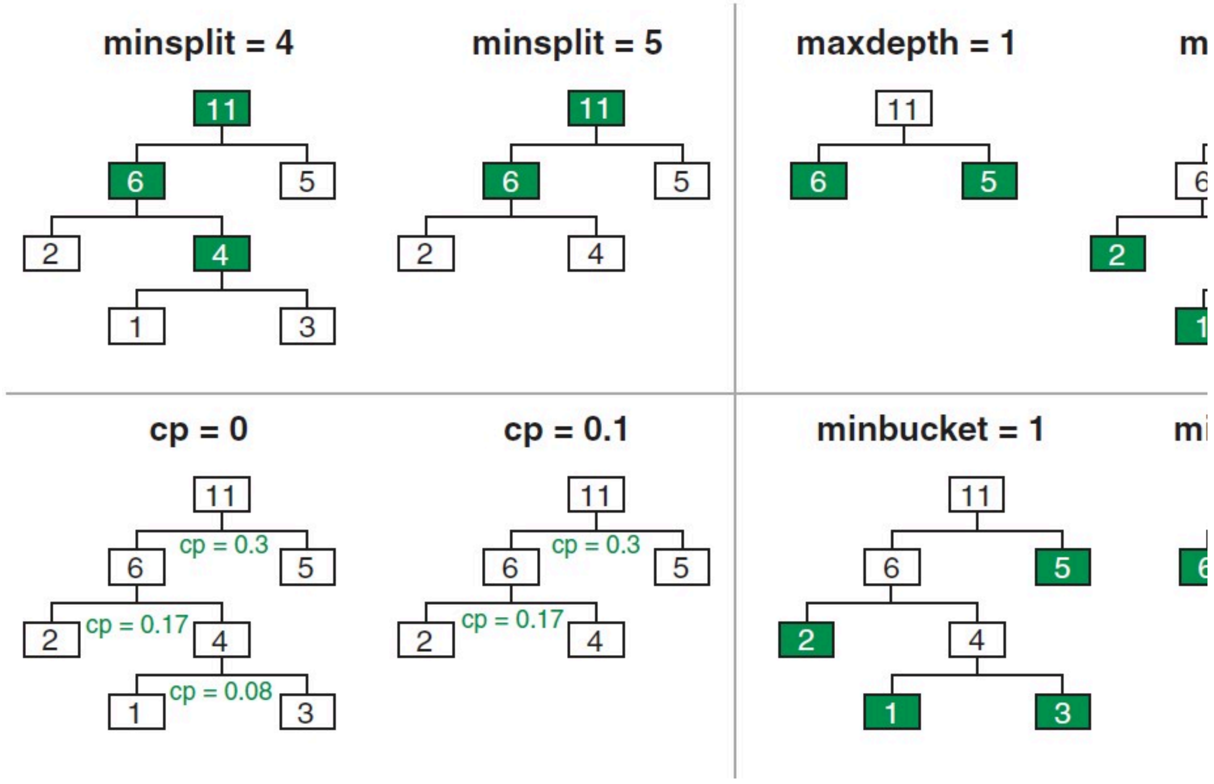
Machine Learning - 1100-ML0ENG (Ćwiczenia informatyczne Z-23/24)

Parameters to fit decision trees

These are the important control parameters that are used by rpart to grow a tree.

- **cp parameters**: The cp (complexity parameter) is an option used in rpart that controls how liberal you want the splitting algorithm to be when it decides whether or not to split a node. Utilizing a small cp value (< 0.01) can generate an enormous tree that can contain unexplainable splits, while a tree with a high cp value (> 0.05) can produce a tree containing only obvious information. Therefore, it is important to set the cp level at a number that generates a fair amount of nodes, yet remains explainable.
- **maximum depth:** If you want the number of levels of the tree to stop at a certain level, specify this number as an option within the function call. Some people will stop a decision tree after 10 levels or so, assuming that there will be no interpretable information beyond that point. However, if you suspect that there may be single nodes that pop up that could contain the information you are looking for, by all means continue to grow the tree. However, remember as you extend the number of levels, that will increase the computation time and memory needed. It is always a good idea to see limits on the growth of the tree, even if it is a high number.

- **minbucket:** Many analysts like to concentrate on terminal nodes. However, a terminal node needs to be a minimum size for it to have statistical meaning. The only exception would be if you were looking for true outliers or anomalies. But generally, for segmentation, we want groups that are large enough to have meaning.
- **minsplit**: This option controls how many observations a node has to have in order to even be considered for a split.



Observe how different parameters affect the appearance of the tree and the correctness of classification in the confiusion matrix.

```
#minsplit
control=rpart.control(minsplit=52)
model.52 <- rpart(Purchased ~ .,data = training_set, control=control)
fancyRpartPlot(model.52)
model.52.pred<-predict(model.52, test_set ,type = "class")
table(test_set$Purchased,model.52.pred)
```

```
#minbucket
control=rpart.control(minbucket=37)
model.37 <- rpart(Purchased ~ .,data = training_set,control=control)
fancyRpartPlot(model.37)
model.37.pred<-predict(model.37, test_set ,type = "class")
table(test_set$Purchased,model.37.pred)
```

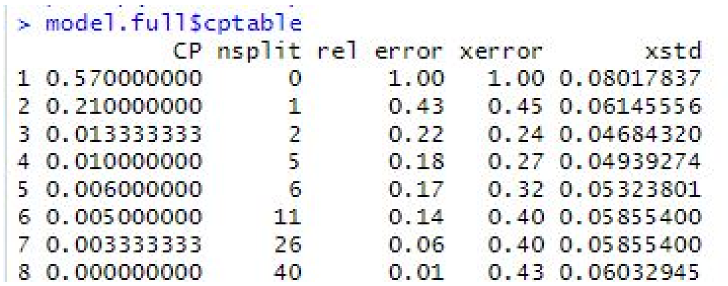
```
#minsplit minbucket
control=rpart.control(minsplit=10, minbucket =3 )
model.103 <- rpart(Purchased ~ .,data = training_set, control=control)
fancyRpartPlot(model.103)
model.103.pred<-predict(model.103, test_set ,type = "class")
table(test_set$Purchased,model.103.pred)
```

```
control=rpart.control(minsplit=253,minbucket = 80)
model.25380 <- rpart(Purchased ~ .,data = training_set, control=control)
fancyRpartPlot(model.25380)
model.25380.pred<-predict(model.25380, test_set ,type = "class")
table(test_set$Purchased,model.25380.pred)
In the following example, we build a full decision tree with cp = and minbucket =
equal to zero.
```

```
control <- rpart.control(cp=0, minbucket=0)
model.full <- rpart(Purchased ~ .,data = training_set,control=control)
model.full
fancyRpartPlot(model.full)
model.full.pred<-predict(model.full , test_set ,type = "class")
table(test_set$Purchased,model.full.pred)
acc(test_set$Purchased,model.full.pred)
```

```
#cptable
```

```
model.full$cptable
```



```
> model.full$cptable
           CP nsplit rel error xerror       xstd
1 0.570000000      0      1.00   1.00 0.08017837
2 0.210000000      1      0.43   0.45 0.06145556
3 0.013333333      2      0.22   0.24 0.04684320
4 0.010000000      5      0.18   0.27 0.04939274
5 0.006000000      6      0.17   0.32 0.05323801
6 0.005000000     11      0.14   0.40 0.05855400
7 0.003333333     26      0.06   0.40 0.05855400
8 0.000000000     40      0.01   0.43 0.06032945
```

- The first column, labelled **CP**, is the value of the cp parameter.
- The second column, **nsplit**, is the number of splits in the tree.
- The **rel error** denotes the relative error associated with the mean squared error (the root of the difference between the value and the modelled value) and the number of splits.
- Both **xerror** and **xstd** are based on tenfold cross-validation, with xerror being the mean error and xstd being the standard deviation of the cross-validation process.

Last modified: wtorek, 19 grudnia 2023, 10:25

Accessibility settings