regardless of the models we are establishing, we would have to choose a *loss function* (*In machine learning and mathematical optimization, loss functions for classification are computationally feasible loss functions representing the price paid for inaccuracy of predictions in classification problems, that is, problems of identifying which category a particular observation belongs to)* to reduce the prediction error rate as much as possible. Thus, after building models, we can choose the model that yields the lowest error rate. Above mentioned loss functions are some of the most commonly used, however, there are a variety of different loss functions available for users to choose from! Some examples are: Square loss (more commonly used in regression), hinge loss, logistic loss, Gini index, twoing, phi coefficient, entropy, and so on.

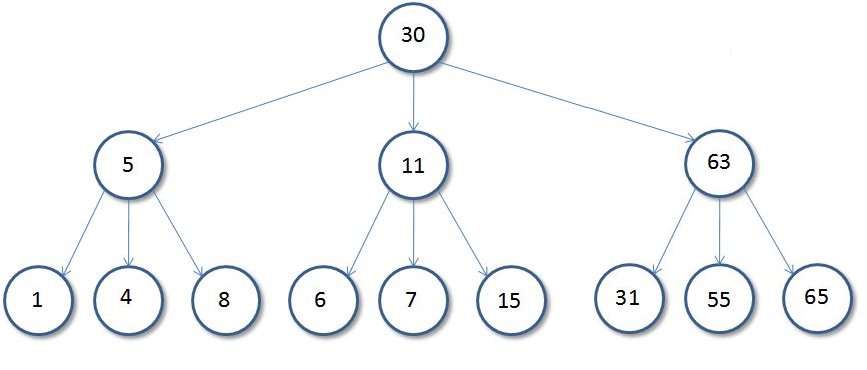
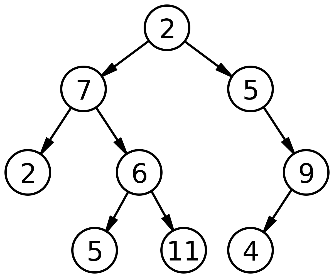
Trees are visualizations beginning with one node, which is known as the root node, and branching into multiple leaves. Ease-of-use and popularity of CART led many people to choose CART over other alternatives for prediction such as regression, discrimination analysis, as well as other algebraic models.

##### 2. Classification Trees and regression trees

**2.1 Growing the Tree and Split Selection**

Before getting to the coding aspect of CART, it is important to understand and visualize the process in our heads. To the right is a graph that I had constructed while working on “Lecture Notes 5 (exploring k-NN with R)”. Black lines are to show an example of a simple partitioning for my data set. When splitting, it is important to choose coordinates (points) that would try to separate different data as much as possible. As can be seen, I split my data into three different parts. Thus, I could separate triangles (dogs), and circles (cats) as much as I could. A good splitting would yield a stronger model with a higher accuracy.

After splitting the data, our next stop would be growing the binary tree. Our tree is going to be called a “binary tree” because each node is split into only two subsamples. Classification trees (or regression trees for that matter) do not have to binary, yet most are. Nonbinary (multibranch) trees do not hold any superiority over binary trees. Here is a difference between a binary and a ternary tree:

 Ternary tree Binary tree

As can be seen above, binary trees have two nodes that derive from each other. However, ternary trees have three nodes deriving from another node. The top node (the root node) contains the entire sample of our data. As we make our leaves go down, each node contains a subset of the sample in the node right above it. Not only that, each node contains the sum of the samples in the nodes right below it. Take binary tree above for example. Node 2 contains the entire sample. However, node 11 is a subset of node 6. Intuitively, node 5 and node 11 combined gives us the node 6 since both node 5 and node 11 are subsets of node 6. Likewise, node 6 and node 2 happens to be the subsets of node 7, which is a subset of node 2.

Now that we have an understanding of how to split the data (or how a given algorithm achieves it), and what nodes are, we can get to building a tree using our knowledge from splitting and understanding of the nodes.

Let’s name our x-coordinate (Weight) Xi1 and our y-coordinate (Whisker) Xi2. Our sequence of splits would look like this: Nodes are in (x,y) format in which x represents the number of triangles, and y represents the number of circle. (9,7) states there are 9 triangles and 7 circles in the sample for given condition. Left lane represents condition being false and right lane represents condition being true.

Xi1>12

No

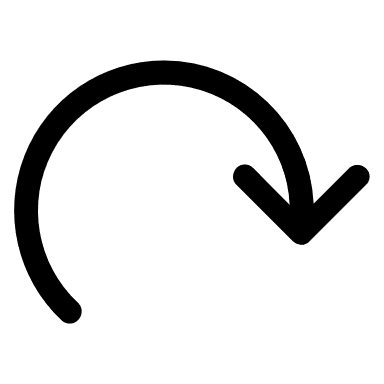
(9,7) (1,6)

Xi2>2

No

(7,1) (2,6)

Although it is not likely for us to encounter such simple decision tree since real life data is often big and complex, understanding the process helps us grasp the idea behind growing trees. This is where ‘pruning the tree’ term comes into action. A classification tree usually consists of two phases. 1) Growing the tree (during which a very large decision tree is constructed from our training data), and 2) Pruning the tree (the ultimate size of the tree that is determined with the goal to minimize Rtree (Rt(P) to represent misclassification rate and it is to be minimal for a more accurate and consistent prediction model). All decision tree algorithms grow the tree in a top-down fashion though they may have other differences. Here’s an example of Decision-Tree Induction Schema by The Handbook of Data Mining by Nong Ye(2003):

 **Classification tree construction:**

n denotes: Node, D denotes: Dataset, SS: denotes our method for splitting

*Input*: Node n, partition D, split selection method SS

*Output*: Decision tree for D rooted at node n

**Top-Down Decision Tree Induction Schema**:

Growing the Tree

1) Apply SS (splitting method) to D (dataset) to find the splitting criterion

2) **if** n splits

3) Use best split to partition D into D1 and D2

4) BuildTree(n1, D1, SS)

5) BuildTree(n2, D2, SS)

6) **endif** (our ending criteria)

At our root node n, our training data is examined, and a criterion for splitting the data is selected. Recursively, at other nodes (not the root node), the node itself and its subsamples are examined and a splitting criterion for other nodes is selected. Above schema depicts these steps.

If confused about different terms such as branches, tree, nodes, and such, one may think of decision tree as a flowchart-like structure in which each node denotes a test, each branch denotes an outcome of the test. Our root node is the uppermost node and holds the entire sample of the data. So, we have our dataset and our algorithm recursively partitions the data. How and when does it stop? The recursive partitioning stops when one of the following stopping (terminating) conditions is true:

1) All the data in a partition D belong to the same class (All cats or all dogs)

2) There are no remaining attributes on which the tuples may be further partitioned. When our algorithm can no longer partition the data, a majority voting system is conducted. That is, a node is converted into a leaf and named whichever class is most common in that node. (Say, when our algorithm divides the plane in such way that within our division we have 5 cats and 6 dogs. A majority-vote implementation is held, and all cats voted for cats, and all dogs voted for dogs. Since dogs outnumber cats, node is labelled dogs, which is unfortunate for cats).

**2.2 Implementation in R using rpart**

rpart is a package that is included in the R and it stands for recursive partitioning and regression trees. To install rpart if it is not included in your R version for some reason, you can run the following code to install it

> install.packages(“rpart”) # to install rpart package

> install.packages(“rpart.plot”) # to install rpart’s plotting package

Although there are other packages that can be used for implementing prediction analysis using decision trees such as *tree* package, we will be using rpart. Syntax and implementation using rpart and tree is not very different than each other and both can be used depending on the personal preference. To download *tree* package, just like we did above, you can write the following code on your R console:

> install.packages(“tree”) # to install tree package

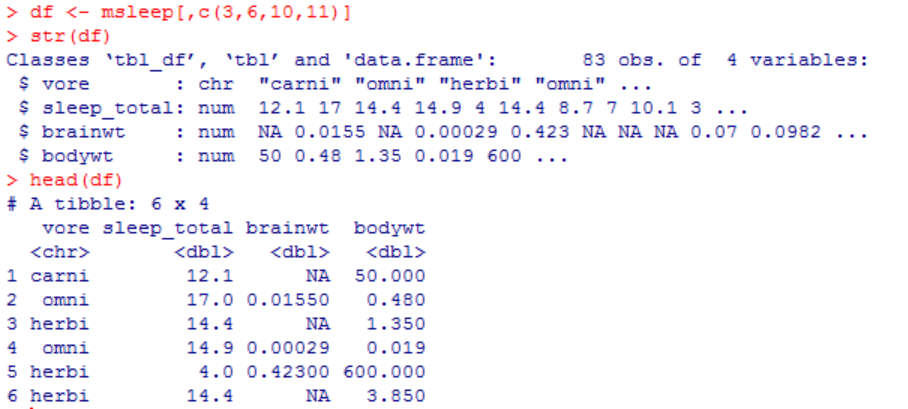
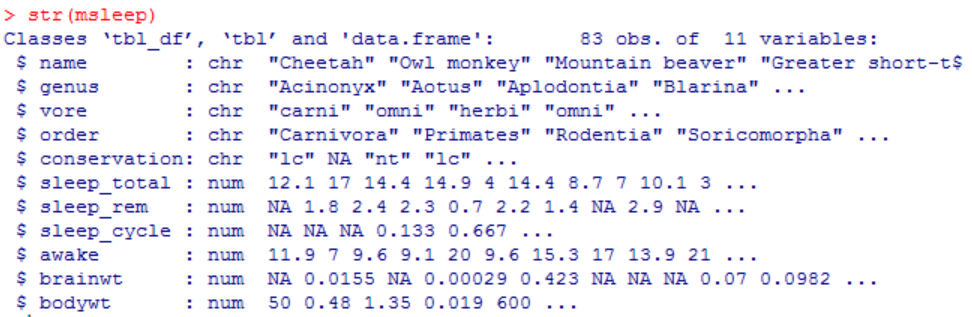
Now that we have downloaded the rpart package, we need to require (load) it to get on with our work. To load the package, all we have to do is execute the following commands on R console:

> library(rpart) # to load rpart package

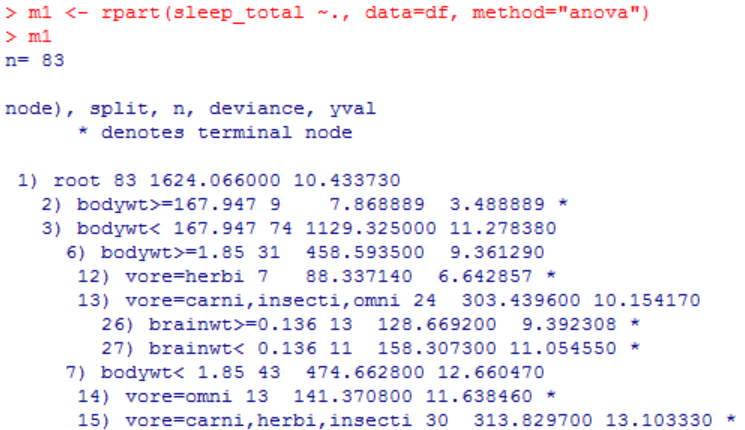
> library(rpart.plot) # to load rpart.plot package

I will be using *msleep* as my dataset which is included in *ggplot2* package.

*msleep: An updated and expanded version of the mammals’ sleep dataset*

str() helps us see the structure of our dataset, which is msleep in this case. Our dataset has 11 variables. However, we will not be including all the variables in our work.

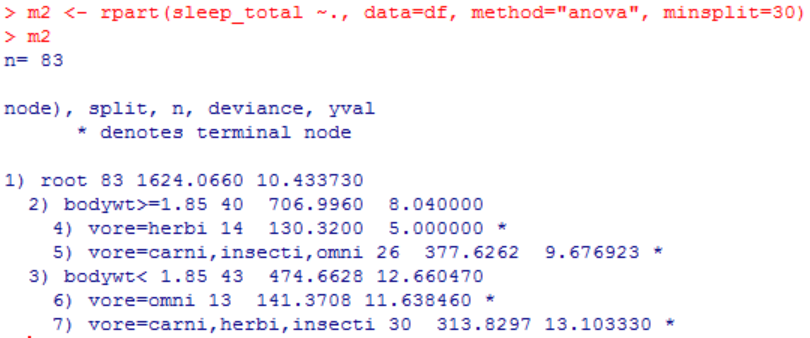
Since I am only interested in using vore, sleep\_total, brainwt, and bodywt as my variables, I have created a data frame and attached it to the name df for easier and faster execution. Then I used str() function again to see the updated structure of my dataset which is named df now. Now we have 83 observations and 4 variables.

Predicting Total Sleeping Hours of Mammals Based on 4 variables

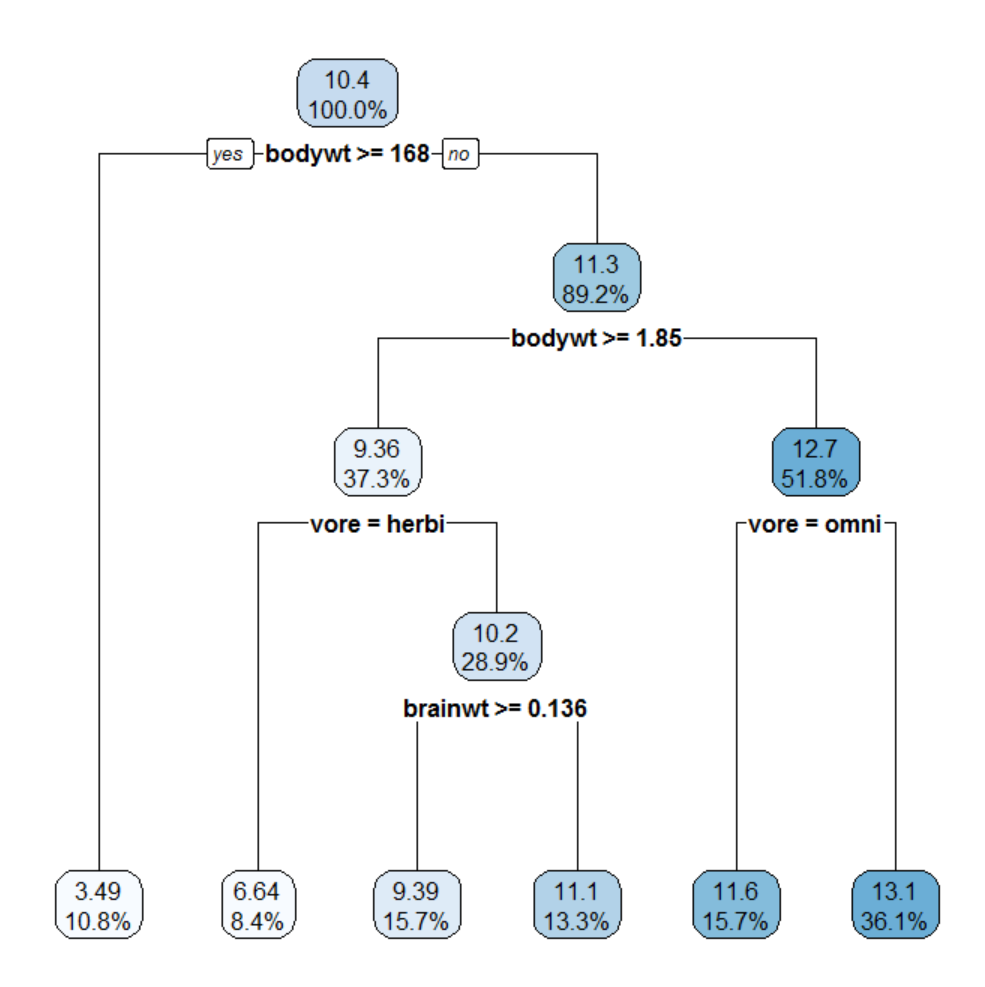
rpart is passing 3 arguments. First is our target feature which is what we want to predict. Since we want to predict total sleeping hours of mammals, first argument we need to pass in our function is sleep\_total. ~ is used to specify predictors, and “.” is used to imply that we want to include all the useful predictors. Second argument we are passing is our data which we named df on the previous page. Last argument we need to pass is our method. Our method can either be “anova” or “class” depending on our variables and what tree we want to grow. To grow a classification tree, our method should be method=”class”, to grow a regression tree, our method should be method=”anova”. Since our target value is numerical (we want to predict the total sleeping hours of mammals based on chosen variables), method we need to use is anova. If our variable were to be a categorical one (such as “yes” and “no”, or “low”, “medium”, and “high”, we could use “class” as our method). I named the function m1 because it is always a good idea in data science to keep track of what you are doing. Hence, I named my outcome m1 which refers to model 1. There is another parameter called control which is optional. You can specify your terminating condition using your control parameter. For example, if we were to run:

> rpart(sleep\_total ~., data=df, method=”anova”, minsplit=30)

This would tell R to observe at least 30 observations in a node before attempting to split it.

With our minsplit condition being 30 minimum observations within our node, the shape of our decision tree would change drastically since we would have a condition which alters the way algorithm splits the data. Number of nodes in model 2 is reduced to 7 from 15 nodes in model 1 due to our minimum splitting condition. Our next step is to visualize our decision tree using one of the packages I mentioned before, rpart.plot.

> rpart.plot(m1, type=2, digits=3, fallen.leaves=TRUE)

Our function is rpart.plot.

Parameters are:

1) our model’s name (m1),

2) type (can be between 0-4). Different type yields different looks for the decision tree.

3) digits=3 tells R to have number in 1 decimal format. digits=2 would yield a percentage without a decimal point.

3) fallen.leaves=TRUE which tells R to make all the leaves fall all the way down. When fallen.leaves is given FALSE, our leaves are not on the same level on the very bottom of the tree which is visually a bit more difficult to read for some people. Content wise, there is no difference between the two choices (it would look more like the basic decision tree we constructed on pg. 5). So, what does this tree tell us? How do read the visual?

Our root node, which holds 100% of the sample, and average sleeping hour for mammals in the entire sample is 10.4 hours. Then, our decision tree diverges to left (for condition being true, that is, is body weight equal to or greater than 168?)

One thing the data scientist should always to is to get an assessment of how the chosen algorithm is performing, as far as accuracy is concerned. Different packages and different methods often use different algorithms and it is important to understand how source code makes the data behave so that we can better analyze, predict, model, and visualize our data. It would be like knowing which port our ship has sailed to so we can expect to know when to make turns and prepare ourselves accordingly. There are several ways to test accuracy and one way to do that is to have a numerical target feature which could be several statistics computations such as Mean Absolute Error, Mean Squared Error, etc. Although there are several *loss functions* readily available, the user can choose to write his own function:

> MAE <- function (actual, predicted) { mean(abs(actual-predicted))} # define the function

> MAE # displays the source code of the function

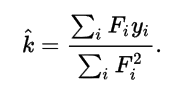
> MAE(df$sleep\_total, p1) # Mean Absolute Error between the actual values of my target value and predicted values.

Yet, it is always more useful to include different models and then choose the model with the lowest error rate (lowest MAE in our case), so that accuracy rate would be higher.

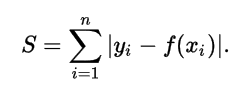
**2.3 Loss Functions**

Loss functions can be thought of as accuracy we are willing to give up for building our model. Different loss functions tend to be more appropriate based on the form of the data and what we are set out to do. For regression trees, the most widely used loss functions are trimmed mean, least absolute deviations, and least squares.

*Least squares:* A standard approach in regression analysis for approximation. “Least squares” means that the overall solution minimizes the sum of the squares of the residuals made in the results of every equation.

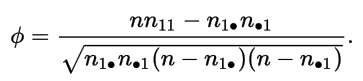


*Least Absolute Deviations:* Also known as Least Absolute Errors (LAE), and Least Absolute Value (LAV). LAD is another statistical optimality criterion and the statistical optimization technique that relies on it. Similar to other celebrity-like optimization techniques, LAD aims to find a function which closely approximate our data.



*Trimmed Mean:* Involves the calculation of the mean after discarding given parts of a probability distribution or sample at the high and low end, and typically discarding an equal amount of both. This number of points to be discarded is usually given as a percentage of the total number of points, but may also be given as a fixed number of points. Also known as Truncated Mean.

For classification trees, phi coefficient, Gini Index, or “twoing” are the most commonly used approximation techniques.

*Phi Coefficient:* Is a measure of association for two binary variables (n is total number of observations).

*Gini Index:* The Gini Index loss measures inequality or dispersion.

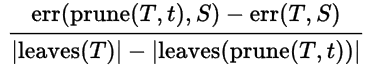
Another step data scientist should consider before finalizing his predictions on data, *pruning the tree* is usually common approach, which I am going to cover in the next section

**2.4 Tree Pruning**

Pruning is a technique we use to reduce the size of decision trees by getting rid of sections of our tree that provide little to no power for our classification. Pruning process helps us reduce the complexity of the finalized classifier, hence providing a better and improved accuracy on our predictions. Since a non-polished final form of our classification may be misleading, and a small, non-pruned tree may be missing important structural data it is important to implement tree pruning accordingly.

Ultimately, the pruning phase in CART decides on the right and most proper tree size to prevent overfitting and minimize the misclassification error Rt(P). In bottom-up fashion pruning in the tree growth phase, the tree is grown until the size of the family of each leaf node *n* falls below a user-specified threshold ‘h’. Some examples for a bottom-up pruning process would be *cost-complexity pruning*, pruning with an additional set of records called a test set (Breiman et al., 1984), and pruning based on the minimum description length (MDL) principle (Mehta, Rissanen, & Agrawal, 1995; Quinlan & Rivest, 1989).

In top-down fashioned pruning process, a statistic *sn* is computed at each node *n*, and based on the value of *sn,* tree growth at *n* is continued or stopped (Quinlan, 1993). Moreover, bottom-up pruning outcome usually in trees of higher quality. However, top-down pruning is computationally more efficient because no parts of the tree are first constructed and later discarded.

***Cost-complexity Pruning****:* generates a series of trees T0…Tm where T0 is our initial tree and Tm is our root tree. At step *i,* the tree is formed by removing a sub-tree from tree *i-*1 and replacing it with a leaf node with a value that is chosen as in our tree building algorithm. The sub-tree that is gotten rid of is chosen as follows. Define the error rate of tree *T* over the data set *S* as err(*T, S).* The sub-tree that tries to minimize  is chosen for removal. The function prune(*T, t)* defines the tree gotten by pruning the sub-trees *t* from the tree *T.* Once the series of trees has been created, the best fit is chosen by generalized accuracy as measured by a training set or cross-validation.

**REFERENCES**

Wei-Yin, Loh. *Bootstrap Calibration, Model Selection and Tree-Structured Methods*. Wisconsin univ-Madison, 1998.

Ye, Nong. *The Handbook of data mining*. Lawrence Erlbaum Associates, 2003.

Han, Jiawei, and Micheline Kamber. *Data mining: concepts and techniques*. Elsevier, 2011.

Groth, Robert. *Data mining: buiding competitive advantage*. Prentice Hall, 1999.

Quinlan, J.R.: *Programs for Machine Learning.* San Mateo: Morgan Kaufmann; 1993.

Breiman L, Friedman JH, Olshen RA, Stone CJ. *Classification and Regression Trees.* CRC Press; 1984.

Therneau TM, Atkinson B. *RPART: recursive portioning.* R port by B. Ripley. R package version 3.1-41, 2008.

Quinlan, J.R.: Induction of decision trees. *Machine Learning,* 1,81-106.

Quinlan, J. R. (1987). "Simplifying decision trees". International Journal of Man-Machine Studies.

“Gini coefficient.” *Wikipedia*, Wikimedia Foundation, 2017, en.wikipedia.org/wiki/Gini\_coefficient.

“Least squares.” *Wikipedia*, Wikimedia Foundation, 11 Dec. 2017, en.wikipedia.org/wiki/Least\_squares#Least\_squares,\_regression\_analysis\_and\_statistics.

“Decision tree.” *Wikipedia*, Wikimedia Foundation, 14 Dec. 2017, en.wikipedia.org/wiki/Decision\_tree.