Sane's STATS Academy of Statistics

Classification Trees

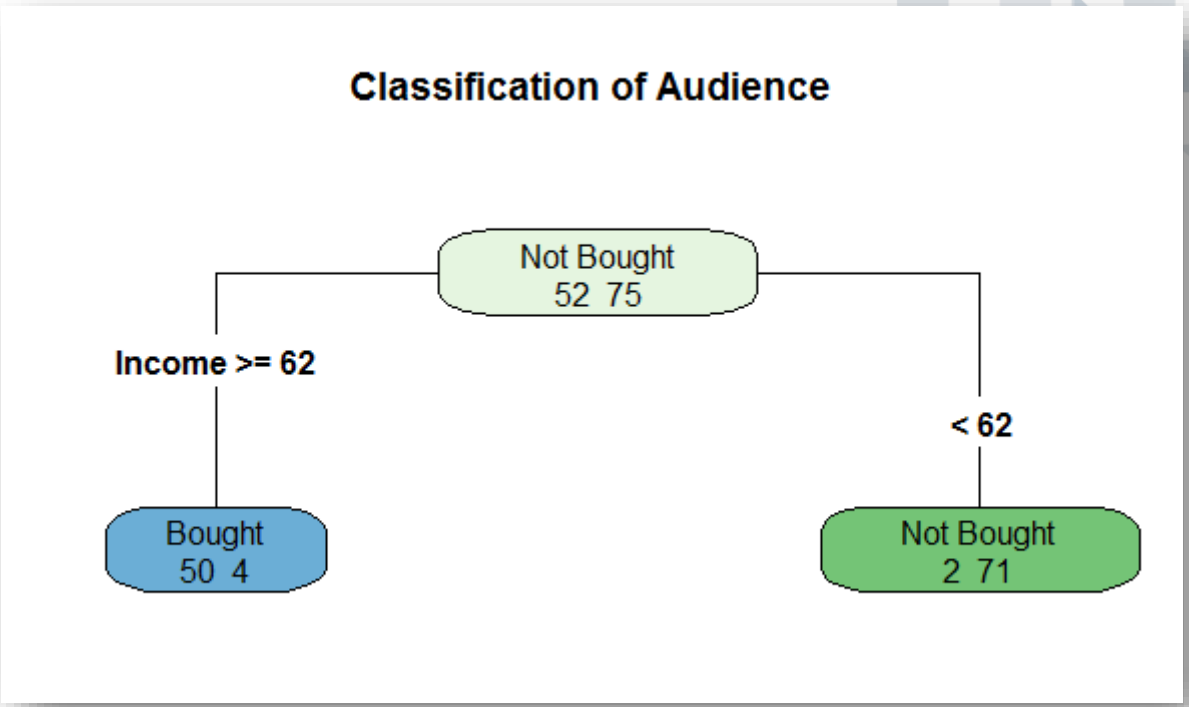
June 7, 2018

Example 1: Riding Mowers

- A riding-mower manufacturer **MOW-EASE** took part in a Industrial Exhibition in which it got an opportunity to show a demo of its product to 180 different audience.
- The land owned by each of the audience and their approximate income have been recorded in the file RidingMowers.csv

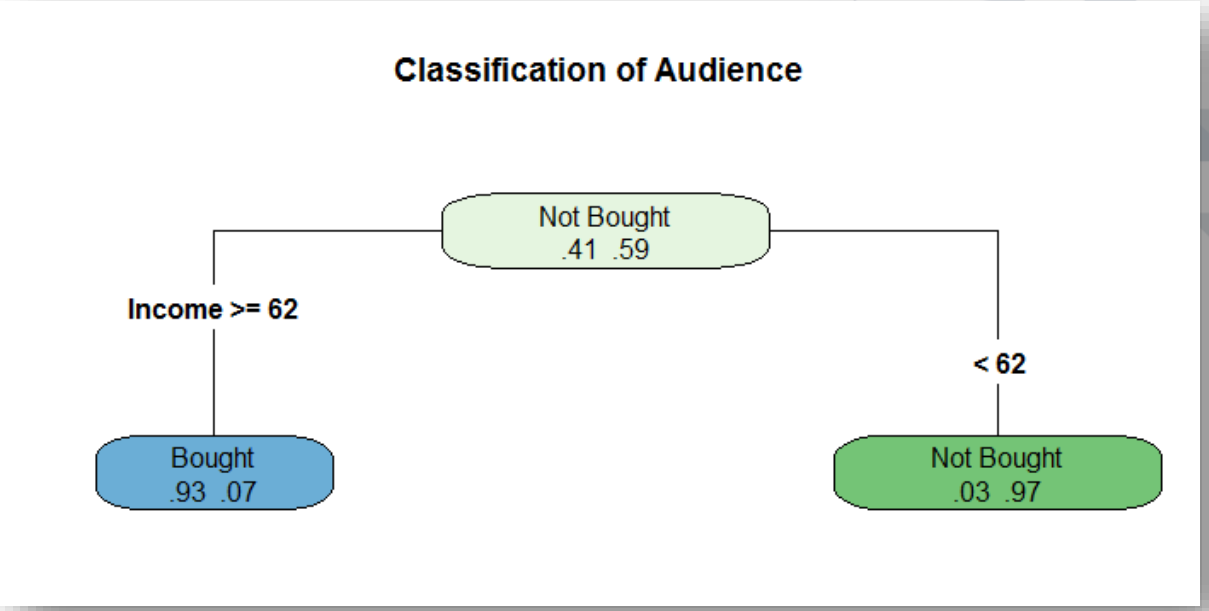
Classification Tree

- For Riding Mowers, data can be classified with tree as follows:



- Here we see that the tree algorithm has classified the audience as Income <= 62 and Income < 62

What did we gain from classification?



- For the whole training data, we had n(Bought)=52 and n(Not Bought)=75 with proportions as 0.41 and 0.59 respectively
- By splitting on Income cut-off as 62, we got two partitions with proportions as [P(B)=0.93,P(NB)=0.07] and [P(B)=0.03, P(NB)=0.97] respectively
- Hence we gained a **purity** or **homogeneity** in classes by this division.

Decision Trees

- Decision Trees create a set of binary splits on the predictor variables
- These splits are used to classify new observations into one of the two groups
- There are two categories of decision trees:
  - Classification Tree for Categorical Response Variable
  - Regression Tree for Numerical Response Variable

Types of Decision Tree Algorithms

- There are many specific decision-tree algorithms. Notable ones include:
  - ID3 (Iterative Dichotomiser 3)
  - C4.5 (successor of ID3)
  - **CART (Classification And Regression Tree)**
  - CHAID (CHi-squared Automatic Interaction Detector). Performs multi-level splits when computing classification trees.
  - MARS: extends decision trees to handle numerical data better.
  - **Conditional Inference Trees:** Statistics-based approach that uses non-parametric tests as splitting criteria, corrected for multiple testing to avoid overfitting. This approach results in unbiased predictor selection and does not require pruning.
- We will be covering CART and Conditional Inference Trees in our learning

Classical Decision Trees Algorithm

- Response : Categorical outcome variable
- Predictors : Categorical and/or Continuous Variables.
- Steps are as follows:
  1. Predictor variable gets chosen in such a way that it best splits the data into two groups with maximized purity.
     A. If the predictor is continuous, then cut-point is chosen for maximizing the purity
     B. If the predictor is categorical, then the categories are combined together to obtain two groups with maximized purity
  2. The data is separated into two groups and the process for each subgroup is continued
  3. Steps 1 and 2 are repeated until a subgroup is obtained containing fewer number of observations than a minimum number specified or the algorithm may terminate if further splitting doesn't increase purity beyond a specific threshold

Sane's STATS Academy of Statistics

Classical Decision Trees Algorithm

- The subgroups in the lowest branch of the tree are called terminal nodes or leaf nodes.
- Each leaf node is classified as one single category of the outcome
- For classifying any observation in the validation / test data set, that observation is traversed through the branches of tree and leaf node at which the traversal stops is predicted as the outcome of the observation.

Sane's STATS Academy of Statistics

Gini's Impurity Index

- Gini impurity can be computed by summing the probability of each item being chosen times the probability of a mistake in categorizing that item.
- It reaches its minimum (zero) when all cases in the node fall into a single target category.
- It is used by CART Algorithm, by package ***rpart*** by default

$$I_G(f) = \sum_{i=1}^{m} f_i(1 - f_i) = \sum_{i=1}^{m} (f_i - {f_i}^2) = \sum_{i=1}^{m} f_i - \sum_{i=1}^{m} {f_i}^2 = 1 - \sum_{i=1}^{m} {f_i}^2 = \sum_{i \neq k} f_i f_k$$

Pruning of Tree

- The tree algorithm often is known to result into overfitting
- Due to overfitting, our predictions are probable to go wrong on a large scale
- To compensate with overfitting, we can prune the created tree

Package rpart

- Classical Decision Tree algorithm can be implemented with function *rpart( )* in package **rpart**
- The word “rpart” stands for **r**ecursive **par**titioning
- Package also provides function prune( ) for pruning the tree

rpart( )

Syntax :

rpart(formula, data, method, control, parms...)

Where

formula: formula with response variable specified

data : training data frame

method : should be specified as “class” when response variable is categorical and as “anova” when response variable is numerical / continuous

control : list of options that tune the algorithm. Should be specified with function rpart.control( )

parms : can be specified with splitting formula

rpart.control( )

Syntax :

rpart.control(minsplit = 20, minbucket = round(minsplit/3), cp = 0.01, ...)

Where

minsplit : the minimum number of observations that must exist in a node in order for a split to be attempted.

minbucket : the minimum number of observations in any terminal <leaf> node. If only one of minbucket or minsplit is specified, the code either sets minsplit to minbucket*3 or minbucket to minsplit/3, as appropriate.

cp : complexity parameter. Any split that does not decrease the overall lack of fit by a factor of cp is not attempted. For instance, with anova splitting, this means that the overall R-squared must increase by cp at each step. The main role of this parameter is to save computing time by pruning off splits that are obviously not worthwhile. Essentially, the user informs the program that any split which does not improve the fit by cp will likely be pruned off by cross-validation, and that hence the program need not pursue it.

rpart.plot()

- The function *rpart.plot( )* is provided in the package **rpart.plot** .

Syntax : rpart.plot (Objrpart , type , extra , …)

Where

Objrpart : Object of the class rpart

type : Type of plot; Possible values are as follows:

0 - Draw a split label at each split and a node label at each leaf

1 - Label all nodes, not just leaves

2 – Draw split labels below the node labels

3 - Draw separate split labels for the left and right directions

4 - Like 3 but label all nodes, not just leaves

Continued….

rpart.plot()

Syntax : ***rpart.plot (Objrpart , type , extra , …)***

Where

extra : Display extra information at the nodes. Possible values are as follows:

0 – No extra information
1 - Display the number of observations that fall in the node
2 - Display the classification rate at the node
3 - Misclassification rate at the node
4 - Probability per class of observations in the node
5 - Like 4 but do not display the fitted class
6 - The probability of the second class only
7 - Like 6 but do not display the fitted class
8 - The probability of the fitted class
9 - The probabilities times the fraction of observations in the node

More information can be found in help of this function

prp()

- The word “prp” stands for plotting rpart model

Syntax: ***prp (Objrpart , type , extra , …)***

Where

extra : Display extra information at the nodes. Possible values are as follows:

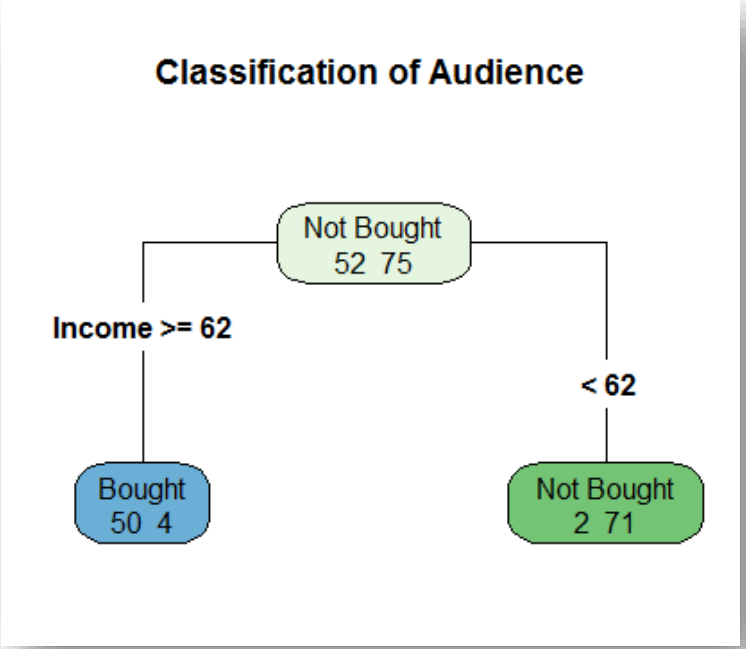
0 – No extra information
1 - Display the number of observations that fall in the node
2 - Display the classification rate at the node
3 - Misclassification rate at the node
4 - Probability per class of observations in the node
5 - Like 4 but do not display the fitted class
6 - The probability of the second class only
7 - Like 6 but do not display the fitted class
8 - The probability of the fitted class
9 - The probabilities times the fraction of observations in the node

Arguments of this function are many more than given. More information can be found in help of this function

Program and Output

```
library(rpart)
library(rpart.plot)

fitRPart <- rpart(Response ~ ., method="class", data=training)

rpart.plot(fitRPart,type=4, extra=1 , main = "Classification of Audience")
```

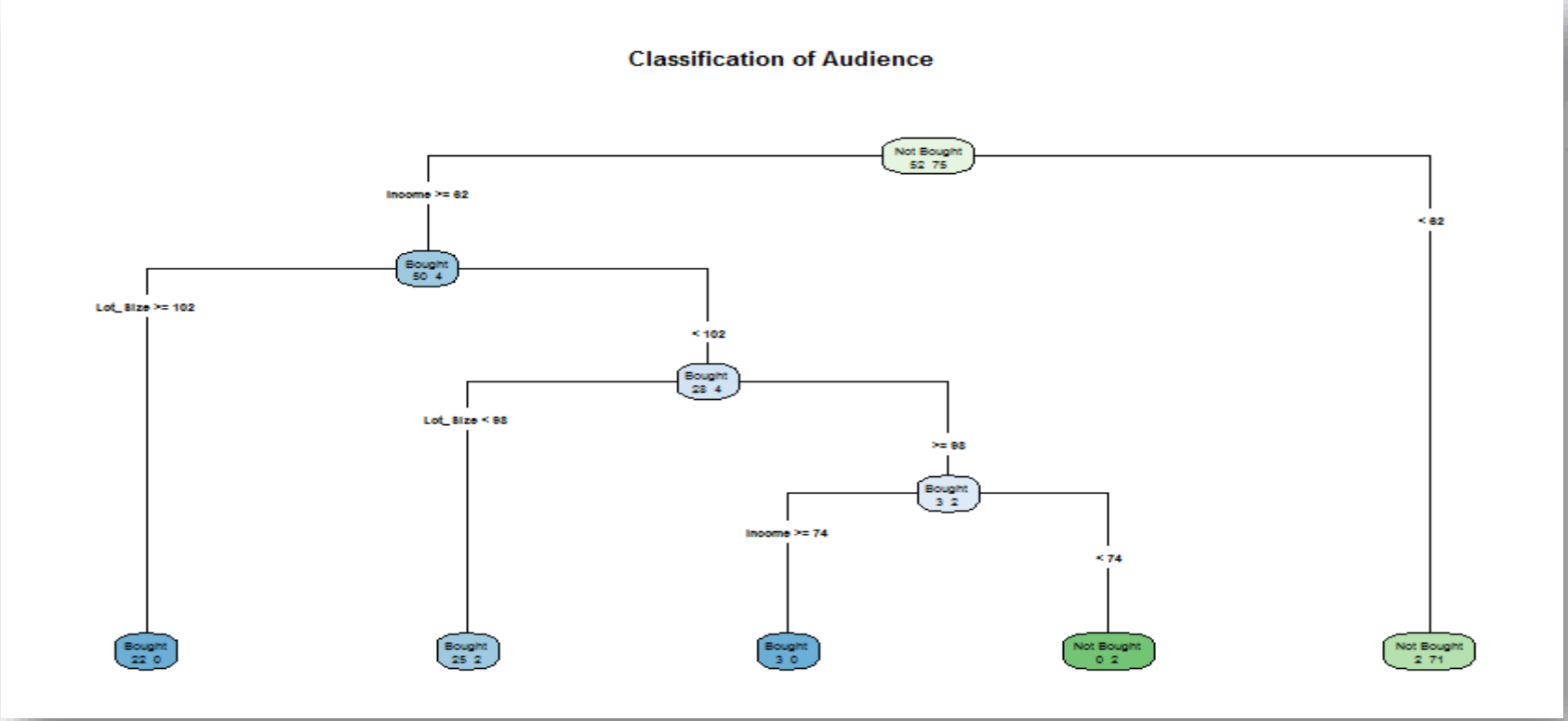
```
pred.rpart <- predict(fitRPart , newdata=validation[,-3],type=c("class"))
tblMowers <- table( pred.rpart , validation$Response)

confusionMatrix(tblMowers)
```




```
Confusion Matrix and Statistics


pred.rpart    Bought Not Bought
  Bought          18           1
  Not Bought       3          31

               Accuracy : 0.9245
                 95% CI : (0.8179, 0.9791)
    No Information Rate : 0.6038
    P-Value [Acc > NIR] : 1.498e-07

                  Kappa : 0.8396
 Mcnemar's Test P-Value : 0.6171

            Sensitivity : 0.8571
            Specificity : 0.9688
         Pos Pred Value : 0.9474
         Neg Pred Value : 0.9118
             Prevalence : 0.3962
         Detection Rate : 0.3396
   Detection Prevalence : 0.3585
      Balanced Accuracy : 0.9129

       'Positive' Class : Bought
```

Program and Output Controlling *minsplit*

```
fitRPart <- rpart(Response ~ ., method="class", data=training,
                  control = rpart.control(minsplit = 5))
```



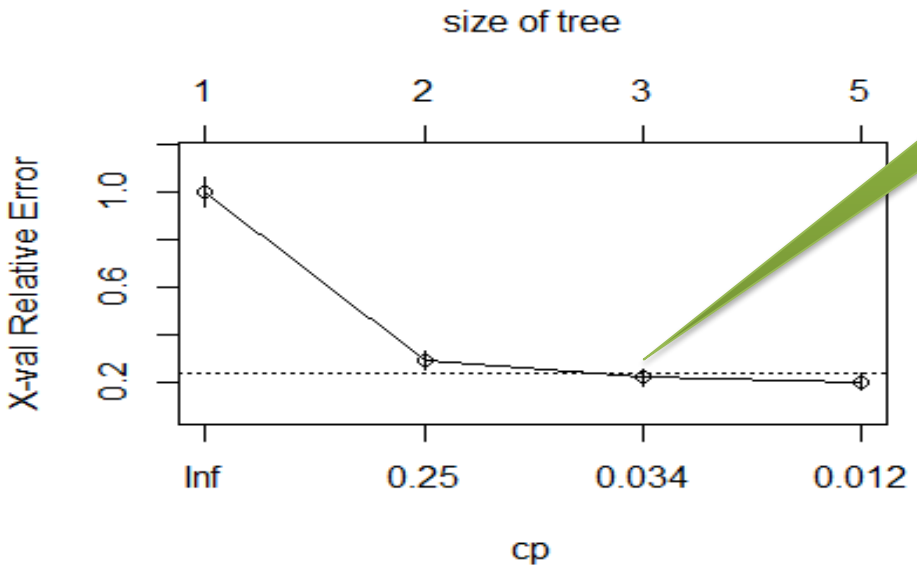
Example 2 : Wisconsin Breast Cancer

- The dataset Cancer in Folder Wisconsin contains 699 fine-needle aspirant samples where 458 (65.5%) are benign and 241 (34.5%) are malignant.
- For each sample, nine cytological characteristics previously found to correlate with malignancy are recorded. These variables are having values ranging from 1 to 10.
- Benign : Not infected ; Malignant : Infected

Program and Output

```
dtree <- rpart(class ~ ., data=training, method="class")

dtree$cptable
plotcp(dtree)
```

```
> dtree$cptable
          CP nsplit rel error    xerror       xstd
1 0.78698225      0 1.0000000 1.0000000 0.06226029
2 0.07692308      1 0.2130178 0.2899408 0.03929457
3 0.01479290      2 0.1360947 0.2189349 0.03460710
4 0.01000000      4 0.1065089 0.2011834 0.03328412
```



Stability of relative error

Pruning

- We can prune the tree by function prune( )
- We prune the tree for that cp value for which we feel that the stability in decrease of relative error has reached

Syntax : prune (ObjRPart , cp)

Where

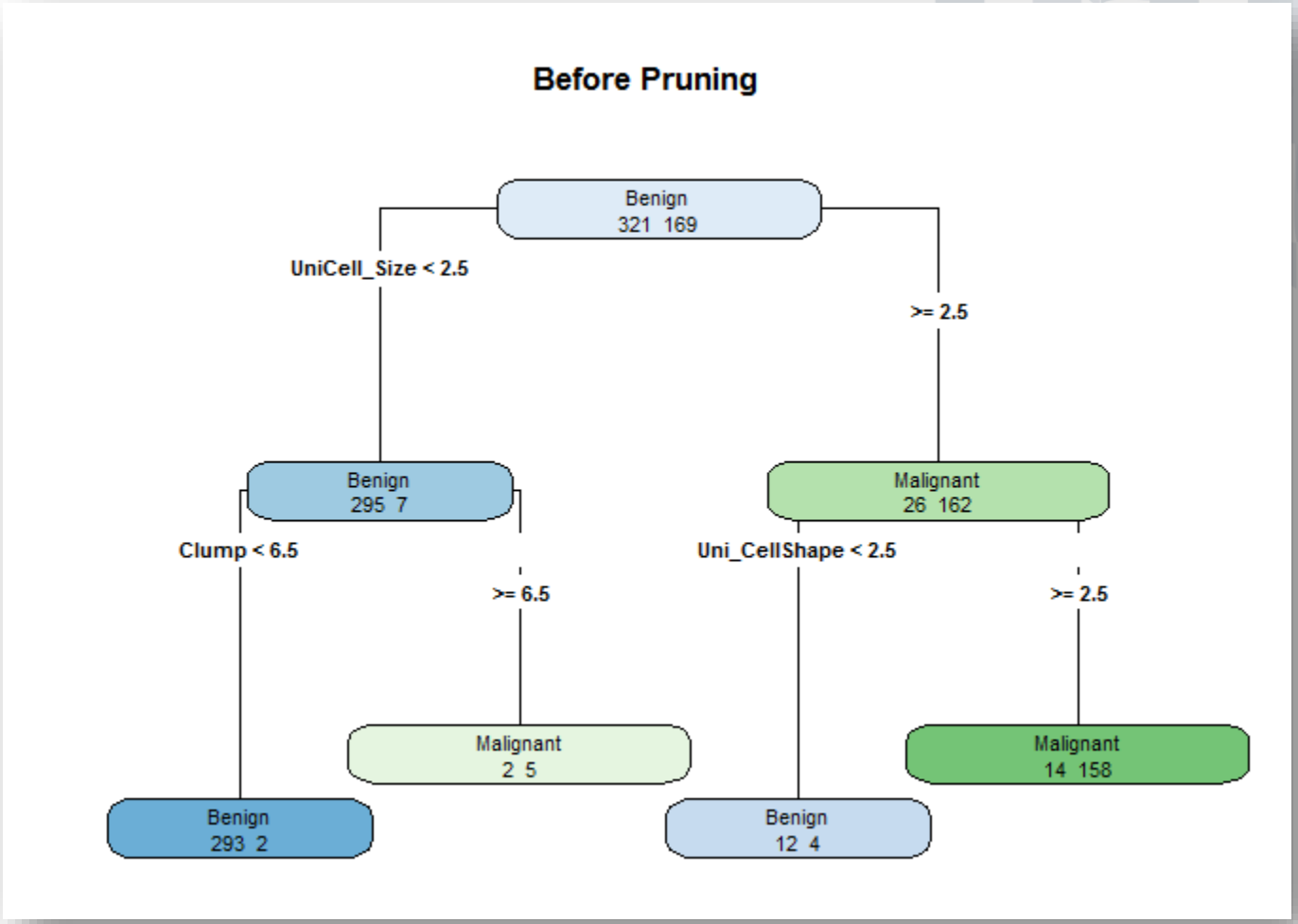
ObjRPart : rpart object

cp : cp value to be specified for pruning

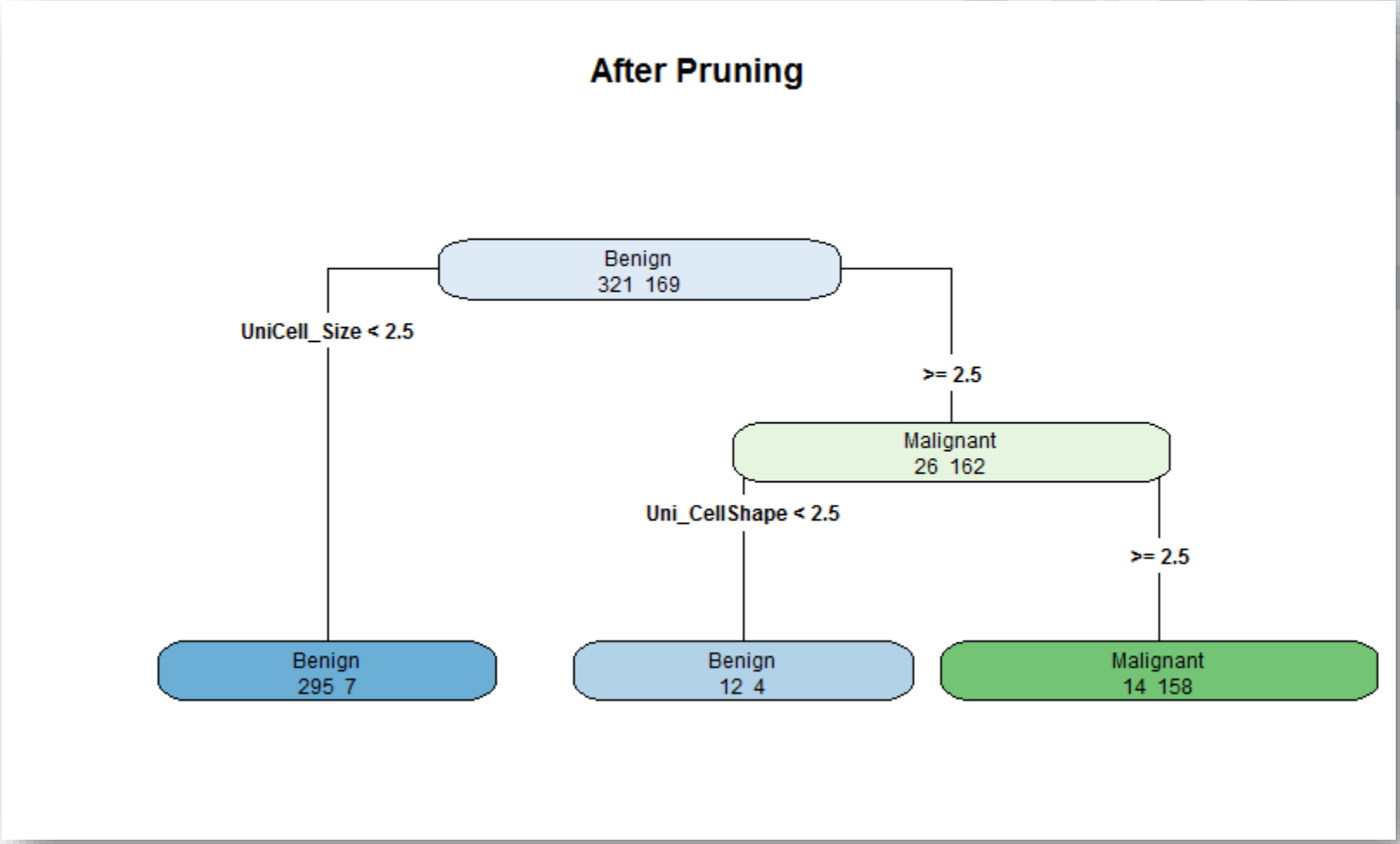
```
dtree.pruned <- prune(dtree, cp= 0.01479290)
```

Sane's STATS Academy of Statistics

Pruning Effect : Before



Pruning Effect : After



How much accuracy did we compromise?

- Not Pruned

```
Confusion Matrix and Statistics

          Actual
Predicted   Benign Malignant
  Benign       128         4
  Malignant      9        68

               Accuracy : 0.9378
                 95% CI : (0.896, 0.9665)
    No Information Rate : 0.6555
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.8645
 Mcnemar's Test P-Value : 0.2673

            Sensitivity : 0.9343
            Specificity : 0.9444
         Pos Pred Value : 0.9697
         Neg Pred Value : 0.8831
             Prevalence : 0.6555
         Detection Rate : 0.6124
   Detection Prevalence : 0.6316
      Balanced Accuracy : 0.9394

       'Positive' Class : Benign
```

- Pruned

```
Confusion Matrix and Statistics

          Actual
Predicted   Benign Malignant
  Benign       128         6
  Malignant      9        66

               Accuracy : 0.9282
                 95% CI : (0.8844, 0.9593)
    No Information Rate : 0.6555
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.8426
 Mcnemar's Test P-Value : 0.6056

            Sensitivity : 0.9343
            Specificity : 0.9167
         Pos Pred Value : 0.9552
         Neg Pred Value : 0.8800
             Prevalence : 0.6555
         Detection Rate : 0.6124
   Detection Prevalence : 0.6411
      Balanced Accuracy : 0.9255

       'Positive' Class : Benign
```

Conditional Inference Trees

- In Conditional Inference Tree, the variables and splits are selected based on significance tests rather than purity measures
- Algorithm is as follows:
  1. P-values are calculated for the relationship between each predictor and the outcome variable
  2. Predictor with lowest p-value is selected
  3. All the possible binary splits are explored on chosen predictor and response variable and most significant split is picked up
  4. The data is separated into these two groups and the process is continued for each subgroup
  5. The process is continued until splits are no longer significant or the minimum node size is reached

Package party

- Conditional Inference Tree can be implemented in R with package ***party***
- Package party provides function ctree()
- Tree can be plotted with function plot( )
- Pruning is not required in this algorithm as it is guided by statistical inference.

Syntax: ctree (formula , data , …)

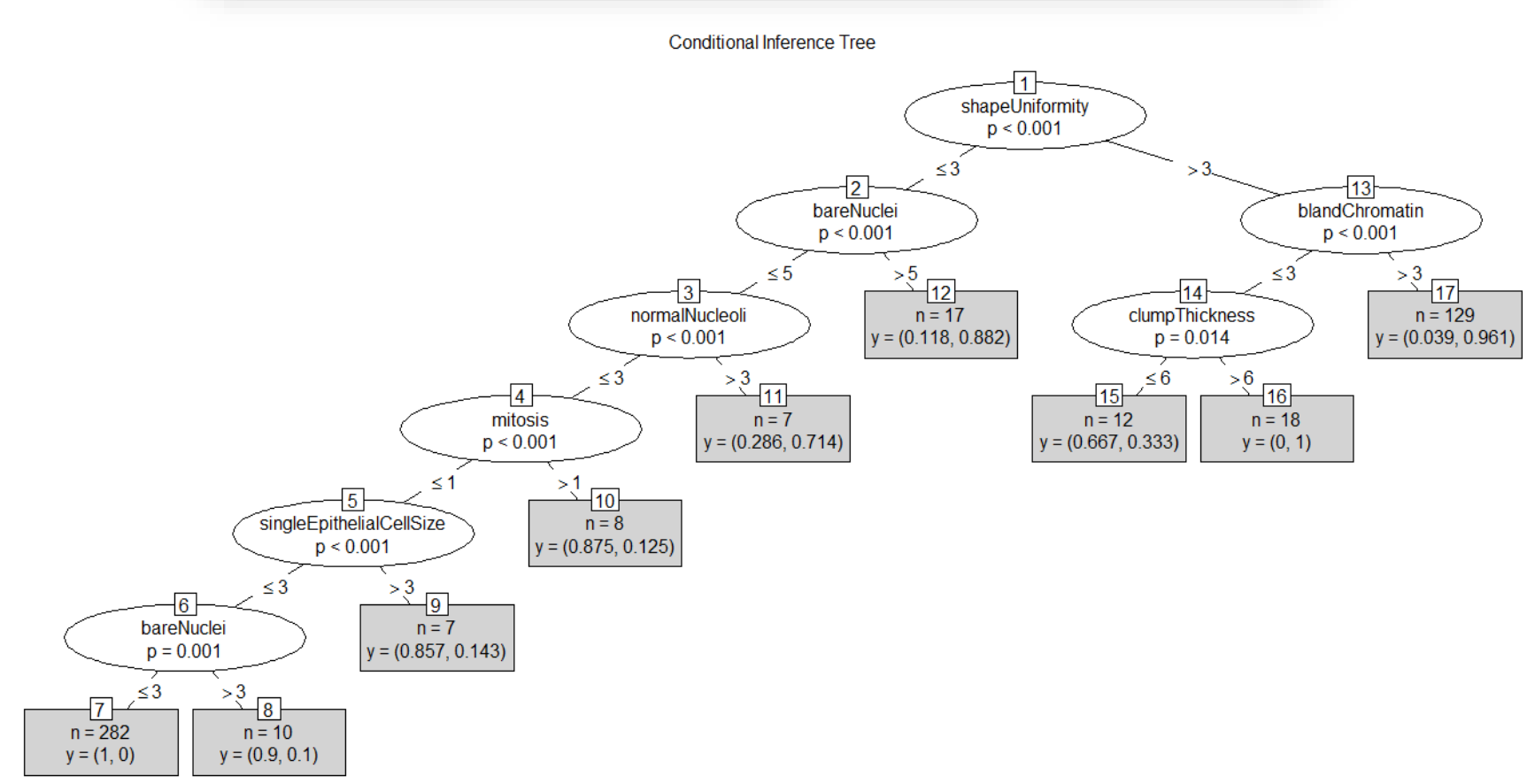
Where

formula : formula for model

data : data frame of training data

Program and Output

```
library(party)
fit.ctree <- ctree(class~., data=training)
plot(fit.ctree, main="Conditional Inference Tree",type="simple")
```



Program and Output

```
ctree.pred <- predict(fit.ctree, validation, type="response")
ctree.perf <- table( ctree.pred,validation$class ,
                     dnn=c("Predicted", "Actual"))

confusionMatrix(ctree.perf)
```

```
Confusion Matrix and Statistics

           Actual
Predicted   benign malignant
  benign       131         8
  malignant      6        64

               Accuracy : 0.933
                 95% CI : (0.8902, 0.9629)
    No Information Rate : 0.6555
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.8507
 Mcnemar's Test P-Value : 0.7893

            Sensitivity : 0.9562
            Specificity : 0.8889
         Pos Pred Value : 0.9424
         Neg Pred Value : 0.9143
             Prevalence : 0.6555
         Detection Rate : 0.6268
   Detection Prevalence : 0.6651
      Balanced Accuracy : 0.9225

       'Positive' Class : benign
```