Seminar 7

Classification I

Overview

In this tutorial document, we will be presenting a classification decision tree with the *rpart* (Recursive Partitioning and Regression Trees) package. Let's install the *rpart* package and activate the package.

> install.packages("rpart")

> library(rpart)

The *rpart* algorithm works by splitting the dataset recursively, which means that the subsets that arise from a split are further split until a pre-defined termination criterion is reached. At each step, the split is made based on the attribute that results in the largest possible reduction in heterogeneity of the target (outcome) variable.

Splitting rules can be constructed in many different ways, all of which are based on the notion of impurity – a measure of the degree of heterogeneity of the leaf nodes. A leaf node that contains a single class is homogenous and has impurity = 0. The *rpart* offers the *Entropy* and *Gini Index* methods as choices (see the lecture slides).

Data

We will use segmentationData from the *caret* package. You don't need to import any data from you project folder for this exercise. Let's install the caret package. We will use this package for presenting a classification tree in the next seminar.

> install.packages("caret")

> library(caret) # Activate the caret package

> data(segmentationData) # Load the data

You can find the descriptions of the variables from the caret package.

> ?segmentationData

The data has a variable "Case" to separate the training set and testing set. We do not need the Case variable for this exercise. Delete the Case variable from the data and rename the data as 'ClassData'

```
> # ! Delete one column #
> ClassData <- segmentationData[, !(colnames(segmentationData) == 'Case') ]
```

Creating Training Set and Testing Set

Next, we have to separate the data into training and testing sets. We will use the former to build the model and the latter to test it. To do this, we randomly select 80% of the data for the training set and assign the remainder to the testing set.

The following commands use the sample ( ) function to select 80% observations at random out of the total observations (i.e., num_obs * 0.8). Make sure that the set.seed() function uses the arbitrary value 123 to reproduce the same subsets for any later purposes. Omitting this seed will cause your training and testing split to differ at every use of sample ().

```
> set.seed(123)
>
> num_obs <- nrow(ClassData)
> train_size <- num_obs * 0.8
> train_sample <- sample(num_obs, train_size)
```

By using this vector to select rows from the ClassData, we can split it into the 80% training and 10 percent test datasets. Recall that the dash (-) operator used to select records that are not in the specified rows. That is, the test data includes only the rows that are not in the training set.

```
> Class_Train <- ClassData[train_sample, ]
> Class_Test  <- ClassData[-train_sample, ]
```

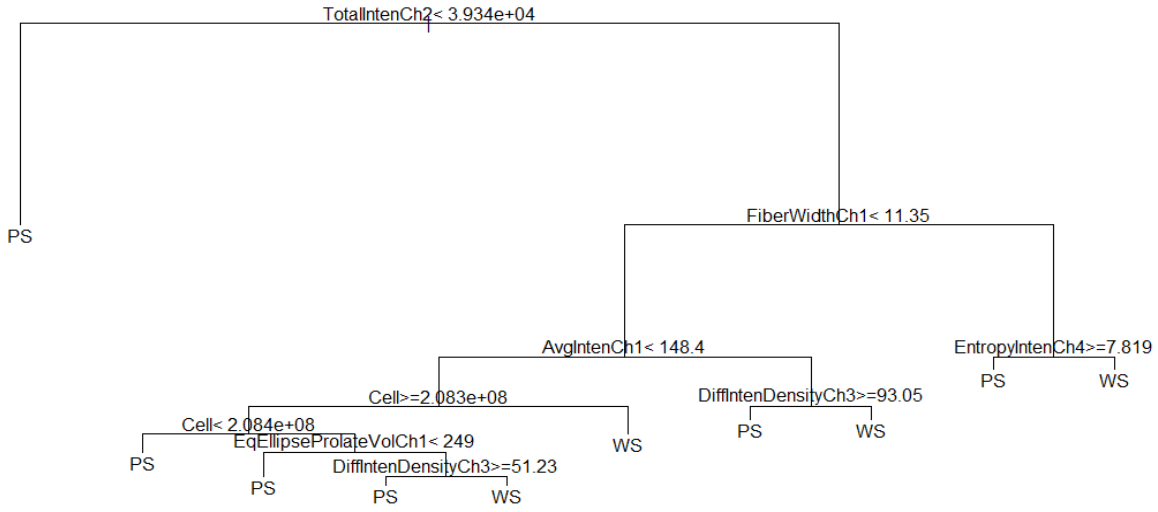
Classification with a Decision Tree

We will use the *rpart* function to train our decision tree model. Make sure to install and activate the *rpart* package. I strongly recommend you take some time to review the documentation (i.e., help(rpart)) and understand the parameters and their default values.

Let's first fit a tree with *rpart*. Note that we set the method parameter to "class", which simply tells the algorithm that the target (outcome) variable is discrete. Finally, *rpart* uses Gini Index for splitting by default, and we will stick with this option.

```
> # build a model
> rpart_model <- rpart(Class ~ ., data = Class_Train, method="class")
>
> # plot tree
> plot(rpart_model); text(rpart_model)
```

The resulting plot is shown in the following figure. It is quite self-explanatory, so I won't explain it.

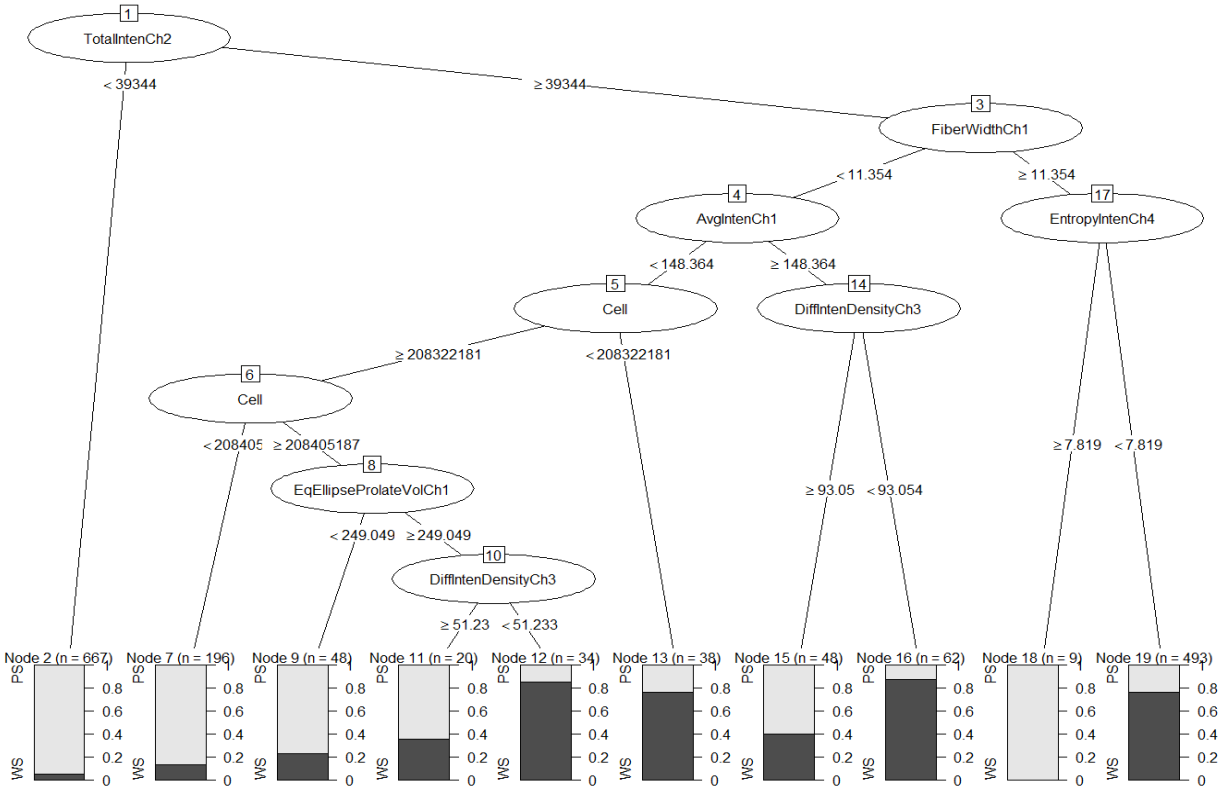


The partykit package provides better plots of classification trees. Install and activate the package.

> install.packages("partykit")

> library(partykit)

> plot(as.party(rpart_model))



Model Validation

Next, we see how good the model is by seeing how it fares against the test data. The predict() function will return a vector of predicted class values (type="class").

```
> rpart_predict <- predict(rpart_model, Class_Test, type="class")
> mean(rpart_predict == Class_Test$Class)
[1] 0.7920792
```

Let's create a confusion matrix.

```
> confusionMatrix(rpart_predict, Class_Test$Class)
Confusion Matrix and Statistics

          Reference
Prediction  PS  WS
        PS 203  23
        WS  61 117

               Accuracy : 0.7921
                 95% CI : (0.7492, 0.8306)
    No Information Rate : 0.6535
    P-Value [Acc > NIR] : 7.570e-10

                  Kappa : 0.5684
 Mcnemar's Test P-Value : 5.413e-05

            Sensitivity : 0.7689
            Specificity : 0.8357
         Pos Pred Value : 0.8982
         Neg Pred Value : 0.6573
             Prevalence : 0.6535
         Detection Rate : 0.5025
   Detection Prevalence : 0.5594
      Balanced Accuracy : 0.8023

       'Positive' Class : PS
```

Note that you may have the different results from the output above, as you are using different training and test sets (from random samplings). You will learn how to read the confusion matrix in the coming class. You can also check out the lecture slides beforehand.

In the next tutorial, you will learn how to prune the tree and to reduce the likelihood of overfitting.