1 CaRT

1.1 Overview

CaRT, a recursive partitioning method, builds classification and regression trees for predicting continuous dependent variables (regression) and categorical predictor variables (classification). The classic C&RT algorithm was popularized by Breiman et al.

1.2 Classification and Regression Problems

- There are numerous algorithms for predicting continuous variables or categorical variables from a set of continuous predictors and/or categorical factor effects. For example, in GLM (General Linear Models) and GRM (General Regression Models), we can specify a linear combination (design) of continuous predictors and categorical factor effects (e.g., with two-way and three-way interaction effects) to predict a continuous dependent variable.
- In GDA (General Discriminant Function Analysis), we can specify such designs for predicting categorical variables, i.e., to solve classification problems.

1.3 Regression-type problems

- Regression-type problems are generally those where we attempt to predict the values of a continuous variable from one or more continuous and/or categorical predictor variables. For example, we may want to predict the selling prices of single family homes (a continuous dependent variable) from various other continuous predictors (e.g., square footage) as well as categorical predictors (e.g., style of home, such as ranch, two-story, etc.; zip code or telephone area code where the property is located, etc.; note that this latter variable would be categorical in nature, even though it would contain numeric values or codes).
- If we used simple multiple regression, or some general linear model (GLM) to predict the selling prices of single family homes, we would determine a linear equation for these variables that can be used to compute predicted selling prices.

- There are many different analytic procedures for fitting linear models (GLM, GRM, Regression), various types of nonlinear models (e.g., Generalized Linear/Nonlinear Models (GLZ), Generalized Additive Models (GAM), etc.), or completely custom-defined nonlinear models (see Nonlinear Estimation), where we can type in an arbitrary equation containing parameters to be estimated. CHAID also analyzes regression-type problems, and produces results that are similar (in nature) to those computed by C&RT. Note that various neural network architectures are also applicable to solve regression-type problems.

1.4 Classification-type problem

- Classification-type problems are generally those where we attempt to predict values of a **categorical dependent variable** (class, group membership, etc.) from one or more continuous and/or categorical predictor variables.

- **Examples** - For example, we may be interested in predicting who will or will not graduate from college, or who will or will not renew a subscription.

- **Binary Outcomes** - These would be examples of simple binary classification problems, where the categorical dependent variable can only assume two distinct and mutually exclusive values.

- **Multichotomous Outcomes** - In other cases, we might be interested in predicting which one of multiple different alternative consumer products (e.g., makes of cars) a person decides to purchase, or which type of failure occurs with different types of engines. In those cases there are multiple categories or classes for the categorical dependent variable.

1.5 Classification and Regression Trees (CaRT)

In most general terms, the purpose of the analyses via tree-building algorithms is to determine a set of ***if-then*** logical (split) conditions that permit accurate prediction or classification of cases.

1.5.1 Classification Trees

- For example, consider the widely referenced Iris data classification problem introduced by Fisher.
- The `iris` data file containts the lengths and widths of sepals and petals of three types of irises: Setosa, Versicolor, and Virginica).
- The purpose of the analysis is to learn how we can discriminate between the three types of flowers, based on the four measures of width and length of petals and sepals.
- **Discriminant function analysis** will estimate several linear combinations of predictor variables for computing classification scores (or probabilities) that allow the user to determine the predicted classification for each observation.
- A classification tree will determine a set of logical ***if-then*** conditions (instead of linear equations) for predicting or classifying cases instead.
- The interpretation of this tree is straightforward:
  - If the petal width is less than or equal to 0.8, the respective flower would be classified as ***Setosa***;
  - if the petal width is greater than 0.8 and less than or equal to 1.75, then the respective flower would be classified as ***Virginica***;
  - else, it belongs to class ***Versicolor***.

```
iris.pw.tree <- tree(Species ~ Petal.Width,data=iris)
plot(iris.pw.tree)
text(iris.pw.tree)
```

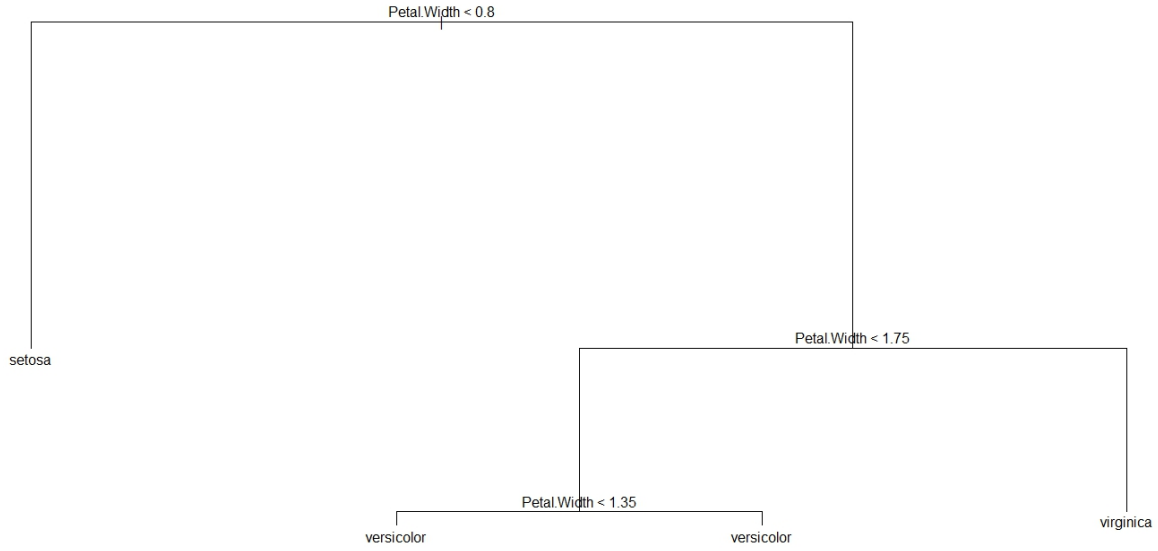

Figure 1.1:

1.6 Iris Tree Example

- Using all four variables and all of the 150 cases

```
iris.tree <- tree(Species ~ Sepal.Length+Sepal.Width+
Petal.Length+Petal.Width,data=iris)
plot(iris.tree)
text(iris.tree)
```

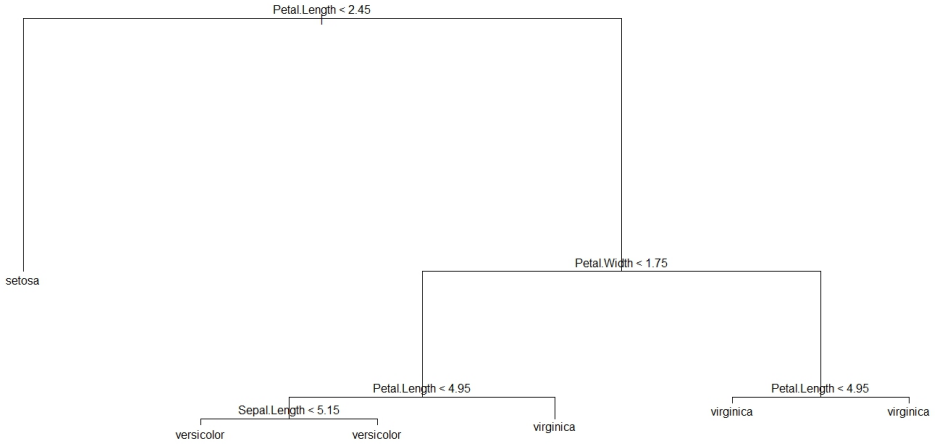

Figure 1.2:

1.6.1 Iris Example - Using a Training Set

- set up a training data set of 100 randomly selected cases
- Compare results with full data set tree.

```
set.seed(1234)
Train1Index = sample(1:150,100)
iristrain1=iris[Train1Index,]

iristrain1.tree <- tree(Species ~ Sepal.Length+Sepal.Width+
Petal.Length+Petal.Width,data=iristrain1)
plot(iristrain1.tree)
text(iristrain1.tree)
```

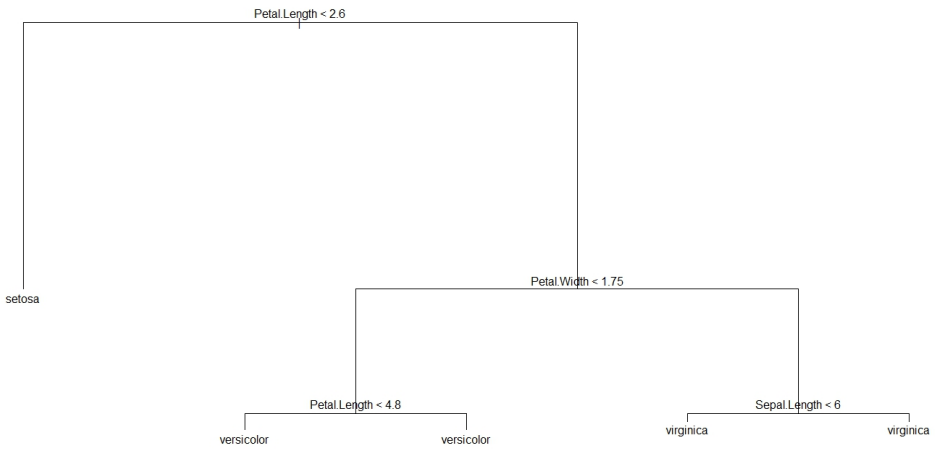

Figure 1.3:

- Another test set (picked using a different seed)

```
set.seed(3456)

Train2Index = sample(1:150,100)
iristrain2=iris[Train2Index,]

iristrain2.tree <- tree(Species ~ Sepal.Length+Sepal.Width+
  Petal.Length+Petal.Width,data=iristtrain2)
plot(iristrain2.tree)
text(iristrain2.tree)
```

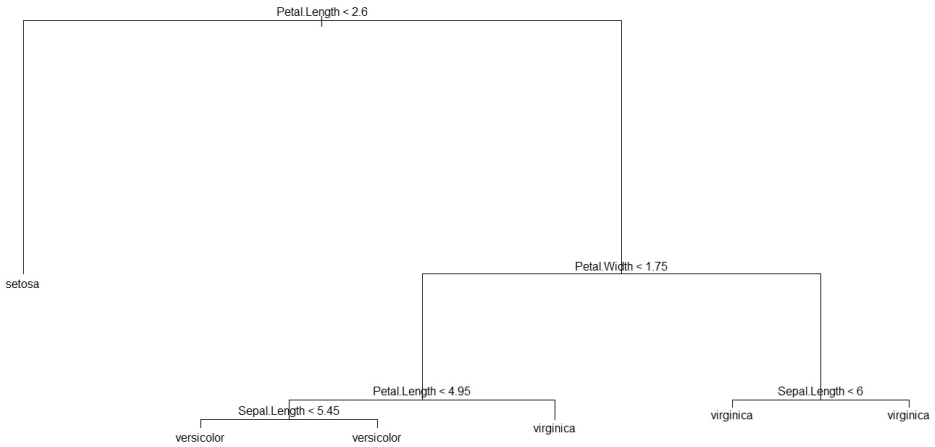

Figure 1.4:

```
> table(iristrain1$Species)

    setosa versicolor  virginica
        38         29         33
> table(iristrain2$Species)

    setosa versicolor  virginica
        34         32         34
```

- Set up a test data set to compare trees
- In this case, probably little difference

```
newdata=iris[131:150,]
predict(iristrain1.tree,newdata)
predict(iristrain2.tree,newdata)
```

1.7 Tree Pruning

The `prune.tree()` command determines a nested sequence of subtrees of the supplied tree by recursively snipping off the least important splits in the classification tree.

```
iris.pruned <- prune.tree(iris.tree,best=3)
plot(iris.pruned); text(iris.pruned)
plot(iris.pruned); text(iris.pruned)
```

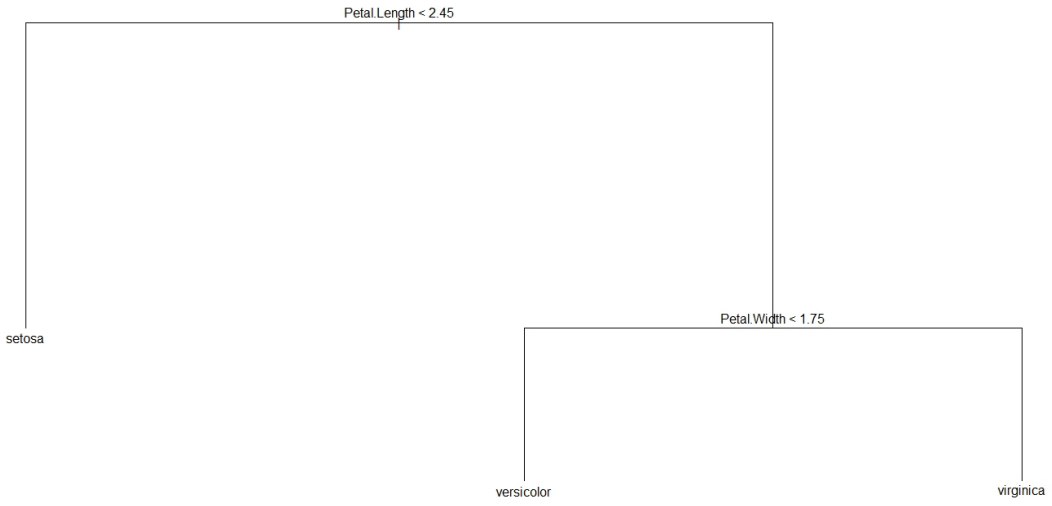

Figure 1.5:

1.8 Random Forests

Random forests improve predictive accuracy by generating a large number of bootstrapped trees (based on random samples of variables), classifying a case using each tree in this new "forest", and deciding a final predicted outcome by combining the results across all of the trees (an average in regression, a majority vote in classification).

```
library(randomForest)
iris.rf <- randomForest(Species ~ Sepal.Length+Sepal.Width+
Petal.Length+Petal.Width,data=iris)
print(iris.rf) # view results
importance(iris.rf) # importance of each predictor
```

```
> iris.rf <- randomForest(Species ~ Sepal.Length+Sepal.Width+
+ Petal.Length+Petal.Width,data=iris)
> print(iris.rf) # view results

Call:
 randomForest(formula = Species ~ Sepal.Length + Sepal.Width +      Petal.Length + Petal.
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 2

        OOB estimate of  error rate: 4%
Confusion matrix:
           setosa versicolor virginica class.error
setosa         50          0         0        0.00
versicolor      0         47         3        0.06
virginica       0          3        47        0.06
```

```
> importance(iris.rf) # importance of each predictor
             MeanDecreaseGini
Sepal.Length         9.654747
Sepal.Width          2.591063
Petal.Length        44.131695
Petal.Width         42.850817
```

Making a Prediction

```
library(randomForest)
iris.t1.rf <- randomForest(Species ~ Sepal.Length+Sepal.Width+
Petal.Length+Petal.Width,data=iristest1)



newdata2index=sample(1:150,20)
newdata2=iris[newdata2index,]

predict(iris.t1.rf,newdata2)
```

1.9 Regression Trees

- The general approach to derive predictions from few simple `if-then` conditions can be applied to regression problems as well.
- In this example, we will use the `cu.summary` data set in the ***rpart*** package.

```
> library(rpart)
> tail(cu.summary)
                     Price Country Reliability Mileage Type
Ford Aerostar V6     12267     USA     average      18  Van
Mazda MPV V6         14944   Japan Much better      19  Van
Mitsubishi Wagon 4   14929   Japan        <NA>      20  Van
Nissan Axxess 4      13949   Japan        <NA>      20  Van
Nissan Van 4         14799   Japan        <NA>      19  Van
Volkswagen Vanagon 4 14080 Germany        <NA>      NA  Van
```

```
fit <- rpart(Mileage~Price + Country + Reliability + Type,
   method="anova", data=cu.summary)

printcp(fit) # display the results
```

Display results of fitted tree

```
> printcp(fit) # display the results

Regression tree:
rpart(formula = Mileage ~ Price + Country + Reliability + Type,
    data = cu.summary, method = "anova")

Variables actually used in tree construction:
[1] Price Type

Root node error: 1354.6/60 = 22.576

n=60 (57 observations deleted due to missingness)

        CP nsplit rel error  xerror     xstd
1 0.622885      0   1.00000 1.00671 0.174008
2 0.132061      1   0.37711 0.52578 0.099645
3 0.025441      2   0.24505 0.36496 0.077946
4 0.011604      3   0.21961 0.35238 0.078224
5 0.010000      4   0.20801 0.35522 0.074295
```

```
# plot the regression tree
plot(fit, uniform=TRUE,
   main="Regression Tree for Mileage ")
text(fit, use.n=TRUE, all=TRUE, cex=1.0)
```

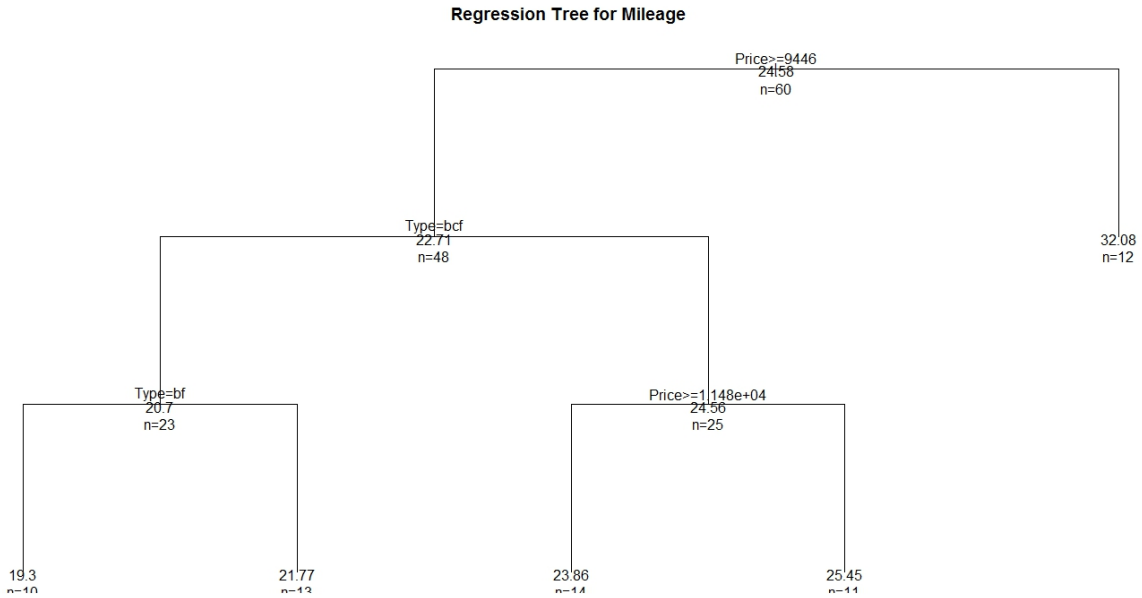

Figure 1.6:

Compare predicted values with observed values.

```
> cbind(predict(fit,cu.summary),cu.summary$Mileage)
                                  [,1] [,2]
Acura Integra 4               23.85714   NA
Dodge Colt 4                  32.08333   NA
Dodge Omni 4                  32.08333   NA
Eagle Summit 4                32.08333   33
Ford Escort   4               32.08333   33
Ford Festiva 4                32.08333   37
GEO Metro  3                  32.08333   NA
GEO Prizm  4                  25.45455   NA
.................
```

1.10 Advantages of Classification and Regression Trees (CaRT) Methods

- As mentioned earlier, there are a large number of methods that an analyst can choose from when analyzing classification or regression problems.

- Tree classification techniques, when they "work" and produce accurate predictions or predicted classifications based on few logical if-then conditions, have a number of advantages over many of those alternative techniques.

- **Simplicity of results** - In most cases, the interpretation of results summarized in a tree is very simple. This simplicity is useful not only for purposes of rapid classification of new observations (it is much easier to evaluate just one or two logical conditions, than to compute classification scores for each possible group, or predicted values, based on all predictors and using possibly some complex nonlinear model equations), but can also often yield a much simpler "model" for explaining why observations are classified or predicted in a particular manner (e.g., when analyzing business problems, it is much easier to present a few simple *if-then* statements to management, than some elaborate equations).

- Tree methods are nonparametric and nonlinear. The final results of using tree methods for classification or regression can be summarized in a series of (usually few) logical if-then conditions (tree nodes). Therefore, there is no implicit assumption about underlying relationships between between variables.

- Thus, tree methods are particularly well suited for data mining tasks, where there is often little a priori knowledge nor any coherent set of theories or predictions regarding which variables are related and how. In those types of data analyses, tree methods can often reveal simple relationships between just a few variables that could have easily gone unnoticed using other analytic techniques.

1.11 Avoiding Over-Fitting: Pruning, Crossvalidation, and V-fold Crossvalidation

- A major issue that arises when applying regression or classification trees to "real" data with much random error noise concerns the decision when to stop splitting. For example, if we had a data set with 20 cases, and performed 19 splits (determined 9 if-then conditions), we could perfectly predict every single case.
- In general, if we only split a sufficient number of times, eventually we will be able to "predict" our original data (from which we determined the splits). Of course, it is far from clear whether such complex results (with many splits) will replicate in a sample of new observations; most likely they will not.
- **Overfitting** - This general issue is relevant to tree classification (as well as neural networks and regression methods)
- If not stopped, the tree algorithm will ultimately "extract" all information from the data, including information that is not and cannot be predicted in the population with the current set of predictors, i.e., random or noise variation.
- The general approach to addressing this issue is first to stop generating new split nodes when subsequent splits only result in very little overall improvement of the prediction.
- For example, if we can predict 90% of all cases correctly from 10 splits, and 90.1% of all cases from 11 splits, then it obviously makes little sense to add that 11th split to the tree.

Criteria

There are many such criteria for automatically stopping the splitting (tree-building) process.

- **Pruning** - Once the tree building algorithm has stopped, it is always useful to further evaluate the quality of the prediction of the current tree in samples of observations that did not participate in the original computations. These methods are used to "prune back" the tree, i.e., to eventually (and ideally) select a simpler tree than the one obtained when

the tree building algorithm stopped, but one that is equally as accurate for predicting or classifying "new" observations.

- **Cross-validation** - One approach is to apply the tree computed from one set of observations (training sample) to another completely independent set of observations (testing sample). If most or all of the splits determined by the analysis of the training sample are essentially based on "random noise," then the prediction for the testing sample will be very poor. Hence, we can see if the selected tree is not very good (useful), or not of the "right size."

- **V-fold crossvalidation** Continuing further along this line of reasoning, we repeat the analysis many times over with different randomly drawn samples from the data, for every tree size starting at the root of the tree, and applying it to the prediction of observations from randomly selected testing samples.

- Then use the tree that shows the best average accuracy for cross-validated predicted classifications or predicted values.

- In most cases, this tree will not be the one with the most terminal nodes, i.e., the most complex tree. This method for pruning a tree, and for selecting a smaller tree from a sequence of trees, can be very powerful, and is particularly useful for smaller data sets.

1.12 Computational Details

The process of computing classification and regression trees can be characterized as involving four basic steps:

1. Specifying the criteria for predictive accuracy
2. Selecting splits
3. Determining when to stop splitting
4. Selecting the "right-sized" tree.

```
library(caret)
set.seed(3456)

#Split the data in training and testing data

trainIndex <- createDataPartition(iris$Species, p = .8,
                                  list = FALSE,
                                  times = 1)
head(trainIndex)

irisTrain <- iris[ trainIndex,]
irisTest  <- iris[-trainIndex,]

# Fit a model using the training data
iris.caret <- train(Species ~ ., data = irisTrain,
   method = "rpart", maxdepth=4)

iris.caret

# Try it out on the testing data
iris.caret.test <- predict(iris.caret,irisTest)



table(iris.caret.test,irisTest$Species)
```

```
> table(iris.caret.test,irisTest$Species)
```

```
iris.caret.test setosa versicolor virginica
     setosa         10          0         0
     versicolor      0         10         0
     virginica       0          0        10
```

1.13 Specifying the Criteria for Predictive Accuracy

The classification and regression trees (C&RT) algorithms are generally aimed at achieving the best possible predictive accuracy. Operationally, the most accurate prediction is defined as the prediction with the minimum costs. The notion of costs was developed as a way to generalize, to a broader range of prediction situations, the idea that the best prediction has the lowest misclassification rate. In most applications, the cost is measured in terms of proportion of misclassified cases, or variance. In this context, it follows, therefore, that a prediction would be considered best if it has the lowest misclassification rate or the smallest variance. The need for minimizing costs, rather than just the proportion of misclassified cases, arises when some predictions that fail are more catastrophic than others, or when some predictions that fail occur more frequently than others.