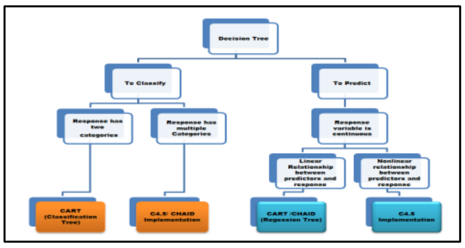
KU LEUVEN

FACULTY OF SCIENCE

Comparing R packages for deriving Classification Trees

(CART, CHAID, C4.5, C5.0)


Decision Tree
To Classify
To Predict
Response has two categories
Response has multiple Categories
Response variable is continuous
Linear Relationship between predictors and response
Nonlinear relationship between predictors and response
CART (Classification Tree)
C4.5/ CHAID Implementation
CART /CHAID (Regession Tree)
C4.5 Implementation

Assam DUKE ABASI TCHAPCHET

Supervisor:

Prof. Martina Vandebroek

Thesis presented in fulfillment of the requirements for the degree of Master of Science in Statistics

Academic year 2013-2014

Preface

This thesis is written and submitted for the award of a Master of Science degree in Statistics at the department of statistics of the KULeuven – University. This thesis seeks to show how some of the best classification tree algorithms like CART, C4.5, C5.0 and CHAID can be applied using the R- statistical software which is open source and would be interesting for users who would like to apply these algorithm but have no access to a licensed statistical software.

First of all, I would like to extend my sincere gratitude to my supervisor Professor Martina Vandebroek of the Research Centre for Operations Research and Business Statistics (ORSTAT, KULeuven) who has been relentless in her efforts to guide, advice and enable me come up with a well-structured paper. Her instructions, discussions and practical tips were quite helpful and I am thankful for that. I would also like to thank my daily assistant Chang Wang of the Research Centre for Operations Research and Business Statistics (ORSTAT - KULeuven)who was always created time from his very busy schedule to listen, assist and guide me in my ideas and approach to this paper.

It is with much gratitude that I thank the R-project team in general for making available its applications and repositories for free to anyone interest in using them. This has been of great help to this project.

I would like to say a special thank you to the staff of the department of statistics, KULeuven for sharing their knowledge and expertise to prepare me for the to-be sexiest job.

Next, I will like to thank my uncle Professor Ketcha Joseph and my brother Assistant Professor Pryseley Assam for their encouragement and support academically, professionally and morally. Notwithstanding, my special thanks go to Sara, Akan, Gussie, Christa, Fergus and my parents for their love and support which has always been an inspiration for me to strive for the best.

Finally, I would like to thank all my family and friends for their encouragement and support in one way or another which has helped me to be the best I can.

Summary

The main aim of classification trees is to predict membership of objects into classes of the response variable. When the response variable is continuous, a regression tree can be produced or categories can be derived from the continuous response variable in which case a classification tree will be produced. The classification tree(or regression tree), a technique used in datamining, is an attractive method for analysis since they are much flexible but it should be stressed that a blind eye should not be given to the traditional methods which exist like cluster analysis, discriminant analysis etc, since when these traditional methods meet their theoretical and distributional assumption, they may perform better. Otherwise, classification trees help to make the situation better.

In this paper, some of the most popular classification tree algorithms like CART, CHAID, C4.5 and its upgrade C5.0 will be examined and applied to some real life datasets downloaded from some repositories which include Car dataset, Churn dataset, nlsy dataset, boston housing dataset. Since these algorithms are very popular and mostly available for application in licensed statistical softwares, this paper also tries to investigate the possibility of applying these algorithm on the open-source(free downloadable) R – statistical software. At the beginning of this paper, not all the classification tree algorithms stated above could be applicable in the R-software. At this time of the end, there are available packages to implement all the above classification tree algorithms which is good news although one package have been changed because it did not represent the underlying theory.

This paper discusses for each classification tree algorithm, the type of response variable and predictor variables, the different types of splitting criteria, the pruning options available, the R-package for its application and an example using real life data.

After carrying out a series of analysis with the above classification tree algorithms in R, it was realized that depending on the data used, some classification tree algorithms perform better than others. Thus it is always advisable when performing classification tree analysis on data, depending on the type of response variable, to try different classification tree algorithms and types of split. From this point, a comparism of the different results is conducted and the best tree is chosen based on accuracy, justifiability and comprehensiveness. Accuracy gives an indication of how much the tree classifies the categories of the response variable correctly. Justifiability deals with

checking if the information we can derive from the tree is a sensible one. This is an important aspect since when overfitting of data occurs, we can end up with insensible trees. The issue of comprehensiveness is relative to the user of the results. It deals with the interpretation of the tree. This implies a very large tree may be okay to a statistician for example due to his expertise and is not comprehensive to non-statistical client. This is the reason why pruning is important.

Nonetheless, pruning tree results to some loss in information leading to reduction in accuracy. Care should be taken when dealing with this for there is a trade-off between accuracy and comprehensiveness.

Definition of key words

Classification: the goal of classification is to accurately predict the target class for each instance in our data or to say, predict and explain responses on a categorical dependent variable.

Classification tree: classification trees are used to divide a dataset into classes(2 or more) of the response variable which implies when the response variable is categorical in nature, classification trees are used

Regression tree: regression trees are used basically for prediction problems whereby the response variable is numeric or continuous in nature.

The generalized gini impurity function: This is a splitting criteria used when ordinal response variables are used for classification. This is similar to the gini criteria used by CART except that scores of the categories of the response are used in absolute differences or quadratic differences.

Pruning: This is a method of cutting of some branches of a classification tree or reducing the size of a classification tree. This process can be carried out using different criteria depending on aech algorithm.

CART: this stands for Classification and regression trees.

Rpart: this stands for Recursive partitioning and regression trees. This is the name of the package that implements classification and regression trees in R statistical software.

Chaid: This stands for Chi-Square automatic interaction detection and is a classification tree algorithm as will be discussed in chapter 2. Its R-package is also called the CHAID package.

C4.5/C5.0: C4.5 generates a classifier in the form of a decision tree and is an extension of Ross Quinlan’s ID3 algorithm. C4.5 has now been superseded by C5.0. This algorithm can produce decision trees or a set of rules.

V-fold cross-validation: this is a method which creates v- equal sub-samples such that for each fold, it uses V-1 sub-samples to train the model and one sub-sample to test. This procedure is repeated until all sub-samples have been used once as a test sub-sample and the cross-validation cost for all the test sub-sample is averaged to give the v-fold estimate for the cross-validation cost.

Minimal cost complexity pruning: The pruning technique used by CART is called minimal cost complexity pruning and it assumes that the bias in the re-substitution error of a tree increases linearly with the number of leaf nodes.

Table of Contents

Chapter 1

1.0) Background

In the statistics community, Belson(1959) wrote probably the first paper on tree-structured methodology though construction of trees began in early 60s in the social sciences. Morgan and Sonquist(1963) and HUNT et al.(1966) developed two algorithm: AID (Automated Interaction Detection) and CLS (Concept Learning System) respectively in order to model the process of concept learning. In the early 70s , statisticians gained more interest when Meisel and Michalopoulos (1973) pointed out the use of tree based methodology in Classification which resulted in several papers and the book by Breiman et al. (1982) called CART. Due to their popularity, simplicity, flexibility and strength, tree based models have become widely used in classification, regression and many other areas . In this paper tree based models for classification and regression will be discussed.

In this section, where literature on classification trees and its algorithms are discussed, a broad view of classification will be considered by looking at some definitions, some of the most popular algorithms applied and their functionality, the reasons why they are so popular and finally by taking into consideration how this technique and these algorithms can be implemented by the application of interest (with R-Software). The main purpose of the paper will be to explore the possible application of these classification tree algorithms based on freely available libraries specifically in R-statistical software (Downloadable from the Comprehensive R Archive Network: CRAN)[1] .

1.1) What is classification?

Classification can be defined as the process of arranging data into homogenous groups or classes according to some common characteristics present in the data based on their similarities and dissimilarities, thus we say a classification tree is a prediction model with the aim of allowing us to predict the unknown value of the variable of interest given known values of other variables. The goal of classification is to accurately predict the target class for each instance in our data or to say, predict and explain responses on a categorical dependent variable. Since the result of a classification process is always

[1] http://cran.r-project.org/

followed by decision-making, it can be referred to as a decision tree[2]. A decision tree can be a classification tree or a regression tree and in this paper, a decision tree will be referred to either or both of them each time it is used.

In classification, we want to learn a mapping from a vector of observed or measured variables x_i taking values {x_1, x_2,…….., x_p} and referred to as features, attributes, explanatory variables etc, to a categorical variable *Y* which is referred to as a class variable taking values {c_1, c_2,…….. , c_m }. Also the value of x which is comprised of p-variables can contain components which can be real-valued, ordinal, categorical etc. “Whenever these interrelationships become very complex containing non linearity and interaction the usefulness of classical approaches is limited." (Press, Rogers, and Shure, 1969, p 364.)

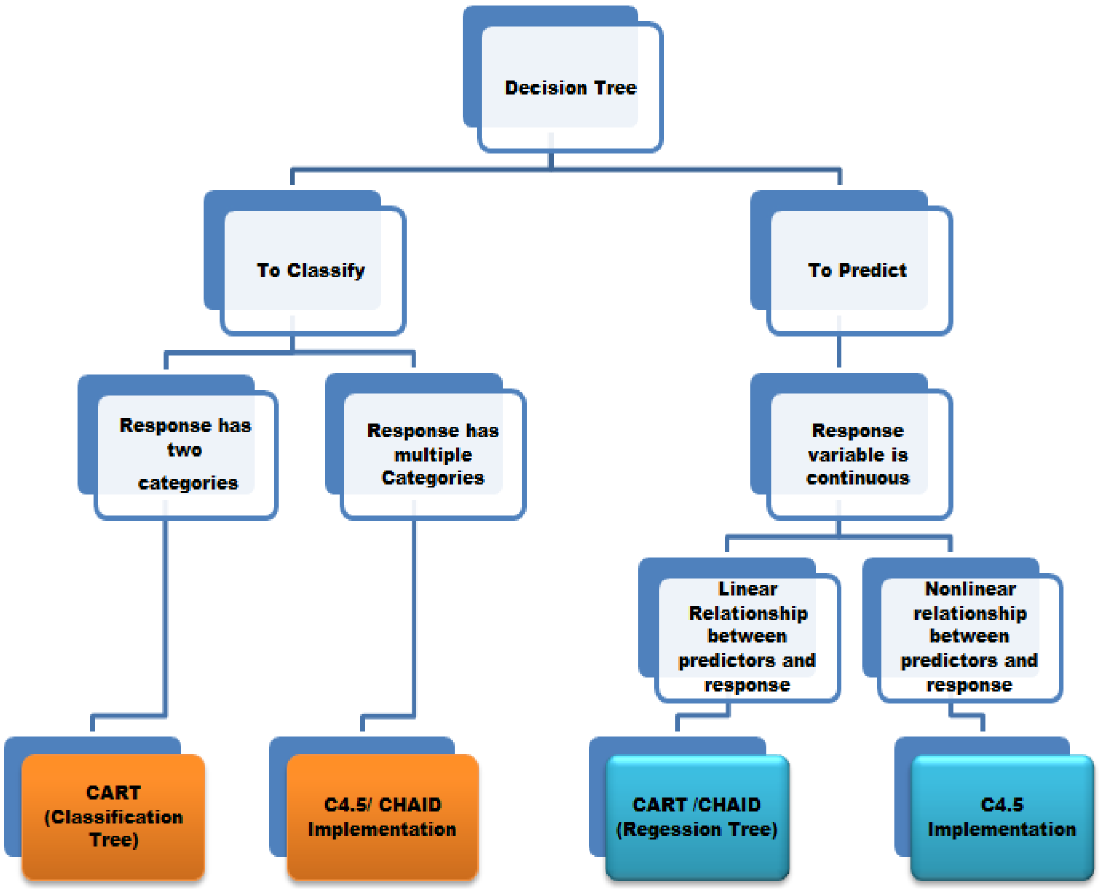
1.2) <u>Difference Between Classification tree and Regression Tree.</u>

As the name implies, classification trees are used to divide a dataset into classes(2 or more) of the response variable which implies when the response variable is categorical in nature, classification trees are used. On the other-hand, regression trees are used basically for prediction problems whereby the response variable is numeric or continuous in nature. It should be noted that in both cases, the predictor variables can be categorical and/or continuous.

Another difference can be seen in the functionality of both. The idea in classification tree is to split a dataset based on homogeneity of data with respect to the response variable and measures of impurity are applied such a Gini-index or entropy based on computing the proportion of the data that belongs to a particular class to quantify the homogeneity. On the other-hand in a regression tree, since the response variable does not have classes, a regression model is fitted to the response variable using each of the predictor variables and for each independent variable, the data is split at several points where at each split point, the error between the predicted and actual values is squared (SSE) and these split points across the variables are compared recursively and the variable point which yields the lowest squared error(SSE) is chosen as the root node or split point. Since regression trees use regression models, they lose one strength of a standard decision tree which is the ability to handle highly non-linear parameters. Nevertheless, in such cases, it may be better to use the C4.5 type implementation as will be seen later to be used for classification trees with more than 2 categories.

[2] Warning: decision trees can also refer to probability trees.

These differences brings us to the notion of decision tree which by its name implies using a tree (either classification tree or regression tree) for decision making. Thus in this paper as already indicated, a decision tree could either refer to a classification tree or a regression tree.


Decision Tree
To Classify
To Predict
Response has two categories
Response has multiple Categories
Response variable is continuous
Linear Relationship between predictors and response
Nonlinear relationship between predictors and response
CART (Classification Tree)
C4.5/ CHAID Implementation
CART /CHAID (Regession Tree)
C4.5 Implementation

3

[3] http://www.simafore.com/blog/bid/62482/2-main-differences-between-classification-and-regression-trees

1.3) Research objective

This paper will consist of an analysis of 3 popular classification algorithms through which the following research questions will be investigated:

- Which are the best classification tree algorithms?
- Which R packages are most suitable for deriving these classification tree algorithms?
- Does an ultimate package exist among the chosen classification tree algorithms?

1.4) Classification Tree Methodology

A classification tree can be basically characterized as involving the following four steps:

1)**Criteria for Predictive Accuracy**

When constructing a classification tree, we want our predictions to be as precise as possible. But talking about precision in this case is hard to define and as such precision could be defined as minimum cost or minimum cost of misclassification or minimum proportion of misclassification. This is important because the misclassification cost for one class may cost more than for another class. Thus we try to minimize cost rather than rates.

2)**Selecting Splits**

Another important step in classification tree analysis is to select splits on the predictor variables such that they are used to predict class membership of the dependent variables for the instances or cases in the analysis. With the hierarchical nature of classification trees, splits are selected one at a time from the root node to the leaf node(where splitting stops). In this light, three split selection methods will be discussed.

a) Discriminant-based univariate splits[4]:

The first step in this method is to which predictor variable should be used to perform the first split. P-values are computed at each terminal node to test the significance of the relationship between class membership with different levels of each predictor variable. When categorical predictors are used, p-values for the Chi-Square test of independence of the classes and levels of the categorical predictor present at the node are computed. When ordered predictors are used, p-values for the ANOVA's of the relationship of classes to values of the ordered predictor present at the node are computed such that if the p-value is smaller than the default bonferroni-adjusted p-value for multiple comparisons(i.e 0.05 or any threshold chosen by the user), the predictor variable with the smallest p-value is chosen to split the corresponding node. If all p-values are smaller than the threshold, p-values for statistical tests which are robust to distributional violations are computed e.g Levene's F (Loh and Shih 1997).

After choosing the best root node, the split has to be chosen. The two-means clustering algorithm(Hartigan and Wong 1979) is applied in the case of ordered predictors to create two superclasses for the node such that two roots are found for a quadratic equation which describes the difference in means on the superclasses of the ordered predictor and the values for corresponding splits for each root are computed with the split closest to a superclass mean selected. In the case of categorical predictors, dummy-coded variables are constructed to represent their levels(categories) and methods for singular value decomposition are used to transform the dummies into a set of non-redundant ordered predictors, then the above procedure for ordered predictors is applied and the split obtained is then mapped back onto the original levels of categorical variables(Loh and Shih 1997). Although this procedure is complicated it reduces the bias in split selection which occurs when the the cart-style exhaustive search method discussed in (c) below is used.. The bias is due to the fact that variables with more levels are selected over those with less levels for splits and this can skew the interpretation of the relative importance of the predictors in explaining responses on the dependent variable(Breiman et. al., 1984)

b) Discriminant-Based Linear Combination Splits[5]: This method is an option for ordered predictor variables where the predictors are assumed to be measured on interval scales. Here, methods for singular value decomposition are applied to transform the continuous predictors into a new set of non-redundant predictors. Next, the same procedure as above for creating superclasses, finding closest split to the

[4] StatSoft, Inc. (2013). Electronic Statistics Textbook. Tulsa, OK

[5] StatSoft, Inc. (2013). Electronic Statistics Textbook. Tulsa, OK

superclass mean are applied, results mapped back to original continuous predictors and represented as a univariate split on a linear combination of predictor variables.

c) CART-style exhaustive search for univariate splits[6]: This method is an option for categorical or ordered predictor variables whereby all possible splits for each predictor variable are examined at each node to find the split which produces the largest improvement in goodness of fit or largest reduction in lack of fit. In order to determine possible splits at each node, the following criteria are used:

-For Categorical predictor variables, with k levels at a node, we have $2^{(k-1)} – 1$ possible contrasts between two sets of levels of the predictor.

-For ordered predictor variables, with k levels at a node, we have k-1 midpoints between distinct levels.

We can clearly see that when we have many predictor variables with many levels to be examined at a node, the number of splits to be examined can be very large. Thus to improve the fit, the goodness of fit measures(e.g the gini measure used by CART) are employed depending on which split method is used.

3) when to stop splitting

An important issue in growing a tree is to decide when to stop splitting, else we end up with pure terminal nodes. In order to decide when to stop splitting a stopping rule has to be specified. The stopping rule could be one of the following:

Minimum n: whereby desired minimum number of cases can be specified and splitting stops when all terminal nodes containing more than one class have no more than specified minimum number of cases.

Fraction by objects: whereby splitting continues until all terminal nodes are pure or contain no more than the specified minimum fraction of the class sizes for one or more classes.

4) Selecting the right size of the tree

When talking about selecting the tree size, it important to state that there is no universal optimum tree size but we can make some generalizations about what constitutes the right sized classification tree. A classification tree should be sufficiently complex to account for the known facts but it should be as simple as possible since

[6] StatSoft, Inc. (2013). Electronic Statistics Textbook. Tulsa, OK

comprehensibility is a key word in this context. In this respect, a right sized tree should incorporate the following:

It should make use of information that increases predictive accuracy and leave out information that does not. It should lead to a better understanding of the phenomena that it describes. A good strategy for growing a right sized tree depends on the user such that through knowledge about previous research, previous analysis, diagnostic information or even intuition, the user could determine when the tree size is comprehensive, justifiable and accurate through pruning. In this regard, some criteria which can be used to select the right size tree are as follows:

- Test sample cross validation: In this case, usually two third of the available dataset is used to train the classification tree model to predict class membership then in order to verify its predictive accuracy, the rest of the dataset(one-third) is used to test the classification tree model. Both models are tried using cross validation and their misclassification costs are compared.

- V-fold cross validation: In this case, the whole dataset is used(usually when the sample is too small to create a test set. Thus, the user specifies a value for V to conduct a V-fold cross-validation which creates v- equal sub-samples such that for each fold, it uses V-1 sub-samples to train the model and one sub-sample to test. This procedure is repeated until all sub-samples have been used once as a test sub-sample and the cross-validation cost for all the test sub-sample is averaged to give the v-fold estimate for the cross-validation cost.

- Minimal Cost-complexity pruning: This is a pruning method used by CART. Here the misclassification cost of each subtree is estimated and the one with the lowest cost is chosen. Although this is more complex, since multiple trees must be built and pruned, it tends to produce smaller trees which is needed for comprehensibility although an eye should be kept on accuracy which should be comparable to the best tree. CART uses the “1 SE rule” which chooses the smallest tree whose estimated mean error rate is within 1 SE of the mean error rate of the best tree. This method will be further discussed in the CART pruning criteria.

CHAPTER 2

2.) Methodology of Classification tree Algorithms

Some of the most popular classification algorithms discussed in this paper include: C4.5, CHAID and CART. They have different tree growing methods with the same goal of maximizing the total purity through identifying sub-segments dominated by a specific outcome. Now we will take a close look into each of these techniques to discover what is the driving force behind them and how they can be implemented in R software environment.

2.1 Chi-Square Automatic Interaction Detection(CHAID)

CHAID is a classification tree algorithm that detects interactions between categorized variables (response variable and classifiers which may or may not be ordered) of a data sample. This technique was developed by Gordon V. Kass 1980 who had completed a PhD thesis on this topic[7]. This method is widely used in survey analysis, Customer profiling, customer targeting etc. An overview of the algorithm is discussed below.

2.1.1 Type of response

Chaid algorithm is used to classify a categorical response variable with many categories based on categorical predictors with many classes. The response variable can also be continuous in which case a transformation is needed into an ordinal variable. Each non-terminal node identifies a split condition to yield optimum prediction of transformed continuous response variable or classification of categorical response variables. This implies chaid can be used to analyze regression type or classification type problems.

2.1.2 Type of classifier variables:

First of all, since this algorithm handles categorical data, it is important to categorize each continuous predictor variable into categories such that each category has approximately the same number of observations . The predictor variables which are categorical may or may not be ordered.

[7] http://en.wikipedia.org/wiki/CHAID

2.1.3 Type of split and variable selection :

The name CHAID which came as a development for an early tree methods stands for **CH**i-squared **A**utomatic **I**nteraction **D**etector and by its name specifically, it uses the Chi-Square splitting criterion and it uses the p-value of the chi-square test when categorical dependent variables are used and with an option for handling quantitative dependent variables, it uses the p-value of the F-statistic for the difference in mean values between the g nodes generated by the non-binary splits such that :

$$F= \frac{BSS/(g-1)}{WSS/(n-g)} \sim F_{(g-1),(n-g)}$$

where WSS= $\sum_{j=1}^{g}\sum_{i=1}^{n_j}(y_{ij} - \bar{y}_j)^2$, g=no of groups, $\bar{y}_j$ =mean values of y_{ij}s in node j.

TSS= $\sum_{j=1}^{g}\sum_{i=1}^{n_j}(y_{ij} - \bar{y})^2$, TSS= total sum of squares before the split.

BSS = TSS – WSS

The second step in the chaid algorithm is to select for each predictor, the pair of least significantly different (predictor)categories with respect to the dependent variable. This procedure is repeated for all predictor variables to determine which predictor variable has the pair of categories with the least significant difference with respect to the response variable. In this step, if the dependent variable is categorical, the pearson chi-square test is used and if the dependent variable is continuous, the F-test is used. If after conducting this test, no pair of predictor categories is significant, the respective pairs are merged and the process repeated . This is done recursively including the merged pairs and if statistical significance is realised in relation to(less than) a specified alpha-to-merge value, it computes a bonferroni adjusted p-value for the pair of categories of the predictor variable in question.

The third step in this algorithm is to select which variable to split and this is done by selecting the variable with the smallest bonferroni adjusted p-value since the variable with the smallest adjusted p-value is the variable with the most significant split. This splitting procedure is continued recursively until the point where the smallest adjusted p-value is larger than some specified alpha-to-split value. At this point, no further splits can be executed and the node becomes an end-node or terminal node. The aim of the Bonferroni adjustment is to account for multiple testing.

A more thorough merging and testing of the predictor variables is performed by the Exhaustive CHAID algorithm which merges categories until only two are left for each predictor and proceeds to select among the predictors, the one with the most significant split. The only problem here is that it requires more computing time which could be a constraint for large datasets.

2.1.4 Type of pruning in CHAID

The following methods can be used to prune a CHAID tree:

a) The chaid tree can be pruned by adjusting the control parameter(minsplit) for minimum number of observations that must exist in a node in order for a split to be attempted.
b) The chaid tree can also be pruned adjusting the control parameter (minprob) for minimum probability i.e p-value for statistical significance.

2.1.5 CHAID Package in R

The version 0.1-1 CHAID package is available from R version R2.15.1 by the FoRt Student Project Team and maintained by Torsten Hothorn since 2009-02-04 and packaged in 2012-07-02. Thanks to this team, the CHAID package in R offers an application of CHAID algorithm and it detects interactions between categorical variables(ordinal or nominal) of a dataset whereby one of the categorical variables is the dependent variable and the other variables may or not be ordered. The Package implements recursive partitioning by maximizing the significance of a chi-squared statistic for cross-tabulation between the dependent variable and the predictors at each partition such that the data is partitioned into mutually exclusive, exhaustive subsets that best describe the dependent variable and splits are multiway by default . This package is available for download at the R-Forge[8] repository. This can be done using the following command line in R:

```
install.packages("CHAID", repos="http://R-Forge.R-project.org")
```

The CHAID algorithm is applied in SAS using the **Treedisc** macro which is a modification. The R-package CHAID offers the application of CHAID in the R-software environment and it is able to detect interactions between categorized variables of a dataset, one of which is the dependent variable and the independent variables could be

[8] **R-Forge** offers a central platform for the development of R packages, R-related software and further projects.

of different forms. In SAS this method can be implemented using the Enterprise Miner tool which is licensed and used mostly by big firms.

2.2 C4.5/C5.0 Algorithms

C4.5 generates a classifier in the form of a decision tree and is an extension of Ross Quinlan's ID3 algorithm. ID3 searches through the attributes of the training instances and extracts the attribute that best separates the given examples. The algorithm uses a greedy search, that is, it picks the best attribute and never looks back to reconsider earlier choices. Note that ID3 may misclassify data. C4.5 which grew to about 9000 lines of C was a successor of ID3 and has now been superseded by C5.0. This algorithm can produce decision trees or a set of rules.

2.2.1 Type of response

The response variable must be categorical(nominal or ordinal).

2.2.2 Type of variables

This algorithm handles non-numeric data like character, factor and ordered data. It is a preferred method for discrete attributes.

2.2.3 Type of split and variable selection

C4.5 uses the information gain criteria for its splitting rule while C5.0 uses the information gain ratio for its splitting rule.

Let S be the sample size to be used,

C_i refers to class I; i =1,2,…..,m

$I(s_1,s_2,\dots..,s_m) = -1*[\sum p_i \log_2(p_i)]$ = information of tree.

S_i= number of samples in class i.

$\log_2$= binary logarithm

P_i= S_i/S

Also, let attribute A have v distinct values such that:

Entropy= $E(A) =\sum[(s_{1j} + s_{2j} + \ldots + s_{mj})/s]*(s_{1j}, .. s_{mj})$; j=1

Where s_{ij} = samples in class *i* and subset *j* of Attribute of attribute A, we have;

$I(s_{1j},s_{2j},\ldots..,s_{mj}) = -1*[\sum p_{ij} \log_2(p_{ij})]$ = information of subtree.

$GAIN(A) = I(s_1,s_2,\ldots..,s_m) - E(A)$

⇨ **Gain(A)** = Information – entropy

Intrinsic Information(A) = $P(A)= \sum_i(S_i/S)\log(S_i/S)$

⇨ **Gain ratio(A)=**Gain(A)/Intrinsic information(A)

From here, information gained by partitioning the data, based on a selected predictor X is measured as follows:

Information Gain(due to split of X) = [Information at Parent node - information (after splitting on x)]

From the above equation, the C5.0 will choose for the split, the predictor with the highest information gain. The C5.0 which is a commercial upgrade of the C4.5 uses a normalized format for the information gain known as the 'information gain ratio' such that it fixes a bias in the previous C4.5 version of the algorithm towards bushy and large trees. Due to this, the algorithm incorporates a pruning procedure for producing smaller trees of equivalent predictive performance. The variable selected for the first split is said to have an importance measurement of 100%.

2.2.4 Pruning options

C5.0 is very famous for its production of small and accurate trees with fast and reliable classifiers making decision trees an important tool for classification. An implementation of C4.5 in R can be done using the Rweka package while the C5.0 is applied in R using the package C50. It is important to state that C5.0 constructs its trees in two phases:

- It grows a large tree to fit the data closely.
- It prunes the large tree by removing the parts of the tree that are predicted to have high error rates. This is applied to every subtree to such that a decision can made whether to subtree can can be replaced by a leaf or by a sub-branch and then it looks globally at the performance of the whole tree.

The C50 package offers three other options for pruning which are regarded as advanced options:

- The option -*g* disables the second pruning aspect and ends up with generally larger trees and rulesets. This option is beneficial for some applications when rulesets are generated.
- The option –*CF*(control factor) affects the estimation of error rates and pruning severity. It has a default value of 0.25 and smaller values results in more pruning while larger values result in less pruning.
- The option –*m* cases with default values equal to two(2) which means that at each branch point in the tree, the minimum number of training cases must follow at least two of the branches. This constrains the degree to which the initial tree can fit the data and is a form of pre-pruning.

2.2.5 RWEKA Package for C4.5 and C50 package for C5.0

The version 0.4-18 RWeka package is available from R version R 2.15.3; by Kurt Hornik [aut, cre], Christian Buchta [ctb], Torsten Hothorn [ctb], Alexandros Karatzoglou [ctb], David Meyer [ctb], Achim Zeileis [ctb] and maintained by Kurt Hornik. It was last updated in 2013-08-11. This package is available for download in the CRAN repository. Thanks to this team, the C4.5 method can be applied in the R software with the RWeka package which interfaces the R software to the machine learning open source tool Weka (Version 3.7.9). J48(C4.5 java version) generates unpruned or pruned C4.5 decision trees (Quinlan, 1993). Weka is a collection of machine learning algorithms for data mining tasks written in Java, containing tools for data pre-processing, classification regression, clustering, association rules, and visualization. Package RWeka contains interface code, the Weka jar is in a separate package RWekajars[9]. For more information on Weka, see http://www.cs.waikato.ac.nz/~ml/weka/.

The version 0.1.0-15 C50 package is available from R version R 2.15.3; by Max Kuhn, Steve Weston, Nathan Coulter using the C code for C5.0 by R. Quinlan and maintained by Max Kuhn. It was last updated in 2013-05-27. This package is available for download in the CRAN repository. Thanks to this team, the C5.0 algorithm can be applied in the R software with the C50 package. The C5.0 algorithm produces as results decision trees and rule-based models.

In order to implement the j48 algorithm in weka using R –programming interface, the first step is to split the data in two parts(65%:35% the proportion of split depends on the user and can be done for all the other classification tree algorithms) such that one part(larger) is used to train the algorithm and the second part used to the test how effective the classifier works. Another option which could be used in this case is the cross-validation (preferably 10 fold) such that it is easy to capture enough information from the data. These options are available in the C50 control in R .This is very important since when we build a classification tree, we want to know how well the tree can predict new data. This step gives us a good simulation of the scenario in which we will have to predict future data. This helps to give us an idea if the machine learning algorithm over-fits the training data and also learns the noise in the data which would lead to higher training error in the classifier.

2.3) CART

CART stands for Classification And Regression Trees .The methodology of C4.5 is also used in CART but the main differences are in the tree structure, splitting criteria, pruning method and the ways missing values are handled. This algorithm produces binary splits and they incorporate the Gini impurity measure for their splits. (Breiman, et al., 1984). The CART algorithm chooses itself significant variables and leaves out insignificant variables when building classification trees.

[9] http://cran.r-project.org/web/packages/RWeka/RWeka.pdf

2.3.1 Response variable type

The response variable for CART could be one of the following:

-Categorical(nominal)response variable for classification using rpart,

-Continuous response variable for regression tree.

-Ordinal response variable for classification tree using rpartScore.

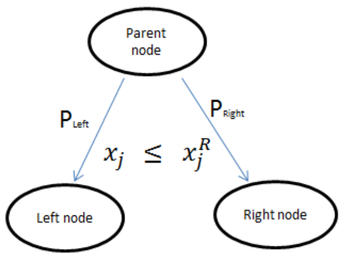
2.3.2 Classification Variable type

Predictor variables can be continuous and categorical.

2.3.3 Splitting rules

This step is the most time consuming as it splits the learning sample until terminal nodes contain observations of only one class. It should be noted that the splitting rule used for classification is different from that of regression.

2.3.3.1 Splitting rule for classification tree



For classification, CART uses the GINI splitting rules. The functionality of these rules can be described as follows:

a) GINI splitting rule(rpart)

The Gini-coefficient is a measure for dispersion that depends on the distribution of the outcome categories with range between 0 & 1 where at maximum value(1), the outcome categories have a balanced distribution (worse split) and has a minimum value(0) when all records of a node are concentrated in a single category(best split= pure node). The Gini measure can be calculated as follows:

$$\textbf{Gini} = 1 - \sum_i P(t_i)^2$$

Where $p(t_i)$ is the proportion of cases in node t that are in output category i. At each branch, all predictors are evaluated and the predictor that results in the maximum impurity reduction or greatest purity improvement is selected for partitioning.

$$\text{Decrease in impurity} = i(t) - P_L i(t_L) - P_R i(t_R)$$

Where t_L and t_R are data splits into left and right node, P_L and P_R are the proportions of split in t_L and t_R.

b) Generalized gini splitting criteria(rpartScore)

This is the splitting rule used by the rpartScore package for building classification trees with ordinal response variable. The generalized gini impurity function(Breiman et al.) for a node t can be given as:

$$I_{GG}(t) = \sum_{k=1}^{J} \sum_{l=1}^{J} C(\omega_k | \omega_l) p(\omega_k|t) p(\omega_l|t)$$

Where

$p(\omega_k|t)$ =proportion of units in node t which belongs to the k-th category of Y, for k=1,..,J.

$C(\omega_k| \omega_l)$ = misclassification cost assigning category ω_k to a sample unit belonging to category ω_l

i) Absolute differences between pairs of scores.

The generalized gini criteria with absolutes differences between pairs of score is derived by replacing in the formula above, $C(\omega_k| \omega_l) = | s_k - s_l|$

$$I_{GG1}(t) = \sum_{k=1}^{J} \sum_{l=1}^{J} | s_k - s_l| p(\omega_k|t) p(\omega_l|t)$$

ii) Squared differences in scores

The generalized gini criteria with absolutes differences between pairs of score is derived by replacing in its formula above $C(\omega_k | \omega_l) = (s_k - s_l)^2$. We get:

$$I_{GG2}(t) = 2\sum_{k=1}^{J} \sum_{l=1}^{J} (s_k - s_l)^2 p(\omega_k|t)p(\omega_l|t))$$

2.3.3.2 Splitting rule for Regression tree

In Regression trees, the response variables is not categorical but continuous in nature(no pre-assigned classes) which implies we cannot implement the GINI criteria. Instead in this case, the squared residuals minimization algorithm is used.

$$\underset{x_j \leq x_j^R, j=1,\ldots,M}{\operatorname{argmin}} [P_L Var(Y_L) + P_R Var(Y_R)]$$

Var(Y_l) , Var(Y_r) are response vectors for left and right child nodes, $x_j \leq x_j^R$, j=1,…,M is the optimal splitting question which satisfies the above function

This implies the expected sum variance for two resulting nodes should be minimized. The twoing splitting rule and the Gini splitting rules are quite similar as seen from the above equation. Assuming objects of class k are given a value of 1 and objects from all other classes are given a value of 0, this would imply the sample variance of these values(0 and 1) would be given as

$$P(k|t) [1 - P(k|t)]$$

Thus for K classes, we would have:

$$i(t)= 1 - \sum_{k=1}^{k} p^2 (k|t)$$

Upto this point, the algorithm produces the largest tree size which sometimes could be very large. The next step will describe to us how to reduce the resulting tree size.

2.3.4 Type of pruning

Maximum trees produced in the previous step could not only be complex but contain a lot of levels which could make it very difficult to interpret, thus it will be a good idea to optimize or reduce such tree sizes such that insignificant branches or leaves can be pruned. This can be done in CART using different methods like:

Specifying the minimum number of points(observations in a node) which in practice is usually set to 10% of the training set size. Here there is a trade-off between impurity and complexity of the tree.

Cross-Validation: This procedure is based on the optimal proportion between the complexity of the tree and the misclassification error. As tree size increases, misclassification error decreases and at maximum tree size, misclassification error of the training set is zero. The problem here is the fact that when a tree is complex, it handles new or independent data poorly. Thus in order to get an optimal proportion between misclassification error and complexity of the tree, a cost-complexity function is used in this method.

Minimal cost complexity pruning: The pruning technique used by CART is called minimal cost complexity pruning and it assumes that the bias in the re-substitution error of a tree increases linearly with the number of leaf nodes.

$R\alpha(T) = R(T) + \alpha(\tilde{T})$

Where

$R\alpha(T)$= cost assigned to a sub-tree,

$R(T)$=re-substitution error,

α=complexity parameter per terminal node and $\alpha \geq 0$

$(\tilde{T})$ is the number of leaves(terminal nodes)

We can see from here that for a small value for the complexity parameter(α), we get a smaller penalty for getting a large number of terminal nodes and also if the value of the complexity parameter(α) is quite large, only the root node will be left as subtree since the full tree will have be pruned completely. This implies, the number of subtrees that can be produced from a full tree(original tree) is finite for a continuum of values of the complexity parameter.

Actually, what we would want is the best value for the complexity parameter(cp=α) which from the cp-plot produced in R, gives us the corresponding size of the subtree to be produced i.e for every value of the complexity parameter (α), we find the subtree $T(\alpha) \leq T\max$ that minimises $R\alpha(T)$ (Breiman et al. 1984).

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

CART uses 10-fold cross-validation to improve error estimates and utilize more data, by generating a sequence of sub-trees through growing a large tree and pruning it recursively and then through cross-validation, it estimates the misclassification cost of each subtree and chooses the one with the lowest cost. Although this is more complex, since multiple trees must be built and pruned, this time complexity of the pruning with 10-fold cross validation is 10 times more expensive than C4.5's pruning, but it tends to produce smaller trees which is needed for comprehensibility although an eye should be kept on accuracy which should be comparable to the best tree. CART uses the "1 SE rule" which chooses the smallest tree whose estimated mean error rate is within 1 SE of the mean error rate of the best tree. Also CART does not penalize its splitting criterion when building a tree unlike C4.5 as it uses only the instances with known values when missing values exist.

The final step in the tree construction using CART is to use the final tree built for classification of new data. Following the questions in the tree, a new observation is assigned a class or response value of terminal node to which it belongs;

2.3.5) CART : Classification and Regression Trees packages in R

The version 3.1-52 of the **Rpart package** by Terry M Therneau and Beth Atkinson is an open source version of **CART** implemented in R programming environment(version R 2.15.0) and available for download in the CRAN repository. The rpart package has been updated since 2012-03-04 and can be used when we want to predict a continuous(regression tree) or classify a binary(classification tree) response variable.

RpartScore package

The version 1.0-1 of the **rpartScore package** by Giuliano Galimberti, Gabriele Soffritti, Matteo Di Masois is an open source version of **CART** (when an ordinal response variable is used) implemented in R programming environment(version R 2.15.0) and available for download in the CRAN repository since 2012-12-08. This is a new package which replaces the RpartOrdinal package used for building classification trees with

ordinal response in R using the CART framework. In this case, it is assumed that a set of numerical scores(not necessarily linear) has been assigned to the ordered categories of the response. This package uses the generalized gini impurity function to build trees such that the misclassification costs are given by the absolute or squared differences in scores assigned to the categories of the response variable(which is ordinal). Since the rpartOrdinal package has been removed from the R programming environment due to malfunctioning, its splitting criteria will also be excluded from this analysis. This is as a result of unexpected results when some real data was used with the rpart-ordinal(Galimberti and So_ritti 2012). These unexpected results were due to various issues related to the theoritical properties of the generalized gini impurity function and the way in which it was implemented in the rpartOrdinal package. This package was created to resolve this problem which will not be elaborated in this paper. In this case, the tree is built using the generalised gini-impurity criterion such that the misclasification cost is given by the absolute or squared differences in the scores assigned to the response categories. In this case, pruning of the tree is in relation to the misclassification cost or rate. This package requires and works as the rpart package as they are similar with the only difference being the presence of the "split" argument which specifies whether to use the absolute differences or the squared difference and the "prune" argument which specifies whether to use the total misclassification cost or total misclassification rate.
These are the splitting methods which should be considered when constructing a classification tree with an ordinal response. This method results in four(4) types of trees depending on the combination of type of split and type of pruning. To compare which of the four trees is the best calls for more research on this topic as discussed by the founders of this package although some measures were proposed like the somer's d to evaluate the ordinal association between the observed and predicted scores(Agresti 2002)

In summary,

Table 1: summary of packages

			Dependent Variable		Splitting Criterion		
R – PACKAGE	Algorithm	Local Split	Quantitative	Categorical	Association	Purity	p-value
CHAID	CHAID	n-ary	**X**	**X**	**X**		**X**
RPART	CART	binary	**X**	**X**		**X**	
RWEKA	C4.5	n-ary	**X**	**X**		**X**	
C50	C5.0	n-ary	**X**	**X**		**X**	

Chapter 3 : DATA ANALYSIS

In this chapter, an analysis will be carried out using the datasets which will be introduced for each algorithm, The analysis will be conducted for all three classification tree algorithms discussed (CART, C4.5 and CHAID) and these will be applied using the R programming environment.

3.1.0) CART Analysis using RPART Package

The CART algorithm will be implemented using the RPART package of the R programming environment using the churn dataset which was downloaded from the UCI datesets(Source: http://www.sgi.com/tech/mlc/db/churn.all). The dataset has 5000 entries with 20 variables where one of the variables (Churn) is the dependent variable. Churning involves customers choosing to go for another company's product over the one they already use. Thus this dataset can be used to understand when customers of a telecom company may decide to churn. The dataset was checked for missing values and it was found that it is complete with no missing values although this does not imply that the dataset is error free.

It is important to note here some simple steps to note when using the RPART package to grow a tree using R. They include:

- Install the package RPART using the R-console.

- Attach the RPART package so its functions can be used: require(rpart)

Fit the model using R-programming language :

```
fit<-rpart(Churn ~.,data=trainUCI,              #specify training data
                method='class',                 #specify categorical data
                parms =list(split="gini"),      #specify  use gini splitting criteria
                cp=0,                          #specify no penalty on the size of tree
                minsplit=20)                 #specify minimum nodes of size 20(default)
```

It should be noted the method depends on the type of response

(continuous='anova', poisson count= 'poisson', survival='exp', categorical='class')

- print the tree in text version: `print(fit)`

- plot the tree: `plot(fit)`

- Summary which examines each node in depth: summary(fit1)

Since growing a classification tree needs discrete values, all numeric attributes in the dataset will be rounded up in order that we do not have decimals in our dataset but rather whole numbers. The dataset was splitted into two parts namely the training set which comprise of two-third of the entire data and a test set which comprise of one third of the whole data. The training set is used to train the model and the test set is used to verify how well the classifier can perform on new data. Figure 1 shows the classification tree produced by rpart when no constrain is set.

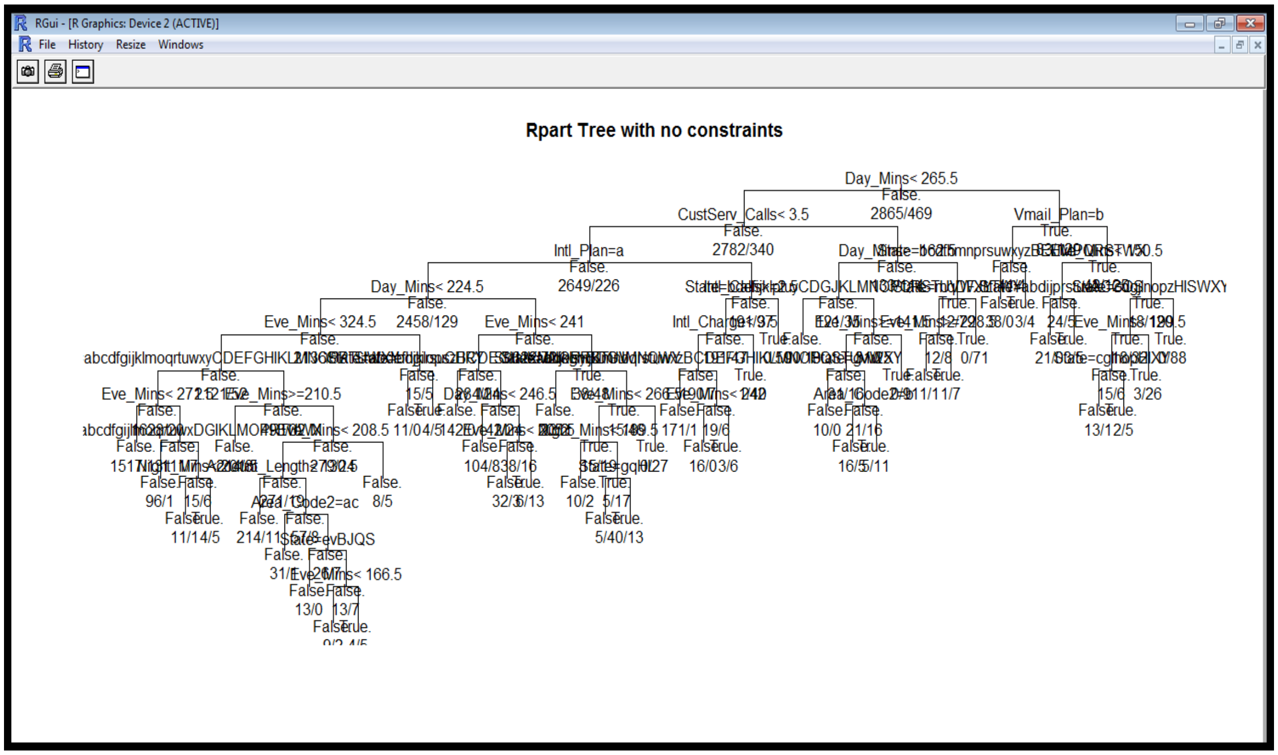


Figure 1:Rpart tree with no constrains.

When parameters are set to default (cp=0.001, minsplit=20), the following tree is produced with 14 leaves.

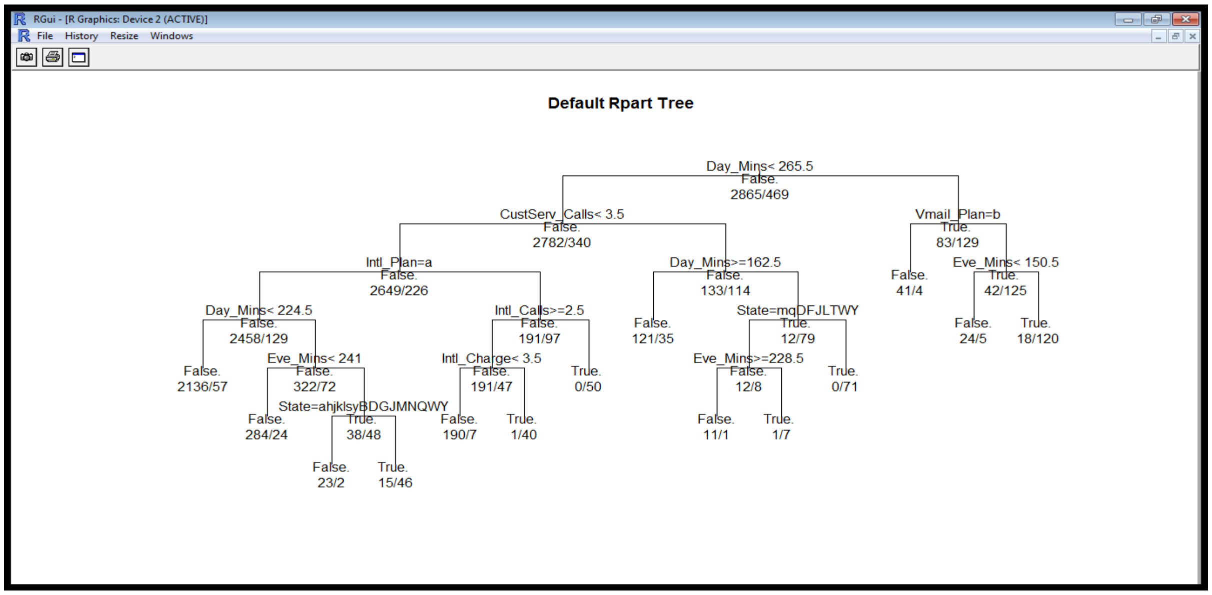


Figure 2: Default Plot of RPart 1

It should be noted that CART algorithm chooses by itself the significant variables for classification and these variables are not to be chosen apriori except for special reasons. The next step is to select a right sized tree. In order to do this, we first choose the tree size by specifying the minimum number of splits and the cost-complexity parameter(cp, which is at default equal to 0.001). These information can be gotten from the rpart plot of the relationships between cost(complexity) parameter(cp), the cross validation error(x-val Relative Error) and the tree size(number of nodes) which is shown below in figure 2.

Code: `Plotcp(fit, minline=TRUE, lty=3, col=1, upper=c("size"))`

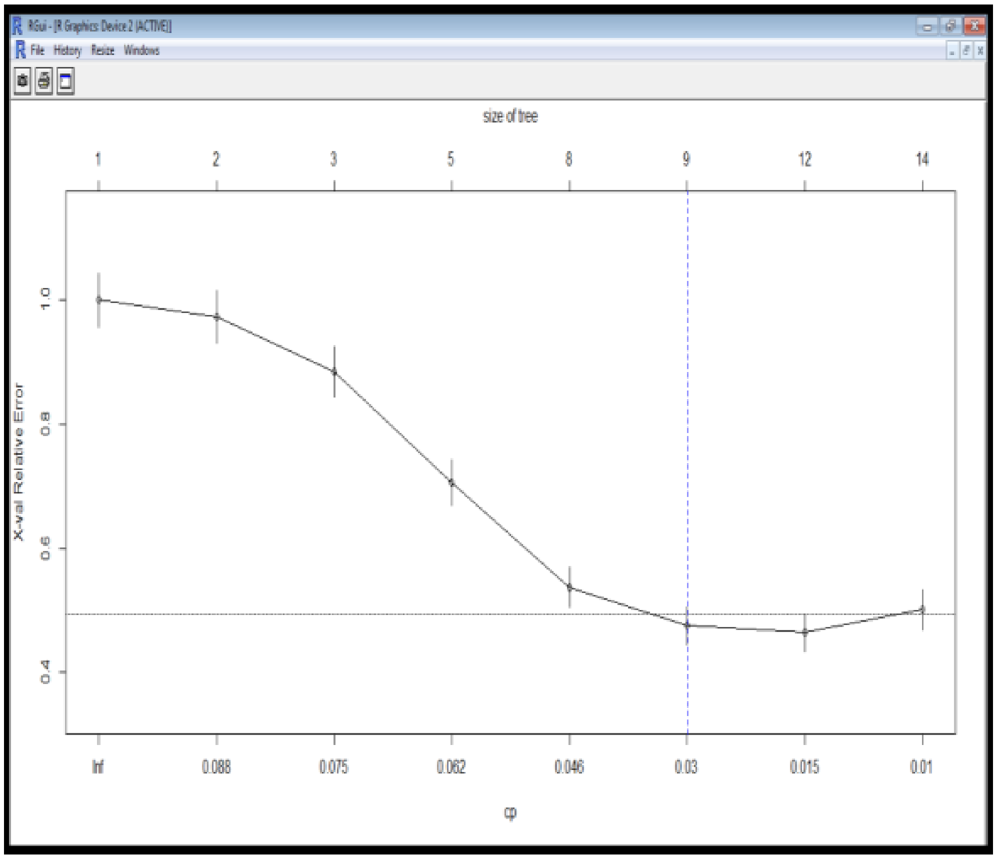


Figure 3: CP Plot of default RPart Tree(figure 2)

The best cp(α) value is chosen as the left-most value below the horizontal line and the vertical line indicates the best cost-complexity(cp) value and the corresponding tree size. As seen earlier, the pruning technique used by CART is called minimal cost complexity pruning and it assumes that the bias in the re-substitution error of a tree increases linearly with the number of leaf nodes.

Variables actually used in tree construction:

CustServ_Calls, Day_Mins, Eve_Mins, Intl_Calls, Intl_Charge, Intl_Plan, State, Vmail_Plan

From the above plot and table, a value of cp=0.03 with a corresponding tree size of 9 nodes gives us the necessary parameters for our right sized tree. Using this information we get the following structure of our final tree.

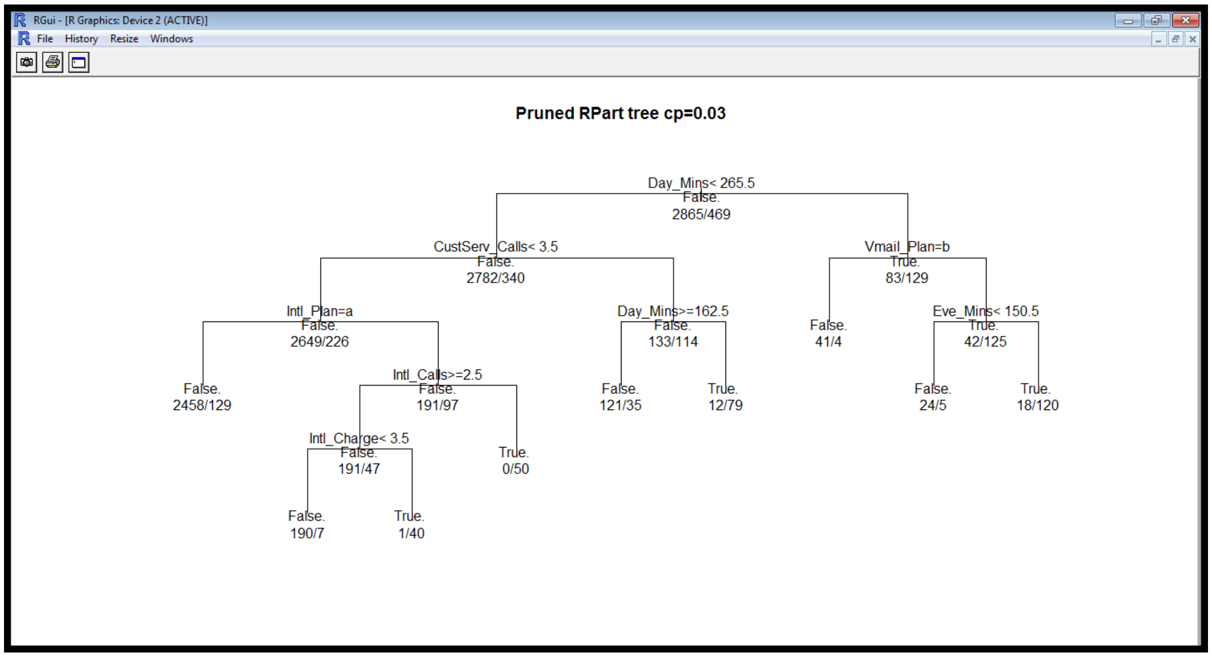


Figure 4: Pruned Tree with 9 leaves(size of tree), cp=0.03

We can see that in the pruned tree, some variables have been dropped e.g state. The re-substitution error gives the error rate computed on the learning sample. The following table shows the confusion matrix for the prediction on the test data.

Code:

```
Test<- predict(fit, type='class', testUCI)
Confmatrix<- table(testUCI$Churn,Test)
resub.error <- 1.0 – (confmatrix[1,1] + confmatrix[2,2])/sum(confmatrix)
```

Resubstitution error = 0.06002401

Accuracy = (True positive + true negative)/total

Accuracy =(1403 + 163)/1666

Accuracy = 93.99% correctly classified instances

Accuracy = 93.99%		
	False.	True.
False.	1403	25
True.	75	163

Table 1: Confusion matrix for observed v predicted instances

Cross Validation

The cross-validation approach is also used to obtain a more reliable error rate estimate of the rpart tree using the entire dataset. This method is preferred except in cases where the sample size is very large) This method will be programmed for rpart trees by specifying a cross-validation parameter in the control object. Hence, by specifying this parameter, the user informs the program that any split which does not improve the fit by cp(cost-complexity) will likely be pruned off by cross validation, thus no need to pursue it(Therneau and Atkinson 1997). Inorder to be sure of minimal errors, another plot is produced with an option for 10-fold cross-validation to verify if it differs from our pruned tree (Tree 2 above) . The following tree is produced:

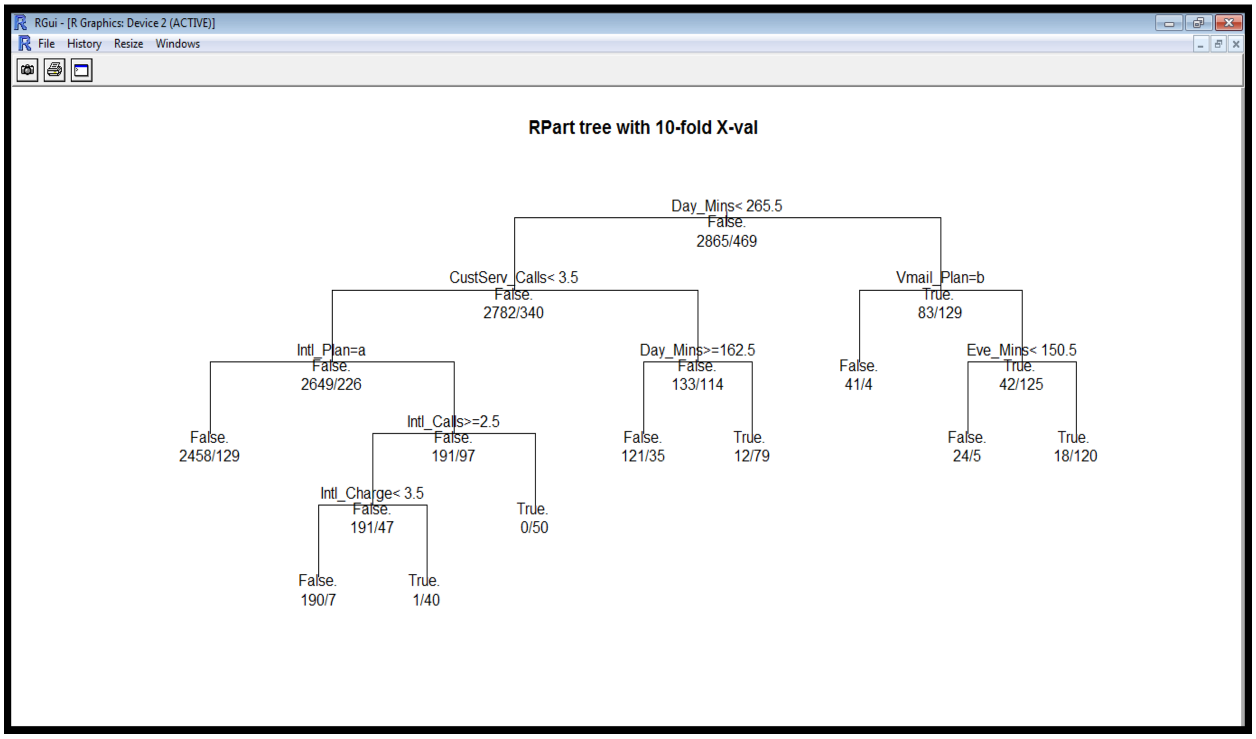


Figure 5: Rpart tree with 10-fold Cross-validation

The resulting pruned tree produced with 10-fold cross validation appears to be similar with the pruned tree without coss-validation. To be more precise, we look at the confusion matrix and misclassification rate to see get an idea of how good the classifier can perform on new data by apply it to predict the test set. The following results were obtained:

Table 2: confusion matrices of default, pruned and Cross-Validation Models

	Default Model		Pruned Model		X-Val Model	
	Accuracy = 93.99%		Accuracy = 93.35%		Accuracy = 93.35%	
	False	True	False	True	False	True
False.	1403	25	1410	18	1410	18
True.	75	163	90	148	90	148

An ROC Curve for the three models is plotted whereby the default tree has the largest area under the curve and the 10-fold cross-validation and pruned tree have the same area under the curve.

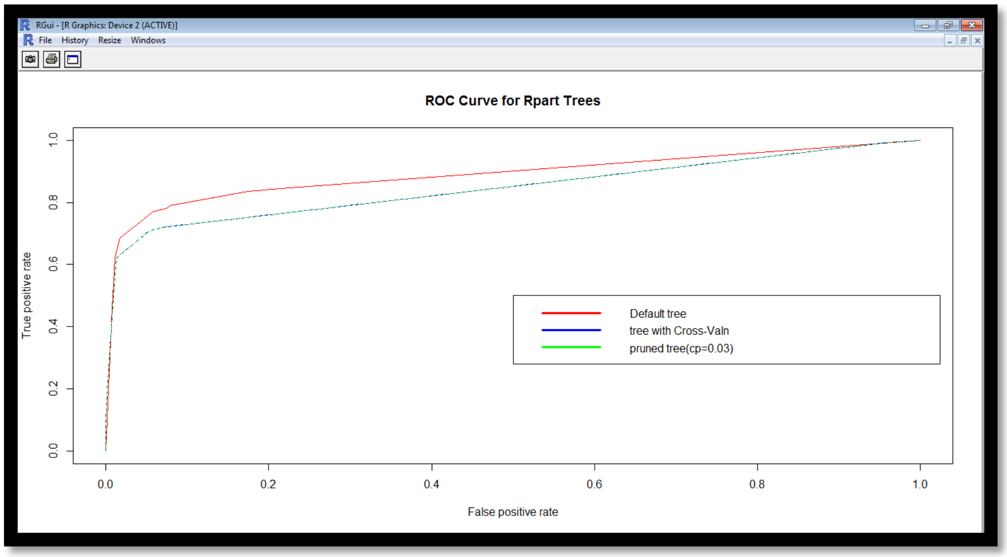


Figure 6: ROC Curve for rpart trees

Note here that the ROC of cross-validation tree and pruned tree are seen to be equal. From the final tree above, the most important variable that can be used to split the data to predict whether a customer will churn or not is seen to be “Day-Mins”(total day minutes). At every node, we see a decision rule used for prediction. At the starting node, the decision rule is :

If “day-mins” is less than 265.5, then check “CustServ_Calls”, else check “Vmail_plan”. It should be noted that the decision rule is related to the left split of the corresponding node and the else statement is related to the right split. After the data is splitted at the first node, the process is repeated for each sub-group until the splitting process stops whereby the subgroup reaches the minimum size or no further improvements can be made. To predict the response, we start from the root or start node to each leaf. For example, Let us consider the leftmost leaf from the above tree. It tell us that:

If “Day_mins”(total day minute) is less than 265.5mins , “CustServ_calls(number of customer service calls) is less than 3.5 and Intl_Plan(International Plan) is equal to “no”, then the customer is not likely to churn(i.e churn =false).

Also, if we consider the right-most node in the above tree, it tell us that:

If “Day_mins”(total day minute) is more than 265.5mins , “vmail_plan”(voicemail plan) is not equal to “yes”, eve_minutes(total evening minutes) is greater than or equal to than 150.5minutes then the customer is likely to churn.

Of all the nine(9) leaves representing 9 decision rules, two of them have been interpreted so far. Five of the decision rules tells us about when a customer is not likely to churn and four(4) of the decision rules tells us about when the customer is likely to churn.

Nevertheless, as in most decision tree algorithms, we want our chosen tree to be understandable, accurate and comprehensive. We have looked at comprehensibility and now let us look at the level of accuracy of the tree. Talking about accuracy in this case, we look at the confusion matrix as seen on table 2.

Also from the plot of the ROC curve, it is plausible to say there is a trade-off between accuracy and comprehensibility. This is due to the fact that as we prune the tree, the level of accuracy drops as well. In this case, the area under the ROC curve can also be check. In this case, it is found to be very close to the full grown tree and equal to the case with cross-validation.

3.2 CART analysis with RpartScore Package

RpartScore is an R package used for deriving classification trees with an ordinal response. In deriving a classification tree with ordinal response, the **generalized gini impurity criterion** which is different from the Rpart gini impurity criterion in that it takes into account the additional information when the response variable is ordinal.

3.2.1 Application of RpartScore Tree with different splitting rules

This NLSY data[10] was used to illustrate a tree with an ordinal response where the response variable was transformed to be ordinal.
For illustrative purposes, response variable 'wage' was transformed to an ordinal variable as follows:

1 =wage > 3500
2 =3000 < wage ≤ 3500
3 =2500 < wage ≤ 3000
3 = bwt ≤ 2500

The data was splitted into the training set(two-third of the data) with 2410 observations which was used to train the model and a test set(one-third of the data) with 1204 observations which was used to test the predictive accuracy of the model. With two types of splitting criteria (absolute or quadratic differences in scores) and two types of pruning criteria (misclassification rate or misclassification cost), we always end up with four types of trees when applying the generalized gini function for constructing trees with an ordinal response variable. The resulting four trees are given as follows:

i) Classification tree for ordinal data with absolute differences in scores for splitting and misclassification cost for pruning.
ii) Classification tree for ordinal data with absolute differences in scores for splitting and misclassification rate for pruning.
iii) Classification tree for ordinal data with quadratic differences in score for splitting and misclassification cost for pruning.
iv) Classification tree for ordinal data with quadratic differences in scores and misclassification rate for pruning.

[10] http://www.oswego.edu/~kane/econometrics/data.htm

Code:

```
Library(rpart)
Library(rpartScore)
s.abs.mr <- rpartscore(Category ~.,data=trainnlsy,
                       prune =”mr”,          #specify misclassification rate
                       split="quad")         #specify quadratic differences in scores
```

3.2.1.1)Absolute differences in Scores with misclassification cost.

When absolute differences in scores is used as the splitting criteria and misclassification cost is used as the criteria for pruning, we get the following tree:

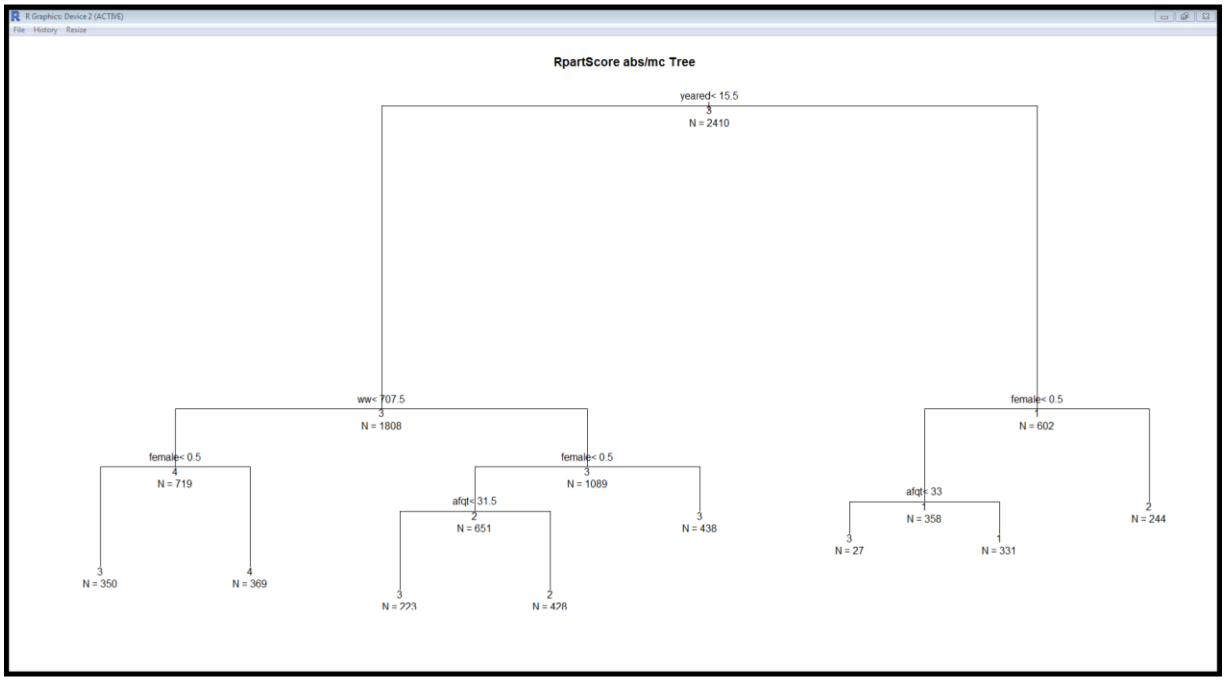


Figure 7: absolute difference in scores vs Misclassification cost

CP-Plot (see appendix figure A1) gives us an idea of the optimal tree size. Applying cp=0.011 as seen on figure A1 of the appendix does not affect the tree(see appendix figure A2). We see this because the algorithm takes into consideration the cost complexity parameter when constructing the tree. Thus the tree size does not change when the cost complexity parameter(cp) value is included in model statement. The final model in this case is a tree with 8 leaves used to predict the wage level. The model uses only four(4) variables to predict the response variable(wage).

3.2.1.2)Absolute Differences in scores with misclassification rate

The second tree uses the absolute differences in score as the splitting criteria and misclassification rate as the pruning method. The resulting tree is seen to have only 5 leaves and when the corresponding complexity parameter is applied to the model no change is given. The resulting trees are shown as follows;

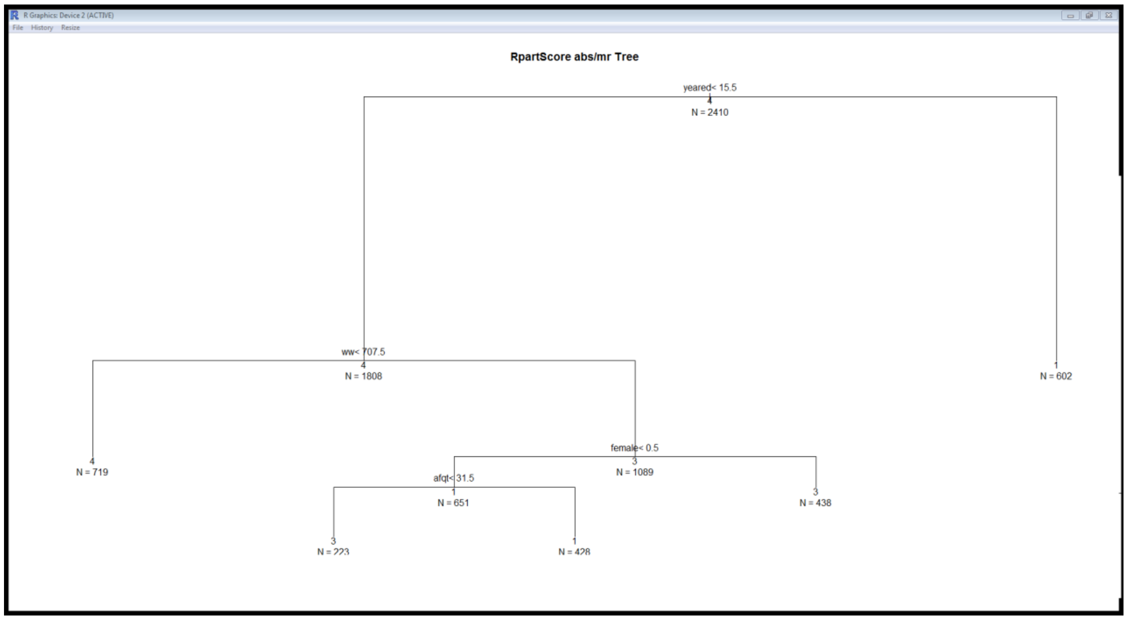


Figure 8: absolute difference vs Misclassification rate

The cp-plot(see appendix figure A3) gives an indication of the best value of the complexity parameter for the tree. When applied to the model, the tree does not change which means that tree model takes the optimal value into consideration(see appendix A4). Still no differences between the tree with cp=0.011 applied. The tree size remains unchanged with 5 leaves.

3.2.1.3)Squared Differences in scores with misclassification cost

As seen in the cases above, the models above the tree does not change when the complexity parameter is introduced with the cp value chosen from the plot. This is because the pruned model takes this into consideration when building the tree.

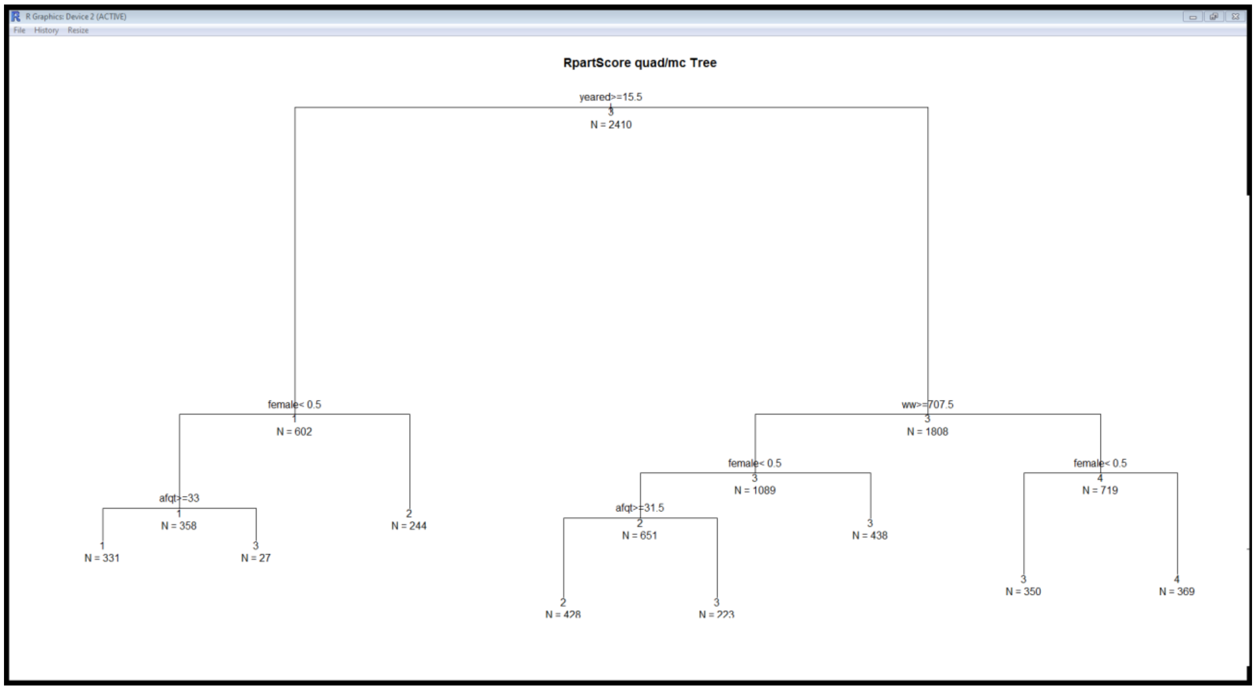


Figure 9: Quadratic difference vs Misclassification cost

The pruned tree with cp=0.011 chosen from the cp-plot does not change the tree size.

3.2.1.4)Squared Differences in scores and Misclassification Rate

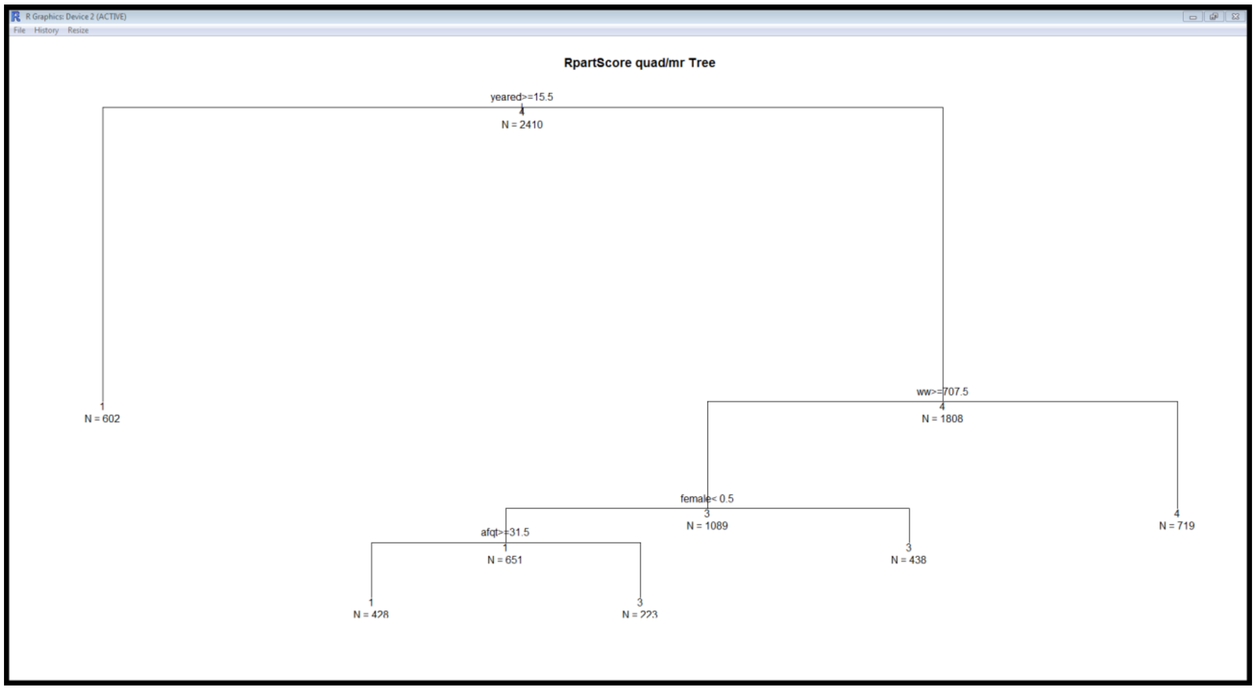


Figure 10: Quadratic differences in scores with Misclassification rate

The resulting tree has 5 leaves and does not change when cp value is introduced.

It can be clearly seen in all the above cases that when using the generalized gini impurity function, it produces the optimal tree since it takes into consideration already the cut-off point of the complexity parameter (cp) when it grows the tree. The somer's d measure for the four different ordinal response tree type discussed above are given as follows:

somers.d1 (abs.mc) = 0.4208102

somers.d2 (abs.mr) = 0.4003806

somers.d3 (quad.mc) = 0.4208102

somers.d4 (quad.mr) = 0.4003806

It can be seen from the above results that the trees with misclassification cost as pruning method have the same results and vice-versa for the case of misclassification rate. It can also be seen that the above values are approximately the same thus the measure of association between the observed and predicted response in all four models are similar. It would be nice to know which combination of splitting rule and pruning method is the best in this case but more extensive research is been carried for this.

It is important to note here that when misclassification rate is used as the pruning measure(see figure 8 and figure 10), only three of the four classes of the response variable are classified but when misclassification cost(see figure 7 and figure 9) is used, all the four classes of the response variable are classified thus the in this case, it will be plausible to interpret any of the trees shown in figure 7 and figure 9 which both have eight leaves. The measure of accuracy in this method of generalized gini splitting criteria is still under research although the somer's d measure was proposed by the founders of the rpartScore package.

3.2.3 Rpart for Regression Trees.

The boston housing dataset(which is about housing values in suburbs of Boston) was used to illustrate the construction of regression trees using rpart. The dataset which was downloaded from the UCI machine learning repository[11], contains 506 instances with 14 attributes. In order to grow a regression tree in the R programming software, the Rpart package is used similarly as in classification trees above with the specification of 'method=anova'. The resulting tree with 10fold-crossvalidation is given as follows:

rpart(formula = medv ~ ., data = boston, method = "anova", xval = 10)

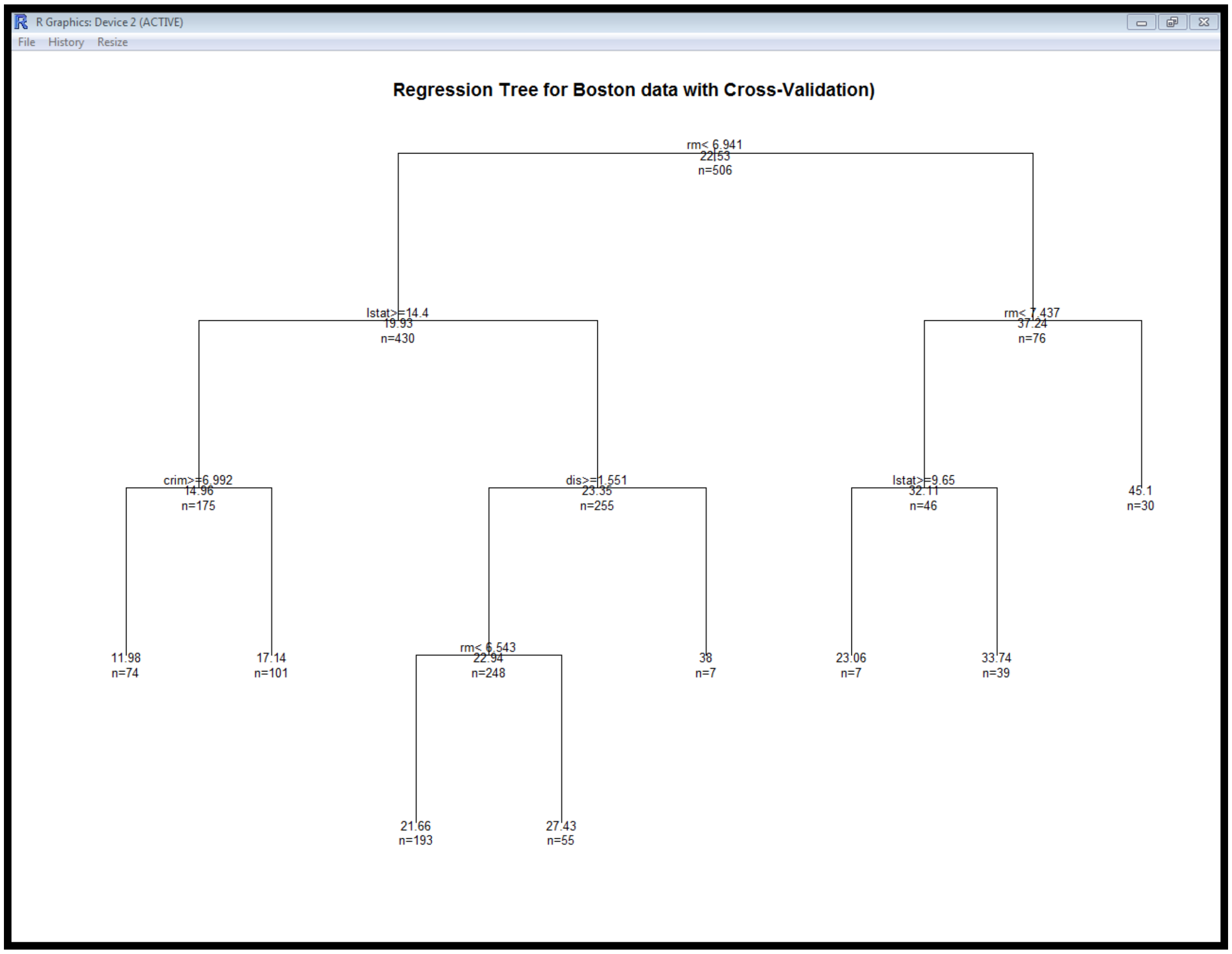


Figure 11: Regression Tree with Cross Validation

[11] http://archive.ics.uci.edu/ml

Regression tree:

```
rpart(formula = medv ~.,data = trainboston, method = "anova",xval=10,
cp=0.021)
```

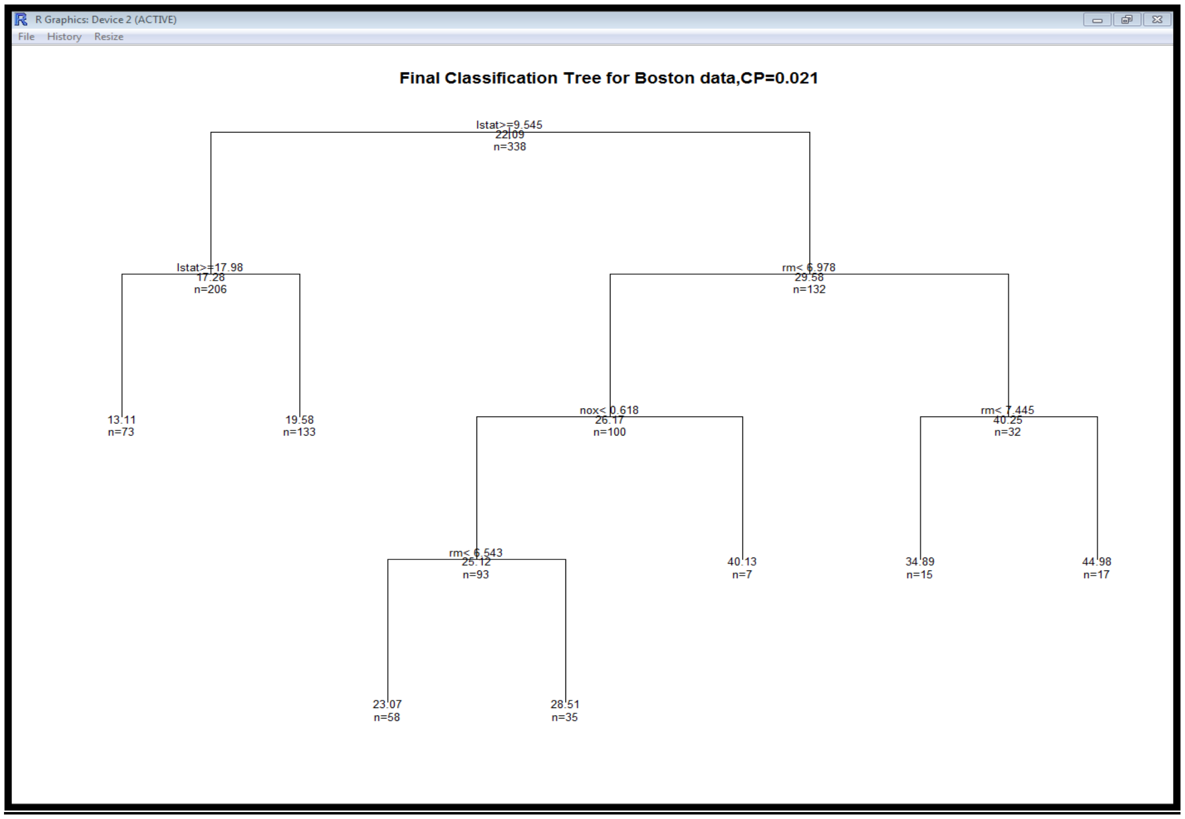


Figure 12: pruned regression tree

After building the final tree based on the complexity parameter(cp=0.021) as seen from the cp plot in figure A9(appendix), we get the pruned tree shown in figure 12. Next, it is worthwhile investigating how well this tree can perform on new data. In order to do this, the testset(30% of the original data) is used. From the results, it was seen that the resulting model had an adjusted R-square of 0.77

Variables actually used in tree construction:

“crim”, “dis”, “ lstat”,”rm”

From the linear model fit between the observed and the predicted response there seems to be a high concordance(r-squared=0.77) between observed and predicted response which is an indication of plausible prediction although this does not prove that the prediction is satisfactory.

Interpreting the regression tree for the Boston dataset with 10 fold cross-validation is chosen since the size of the data set is not large and 10 fold cross-validation work better in this situation. can be seen as follows:

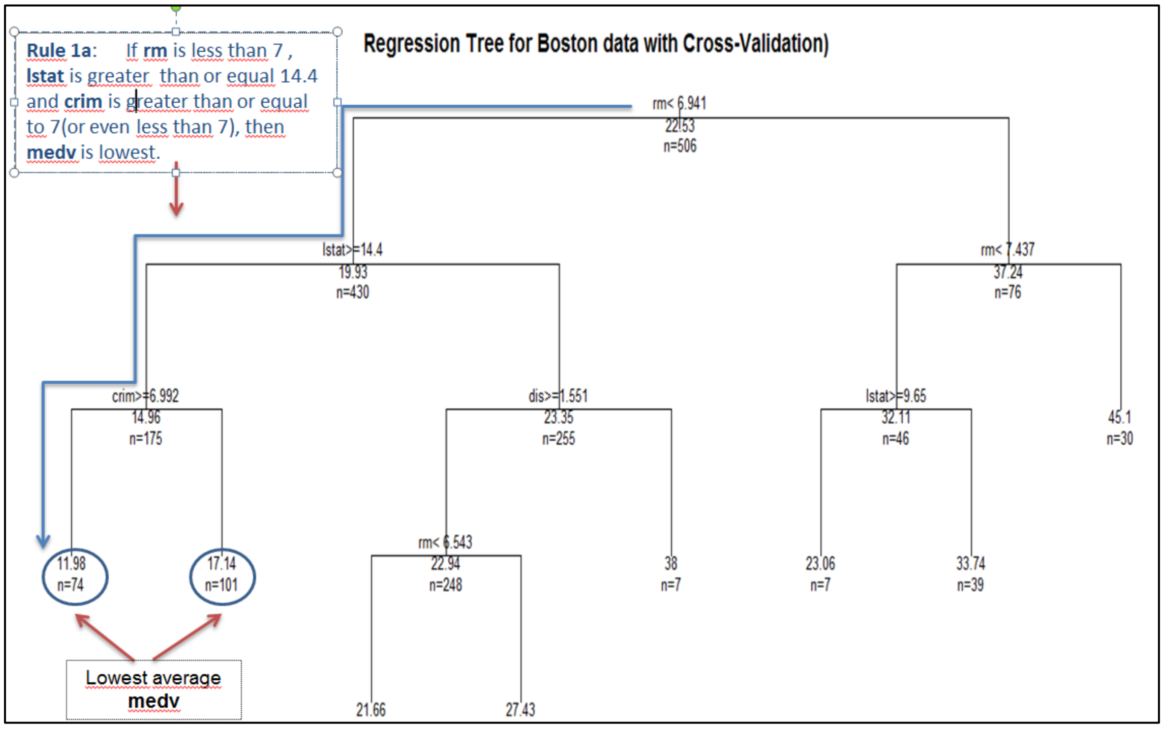


figure 13: interpretation of regression tree

The blue line above from the top node to the root node results in a decision rule that could be used to predict ‘medv’. It is worth mentioning here that in order in predict the response variable when using regression trees where the response variable is continuous, we interprete the response in terms of the average. Interpretations here will be done using the tree with 10-fold cross validation. The only variables which were seen to be retained by the algorithm are “Lstat”, “crim”, “dis” and “rm”. Eight tree rules to explain “medv”. The adjusted R-square = 0.8003 with cross-validation error of 0.2981 and R-square=0.702

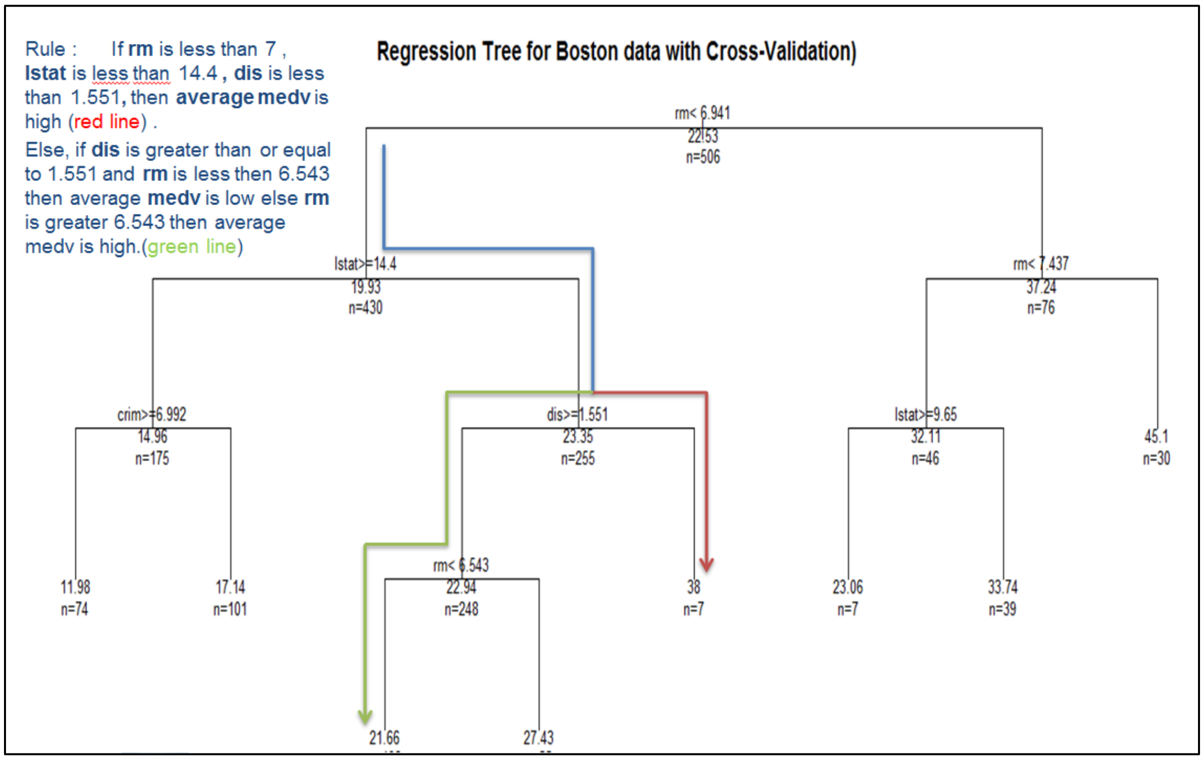

Rule : If rm is less than 7 , lstat is less than 14.4 , dis is less than 1.551, then average medv is high (red line) .
Else, if dis is greater than or equal to 1.551 and rm is less then 6.543 then average medv is low else rm is greater 6.543 then average medv is high.(green line)
Regression Tree for Boston data with Cross-Validation)
rm< 6.941
22.53
n=506
lstat>=14.4
19.93
n=430
rm< 7.437
37.24
n=76
crim>=6.992
14.96
n=175
dis>=1.551
23.35
n=255
lstat>=9.65
32.11
n=46
45.1
n=30
rm< 6.543
22.94
n=248
11.98
n=74
17.14
n=101
38
n=7
23.06
n=7
33.74
n=39
21.66
27.43

Figure 14

Finally, the middle branch of the tree(figure 15) can be interpreted as follows :

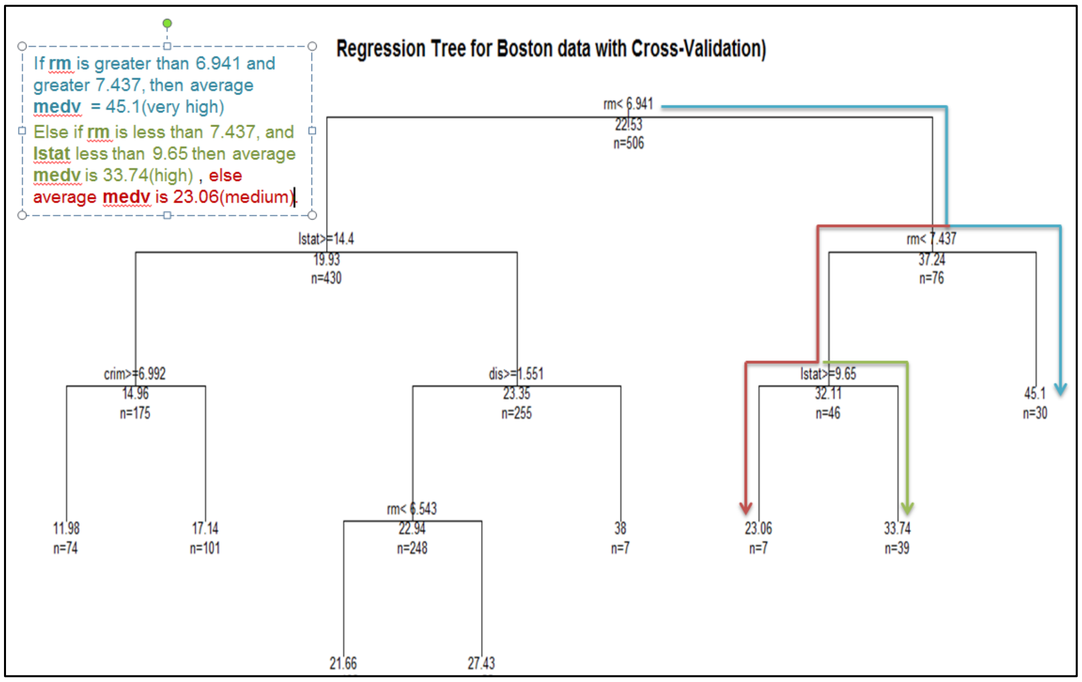

If rm is greater than 6.941 and greater 7.437, then average medv = 45.1(very high)
Else if rm is less than 7.437, and lstat less than 9.65 then average medv is 33.74(high) , else average medv is 23.06(medium)
Regression Tree for Boston data with Cross-Validation)
rm< 6.941
22.53
n=506
lstat>=14.4
19.93
n=430
rm< 7.437
37.24
n=76
crim>=6.992
14.96
n=175
dis>=1.551
23.35
n=255
lstat>=9.65
32.11
n=46
45.1
n=30
rm< 6.543
22.94
n=248
11.98
n=74
17.14
n=101
38
n=7
23.06
n=7
33.74
n=39
21.66
27.43

Figure 15

3.2) C4.5/C5.0 Analysis

3.2.1) C4.5 Analysis using RWeka Package

3.2.1.1) C4.5 Analysis using Car Data

The C4.5 algorithm for decision tree will be implemented using the car evaluation dataset from the UCI list of datasets(http://archive.ics.uci.edu/ml/machine-learning-databases/car/). The dataset consists of 1718 instances and 6 variables. These variables were collected based on price, technical characteristics and comfort.

The dataset was split into two parts namely the training set(two-third of data) and the test set(one-third of data). The dataset was also seen to have no missing values. Thus we can proceed to build a classifier based on the training data. The classifier produces a default tree with 97 number of leaves and 134 nodes(size of tree). The resulting tree is quite a large tree and has to be pruned to reduced the tree size in order to get a simpler tree. In order to do this some control parameters are included in the model.

First, a parameter R is included to use reduced error pruning in the decision tree. The number of leaves are seen to reduce to 77 and the tree size reduces to 105 nodes. Since the tree size is still quite large, pruning is continued by increasing the minimum number of instances per split(M) to 10 and the resulting tree is seen to have a tree size of 21 with 15 leaves. The tree size of 21 from 182(original size) can be thought to be reasonable as the tree needs to be simple.

Another pruning method is carried out on the original tree using the confidence threshold(default = 0.25). This pruning confidence is fluctuated by very small variations of about 0.05 until a satisfactory tree is gotten; At a value of C=0.15, a tree of size 89 with 65 leaves and at a value of C=0.05 the size reduce to tree size of 63 with 46 leaves and the parameter is again reduced for further pruning. Finally , at a value of 0.001 a tree of size 17 with 12 leaves is produced.

It is worth mentioning that since our response variable is categorical with 4 levels, J48 trees also contain multiple-way splits as many as four(4). Currently, plotting trees with multi-way splits are not implemented in R. But, the tree can be given in the form of rules and some statistics about the tree can also be gotten as shown below:

```
------------------
safety = high
|   persons = 2: unacc (124.0)
|   persons = 4: acc (128.0/52.0)
|   persons = more: acc (128.0/62.0)
safety = low: unacc (368.0)
safety = med
|   persons = 2: unacc (132.0)
|   persons = 4
|   |   maint = high: acc (32.0/15.0)
|   |   maint = low: acc (30.0/13.0)
|   |   maint = med: acc (34.0/13.0)
|   |   maint = vhigh: unacc (35.0/6.0)
|   persons = more
|   |   lug_boot = big: acc (45.0/17.0)
|   |   lug_boot = med: acc (43.0/19.0)
|   |   lug_boot = small: unacc (47.0/13.0)
```

Figure 16: pruned J48 tree

```
Number of Leaves  :     12
Size of the tree :      17
```

Table 4

=== Confusion Matrix ===
a   b   c   d   <-- classified as
249   0   19   0 | a = acc
43   0   0   0 | b = good
106   0 687   0 | c = unacc
42   0   0   0 | d = vgood

The above J48 pruned tree as seen on figure 16 and table 4 does classify all the classes of the response variable(evaluatn) as seen in the case of class 'good' which was not classified in the final model where all the instances classifed as 'good' were wrongly class as 'acc'. This information can be seen in table 4 above.

This implies that the resulting model does properly classify the response variable although it has an accuracy of 81.67% of correctly classified instances. This explains why when a lot of pruning is made on a tree, much information can be lost. To illustrate how these tree which is given in the form of rules can look like in tree format, a tree was manually drawn as seen in figure 17 below.

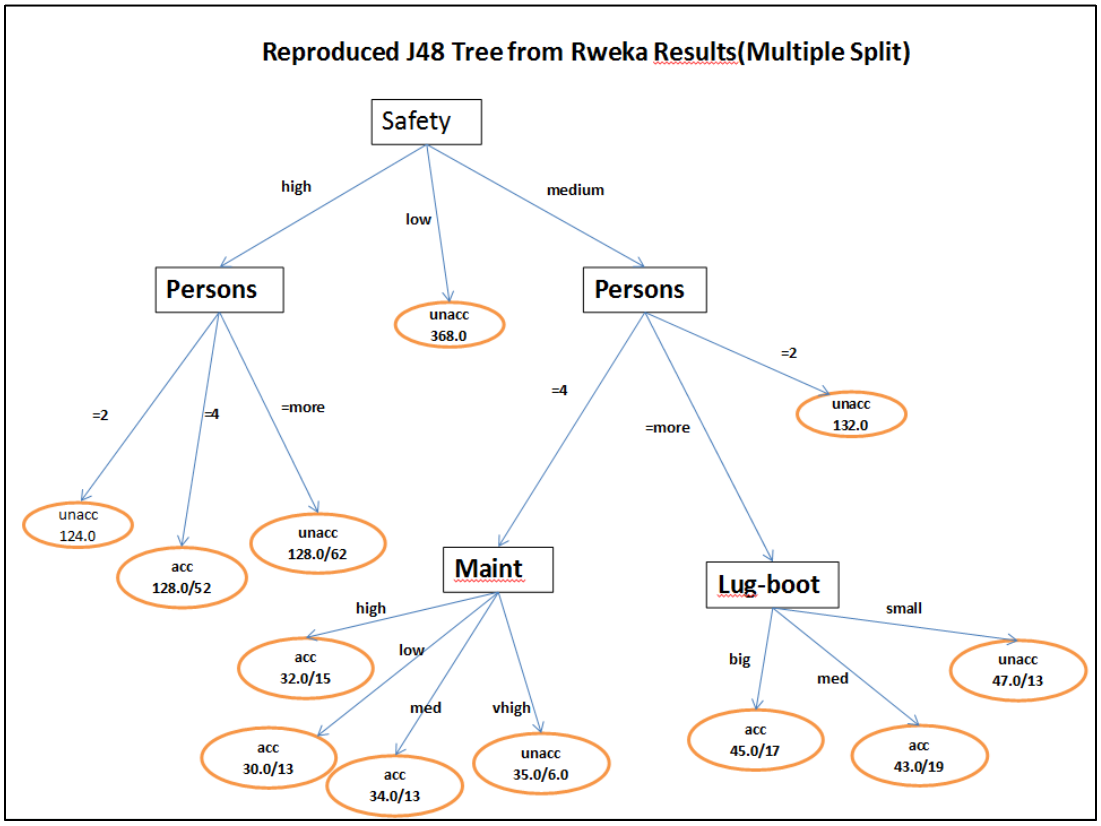


Figure 17: Reproduced J48 pruned tree 1

From the above tree diagram(which is represented in the form of rules), we can represent it in the form of a tree by hand as seen below. This is to make it interpretable by the user. For very large trees, this would be difficult to draw. The main idea here is to help in interpretation although in the previous format, the tree can still be read.

The numbers written under leaves indicate how many instances fall in that category and the second number indicates how many were incorrectly classified. The square figures represent the nodes and the round(orange) figures represent the leaves. The left-most left can be interpreted as;

-If “safety” is high and the capacity of the car(“persons”=2), then the buying price is inaccurate(unacc). At this split, we can see that all the instances were correctly classified which indicates a perfect split. The same procedure can be repeated for the 12 leaves and we would end up in 12different decision rule to evaluate the car price.

3.2.1.2 Churn Analysis using RWEKA

The UCI Churn dataset is used to illustrate the C4.5 algorithm in R. This data is used here in order to illustrate how the J48 algorithm can produce trees with binary splits a opposed to multiple splits it produced above. When the splits are binary, it is possible to get a tree diagram in the R-programming software as opposed to multiple split tree diagrams which cannot be produce directly as of now. The dataset was split into two parts namely the training set (two-third of data) and the test set(one-third of data). The dataset was also seen to have no missing values. Thus we can proceed to build a classifier based on the training data using the J48 algorithm to classify the UCI dataset, we get the resulting plot;

Default tree on training set:

The resulting tree is seen below to have a tree size of 53 with 27 leaves represented as rules. A plot of this tree is shown on Figure B1(See appendix).

Code:

```
tree1 <- J48(Churn ~ ., data = UCI, control = Weka_control(R = TRUE))

plot(tree1)

text(tree1)

tree2 <- J48(Churn ~ ., data = UCI, control = Weka_control(R = TRUE, M = 5))

tree3 <- J48(Churn ~ ., data = UCI, control = Weka_control(M = 6, C=0.10))
```

The resulting tree in Figure B1(see appendix) is seen to be very large and difficult to read or interprete thus it would be a nice idea to prune it. Using an option for reduced error pruning(R=TRUE , M=20) results in the following tree of size 31 with 16 leaves. As compared to the previous tree, this pruned tree looks more easier to interpret and read. The post-pruning method of reduced error pruning uses a hold-out set of the training data to make pruning decisions. As compared to other pruning methods, this uses just a fraction of the data and once the structure of the tree is learned, the full training set is used against this structure. It is worthwhile mentioning that to regulate the tree size in this case, the minimum number of instance M,(2,…,20) and the pruning confidence(C=0.25 default) can also be used but this will not be used in this case since after a number of trials regulating these two constructs, the resulting tree was still much larger than that used with reduced-error pruning. We have to continue pruning until we get a comprehensive tree keeping in mind the associated error. Varying these two parameters(M=20 and C=0.10) have led us to the following final tree(figure 18) with 16 leaves.

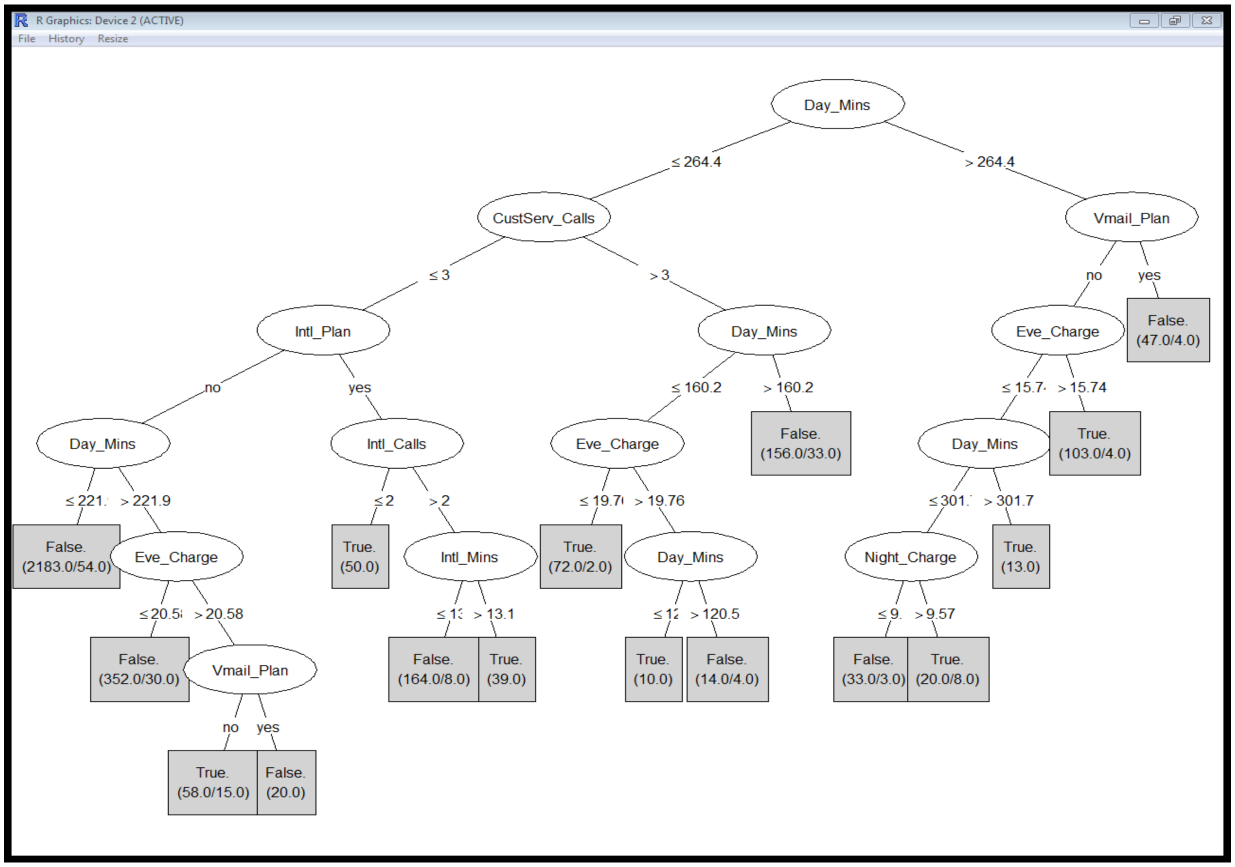


Figure 18: pruned C4.5 tree with churn data

It is worthwhile mentioning that to regulate the tree size in this case, the minimum number of instance M,(2,…,20) and the pruning confidence(C=0.25 default) can also be used but this will not be used in this case since after a number of trials regulating these two constructs, the resulting tree was still much larger than that used with reduced-error pruning. We have to continue pruning until we get a comprehensive tree keeping in mind the associated error. Varying these two parameters have led us to the following final tree.

The next thing to do once a final tree has been chosen is to evaluate how good is its performance or how good it can predict new data. Thus the test set is used to evaluate the classifier.

Table 5: confusion matrix

pruned tree				evaluation of classifier	
Accuracy= 95.31%				Accuracy= 89.61%	
a	b	<-- classified as		a	b
1427	16	|	a = False.	1370	63
62	161	|	b = True.	110	123

From the above results, it is plausible to conclude that the classifier perform well on the test-set since we can see that there was 95.31% of correctly classified instances. This algorithm produced a tree of size 53 with 27 leaves and after pruning using reduced error pruning method, the final tree of size 31 with 16 leaves as seen below on figure 19.

This tree seems to have a high accuracy level of 95.32 % of corrrectly classified instances and only 4.68% of incorrectly classified instances. The classifier was evaluated using the test set and it produced an accuracy of 89.61% of correctly classified instances. The tree gives 16 decision rules to predict whether a customer is likely to churn or not. The blue lines above leads to the following decision rules:

a) If a customer's total day call (Day_Mins) is less than or equal to 264.4mins and total customer service call(CustServ_Calls) is less than or equal to 3 with no international call lan(Intl_Plan) and his total day call (Day_Mins) is also less than or equal221mins, then the customer is unlikely to churn.
b) From (a) above, if the customer's total day call (Day_Mins) is greater than 221.9mins and his total evening charge(eve_charge) is less than or equal to 20.5units, then the customer is also unlikely to churn.
c) From (b) above, if there is no voicemail plan(vmail_Plan), the customer is likely to churn
d) From(b) above, if there is a voicemail plan(vmail_Plan), the customer is not likely to churn.

The same process is carried out for all the different nodes using the top-down approach.

How can this become interesting for the Company?

When making business rules for a telecom company, all the above decisions for the full grown tree will important especially relating to promotions geared at preventing customers from churning and also towards building a stronger relationship with customers even those they are sure will not churn. For the above four decision rules, there is an order of how to check them as follows:

First Check (a), then (b), from (b) check (c) or (d)

At this point, it is plausible to say that the tree in figure 19 is accurate and comprehensive.

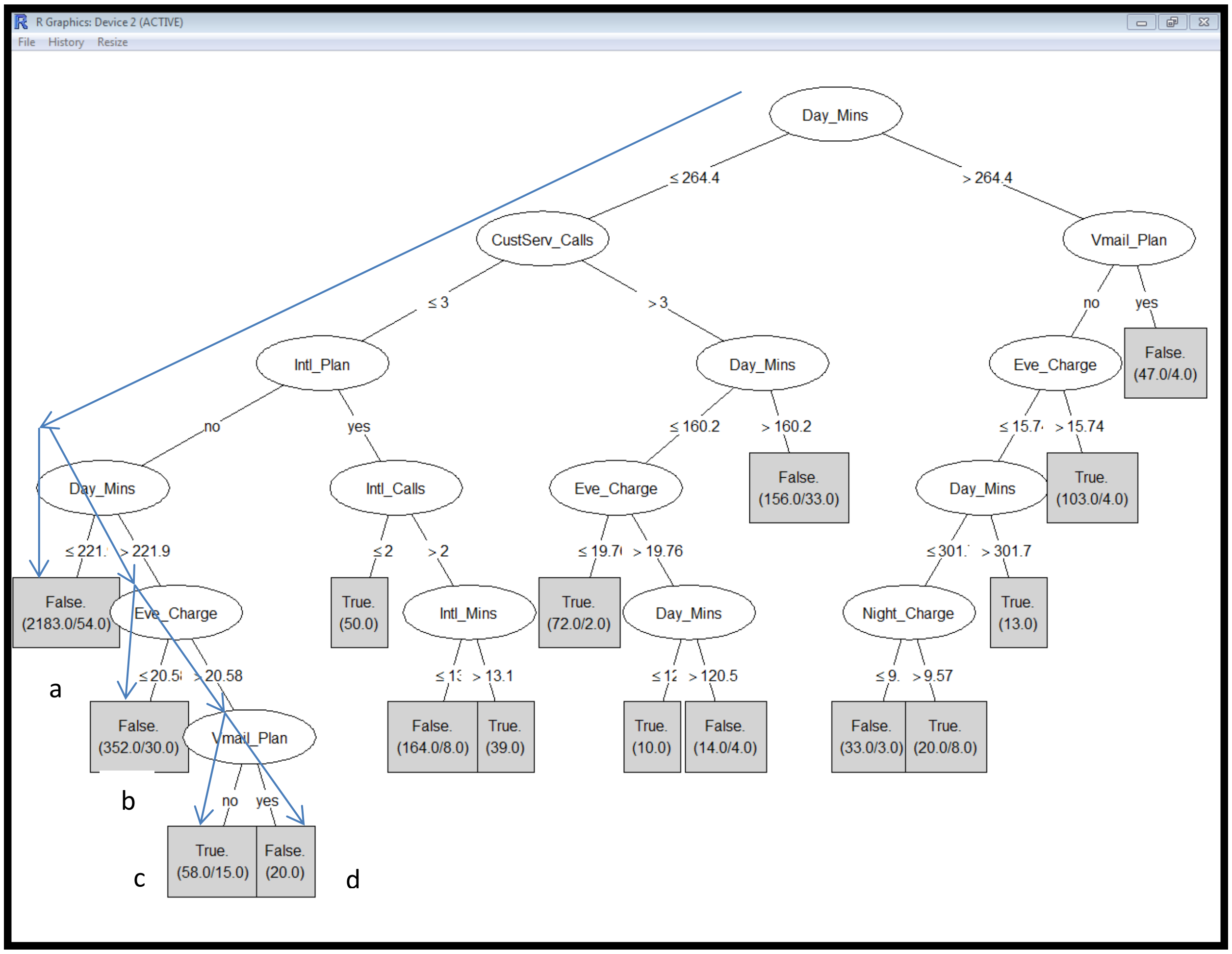


Figure 19

Churn Analysis Using C50 Package(C5.0)[12]

The Churn dataset was used as in the implementation of the C4.5 algorithm above using Rweka. Two third of the data was used to train the model and the following tree was produced. It is also important to state that C5.0 produces tree-based and rule-based models.

The C50 package allows for the user to represent the corresponding tree produced by the model in two forms:

a)-the tree format

b)-the rule set format

It should be noted that these two forms do not produce the same results. The rule set format is usually fewer than the leaves
The C50 package allows the user to split the dataset by including 'sample=0.65' in the C5.0 Control which tell it to use 65%percent of the data to train and the rest 35% to test the model.
Code:

```
tree <- C5.0(x = UCI[, -20], y = UCI$Churn,
             control = C5.0Control(sample = 0.65))
test<- predict(tree, head(testUCI[, -20]), type = "prob")
```

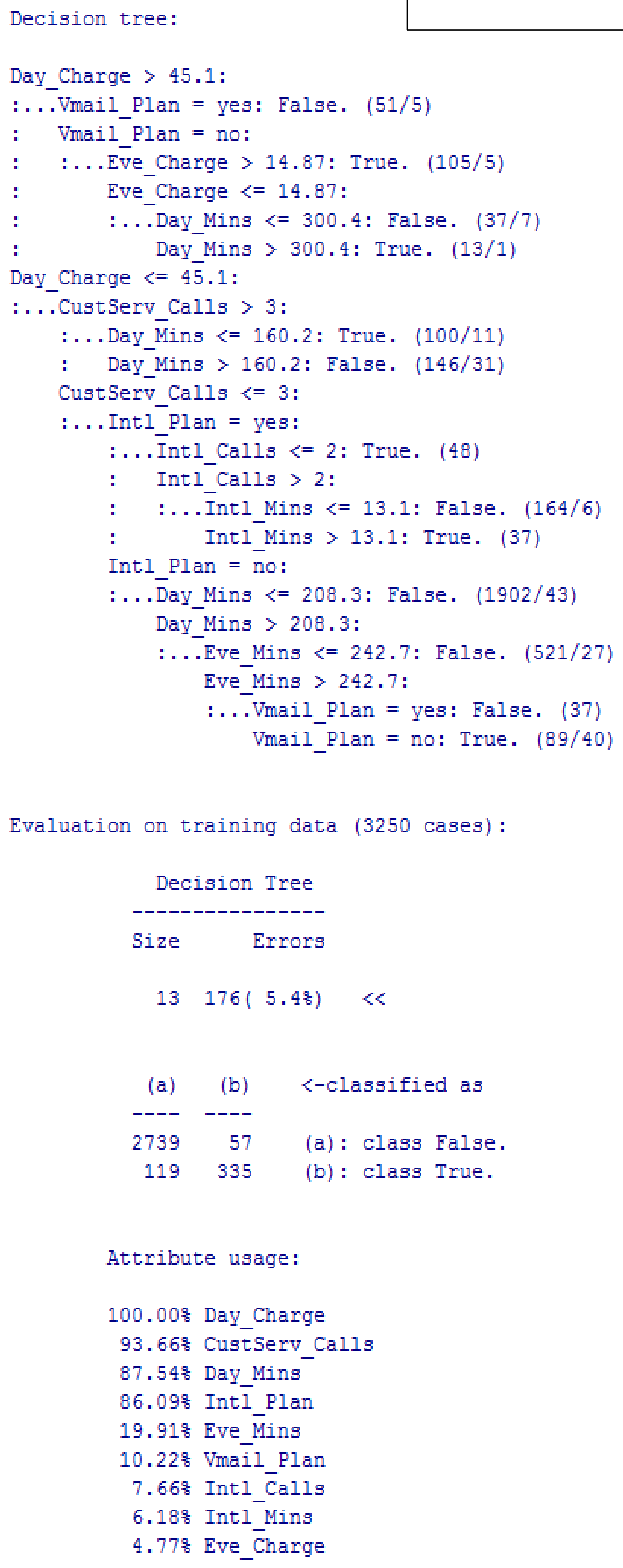
The problem here is that each time the model is run, you are likely to get different results due to the fact that the training set is not always composed of the same data. When this option is used, the user is advised to set seed which will make sure that the same training set is used or the user may as well split the data apriori into a training set and a test set and then use the training set to train your model. The resulting tree and rule models are presented below.

[12] http://www.rulequest.com/see5-unix.html

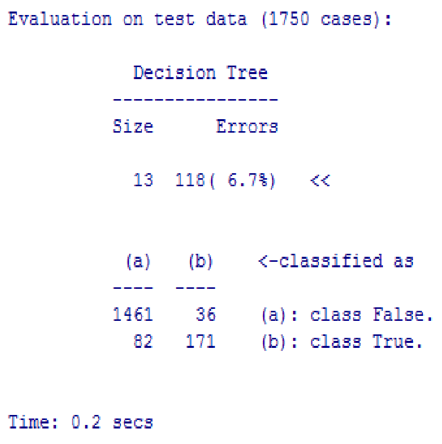
The tree model seems to perform well with 5.4 % misclassification error(94.6% correctly classified instances) on the training set and 6.7% misclassification error(93.3% correctly classified instances) on the test set with a tree size of 13 leaves.

The rule model also seems to perform better with 4.0% misclassification error(96%) correctly classified instance on the training set and 6.2% misclassification error(93.8% correctly classified instances) on the test set. It should be noted that the rule sets produced by C5.0 are usually fewer and more easy to interprete than the leaves of the tree .

C50 Decision Tree



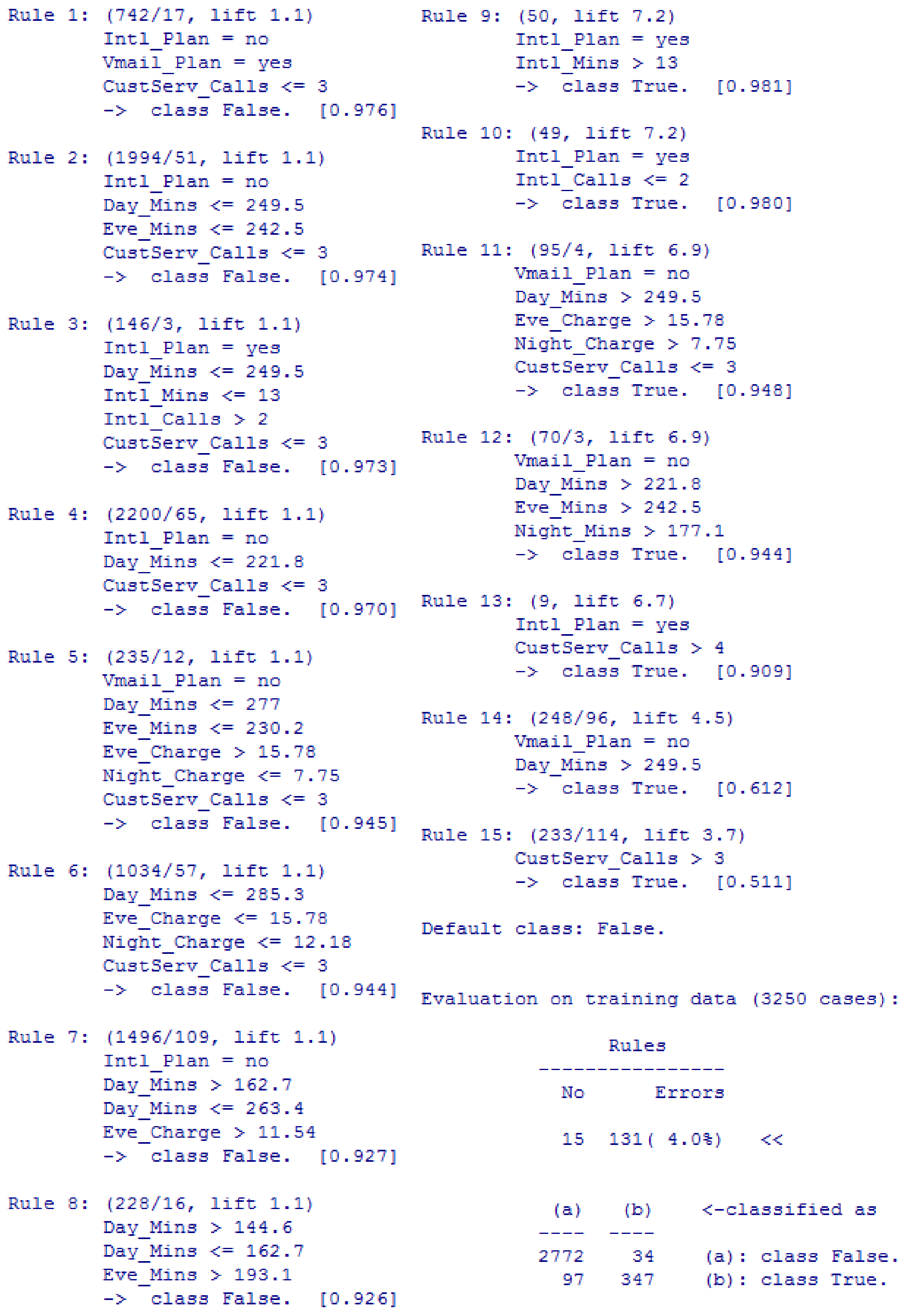
```
Decision tree:

Day_Charge > 45.1:
:...Vmail_Plan = yes: False. (51/5)
:   Vmail_Plan = no:
:   :...Eve_Charge > 14.87: True. (105/5)
:       Eve_Charge <= 14.87:
:       :...Day_Mins <= 300.4: False. (37/7)
:           Day_Mins > 300.4: True. (13/1)
Day_Charge <= 45.1:
:...CustServ_Calls > 3:
    :...Day_Mins <= 160.2: True. (100/11)
    :   Day_Mins > 160.2: False. (146/31)
    CustServ_Calls <= 3:
    :...Intl_Plan = yes:
        :...Intl_Calls <= 2: True. (48)
        :   Intl_Calls > 2:
        :   :...Intl_Mins <= 13.1: False. (164/6)
        :       Intl_Mins > 13.1: True. (37)
        Intl_Plan = no:
        :...Day_Mins <= 208.3: False. (1902/43)
            Day_Mins > 208.3:
            :...Eve_Mins <= 242.7: False. (521/27)
                Eve_Mins > 242.7:
                :...Vmail_Plan = yes: False. (37)
                    Vmail_Plan = no: True. (89/40)


Evaluation on training data (3250 cases):

            Decision Tree
          ----------------
          Size      Errors

            13  176( 5.4%)   <<


           (a)   (b)    <-classified as
          ----  ----
          2739    57    (a): class False.
           119   335    (b): class True.


        Attribute usage:

        100.00% Day_Charge
         93.66% CustServ_Calls
         87.54% Day_Mins
         86.09% Intl_Plan
         19.91% Eve_Mins
         10.22% Vmail_Plan
          7.66% Intl_Calls
          6.18% Intl_Mins
          4.77% Eve_Charge
```



```
Evaluation on test data (1750 cases):

            Decision Tree
          ----------------
          Size      Errors

            13  118( 6.7%)   <<


           (a)   (b)    <-classified as
          ----  ----
          1461    36    (a): class False.
            82   171    (b): class True.


Time: 0.2 secs
```

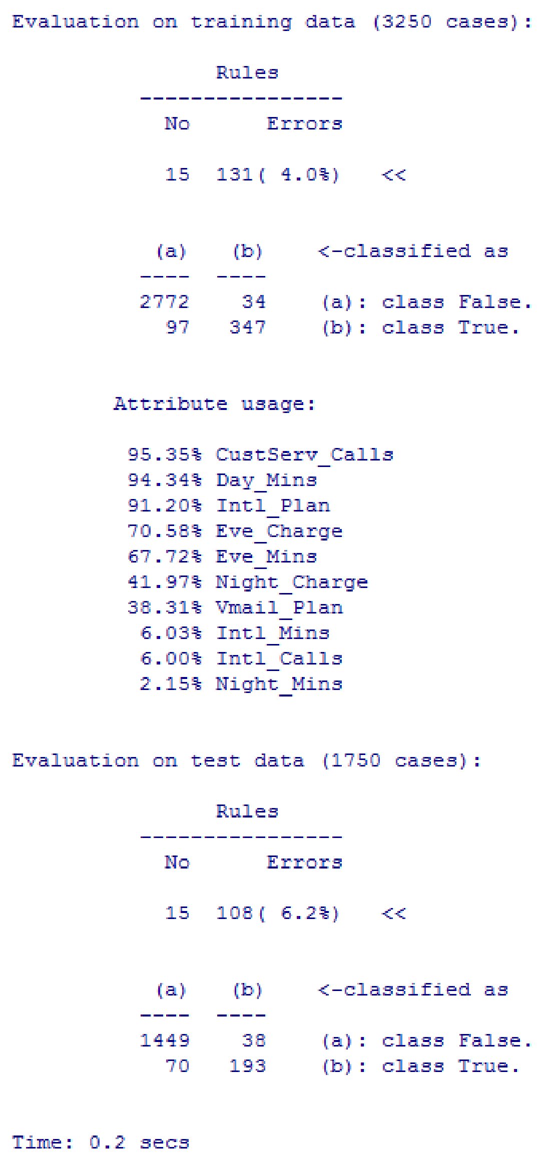
Figure 20: C50 T

C50 Decision Rules



```
Rule 1: (742/17, lift 1.1)
        Intl_Plan = no
        Vmail_Plan = yes
        CustServ_Calls <= 3
        ->  class False.  [0.976]

Rule 2: (1994/51, lift 1.1)
        Intl_Plan = no
        Day_Mins <= 249.5
        Eve_Mins <= 242.5
        CustServ_Calls <= 3
        ->  class False.  [0.974]

Rule 3: (146/3, lift 1.1)
        Intl_Plan = yes
        Day_Mins <= 249.5
        Intl_Mins <= 13
        Intl_Calls > 2
        CustServ_Calls <= 3
        ->  class False.  [0.973]

Rule 4: (2200/65, lift 1.1)
        Intl_Plan = no
        Day_Mins <= 221.8
        CustServ_Calls <= 3
        ->  class False.  [0.970]

Rule 5: (235/12, lift 1.1)
        Vmail_Plan = no
        Day_Mins <= 277
        Eve_Mins <= 230.2
        Eve_Charge > 15.78
        Night_Charge <= 7.75
        CustServ_Calls <= 3
        ->  class False.  [0.945]

Rule 6: (1034/57, lift 1.1)
        Day_Mins <= 285.3
        Eve_Charge <= 15.78
        Night_Charge <= 12.18
        CustServ_Calls <= 3
        ->  class False.  [0.944]

Rule 7: (1496/109, lift 1.1)
        Intl_Plan = no
        Day_Mins > 162.7
        Day_Mins <= 263.4
        Eve_Charge > 11.54
        ->  class False.  [0.927]

Rule 8: (228/16, lift 1.1)
        Day_Mins > 144.6
        Day_Mins <= 162.7
        Eve_Mins > 193.1
        ->  class False.  [0.926]

Rule 9: (50, lift 7.2)
        Intl_Plan = yes
        Intl_Mins > 13
        ->  class True.  [0.981]

Rule 10: (49, lift 7.2)
        Intl_Plan = yes
        Intl_Calls <= 2
        ->  class True.  [0.980]

Rule 11: (95/4, lift 6.9)
        Vmail_Plan = no
        Day_Mins > 249.5
        Eve_Charge > 15.78
        Night_Charge > 7.75
        CustServ_Calls <= 3
        ->  class True.  [0.948]

Rule 12: (70/3, lift 6.9)
        Vmail_Plan = no
        Day_Mins > 221.8
        Eve_Mins > 242.5
        Night_Mins > 177.1
        ->  class True.  [0.944]

Rule 13: (9, lift 6.7)
        Intl_Plan = yes
        CustServ_Calls > 4
        ->  class True.  [0.909]

Rule 14: (248/96, lift 4.5)
        Vmail_Plan = no
        Day_Mins > 249.5
        ->  class True.  [0.612]

Rule 15: (233/114, lift 3.7)
        CustServ_Calls > 3
        ->  class True.  [0.511]

Default class: False.


Evaluation on training data (3250 cases):

                Rules
          ----------------
            No      Errors

            15   131( 4.0%)   <<


           (a)   (b)    <-classified as
          ----  ----
          2772    34    (a): class False.
            97   347    (b): class True.
```

Figure 21: C50 Rule set



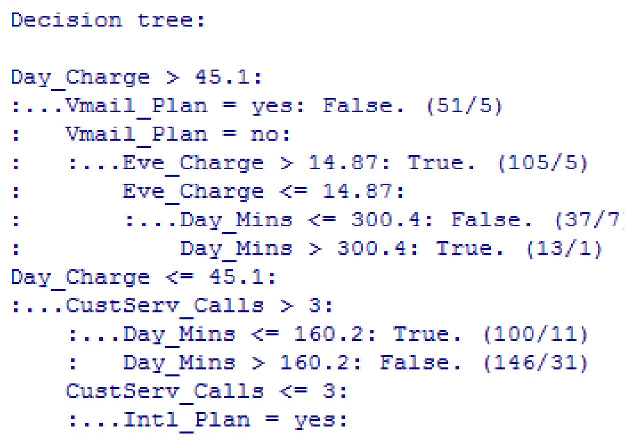
```
Evaluation on training data (3250 cases):

                Rules
          ----------------
            No      Errors

            15  131( 4.0%)   <<


           (a)   (b)    <-classified as
          ----  ----
          2772    34    (a): class False.
            97   347    (b): class True.


        Attribute usage:

         95.35% CustServ_Calls
         94.34% Day_Mins
         91.20% Intl_Plan
         70.58% Eve_Charge
         67.72% Eve_Mins
         41.97% Night_Charge
         38.31% Vmail_Plan
          6.03% Intl_Mins
          6.00% Intl_Calls
          2.15% Night_Mins


Evaluation on test data (1750 cases):

                Rules
          ----------------
            No      Errors

            15  108( 6.2%)   <<


           (a)   (b)    <-classified as
          ----  ----
          1449    38    (a): class False.
            70   193    (b): class True.


Time: 0.2 secs
```

Figure 21: C50 Rule set continued

After applying the C5.0 algorithm to the churn dataset, two types of models were produced:

a) The Tree Model

The model was trained with 65% of the dataset and tested on 35% of the dataset and evaluated to perform well on new data although there is a slight difference in accuracy between the evaluation on the training data and the evaluation on the test data. Interpretation of the tree is carried out as follows:



```
Decision tree:

Day_Charge > 45.1:
:...Vmail_Plan = yes: False. (51/5)
:   Vmail_Plan = no:
:   :...Eve_Charge > 14.87: True. (105/5)
:       Eve_Charge <= 14.87:
:       :...Day_Mins <= 300.4: False. (37/7
:           Day_Mins > 300.4: True. (13/1)
Day_Charge <= 45.1:
:...CustServ_Calls > 3:
    :...Day_Mins <= 160.2: True. (100/11)
    :   Day_Mins > 160.2: False. (146/31)
    CustServ_Calls <= 3:
    :...Intl_Plan = yes:
```

As for the other classification tree types above, just one part of the full tree will be interpreted for illustration. Surprising, compared to the C4.5 tree(with first split on Days_Mins), the C5.0 tree has top node splitted on Day_Charge. The numbers in bracket at the end of each leaf(e.g 51,5) indicates the number of instances the fall in that leaf(first number) and the number of misclassified cases(second number).

The above tree tell us that:

- If a customers 'Day_ Charge' is greater than 45.1 and he has a voicemail plan(Vmail_plan=yes) , then the customer is unlikely to churn. In this case 51 instances were classified whereby 5 of them were misclassified.

- If a customers 'Day_ Charge' is greater than 45.1 and he has no voicemail plan(Vmail_plan=no), the customer is likely to churn if 'Eve_Charge' is greater than 14.87. In this case 105 instances were classified with 5 cases of misclassification.

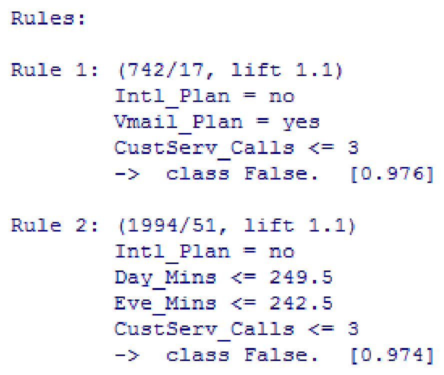
-If a customers 'Day_ Charge' is greater than 45.1 and he has no voicemail plan(Vmail_plan=no), the customer is unlikely to churn if 'Eve_Charge' is less than or equal to 14.87 and total day minutes is less than or equal to 300.4 minutes.

-If a customers 'Day_ Charge' is greater than 45.1 and he has no voicemail plan(Vmail_plan=no), the customer is likely to churn if 'Eve_Charge' is less than or equal to 14.87 and total day minutes is greater 300.4 minutes.

The same method of interpretion is applicable to the rest of the tree.

b) The Rule Model

The rule model was trained with 65% of the dataset and tested on 35% of the dataset and evaluated to perform well on new data although there is a slight difference in accuracy between the evaluation on the training data and the evaluation on the test data. Interpretation of the first two rules will be carried illustrated.



```
Rules:

Rule 1: (742/17, lift 1.1)
        Intl_Plan = no
        Vmail_Plan = yes
        CustServ_Calls <= 3
        ->  class False.  [0.976]

Rule 2: (1994/51, lift 1.1)
        Intl_Plan = no
        Day_Mins <= 249.5
        Eve_Mins <= 242.5
        CustServ_Calls <= 3
        ->  class False.  [0.974]
```

As seen above, each rule is consists of an arbitrary rule number, some statistics about the rule given usually as (n/m, lift x) which gives a summary of the performance of each rule, the conditions for the rule to be satisfied, the class to which the rule predicts and the confidence with which the prediction is made.

Rule 1: If a customer has no international plan(Intl_Plan=no) but has a voicemail plan (Vmail_Plan=yes) and the customer's total customer service calls(CustServ_Calls) is less than or equal to three(3), the customer is unlikely to churn. In this rule, 742 instances were classified with 17 cases wrongly classified. Lift 1.1 indicates that dividing the estimated accuracy of rule 1(0.976) by the relative frequency of the the class 'False' in the training set is equal to 1.1 and the confidence with which the prediction of rule 1 is made is 97.6%.

Rule 2: If a customer has no international plan(Intl_Plan=no) , his 'Day_Mins' is less than or equal to 249.5, his 'Eve_Mins' is less than or equal to 242.5 and the customer's total customer service calls(CustServ_Calls) is less than or equal to three(3), the customer is unlikely to churn. In this rule, 1994 instances were classified with 51 cases wrongly classified. Lift 1.1 indicates that dividing the estimated accuracy of rule 1(0.976) by the relative frequency of the the class 'False' in the training set is equal to 1.1 and the confidence with which the prediction of rule 1 is made is 97.4%.

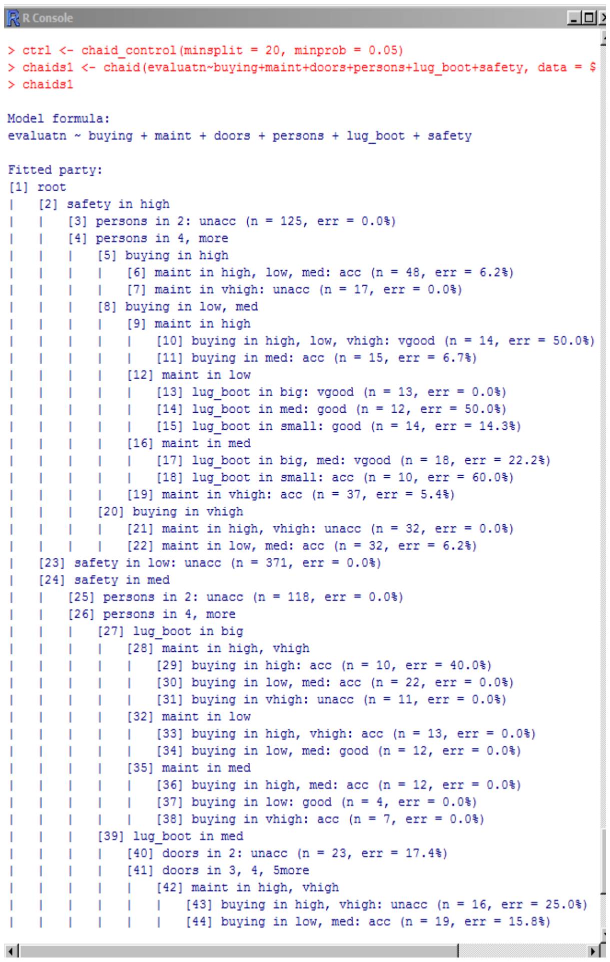
3.3 CHAID Analysis in R

The chaid analysis will be illustrated using the car dataset was used previously to implement the j48 algorithm using rweka. UCI list of datasets(http://archive.ics.uci.edu/ml/machine-learning-databases/car/). The dataset consists of 1718 instances and 6 attributes. These variables were collected based on price, technical characteristics and comfort.

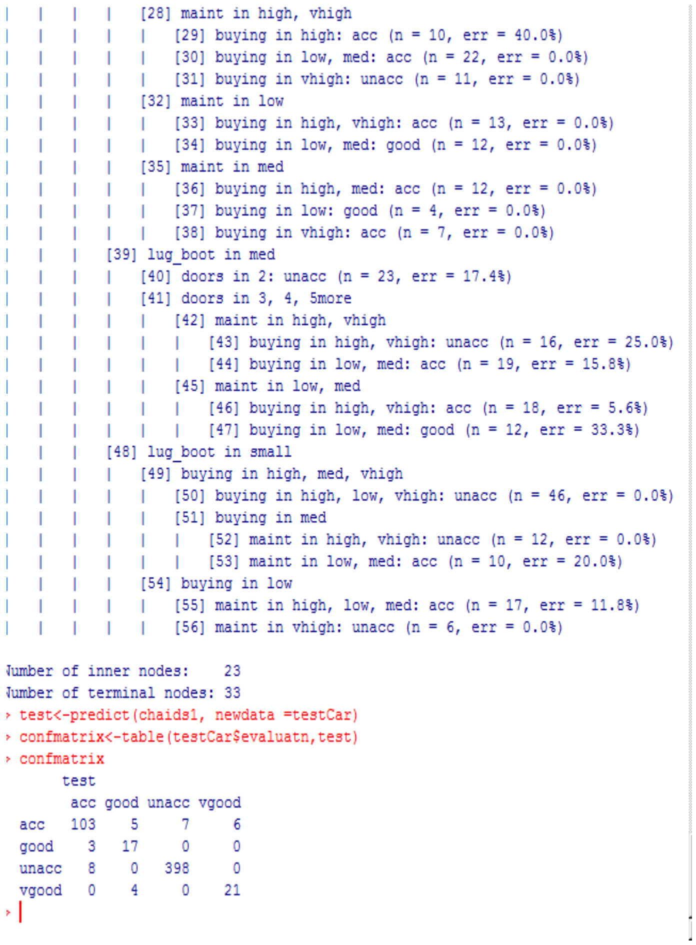
The table below shows the results of the chaid analysis using the car dataset. The resulting tree has 43 terminal nodes and 30 inner nodes. Due to the presence of multiple splits the R-programming software presents the tree in the form of rules due to its incapability to make proper tree diagrams for trees which have more than binary splits. However, interfacing other programs with R may be produce the tree diagram. In the next chapter, an interpretation of part of the tree will be made in order to illustrate how trees presented in form can be interpreted.

```
Instal command

install.packages("CHAID", repos="http://R-Forge.R-project.org")

chaids <- chaid(evaluatn~buying+maint+doors+persons+lug_boot+safety, data=trainCar)

pred<-predict(chaids, newdata =testCar)

print(chaids)

ctrl <- chaid_control(minsplit = 20, minprob = 0.05) # for pruning

chaids1 <- chaid(evaluatn~buying+maint+doors+persons+lug_boot+safety, data =
trainCar, control=ctrl)

test<-predict(chaids1, newdata =testCar)

confmatrix<-table(testCar$evaluatn,test)

confmatrix
```

Figure 22: Chaid analysis on Car Data



```
> ctrl <- chaid_control(minsplit = 20, minprob = 0.05)
> chaids1 <- chaid(evaluatn~buying+maint+doors+persons+lug_boot+safety, data = $
> chaids1

Model formula:
evaluatn ~ buying + maint + doors + persons + lug_boot + safety

Fitted party:
[1] root
|   [2] safety in high
|   |   [3] persons in 2: unacc (n = 125, err = 0.0%)
|   |   [4] persons in 4, more
|   |   |   [5] buying in high
|   |   |   |   [6] maint in high, low, med: acc (n = 48, err = 6.2%)
|   |   |   |   [7] maint in vhigh: unacc (n = 17, err = 0.0%)
|   |   |   [8] buying in low, med
|   |   |   |   [9] maint in high
|   |   |   |   |   [10] buying in high, low, vhigh: vgood (n = 14, err = 50.0%)
|   |   |   |   |   [11] buying in med: acc (n = 15, err = 6.7%)
|   |   |   |   [12] maint in low
|   |   |   |   |   [13] lug_boot in big: vgood (n = 13, err = 0.0%)
|   |   |   |   |   [14] lug_boot in med: good (n = 12, err = 50.0%)
|   |   |   |   |   [15] lug_boot in small: good (n = 14, err = 14.3%)
|   |   |   |   [16] maint in med
|   |   |   |   |   [17] lug_boot in big, med: vgood (n = 18, err = 22.2%)
|   |   |   |   |   [18] lug_boot in small: acc (n = 10, err = 60.0%)
|   |   |   |   [19] maint in vhigh: acc (n = 37, err = 5.4%)
|   |   |   [20] buying in vhigh
|   |   |   |   [21] maint in high, vhigh: unacc (n = 32, err = 0.0%)
|   |   |   |   [22] maint in low, med: acc (n = 32, err = 6.2%)
|   [23] safety in low: unacc (n = 371, err = 0.0%)
|   [24] safety in med
|   |   [25] persons in 2: unacc (n = 118, err = 0.0%)
|   |   [26] persons in 4, more
|   |   |   [27] lug_boot in big
|   |   |   |   [28] maint in high, vhigh
|   |   |   |   |   [29] buying in high: acc (n = 10, err = 40.0%)
|   |   |   |   |   [30] buying in low, med: acc (n = 22, err = 0.0%)
|   |   |   |   |   [31] buying in vhigh: unacc (n = 11, err = 0.0%)
|   |   |   |   [32] maint in low
|   |   |   |   |   [33] buying in high, vhigh: acc (n = 13, err = 0.0%)
|   |   |   |   |   [34] buying in low, med: good (n = 12, err = 0.0%)
|   |   |   |   [35] maint in med
|   |   |   |   |   [36] buying in high, med: acc (n = 12, err = 0.0%)
|   |   |   |   |   [37] buying in low: good (n = 4, err = 0.0%)
|   |   |   |   |   [38] buying in vhigh: acc (n = 7, err = 0.0%)
|   |   |   [39] lug_boot in med
|   |   |   |   [40] doors in 2: unacc (n = 23, err = 17.4%)
|   |   |   |   [41] doors in 3, 4, 5more
|   |   |   |   |   [42] maint in high, vhigh
|   |   |   |   |   |   [43] buying in high, vhigh: unacc (n = 16, err = 25.0%)
|   |   |   |   |   |   [44] buying in low, med: acc (n = 19, err = 15.8%)
```



```
|   |   |   |   [28] maint in high, vhigh
|   |   |   |   |   [29] buying in high: acc (n = 10, err = 40.0%)
|   |   |   |   |   [30] buying in low, med: acc (n = 22, err = 0.0%)
|   |   |   |   |   [31] buying in vhigh: unacc (n = 11, err = 0.0%)
|   |   |   |   [32] maint in low
|   |   |   |   |   [33] buying in high, vhigh: acc (n = 13, err = 0.0%)
|   |   |   |   |   [34] buying in low, med: good (n = 12, err = 0.0%)
|   |   |   |   [35] maint in med
|   |   |   |   |   [36] buying in high, med: acc (n = 12, err = 0.0%)
|   |   |   |   |   [37] buying in low: good (n = 4, err = 0.0%)
|   |   |   |   |   [38] buying in vhigh: acc (n = 7, err = 0.0%)
|   |   |   [39] lug_boot in med
|   |   |   |   [40] doors in 2: unacc (n = 23, err = 17.4%)
|   |   |   |   [41] doors in 3, 4, 5more
|   |   |   |   |   [42] maint in high, vhigh
|   |   |   |   |   |   [43] buying in high, vhigh: unacc (n = 16, err = 25.0%)
|   |   |   |   |   |   [44] buying in low, med: acc (n = 19, err = 15.8%)
|   |   |   |   |   [45] maint in low, med
|   |   |   |   |   |   [46] buying in high, vhigh: acc (n = 18, err = 5.6%)
|   |   |   |   |   |   [47] buying in low, med: good (n = 12, err = 33.3%)
|   |   |   [48] lug_boot in small
|   |   |   |   [49] buying in high, med, vhigh
|   |   |   |   |   [50] buying in high, low, vhigh: unacc (n = 46, err = 0.0%)
|   |   |   |   |   [51] buying in med
|   |   |   |   |   |   [52] maint in high, vhigh: unacc (n = 12, err = 0.0%)
|   |   |   |   |   |   [53] maint in low, med: acc (n = 10, err = 20.0%)
|   |   |   |   [54] buying in low
|   |   |   |   |   [55] maint in high, low, med: acc (n = 17, err = 11.8%)
|   |   |   |   |   [56] maint in vhigh: unacc (n = 6, err = 0.0%)

Number of inner nodes:    23
Number of terminal nodes: 33
> test<-predict(chaids1, newdata =testCar)
> confmatrix<-table(testCar$evaluatn,test)
> confmatrix
       test
        acc good unacc vgood
  acc   103    5     7     6
  good    3   17     0     0
  unacc   8    0   398     0
  vgood   0    4     0    21
>
```

From figure 22 above the percentage of correctly classified instances is 94.23% with a resulting tree size of 33 leaves after pruning have been applied using the chaid control. Further pruning was tried to produce a smaller tree size which resulted in one of the classes(‘good’) of the response not classified in the tree. Thus care should be taken when pruning a tree. Note should be taken in this case since this is just an illustration. The different categories of the response variable in the dataset should have equal number of observations in order to get a good classification tree model using chaid analysis.

From the CHAID analysis of the car dataset which was also used in J48 analysis in Rweka, the final tree was seen to have 30 inner nodes and 43 terminal nodes(or leaves) . This large numbers are as a result of multiple splits by the chaid algorithm. Chaid analysis can also be used for exploratory purpose. This package is available for download in r-forge and not present in the R-cran list of packages. An ROC curve was attempted but the package only supports evaluation of binary classification tasks.

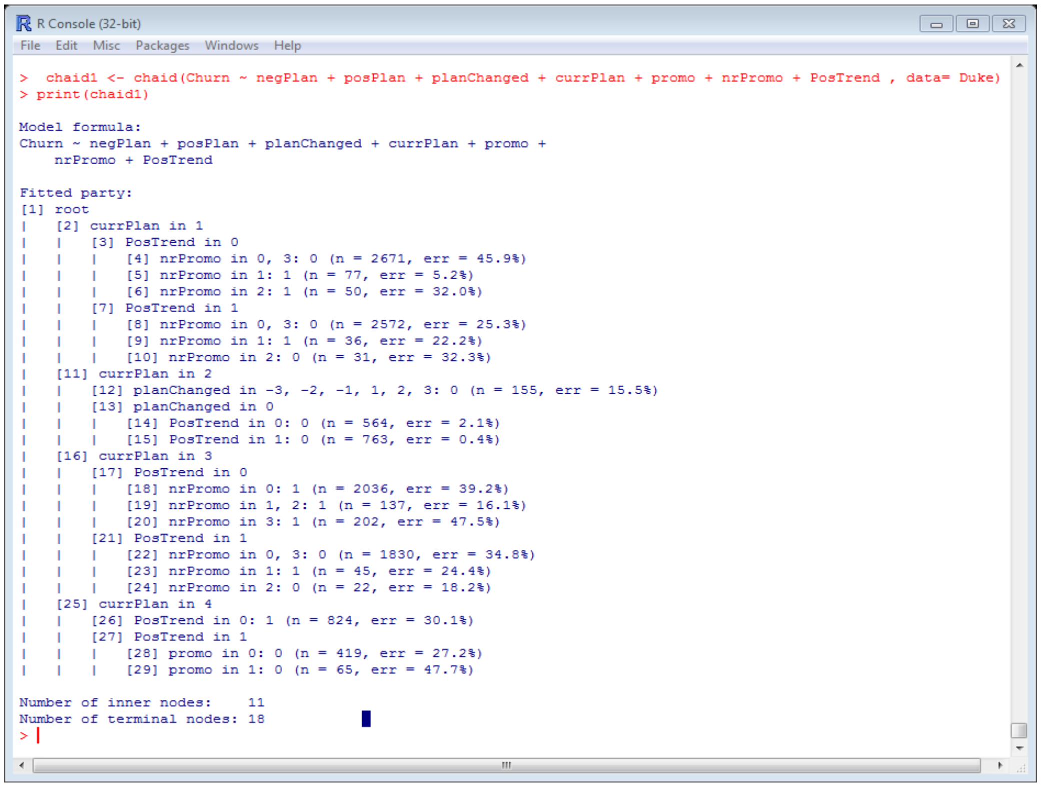
The resulting tree had a re-substitution error of 0.07867133. Accuracy of the splits in this case can also be seen at the level of the errors associated with each leaf. At each leaf node, the number of instances and the corresponding error rates are reported.

For illustration purposes, a branch of the tree in figure 22 can be interpreted as follows:

a) If the car "safety" is 'high' and the capacity of "persons" is two(2), then the car is unacceptable. There were 189 correctly classified instances i.e zero error.
b) Otherwise from (a), if the capacity of persons is four(4) or more, and "buying" price is high, then we check the maintenance price "maint". If maintenance price is high, low or medium, then the car is acceptable.
c) Otherwise from (b), if the maintenance price is very high, then the car is unacceptable.

This process is continued to until the full tree has been interpreted using a top-down approach.

The Churn dataset, used for cart analysis above with only the categorical variables, we get a tree structure as seen below. This is used just for an illustration purpose. Inorder to produce a tree with all the predictors, all the other non-categorical variables will have to be transformed into ordinal variables.



```
>  chaid1 <- chaid(Churn ~ negPlan + posPlan + planChanged + currPlan + promo + nrPromo + PosTrend , data= Duke)
> print(chaid1)

Model formula:
Churn ~ negPlan + posPlan + planChanged + currPlan + promo +
    nrPromo + PosTrend

Fitted party:
[1] root
|   [2] currPlan in 1
|   |   [3] PosTrend in 0
|   |   |   [4] nrPromo in 0, 3: 0 (n = 2671, err = 45.9%)
|   |   |   [5] nrPromo in 1: 1 (n = 77, err = 5.2%)
|   |   |   [6] nrPromo in 2: 1 (n = 50, err = 32.0%)
|   |   [7] PosTrend in 1
|   |   |   [8] nrPromo in 0, 3: 0 (n = 2572, err = 25.3%)
|   |   |   [9] nrPromo in 1: 1 (n = 36, err = 22.2%)
|   |   |   [10] nrPromo in 2: 0 (n = 31, err = 32.3%)
|   [11] currPlan in 2
|   |   [12] planChanged in -3, -2, -1, 1, 2, 3: 0 (n = 155, err = 15.5%)
|   |   [13] planChanged in 0
|   |   |   [14] PosTrend in 0: 0 (n = 564, err = 2.1%)
|   |   |   [15] PosTrend in 1: 0 (n = 763, err = 0.4%)
|   [16] currPlan in 3
|   |   [17] PosTrend in 0
|   |   |   [18] nrPromo in 0: 1 (n = 2036, err = 39.2%)
|   |   |   [19] nrPromo in 1, 2: 1 (n = 137, err = 16.1%)
|   |   |   [20] nrPromo in 3: 1 (n = 202, err = 47.5%)
|   |   [21] PosTrend in 1
|   |   |   [22] nrPromo in 0, 3: 0 (n = 1830, err = 34.8%)
|   |   |   [23] nrPromo in 1: 1 (n = 45, err = 24.4%)
|   |   |   [24] nrPromo in 2: 0 (n = 22, err = 18.2%)
|   [25] currPlan in 4
|   |   [26] PosTrend in 0: 1 (n = 824, err = 30.1%)
|   |   [27] PosTrend in 1
|   |   |   [28] promo in 0: 0 (n = 419, err = 27.2%)
|   |   |   [29] promo in 1: 0 (n = 65, err = 47.7%)

Number of inner nodes:    11
Number of terminal nodes: 18
> 
```

Figure 23: Chaid analysis on Churn data with selected variables.

Confusion Matrix					
	acc	good	unacc	vgood	Accuracy= 92.13%
acc	105	5	10	1	
good	2	19	0	0	
unacc	10	0	391	0	
vgood	7	10	0	12	

Table 6

Table 7:

R – PACKAGE	Algorithm	Data	Accuracy: Correctly classified Instances
CHAID	CHAID	Car	**94.23**
		Churn	**92.13%**
RPART	CART	Churn	**93.53%**
RPART	CART	Boston	Adj R Sq=0.77
rpartScore	CART	NLSY	N/A
RWEKA	C4.5	Churn	**89.61%**
C50	C50	Churn	**96% on rules**
C50	C5.0	Churn	**94.6% on tree**

The C50 classification algorithm based on the churn dataset is the best classifier with its rule model a better classifier than the its tree model and all other tree models discussed with the hurn dataset.The C5.0 tree model is also the best tree model among all models in which the churn dataset was used followed the chaid algorithm. Note should be taken as it is important o try different models for a dataset when using tree models and the best model can be chosen based on its classification accuracy.

Chapter 4

Discussion and Recommendations

Implementing classification tree algorithms in the R –programming software could be an interesting or a hectic process. Depending on the kind of dataset to be analyzed and classification tree algorithm to be implemented, there exist possible packages in R which can be used to grow , prune and evaluate a tree. This paper has illustrated the implementation of three classification tree algorithms namely: CART, C4.5 and CHAID analysis using the Rpart, Rweka and chaid packages in R. Using the Rpart package for CART analysis, an implementation was carried out using different datasets with binary response variable for classification trees, ordinal response variable for ordinal trees(alongside the rpartOrdinal and rpartScore) and continuous response data for regression trees. The rweka package for C4.5 analysis was illustrated using binary and categorical response variables while the chaid package for chaid analysis was illustrated using the categorical response data.

The implementation of CART in R is quite extensive with a lot of possibilities. It also has an advantage in R due to the fact that it presents the best user interface as all the trees it produces can be plotted in R. The R programming software can only plot trees with binary splits otherwise, it produces the tree in the form of rules which sometimes can be more difficult to interprete. Notwithstanding, whether binary splits or multiple splits are used in a tree, we are able to get a tree. C4.5 also works well in R since it can also produce trees with binary splits in which case a plot of the tree can given. Although CHAID and RWEKA packages may result in multiple splits, an illustration has been given show how the results can be interpreted whether in the form of a tree or in the form of rules. C50 was seen to have the best results on churn data as seen in table 7. The algorithm is quite fast to execute in R. The rpartScore package which implements the generalized gini split, takes more time to train a model than the other algorithms.

When carrying out classification tree analysis it is important to note that although we want to grow a tree which is easy to understand for the user. We also want to know if the rules drawn from the tree are interpretable and comprehensive. Overfitting and underfitting of the data should be avoided. As seen in Chapter three(3) and Chapter four(4), a very large tree is difficult to interpret and is less comprehensive than a small tree. Also, as we try to downsize a large tree to make is comprehensive, we lose some information such that sometimes, a class of the response variable is not classified especially when they have least number of observations. When we talk about comprehensiveness, it does not mean that the small and interpretable pruned tree is the best tree.

On the otherhand, when we talk about accuracy, it does not mean that the very large unpruned more accurate tree is the best tree. The best tree however, depends on the user of the tree. For someone with a strong statistical background, a large complex tree could still be interpretable, comprehensive and ofcourse accurate. Meanwhile for someone with little or no statistical knowledge, a very small and less accurate, comprehensive tree is better. In this regards, it is right to say that there is trade-off between accuracy and comprehensibility which the user should consider when choosing the right tree.

Classification tree algorithms are interesting to use but one should be very keen to be sure that the right tree model is chosen.

Finally, it is worth mentioning that classification tree is a very attractive predictive or exploratory method but its use is not recommended over the traditional methods like discriminant analysis, cluster analysis etc, since when these traditional methods meet their distributional and theoretical assumptions, they are most preferred. Otherwise classification tree methods become very useful.

Bibliography

0) en.wikipedia.org/wiki/Classification_Tree
1) Data Mining: Concepts, Models and Technique; Florin Gorunescu, (2001). , http://books.google.com/books?id=yJvKY-sB6zkC&pg=PA158&dq=Classification+Tree+Techniques&hl=en&ei=aSHKTo2YBc_m-ga5k4g7&sa=X&oi=book_result&ct=result&resnum=5&ved=0CEYQ6AEwBA#v=onepage&q=Classification%20Tree%20Techniques&f=false
2) Principles of Data Mining; David Hand, Heikki Mannila, Padhraic Smyth (2001)

3) Breiman,Friedman,Olshen,Stone: Classification and Decision Trees Wadsworth, 1984
4) Quinlan,J.R.: C4.5: Programs for Machine Learning , Morgan Kauffman, 1993
5) Winston,P.H.: Artificial Intelligence, Third Edition ,Addison-Wesley, 1992
6) Hill, T. & Lewicki, P. (2007). STATISTICS: Methods and Applications. StatSoft, Tulsa, OK
7) CHAID and Earlier Supervised Tree Methods; July 2010, Gilbert Ritschard http://www.unige.ch/ses/metri/cahiers/2010_02.pdf
8) Giorgio Ingargiola, http://www.cis.temple.edu/~ingargio/cis587/readings/id3-c45.html
9) Gamberger+Smuc:2001, author="D.Gamberger and T. Smuc, year = "2001" ,title = "{DMS}DataMiningServer url="http://dms.irb.hr/ institution="Rudjer Boskovic Institute,Lab.forinformationsystems (http://dms.irb.hr/tutorial/tut_dtrees.php)
10) Data Mining Techniques in CRM Inside Customer Segmentation; Konstantinos Tsiptsis Antonios Chorianopoulos(2009)
11) Classification, Clustering and Data Analysis, k. Jajuga, A. Sokolowski, H.-H. Bock (2002)
12) R-forge: https://r-forge.r-project.org/R/?group_id=343

13) Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, Ian H. Witten (2009); The WEKA Data Mining Software: An Update; SIGKDD Explorations, Volume 11, Issue 1.
14) Frank, A. & Asuncion, A. (2010). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.
15) Galimberti G., Soffritti G., Di Maso M. 2012 Classification Trees for Ordinal Responses in **R**: The rpartScore Package. *Journal of Statistical Software*, **47**(10), 1-25. http://www.jstatsoft.org/v47/i10/
16) Data Mining: Concepts, Models, Methods, and Algorithms By Mehmed Kantardzic.
17) SimaFore: analytics made accessible http://www.simafore.com/blog/bid/62482/2-main-differences-between-classification-and-regression-trees
18) Dr Saed Sayad :
http://www.saedsayad.com/
http://chem-eng.utoronto.ca/~datamining/dmc/decision_tree_reg.htm
19) Discretization Techniques: A recent survey ;Sotiris Kotsiantis, Dimitris Kanellopoulos (2006)
20) Classification/Decision Trees (I), Jia Li,
http://sites.stat.psu.edu/~jiali/course/stat597e/notes2/trees.pdf

21) Agresti AA. Categorical Data Analysis. 2nd edition John Wiley & Sons; Hoboken, NJ: 2002.
22) Piccarreta R. Classification Trees for Ordinal Variables. Computational Statistics. 2008;23:407–427.
23) Therneau TM, Atkinson EJ. Technical Report 61. Mayo Clinic; Rochester: 1997. An Introduction to Recursive Partitioning Using the **rpart** Routine. Section of Biostatistics. URL http://www.mayo.edu/hsr/techrpt/61.pdf
24) rpartOrdinal: An R Package for Deriving a Classification Tree for Predicting an Ordinal Response, Kellie J. Archer 2010 http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2899711/
25) http://cran.r-project.org/web/packages/rpartOrdinal/vignettes/rpartOrdinal.pdf
26) Applying CHAID for logistic regression diagnostics and classi_cation accuracy improvement; Evgeny Antipov and Elena Pokryshevskaya 2009 http://mpra.ub.uni-muenchen.de/21499/1/MPRA_paper_21499.pdf
27) Package ‘C50’: Max Kuhn, Steve Weston, Nathan Coulter. C code for C5.0 by R. Quinlan(2013) http://cran.r-project.org/web/packages/C50/C50.pdf
28) Classification tree for Ordinal Response in R: The rpartScore package; Giuliano Galimberti, Gabriele So_ritti, Matteo Di Maso (2012) **http://www.jstatsoft.org/v47/i10/paper**
29) C5.0: An informal tutorial, **RULEQUEST RESEARCH 2013** http://www.rulequest.com/see5-unix.html#OTHER
30) Statistics: Methods and Applications : a Comprehensive Reference for Science, industry and data mining, By Thomas Hill, Pawel Lewicki 2006. http://books.google.be/books?id=TI2TGjeilMAC&pg=PA79&dq=CHAID&hl=en&sa=X&ei=D31GT_eSEIjC0QXBjYWMDg&sqi=2&ved=0CDIQ6AEwAA#v=onepage&q=CHAID&f=false
31) IBM info center: http://pic.dhe.ibm.com/infocenter/spssstat/v20r0m0/index.jsp?topic=%2Fcom.ibm.spss.statistics.help%2Ftree_growing_criteria.htm
32) R-Forge SCM Repository https://r-forge.r-project.org/scm/viewvc.php/pkg/tests/Examples/CHAID-Ex.Rout.save?view=markup&root=chaid
33) http://www.public.iastate.edu/~kkoehler/stat557/tree14p.pdf
34)

Appendix

Section A

Table A1: summary of Churn dataset

Item	Type	Missing	Min	Max	Mean	StdDev
State	Factor	0	NA	NA	NA	NA
Account Length	Integer	0	1	243	100.3	39.7
Area Code	Factor	0	NA	NA	NA	NA
Int'l Plan	Factor	0	NA	NA	NA	NA
VMail Plan	Factor	0	NA	NA	NA	NA
VMail Msg	Integer	0	0	52	7.8	13.5
Day Mins	Numeric	0	0	351.5	180.3	54
Days Calls	Integer	0	0	165	100	20
Days Charge	Numeric	0	0	59.76	30.65	9.2
Eve Mins	Numeric	0	0	363.7	200.6	50.6
Eve Calls	Integer	0	0	170	100.2	20
Eve Charge	Numeric	0	0	30.91	17	4.3
Night Mins	Numeric	0	0	395	200.4	50.5
Night Calls	Integer	0	0	175	99.92	20
Night Charge	Numeric	0	0	17.8	9	2.3
Intl Mins	Numeric	0	0	20	10.26	2.8
Intl Calls	Integer	0	0	20	4.4	2.5
Intl Charge	Numeric	0	0	5.4	2.8	0.7
CusServ Calls	Integer	0	0	9	1.6	1.3
Churn	Factor	0	NA	NA	NA	NA

Root node error: 469/3334 = 0.14067

Code: printcp(fit)

n= 3334

CP	nsplit	rel error	xerror	xstd
1: 0.098081	0	1.00000	1.00000	0.042805
2: 0.078891	1	0.90192	0.96162	0.042107
3: 0.071429	2	0.82303	0.81663	0.039258
4: 0.053305	4	0.68017	0.68657	0.036366
5: 0.040512	7	0.49041	0.49680	0.031389
6: 0.022033	8	0.44989	0.46269	0.030370
7: 0.010661	11	0.38380	0.46695	0.030500
8: 0.010000	13	0.36247	0.47335	0.030693

Table A2: Print of CP Table

ww	= total weeks worked 1979-1995
wage	= respondent's total wage and salary income for 1995
afqt	= percentile score on U.S. Armed Forces Qualifying Exam
yeared	= years of education completed by the respondent
med	= years of education completed by the respondent's mother
fed	= years of education completed by the respondent's father
female	= 1 if the respondent is female (=0 otherwise)
black	= 1 if the respondent is black (=0 otherwise)
hisp	= 1 if the respondent is Hispanic (=0 otherwise)
asian	= 1 if the respondent is Asian (=0 otherwise)
natamer	= 1 if the respondent is native American (=0 otherwise)

health	= 1 if the respondent has a health condition that interferes with his or her ability to work (=0 otherwise)
ne	= 1 if the respondent lived in the northeast in 1996 (=0 otherwise)
nortcent	= 1 if the respondent lived in the north central states in 1996 (=0 otherwise)
south	= 1 if the respondent lived in the south in 1996 (=0 otherwise)
rural	= 1 if the respondent lived in a rural area in 1996 (=0 otherwise)
aa	= 1 if the respondent's highest level of education is an AA or AS degree (=0 otherwise)
ba	= 1 if the respondent's highest level of education is a BA degree (=0 otherwise)
bs	= 1 if the respondent's highest level of education is a BS degree (=0 otherwise)
masters	= 1 if the respondent's highest level of education is an MA or MS degree (=0 otherwise)
phd	= 1 if the respondent's highest level of education is a PhD (or equivalent) degree (=0 otherwise)
profdeg	= 1 if the respondent's highest level of education is an MD, LLD, DDS (or equivalent) degree (=0 otherwise)

Table A3: NLSY definition of variables

http://www.oswego.edu/~kane/econometrics/nlsy.html

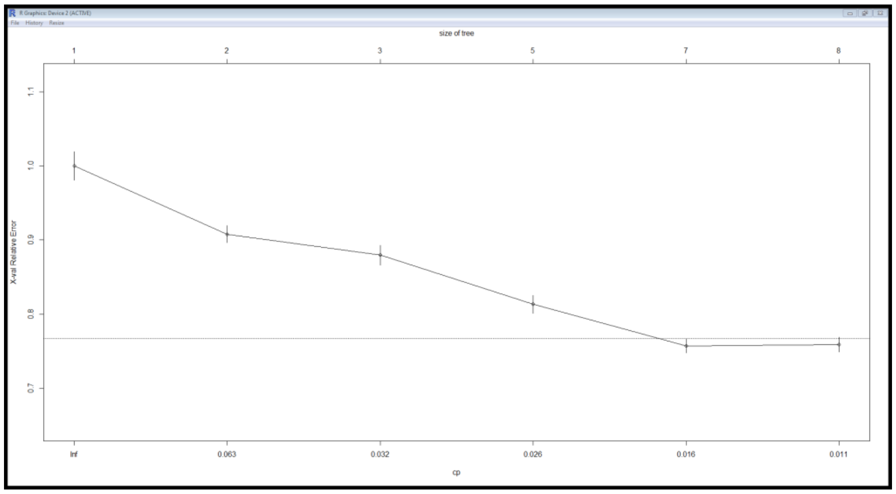


Figure A1: CP plot of fig 7

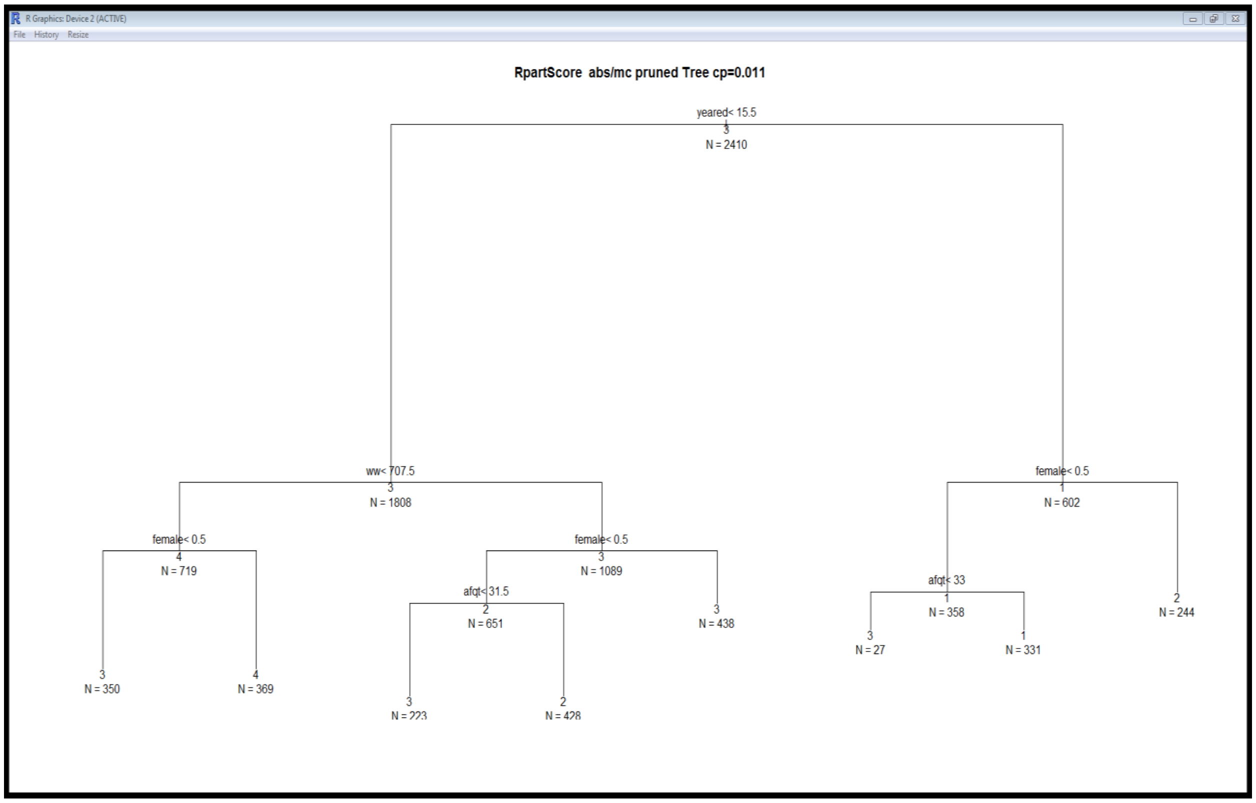


Figure A2: pruned tree of fig 7

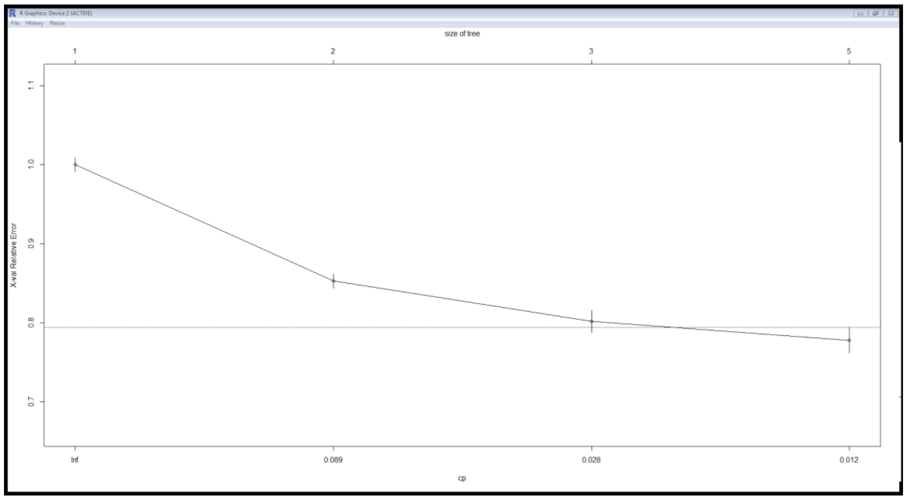


Figure A3: CP plot of fig 8

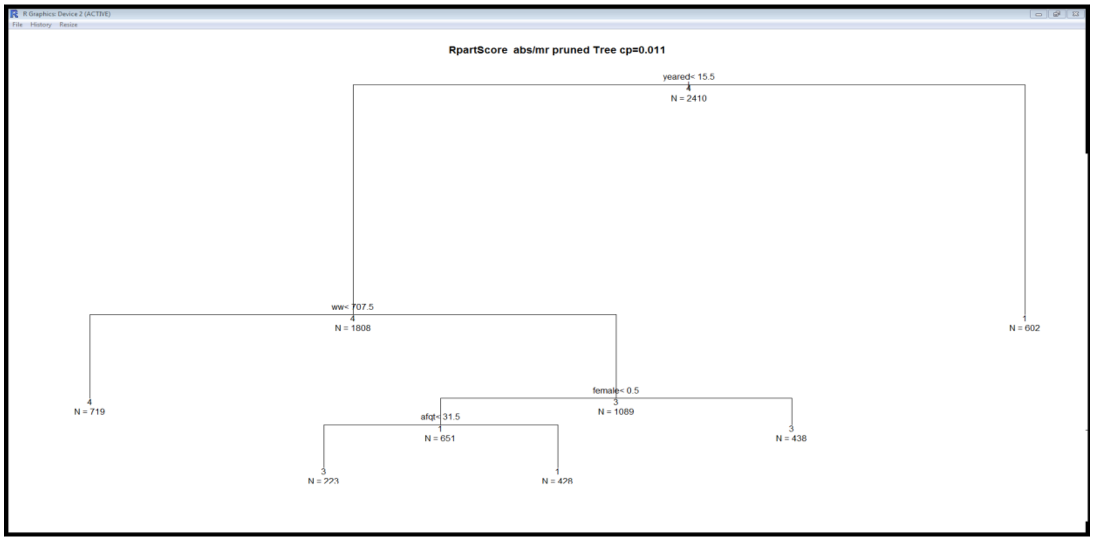


Figure A4: abs diff vs Miscl. Rate

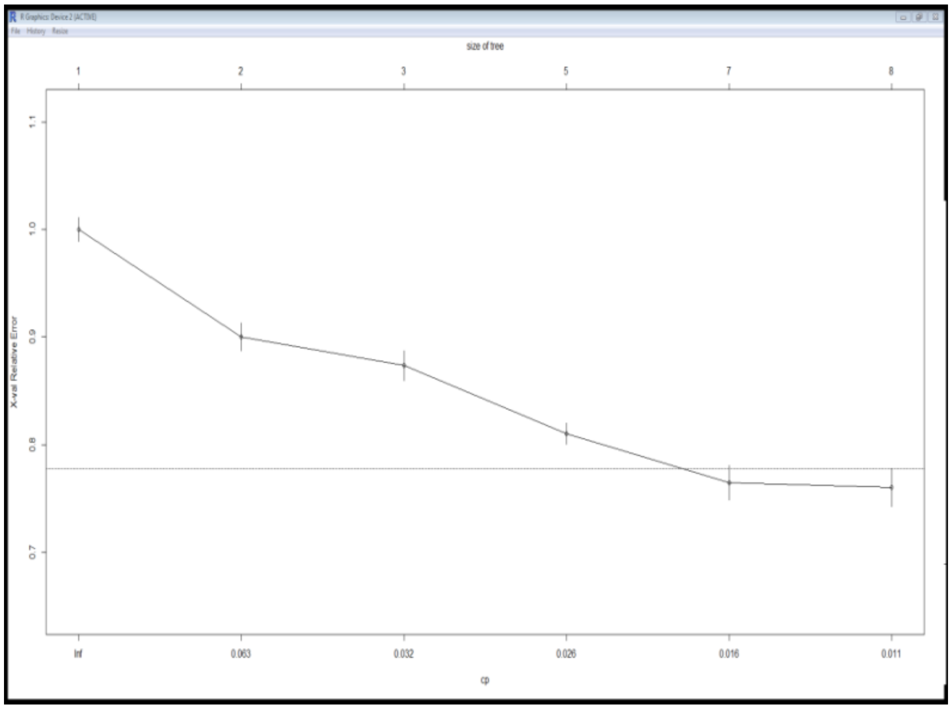


Figure A5: CP plot of fig 9

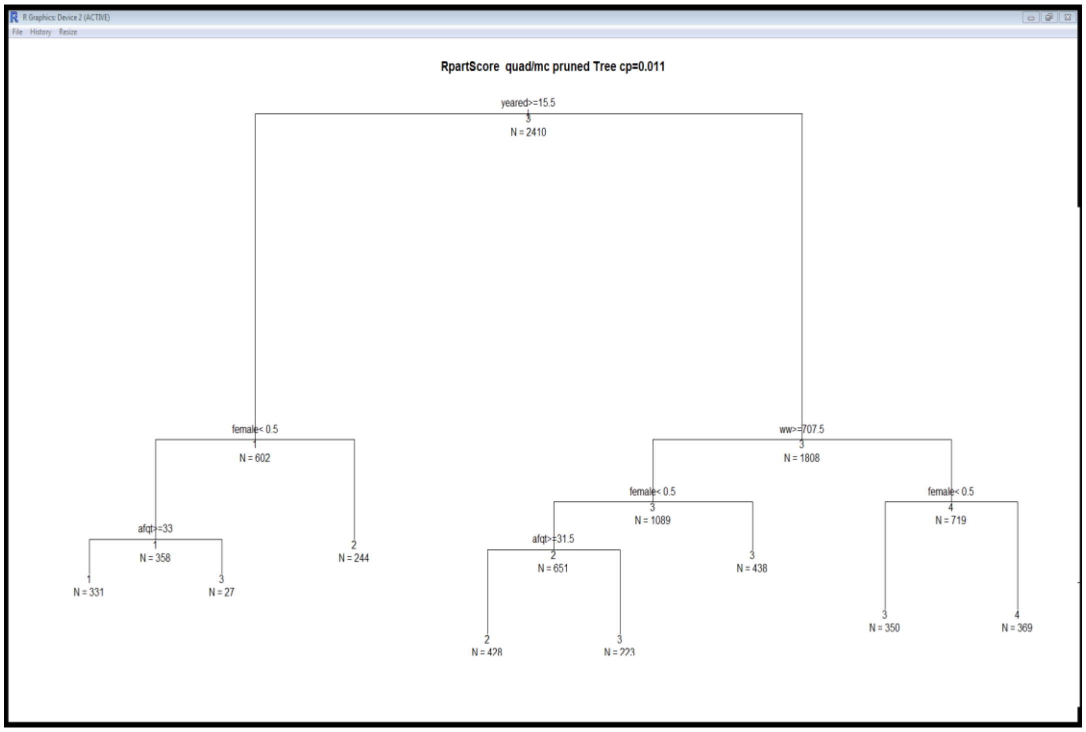


Figure A6: rpartScore pruned tree with quadratic differences of the score and misclassification cost with cp=0.011

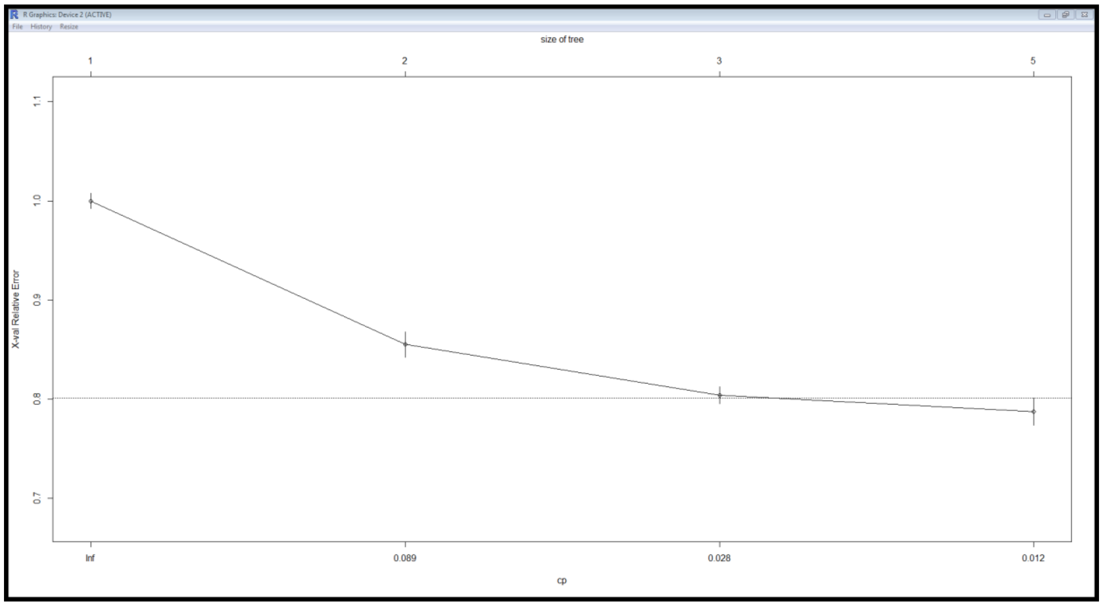


Figure A7: CP plot of fig 10

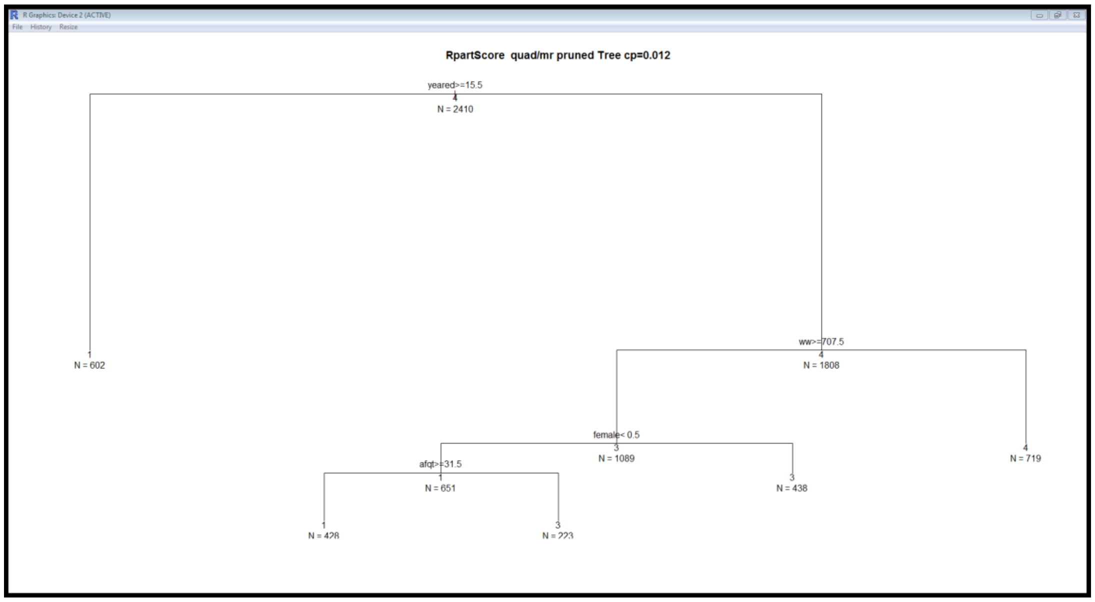


Figure A8: rpartScore pruned tree with quadratic differences of the score and misclassification rate with cp=0.012

Attribute Information:
1. CRIM: per capita crime rate by town
2. ZN: proportion of residential land zoned for lots over 25,000 sq.ft.
3. INDUS: proportion of non-retail business acres per town
4. CHAS: Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
5. NOX: nitric oxides concentration (parts per 10 million)
6. RM: average number of rooms per dwelling
7. AGE: proportion of owner-occupied units built prior to 1940
8. DIS: weighted distances to five Boston employment centres
9. RAD: index of accessibility to radial highways
10. TAX: full-value property-tax rate per $10,000
11. PTRATIO: pupil-teacher ratio by town
12. B: 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
13. LSTAT: % lower status of the population
14. MEDV: Median value of owner-occupied homes in $1000's

Table A4: Boston housing data definition of variables

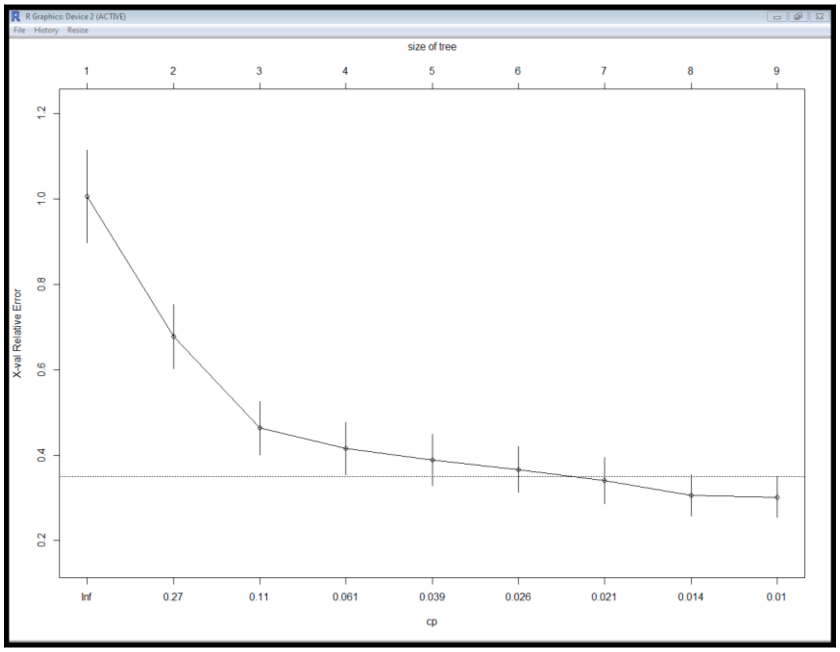


Figure A9: CP plot of regression tree

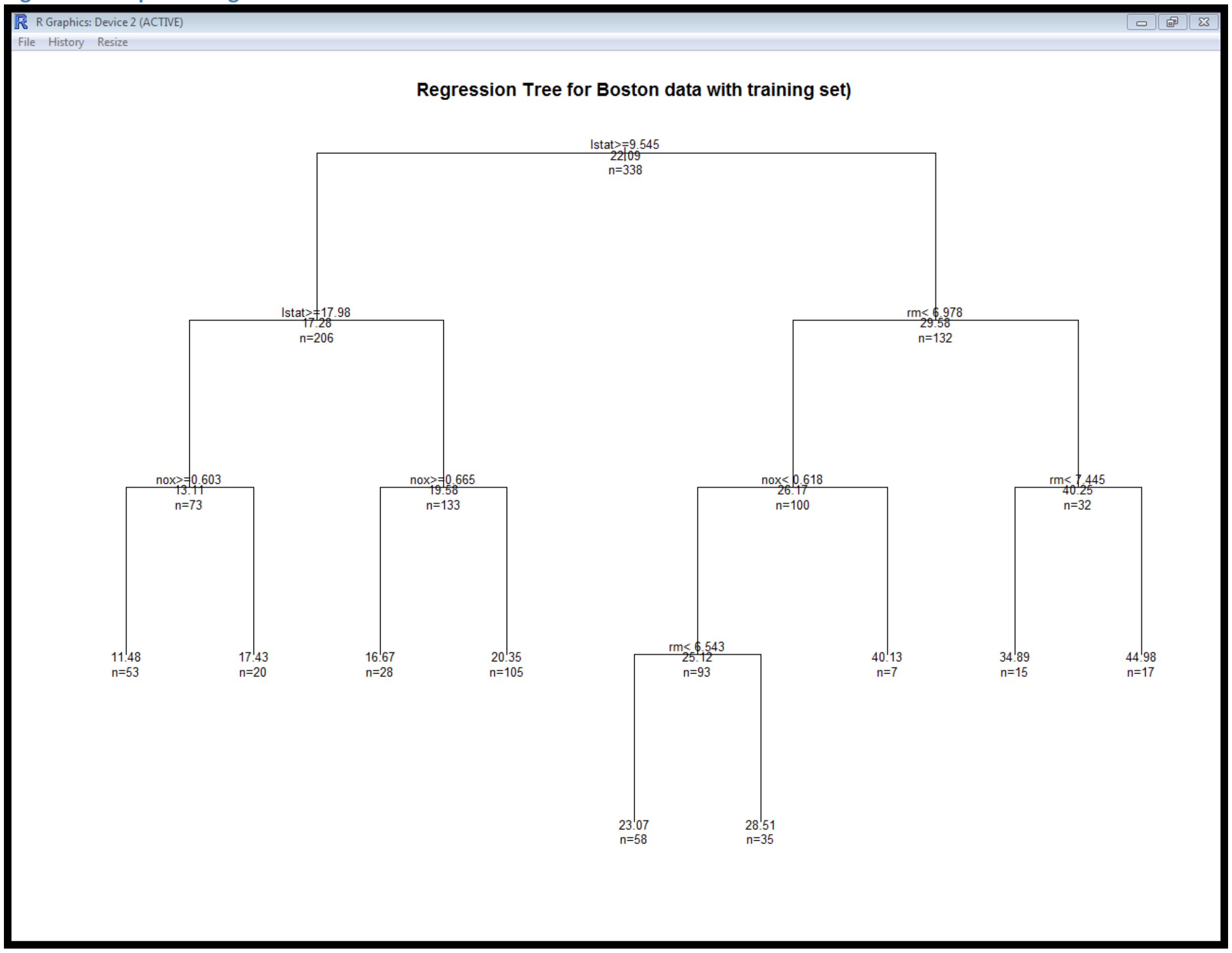


Figure A10: Regression tree using training set.

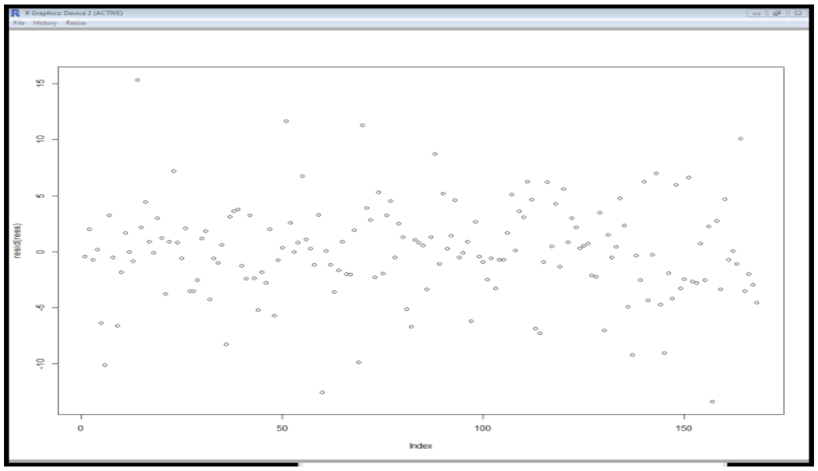


Figure A11: residual plot of observed vs predicted of the response

lm(formula = testb ~ testboston$medv)
Residuals:
Min 1Q Median 3Q Max
-13.4148 -2.3745 0.0338 2.3865 15.3620
Coefficients:
Estimate Std. Error t value Pr(>|t|)
(Intercept) 3.82377 0.89507 4.272 3.25e-05 ***
testboston$medv 0.83771 0.03548 23.611 < 2e-16 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error: 4.303 on 166 degrees of freedom
Multiple R-squared: 0.7706, Adjusted R-squared: 0.7692
F-statistic: 557.5 on 1 and 166 DF, p-value: < 2.2e-16

Table A5: results to test predicted vs observed

Codes:

Section B

Table B1: Description of the Car dataset 1

Attribute Information	
class values(evaluatn):	unacc, acc, good, vgood
buying:	vhigh, high, med, low.
maint:	vhigh, high, med, low.
doors:	2, 3, 4, 5more.
persons:	2, 4, more.

lug_boot:	small, med, big.
safety:	low, med, high.

Table B2: Summary statistics of C4.5(J48) pruned tree

=== Summary ===		
Correctly Classified Instances	936	81.6754 %
Incorrectly Classified Instances	210	18.3246 %
Kappa statistic	0.6213	
Mean absolute error	0.1247	
Root mean squared error	0.2497	
Relative absolute error	53.6729 %	
Root relative squared error	73.3401 %	
Coverage of cases (0.95 level)	100 %	
Mean rel. region size (0.95 level)	50.8726 %	
Total Number of Instances	1146	

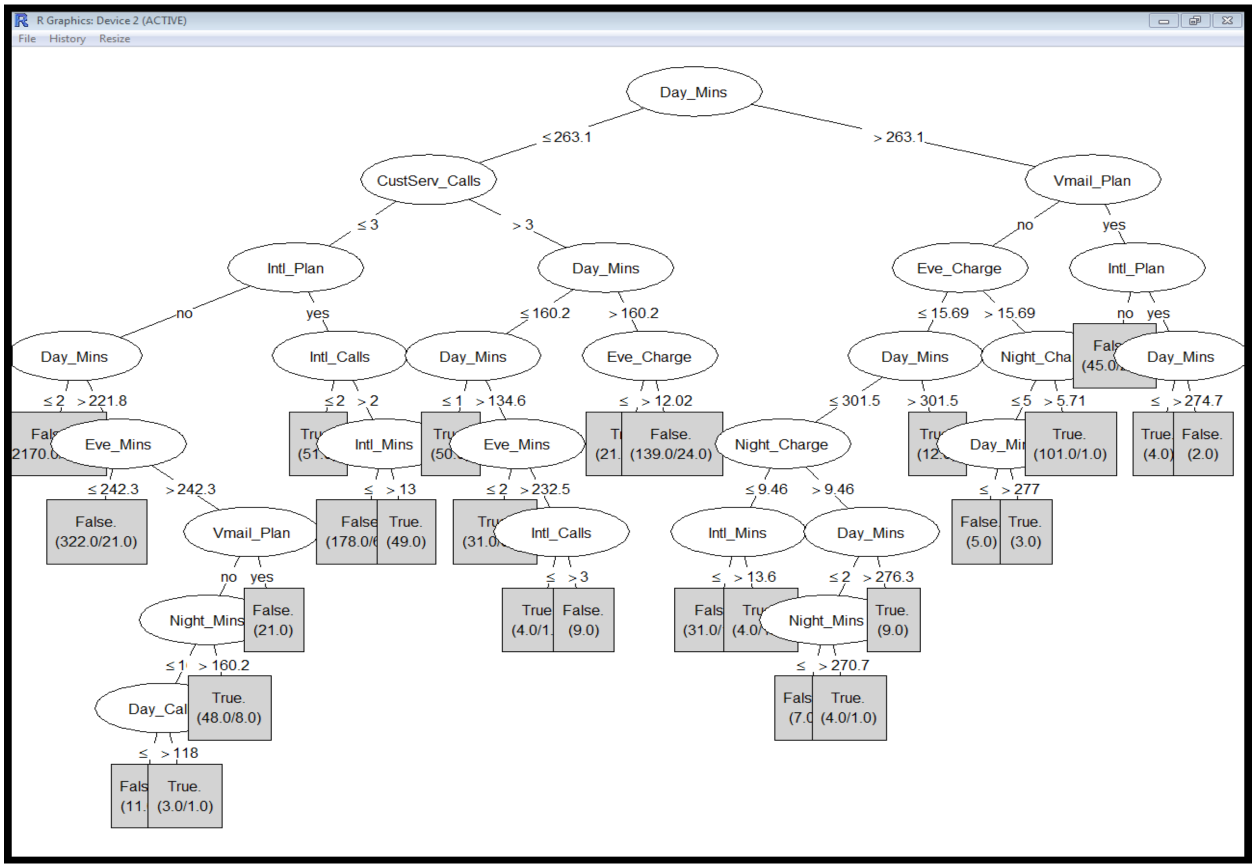
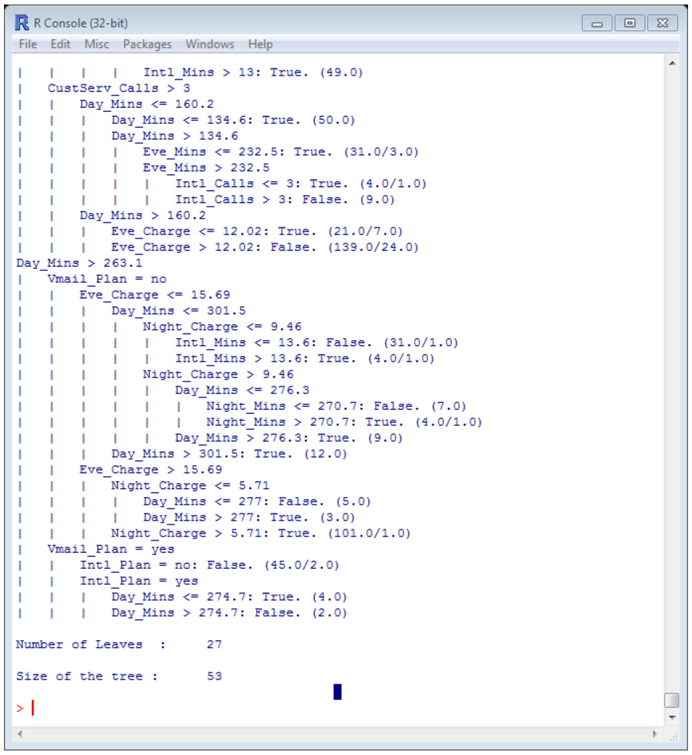


Figure B1: C4.5 for Churn data



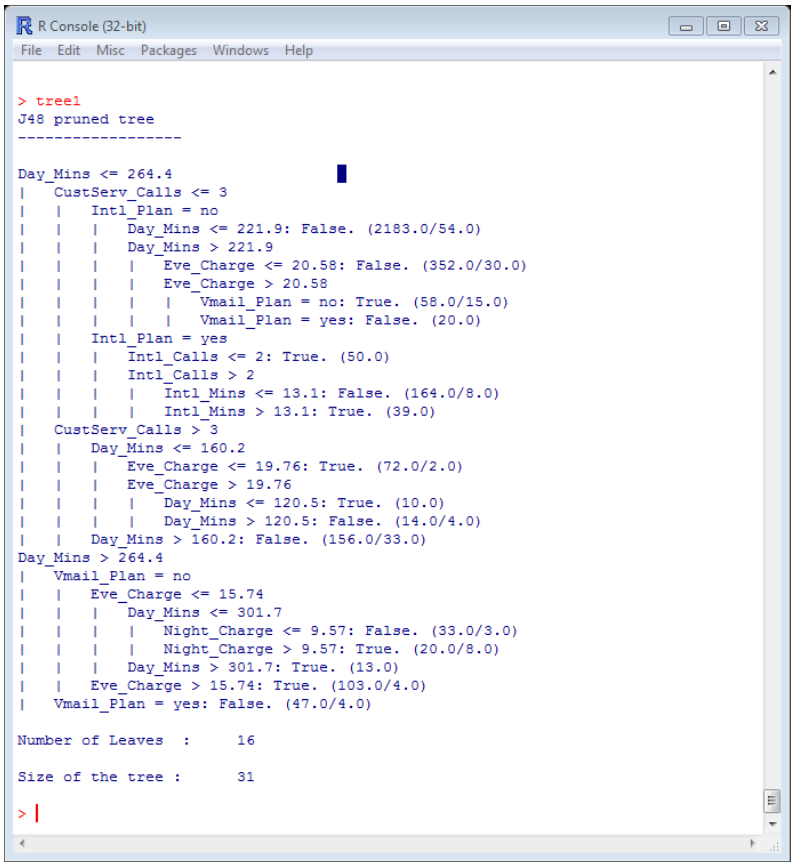
```
|   |   |   |   Intl_Mins > 13: True. (49.0)
|   CustServ_Calls > 3
|   |   Day_Mins <= 160.2
|   |   |   Day_Mins <= 134.6: True. (50.0)
|   |   |   Day_Mins > 134.6
|   |   |   |   Eve_Mins <= 232.5: True. (31.0/3.0)
|   |   |   |   Eve_Mins > 232.5
|   |   |   |   |   Intl_Calls <= 3: True. (4.0/1.0)
|   |   |   |   |   Intl_Calls > 3: False. (9.0)
|   |   Day_Mins > 160.2
|   |   |   Eve_Charge <= 12.02: True. (21.0/7.0)
|   |   |   Eve_Charge > 12.02: False. (139.0/24.0)
Day_Mins > 263.1
|   Vmail_Plan = no
|   |   Eve_Charge <= 15.69
|   |   |   Day_Mins <= 301.5
|   |   |   |   Night_Charge <= 9.46
|   |   |   |   |   Intl_Mins <= 13.6: False. (31.0/1.0)
|   |   |   |   |   Intl_Mins > 13.6: True. (4.0/1.0)
|   |   |   |   Night_Charge > 9.46
|   |   |   |   |   Day_Mins <= 276.3
|   |   |   |   |   |   Night_Mins <= 270.7: False. (7.0)
|   |   |   |   |   |   Night_Mins > 270.7: True. (4.0/1.0)
|   |   |   |   |   Day_Mins > 276.3: True. (9.0)
|   |   |   Day_Mins > 301.5: True. (12.0)
|   |   Eve_Charge > 15.69
|   |   |   Night_Charge <= 5.71
|   |   |   |   Day_Mins <= 277: False. (5.0)
|   |   |   |   Day_Mins > 277: True. (3.0)
|   |   |   Night_Charge > 5.71: True. (101.0/1.0)
|   Vmail_Plan = yes
|   |   Intl_Plan = no: False. (45.0/2.0)
|   |   Intl_Plan = yes
|   |   |   Day_Mins <= 274.7: True. (4.0)
|   |   |   Day_Mins > 274.7: False. (2.0)

Number of Leaves  :     27

Size of the tree :      53

> 
```

Figure B2: C4.5 pruned tree for Churn data

Evaluating RWeka classifier
=== Summary ===
Correctly Classified Instances   1493   89.6158 %
Incorrectly Classified Instances   173   10.3842 %
Kappa statistic   0.5286
K&B Relative Info Score   52508.4949 %
K&B Information Score   307.4335 bits   0.1845 bits/instance
Class complexity | order 0   972.7306 bits   0.5839 bits/instance
Class complexity | scheme   668.4986 bits   0.4013 bits/instance
Complexity improvement   (Sf)   304.2321 bits   0.1826 bits/instance
Mean absolute error   0.1478
Root mean squared error   0.2832
Relative absolute error   61.323 %
Root relative squared error   81.6379 %
Coverage of cases (0.95 level)   97.8992 %
Mean rel. region size (0.95 level)   69.7479 %
Total Number of Instances   1666
=== Detailed Accuracy By Class ===
TP Rate FP Rate Precision Recall F-Measure MCC ROC Area PRC Area Class
0,956 0,472 0,926 0,956 0,941 0,533 0,849 0,957 False.
0,528 0,044 0,661 0,528 0,587 0,533 0,849 0,602 True.
Weighted Avg. 0,896 0,412 0,889 0,896 0,891 0,533 0,849 0,908

=== Confusion Matrix ===
a b <-- classified as
1370 63 | a = False.
110 123 | b = True.

>



```
> tree1
J48 pruned tree
------------------

Day_Mins <= 264.4
|   CustServ_Calls <= 3
|   |   Intl_Plan = no
|   |   |   Day_Mins <= 221.9: False. (2183.0/54.0)
|   |   |   Day_Mins > 221.9
|   |   |   |   Eve_Charge <= 20.58: False. (352.0/30.0)
|   |   |   |   Eve_Charge > 20.58
|   |   |   |   |   Vmail_Plan = no: True. (58.0/15.0)
|   |   |   |   |   Vmail_Plan = yes: False. (20.0)
|   |   Intl_Plan = yes
|   |   |   Intl_Calls <= 2: True. (50.0)
|   |   |   Intl_Calls > 2
|   |   |   |   Intl_Mins <= 13.1: False. (164.0/8.0)
|   |   |   |   Intl_Mins > 13.1: True. (39.0)
|   CustServ_Calls > 3
|   |   Day_Mins <= 160.2
|   |   |   Eve_Charge <= 19.76: True. (72.0/2.0)
|   |   |   Eve_Charge > 19.76
|   |   |   |   Day_Mins <= 120.5: True. (10.0)
|   |   |   |   Day_Mins > 120.5: False. (14.0/4.0)
|   |   Day_Mins > 160.2: False. (156.0/33.0)
Day_Mins > 264.4
|   Vmail_Plan = no
|   |   Eve_Charge <= 15.74
|   |   |   Day_Mins <= 301.7
|   |   |   |   Night_Charge <= 9.57: False. (33.0/3.0)
|   |   |   |   Night_Charge > 9.57: True. (20.0/8.0)
|   |   |   Day_Mins > 301.7: True. (13.0)
|   |   Eve_Charge > 15.74: True. (103.0/4.0)
|   Vmail_Plan = yes: False. (47.0/4.0)

Number of Leaves  :     16

Size of the tree :      31

> 
```

Figure B1: C4.5 for Churn data

Table B2: Summary statistics

=== Summary ===		
Correctly Classified Instances	1588	95.3181 %
Incorrectly Classified Instances	78	4.6819 %
Kappa statistic	0.7788	
Mean absolute error	0.0833	
Root mean squared error	0.2025	
Relative absolute error	35.8689 %	
Root relative squared error	59.4629 %	
Coverage of cases (0.95 level)	97.9592 %	
Mean rel. region size (0.95 level)	59.7539 %	
Total Number of Instances	1666	

Section C: Codes

```
##################################################################################################
#################
 #                       THESIS: Master in Business Statistics KULeuven
                            #
 #                          Comparing Classification Tree Packages in R
                              #
 #                             by:    Duke Abasi Tchapchet Assam
                                     #

##################################################################################################
#################


 #----------------------------#
 #     Importing Data sets    #
 #----------------------------#

Car <- read.csv("E:/Classification trees in R/ThesisDataNew/Car.csv" , sep=";")
names(Car)
str(Car)

Duke<- read.csv("E:/Classification trees in R/ThesisDataNew/UCI.csv" , sep=";")
names(Duke)


#-------------------------------------- CHAID -------------------------------------------------
---------------------------------#

 #----------------------------------------#
 #                CHAID                    #
 #----------------------------------------#


/*INSTAL PACKAGE FROM R-FORGE WEBSITE. NB: package is not available to download directly*/

install.packages("CHAID", repos="http://R-Forge.R-project.org")




library("CHAID")
require("partykit")
require("grid")

is.na(Duke)  # Turns true if missing values exist in the dataframe
str(Duke)

/*split function*/

splitdf <- function(dataframe, seed=NULL) {
    if (!is.null(seed)) set.seed(seed)
    index <- 1:nrow(dataframe)
    trainindex <- sample(index, trunc(length(index)/3))
    trainset <- dataframe[-trainindex, ]
    testset <- dataframe[trainindex, ]
    list(trainset=trainset,testset=testset)
}

#applying the split function
splits <- splitdf(Car, seed=808)

/* This returns a list - two data frames called trainset and testset */

str(splits)

/*Giving names to training set and test set */
```

```
trainCar <- splits$trainset
testCar <- splits$testset
names(testCar)

################################################################################################
#####
chaids <- chaid(evaluatn~buying+maint+doors+persons+lug_boot+safety, data = trainCar)

ctrl <- chaid_control(minsplit = 20, minprob = 0.05)
chaids1 <- chaid(evaluatn~buying+maint+doors+persons+lug_boot+safety, data = trainCar,
control=ctrl)
chaids1
test<-predict(chaids1, newdata =testCar)
print(test)

confmatrix<-table(testCar$evaluatn,test)
confmatrix
resub.error <- 1.0 -
(confmatrix[1,1]+confmatrix[2,2]+confmatrix[3,3]+confmatrix[4,4])/sum(confmatrix)
resub.error
library(ROCR)

#----------------------------------------#
 #                 C50                    #
 #----------------------------------------#

/*  Load C50 Package */
/*First instal package from CRAN then do following */
library(C50)
require(C50)

 #----------------------------#
 #     Importing Data sets   #
 #----------------------------#
 Address <- "E:\\Classification trees in R\\Thesis Data\\"   ## Specifies the location of the
datasets

/* reading each data from the specified location */

 UCI <- read.csv(paste(Address,"UCI.csv", sep=""))
 Duke <- read.csv(paste(Address,"Duke.csv", sep=""))
duke <- read.csv("E:/Classification trees in R/ThesisData/Duke.csv" , sep=";")
UCI <- read.csv("E:/Classification trees in R/ThesisData/UCI1.csv" , sep=";")

/* Attach Data*/

attach(UCI)
attach(duke)

/* view data*/
UCI
Duke
names(duke)
names(UCI)



############################################################

/* Split Data into training set and test set*/

##############################################################

/*split function*/

splitdf <- function(dataframe, seed=NULL) {
    if (!is.null(seed)) set.seed(seed)
    index <- 1:nrow(dataframe)
    trainindex <- sample(index, trunc(length(index)/3))
    trainset <- dataframe[-trainindex, ]
    testset <- dataframe[trainindex, ]
    list(trainset=trainset,testset=testset)
}

#apply the function
splits <- splitdf(UCI, seed=403)

/* This returns a list - two data frames called trainset and testset */

str(splits)

/*Giving names to training set and test set */

trainUCI <- splits$trainset
testUCI <- splits$testset
###############################################################################################

# Note that it is also possible to split data in C50 with the option 'sample = 0.65' in the
C5.0Control
#----------------------------------------#
 #         RPART : Regression Tree         #
 #----------------------------------------#

#----------------------------#
 #      Importing Data sets   #
 #----------------------------#
 boston1 <- read.csv("C:/Users/hebbar/Desktop/boston.csv")
 boston <- read.csv("E:/Statistical Consulting/boston.csv")
names(boston)
boston

/*  Load rpart Package */

library(rpart)
require(rpart)

/* Attach Data*/
attach(boston)

/*view variable names*/
names(boston)


##############################################################

/* Split Data into training set and test set*/

##############################################################

/*split function*/

splitdf <- function(dataframe, seed=NULL) {
    if (!is.null(seed)) set.seed(seed)
    index <- 1:nrow(dataframe)
    trainindex <- sample(index, trunc(length(index)/3))
    trainset <- dataframe[-trainindex, ]
    testset <- dataframe[trainindex, ]
    list(trainset=trainset,testset=testset)
}

#apply the function
splits <- splitdf(boston, seed=202)

/* This returns a list - two data frames called trainset and testset */

str(splits)

/*Giving names to training set and test set */

trainboston <- splits$trainset
testboston <- splits$testset
names(trainboston)
```

```
/*  rpart function: run rpart using traning set */

fit1<-rpart(medv ~.,data=trainboston, method='anova')
fit22<-rpart(medv ~.,data=boston, method='anova', xval=10)
plot(fit22)
text(fit22)
summary(fit22)
par(mfrow=c(1,2), xpd=NA) # otherwise on some devices the text is clipped



plot(fit1, uniform=TRUE, compress=T, margin=0, main="Regression Tree for Boston data with
training set)")
text(fit1, use.n=TRUE, all=TRUE, cex=.8)
rsq.rpart(fit1); par(mfcol=c(1,1))

/*Summary which examines each node in depth
summary(fit1)

/* Using test data to test model prediction*/
test<-predict(fit1,type='class',testUCI)
test
test1





/*prints complexity and out of sample error*/
printcp(fit1)

/*plots complexity vs. error*/
plotcp(fit1)

# Tree pruning after selecting the cp from cpplot
pruned<-prune(fit1,cp=0.021)
dev.set(which=3);
plot(pruned, uniform=TRUE, compress=T, margin=0, main="Final Classification Tree for Boston
data,CP=0.021")
text(pruned, use.n=TRUE, all=TRUE, cex=.8)
summary(pruned)


xmat <- xpred.rpart(pruned)
testb<- predict(pruned, newdata=testboston)
testb
summary(resid(testb))
plot((testb),testboston$medv)
ress<-lm(testb ~testboston$medv)
plot(resid(ress))
summary(ress)
summary(residuals(ress))


# Cross-validation plots
dev.set(which=5); par(mfcol=c(2,1));
rsq.rpart(fit1); par(mfcol=c(1,1))
#predict the outcome of the testing data
fit3 <- (fit,newdata=testboston)

summary(residuals(fit1))
plot(fit3),residuals(fit3))


#----------------------------------------#
 #          RPART SCORE                  #
 #----------------------------------------#

nlsy <- read.csv("E:/Classification trees in R/ThesisData/nlsy.csv" , sep=";")
```

```
names(nlsy)
str(nlsy)
/*  Load rpart Package */
library(rpart)
library(rpartScore)
require(rpartScore)
require(rpart)


/* Attach Data*/
attach(nlsy) # to attach the data
nlsy    #to view the dataset
/*view variable names*/
names(nlsy)   #gives the variable names of the dataset
summary(nlsy) # give a summary of the data
is.na(nlsy)  # Turns true if missing values exist in the dataframe

dim(nlsy)


names(nlsy)
/* Create ordinal response */

nlsy$Category<-ifelse(nlsy$wage<=20000,4,
ifelse(nlsy$wage<=30000,3,
ifelse(nlsy$wage<=40000,2,1)))
summary(nlsy$Category)

###############################################################

/* Split Data into training set and test set*/

################################################################



/*split function*/

splitdf <- function(dataframe, seed=NULL) {
    if (!is.null(seed)) set.seed(seed)
    index <- 1:nrow(dataframe)
    trainindex <- sample(index, trunc(length(index)/3))
    trainset <- dataframe[-trainindex, ]
    testset <- dataframe[trainindex, ]
    list(trainset=trainset,testset=testset)
}

#applying the split function
splits <- splitdf(nlsy, seed=606)

/* This returns a list - two data frames called trainset and testset */

str(splits)

/*Giving names to training set and test set */

trainnlsy <- splits$trainset
testnlsy <- splits$testset
names(testnlsy)
##################################################################################################
#########################################



############### rpartScore #########################################
set.seed(16112013)



####################  (1)absolute differences in scores(default) ################################
/*misclassification cost(default)*/
```

```
s.abs.mc <-rpartScore(Category ~ ww + afqt + yeared + med + fed + female + black + hisp + asian +
natamer + health + ne + nortcent + south + rural + aa + ba + bs + masters + phd + profdeg,
data=trainnlsy)
plot(s.abs.mc, main="RpartScore abs/mc Tree")
text(s.abs.mc,pretty=TRUE,use.n=TRUE, all=TRUE)
text(s.abs.mc)
plotcp(s.abs.mc)

s.abs.mc.pruned<-prune(s.abs.mc,cp=0.011)
summary(s.abs.mc.pruned)

plot(s.abs.mc.pruned, main="RpartScore  abs/mc pruned Tree cp=0.011")

text(s.abs.mc.pruned,pretty=TRUE,use.n=TRUE, all=TRUE)


/* misclassification rate*/

s.abs.mr <-rpartScore(Category ~ ww + afqt + yeared + med + fed + female + black + hisp + asian +
natamer + health + ne + nortcent + south + rural + aa + ba + bs + masters + phd + profdeg,
data=trainnlsy, prune = "mr")
plot(s.abs.mr, main="RpartScore abs/mr Tree")
text(s.abs.mr,pretty=TRUE,use.n=TRUE, all=TRUE)
plotcp(s.abs.mr)
s.abs.mr.pruned<-prune(s.abs.mr,cp=0.012)

plot(s.abs.mr.pruned, main="RpartScore  abs/mr pruned Tree cp=0.011")

text(s.abs.mr.pruned,pretty=TRUE,use.n=TRUE, all=TRUE)


##########################   (2) squared differences in scores
####################################

/* misclassification cost(default)*/

s.quad.mc <-rpartScore(Category ~ ww + afqt + yeared + med + fed + female + black + hisp + asian
+ natamer + health + ne + nortcent + south + rural + aa + ba + bs + masters + phd + profdeg,
data=trainnlsy, split="quad")
plot(s.quad.mc, main="RpartScore quad/mc Tree")
text(s.quad.mc,pretty=TRUE,use.n=TRUE, all=TRUE)
text(s.quad.mc)
plotcp(s.quad.mc)

s.quad.mc.pruned<-prune(s.quad.mc,cp=0.011)

plot(s.quad.mc.pruned, main="RpartScore  quad/mc pruned Tree cp=0.011")

text(s.quad.mc.pruned,pretty=TRUE,use.n=TRUE, all=TRUE)


/* misclassification rate*/

s.quad.mr <-rpartScore(Category ~ ww + afqt + yeared + med + fed + female + black + hisp + asian
+ natamer + health + ne + nortcent + south + rural + aa + ba + bs + masters + phd + profdeg,
data=trainnlsy,split="quad", prune = "mr")
plot(s.quad.mr, main="RpartScore quad/mr Tree")
text(s.quad.mr,pretty=TRUE,use.n=TRUE, all=TRUE)
plotcp(s.quad.mr)
s.quad.mr.pruned<-prune(s.quad.mr,cp=0.012)

plot(s.quad.mr.pruned, main="RpartScore  quad/mr pruned Tree cp=0.012")

text(s.quad.mr.pruned,pretty=TRUE,use.n=TRUE, all=TRUE)




#----------------------------------------#
```

```
 #                  RWEKA                    #
 #---------------------------------------#

/*  Load RWeka Package */
/*First instal package from CRAN then do following */
library(RWeka)
require(RWeka)



###############################################################################################

/* Used in case you want to use one of the datasets already installed in weka tool */

read.arff(system.file("arff", "data.arff", package = "RWeka"))

Example
DF2 <- read.arff(system.file("arff", "contact-lenses.arff",
package = "RWeka"))
m2 <- J48(‘contact-lenses‘ ~ ., data = DF2)
m2
table(DF2$‘contact-lenses‘, predict(m2))
if(require("party", quietly = TRUE)) plot(m2)

###############################################################################################

################################################################

/* Split Data into training set and test set*/

################################################################

/*split function*/

splitdf <- function(dataframe, seed=NULL) {
    if (!is.null(seed)) set.seed(seed)
    index <- 1:nrow(dataframe)
    trainindex <- sample(index, trunc(length(index)/3))
    trainset <- dataframe[-trainindex, ]
    testset <- dataframe[trainindex, ]
    list(trainset=trainset,testset=testset)
}

#apply the function
split <- splitdf(duke, seed=7073)

/* This returns a list - two data frames called trainset and testset */

str(splits)

/*Giving names to training set and test set */

trainUCI <- split$trainset
testUCI <- split$testset
names(trainUCI)

trainduke <- splits$trainset
testduke<- splits$testset

/*C4.5 Implementation with J48 OF WEKA*/

J48(formula, data, subset, na.action,control = Weka_control(), options = NULL)



/* Learn J4.8 tree on UCI data with default settings:*/
tree <- J48(Churn ~ ., data =trainUCI)
tree      ## to view tree details
plot(tree)  ##Plots tree
```

```
## Query J4.8 options:

WOW("J48")        # Gives a list with all options available for J48

## Learn J4.8 tree on UCI data with reduced error pruning:
tree1 <- J48(Churn ~ ., data = UCI, control = Weka_control(R = TRUE))
plot(tree1)

## Learn J4.8 tree with reduced error pruning (-R) and
## minimum number of instances set to 5 (-M 5):
tree2 <- J48(Churn ~ ., data = UCI, control = Weka_control(R = TRUE, M = 20))
tree3 <- J48(Churn ~ ., data = UCI, control = Weka_control(M = 20, C=0.10))

plot(tree3)


e <- evaluate_Weka_classifier(tree2,newdata = testUCI,
cost = matrix(c(0,2,1,0), ncol = 2),
numFolds = 10, complexity = TRUE,
seed = 123, class = TRUE)
e
summary(e)



#----------------------------------------#
 #                 RPART                 #
 #----------------------------------------#

/*  Load rpart Package */

library(rpart)
require(rpart)

/* Attach Data*/
attach(UCI)
UCI
/*view variable names*/
names(UCI)
summary(UCI)
 sd(UCI)
is.na(UCI)  # Turns true if missing values exist in the dataframe

length(UCI$State)
str(UCI)
table(UCI$State)

#########################################################################
/* Rounding-up continuous data into whole numbers */


omit<-
c("State","Account_Length","Area_Code2","Intl_Plan","Vmail_Plan","Vmail_Message","Day_Calls","Eve
_Calls","Night_Calls","Intl_Calls","CustServ_Calls","Churn")
leave<-match(omit,names(UCI))
out<-UCI
out[-leave]<-round(UCI[-leave])
out
UCInew <- out  #Giving the rounded up data a new name
UCInew


##############################################################

/* Split Data into training set and test set*/

###############################################################

/*split function*/

splitdf <- function(dataframe, seed=NULL) {
    if (!is.null(seed)) set.seed(seed)
    index <- 1:nrow(dataframe)
    trainindex <- sample(index, trunc(length(index)/3))
    trainset <- dataframe[-trainindex, ]
    testset <- dataframe[trainindex, ]
    list(trainset=trainset,testset=testset)
}

#applying the split function
splits <- splitdf(UCInew, seed=808)

/* This returns a list - two data frames called trainset and testset */

str(splits)

/*Giving names to training set and test set */

trainUCI <- splits$trainset
testUCI <- splits$testset
names(testUCI)
#################################################################################################
########################################

/*  rpart function: run rpart using training set with no control on size */

fit1<-rpart(Churn ~.,data=trainUCI )
fit1c<-rpart(Churn ~.,data=trainUCI,                    #specify training data
             method="class",                            #classification tree
             parms =list(split = "gini"),               #use entropy/deviance
             maxsurrogate = 0.001,                      #due to no missing values
             cp=0,                                      #no penalty on the size of tree
             minsplit= 20)                              #nodes of size 20(Default) or more can be
splitted

            # minbucket = 7,                             #each sub-node contains at least
(Default= minsplit/3)7 observations



plot(fit1c, uniform =TRUE, margin= .01, main="Rpart Tree with no constraints")
text(fit1c,use.n=TRUE, all=TRUE)

#################################################################################################
#######################################

/*  rpart function: run rpart using training set */


fit1<-rpart(Churn ~.,data=trainUCI )

par(mfrow=c(1,2), xpd=NA) # otherwise on some devices the text is clipped
dev.set(which=1);
plot(fit1, uniform=TRUE, margin=0.1, main="Default Rpart Tree"); #this plots an empty tree
text(fit1); #this puts in text on the tree
text(fit1, use.n=T, all=TRUE); #this puts in text on the tree


/* print a text version of the tree */
print(fit1)

/*Summary which examines each node in depth
summary(fit1)

#################################################################################################
#######
/*prints complexity and out of sample error*/
printcp(fit1)

/*plots complexity vs. error*/
plotcp(fit1, minline=TRUE, lty=3, col=1,
        upper= c("size"))
```

```
coord<- locator(type="1")             #helps to use mouse to determine point where you want to
plot a line
abline(v=6.003429, lty=2, col="blue") #to plot vertical line on desired point on the plot

##################################################################################################
################

/*Applying The threshold cost-complexity parameter and cross-validation */

contr <- rpart.control(xval=10, minsplit=20, cp=0.03)
fitcv <-rpart(Churn ~.,data=trainUCI,method='class', control=contr)
plot(fitcv,uniform=TRUE, margin=0.1, main="RPart tree with 10-fold X-val")
text(fit1a, use.n=T, all=TRUE); #this puts extra information on the tree


/* Tree pruning after selecting the cp from cpplot*/
fitpr<-prune(fit1,cp=0.03)
plot(pruned, uniform=TRUE, margin=0.1, main="Pruned RPart tree cp=0.03");
text(pruned, use.n=T, all=TRUE)

#############################################################################################
/* Using test data to test model prediction*/
##predict the outcome of the testing data
##predicted <- predict(model, newdata=testing[ ,-1])


test1<-predict(fit1,type='class',testUCI)
testcv<-predict(fitcv,type='class',testUCI)    #test using cross-validation model
testpr<-predict(fitpr,type='class',testUCI)    #test using pruned model

######################################################################

/* Confusion Matrix */
confmatrix<- table(testUCI$Churn,test)
print(confmatrix)

/*Calculate resubstitution error */
resub.error <- 1.0 - (confmatrix[1,1]+confmatrix[2,2])/sum(confmatrix)
print(resub.error)


# Low-resolution plov.set(which=1);
plot(fit1);
text(fit1)
# High-resolution plot: Prettier version of the tree
dev.set(which=2);
post(fit1,file="E:/Classification trees in R/Thesis Diagrams/rpart1.pdf")


# what is the proportion variation explained in the outcome of the testing data?
# i.e., what is 1-(SSerror/SStotal)
actual <- testing$Sepal.Length
rsq <- 1-sum((actual-predicted)^2)/sum((actual-mean(actual))^2)
print(rsq)

## Using Prior probabilities
fit2<-rpart(Churn ~.,data=trainUCI,params=list(prior(.9,.1)),cp=.03)
plot(fit2)
text(fit2)
##################################################################################################
##############
/*ROC Curve*/
library(ROCR)
testUCI

fit1<-rpart(Churn ~.,data=trainUCI)
pred<-predict(fit1,newdata=testUCI,type='prob')[,2]
pred
testUCI$Churn
pred1<- prediction(pred,testUCI$Churn)
perf1<-performance(pred1, 'tpr' , 'fpr')
```

```
print(perf1)
perf.fpr <- performance(pred1, "fpr")
perf.fnr <- performance(pred1, "fnr")

testUCI$Churn
pred1<- prediction(pred,testUCI$Churn)
perf1<-performance(pred1, 'tpr' , 'fpr')
print(perf1)

fit2<-rpart(Churn ~.,data=trainUCI,parms=list(prior(.9,.1)),cp=.03)

test2<-predict(fit2,type='prob',testUCI)
pred2<- prediction(test2,testUCI$Churn)
perf2<-performance(pred2, 'tpr' , 'fpr')
print(perf)

fit12 <-rpart(Churn ~.,data=trainUCI, parms=list(prior=c(.65,.35), split= 'information' ))
fit13 <- rpart(Churn ~.,data=trainUCI, control=rpart.control(cp=.03))

pred12<-predict(fit12,newdata=testUCI,type='prob')[,2]
pred13<-predict(fit13,newdata=testUCI,type='prob')[,2]
predn12<- prediction(pred12,testUCI$Churn)
predn13<- prediction(pred13,testUCI$Churn)

perf12<-performance(predn12, 'tpr' , 'fpr')
perf13<-performance(predn13, 'tpr' , 'fpr')



/* Comparing ROC for tree vs tree with prior */

plot(perf1, col='red', lty=1, main ='ROC Curve for Rpart Trees')
plot(perf12, col='blue' ,add = TRUE, lty=2)
plot(perf13, col='green', add =TRUE, lty=3)
legend(0.6,0.6,c('simple tree','tree with 90/10 prior','pruned
tree'),col=c('red','blue','green'),lwd=3)

# calculating AUC
auc1 <- performance(pred1,"auc")
auc2 <- performance(predn12,"auc")
auc3 <- performance(predn13,"auc")

minauc<-min(round(auc1, digits = 2))
maxauc<-max(round(auc1, digits = 2))
minauct <- paste(c("min(AUC) = "),minauc,sep="")
maxauct <- paste(c("max(AUC) = "),maxauc,sep="")
legend(0.3,0.6,c(minauct,maxauct,"\n"),border="white",cex=1.7,box.col = "white")


/* Transform tree to rules*/
list.rules.rpart(fit1)
list.rules.rpart(fit2)

listrules(fit1)  #only prints rules for one class of churning.
```