workshop10

February 3, 2019

```
In [48]: library(tidyverse)
         library(tree)
         library(rpart)
         library(rpart.plot)
         library(caret)

         install.packages('precrec',lib='.', verbose=TRUE)
         library(precrec,lib.loc='.')
```



```


         # Plot size depending on your screen resolution to 5 x 3
         options(repr.plot.width=6, repr.plot.height=6)
```



```
system (cmd0): /usr/lib/R/bin/R CMD INSTALL
foundpkgs: precrec, /tmp/Rtmp2fiDjC/downloaded_packages/precrec_0.9.1.tar.gz
files: /tmp/Rtmp2fiDjC/downloaded_packages/precrec_0.9.1.tar.gz
1): succeeded '/usr/lib/R/bin/R CMD INSTALL -l '/srv/home/whtam4' /tmp/Rtmp2fiDjC/downloaded_pac
```


1 Welcome to Workshop 10

1.0.1 Exercise 1: Build Classification Trees

Read in the `Mower01.csv` file. Since there are only 24 observations in the data set, print the whole data set. Notice that the variable Owner in the data frame Mower takes the values 1 and 0. 1 means the relevant household owns a ride-on mower, while 0 means it does not. Create a factor variable out of `Owner`. Hint: `labels = c("Noowner", "Owner")`

```
In [14]: mower.df <- read.table(file = "Mower01.csv",
                                header = TRUE,
                                sep = ",")
         mower.df$Owner <- factor(mower.df$Owner, levels=c(0,1), labels = c("Noowner", "Owner"))
         mower.df
```

HH_ID	Income	Lot_Size	Owner
1	60.0	18.4	Owner
2	85.5	16.8	Owner
3	64.8	21.6	Owner
4	61.5	20.8	Owner
5	87.0	23.6	Owner
6	110.1	19.2	Owner
7	108.0	17.6	Owner
8	82.8	22.4	Owner
9	69.0	20.0	Owner
10	93.0	20.8	Owner
11	51.0	22.0	Owner
12	81.0	20.0	Owner
13	75.0	19.6	Noowner
14	52.8	20.8	Noowner
15	64.8	17.2	Noowner
16	43.2	20.4	Noowner
17	84.0	17.6	Noowner
18	49.2	17.6	Noowner
19	59.4	16.0	Noowner
20	66.0	18.4	Noowner
21	47.4	16.4	Noowner
22	33.0	18.8	Noowner
23	51.0	14.0	Noowner
24	63.0	14.8	Noowner

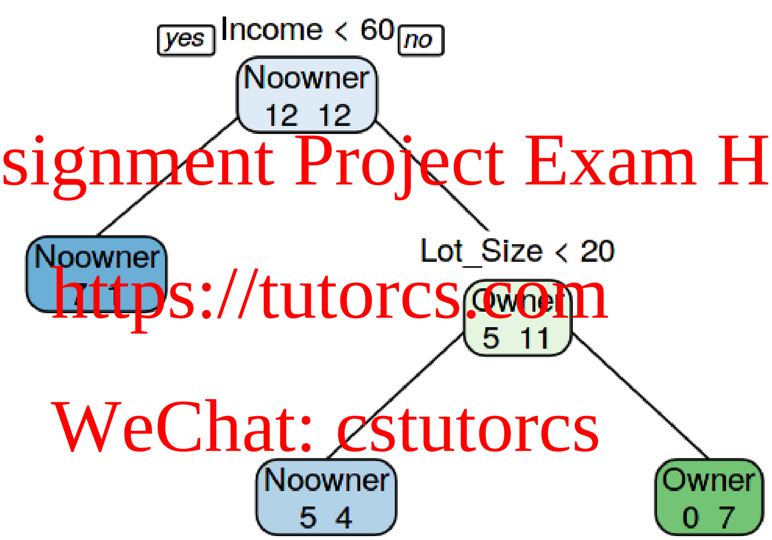
Let's now build a tree with only `Income` and `Lot_Size` as predictors.

```
In [15]: tree_model<-rpart(Owner ~ Income + Lot_Size,
                           data=mower.df,
                           method="class",
                           control=rpart.control(minsplit  = 5, #the minimum number of observatio
                                                 minbucket = 5, # the minimum number of observati
                                                 xval = 1)) # to use all samples for the first tr

         tree_model
```

```
n= 24

node), split, n, loss, yval, (yprob)
      * denotes terminal node

1) root 24 12 Noowner (0.5000000 0.5000000)
  2) Income< 59.7 8  1 Noowner (0.8750000 0.1250000) *
  3) Income>=59.7 16  5 Owner (0.3125000 0.6875000)
    6) Lot_Size< 19.8 9  4 Noowner (0.5555556 0.4444444) *
    7) Lot_Size>=19.8 7  0 Owner (0.0000000 1.0000000) *
```

Let's plot the tree, this is one of the few times we make use of the R-base graphics

```
In [16]: rpart.plot(tree_model,
                    fallen.leaves = FALSE, # to position the leaf nodes at the bottom of the gra
                    type  = 1, # 1 Label all nodes, not just leaves.
                    extra = 1, # 1 Display the number of observations that fall in the node
                    split.font = 1, # Font for the split labels. 1=normal 2=bold
                    varlen = -10) # Length of variable names in text at the splits (and, for cla
```



- Your plotted tree should have 3 leaf nodes and three decision nodes.
- Note that at each decision node, as regards the answer to the question at the node, Yes is to the left and No is to the right.
- You can see the number of observation at each leaf node that represent the actual observations of each partition

Using the tree to "predict" the value of Owner in the dataset Mower Having built and graphed the model, use it to predict the value of the target variable for all the cases in the original data frame. This is done as follows:

```
In [17]: treePred.class <- predict(tree_model, data = Mower, type = "class")
         head(treePred.class)
```

1 Noowner **2** Noowner **3** Owner **4** Owner **5** Owner **6** Noowner
Levels: 1. 'Noowner' 2. 'Owner'

Thus the arguments of predict are: * tree_model , the tree being used for the prediction; * Mower, the data from which the prediction is being made. For this example, we are using the data that was used to build the tree, but there are good reasons to use other data here that has not been used in constructing the tree. See later. * Since this is a classification rather than a regression tree, the type is "class" * The predicted classes got created assuming a **cutoff** of 0.5 of the score.

This prediction though, represent an underlying score cut off. Compare the result below and above. To see the predicted score use:

```
In [18]: treePred.score <- predict(tree_model, data = Mower, type = "prob")
         head(treePred.score)
```


	Noowner	Owner
1	0.5555556	0.4444444
2	0.5555556	0.4444444
3	0.0000000	1.0000000
4	0.0000000	1.0000000
5	0.0000000	1.0000000
6	0.5555556	0.4444444

It is common to analyse the score in terms of the 1 Class, in our case **Owner**

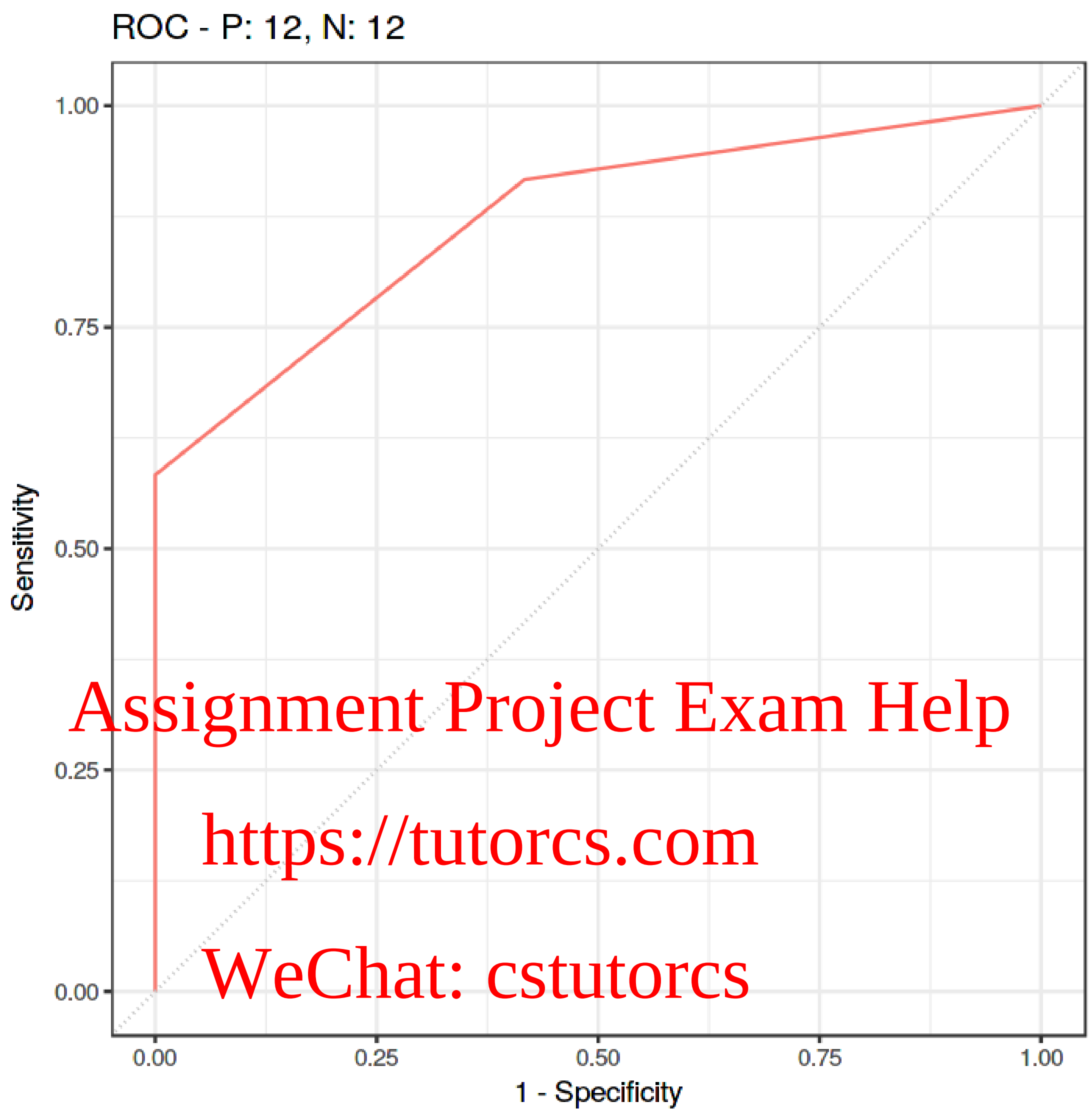

```
In [19]: ownerScore <- treePred.score[,2]
         head(ownerScore)
```

1 0.444444444444444 **2** 0.444444444444444 **3** 1 **4** 1 **5** 1 **6** 0.444444444444444

A common way of plotting the predicted score is the Receiver Operating Characteristic Curve **ROC** Curve. The area under the curve **AUC** is a possible measure of comparision. Higher number means better accuracy. This is a good measure if you are interested to be highly accurate across all cases. If you are only interested in a few cases, like the ones you are the most sure about (highest score), the criteria is called **top k preciscion**. We will discuss this later. To plot an ROC curve:

```
In [20]: sscurves <- evalmod(scores = ownerScore, labels = mower.df$Owner)
         autoplot(sscurves, "ROC")
         auc(sscurves) %>% filter(curvetypes=='ROC')
```

modnames	dsids	curvetypes	aucs
m1	1	ROC	0.8715278



The default cutoff is 0.5. Depending on your problem you are trying to solve, a different cutoff results in higher accuracy. To calculate it with 0.5:

```
In [21]: confusionMatrix(mower.df$Owner,factor( ifelse(ownerScore > 0.5, "Owner", "Noowner") ),p
```

```
Confusion Matrix and Statistics

          Reference
Prediction Noowner Owner
   Noowner      12     0
   Owner         5     7

               Accuracy : 0.7917
                 95% CI : (0.5785, 0.9287)
```

```
    No Information Rate : 0.7083
    P-Value [Acc > NIR] : 0.25644

                  Kappa : 0.5833
 Mcnemar's Test P-Value : 0.07364

            Sensitivity : 1.0000
            Specificity : 0.7059
         Pos Pred Value : 0.5833
         Neg Pred Value : 1.0000
             Prevalence : 0.2917
         Detection Rate : 0.2917
   Detection Prevalence : 0.5000
      Balanced Accuracy : 0.8529

       'Positive' Class : Owner
```

Or 0.75:


```
In [22]: confusionMatrix(mower.df$Owner,factor( ifelse(ownerScore > 0.3, "Owner", "Noowner") ),p
```

```
Confusion Matrix and Statistics
```



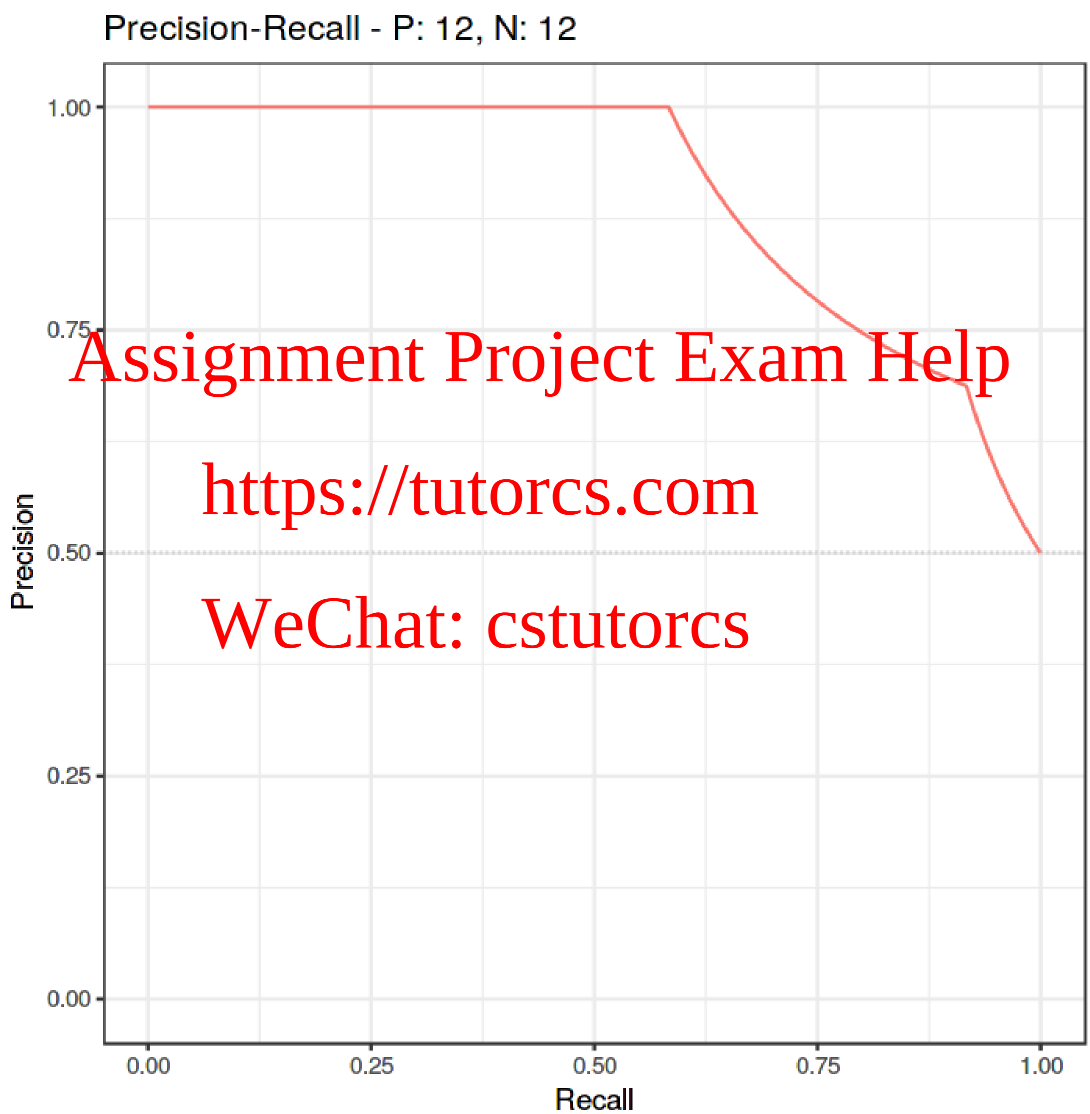
```
          Reference
Prediction Noowner Owner
   Noowner       7     5
   Owner         1    11
```



```
               Accuracy : 0.75
                 95% CI : (0.5329, 0.9023)
    No Information Rate : 0.6667
    P-Value [Acc > NIR] : 0.2632

                  Kappa : 0.5
 Mcnemar's Test P-Value : 0.2207

            Sensitivity : 0.6875
            Specificity : 0.8750
         Pos Pred Value : 0.9167
         Neg Pred Value : 0.5833
             Prevalence : 0.6667
         Detection Rate : 0.4583
   Detection Prevalence : 0.5000
      Balanced Accuracy : 0.7812

       'Positive' Class : Owner
```

Let's say we want to reach out only to those owners we are very sure about owning a mower; the purpose is to offer them an upgrade. As our marketing team has limited resources, we cannot contact all 24 but can contact 5. In this case, we are interested in the precision at top k, where k equals 5. The precision-recall curve plot helps us with that:

```
In [23]: autoplot(sscurves, "PRC")
```



You can see above that if we rank all our predictions in terms of score, and look at the first ones, we would be correct. Note: The reason is, as this is not a real test, we are using the same data we used for learning the tree for testing.

1.0.2 Exercise 2: Cross-validation

In the current example, we have just 24 observations. We want to create a tree and test how well it does on test data that was not used in the construction. We will demonstrate how to choose the size of the tree by performing 4-fold cross-validation.

The average across folds yields a more stable result than just using the validation set once. We divide the data into four groups of 6 observations each. Call them A, B, C, and D. We then use only the 18 observations from B, C, and D in choosing trees pruned to various sizes. We test how well these trees do at predicting the observations in A. This is one "fold". We then use the 18 observations in A, C and D, and test how well these trees do in predicting the observations in B. This is another "fold". And so on.

In this case, we do not have a time component, so we can randomly split up the data. To make sure that we are having reproducable results, set the random seed.

```
In [49]: set.seed(3800)
         cv.ct<-rpart(Owner ~ Income + Lot_Size,
                         data=mower.df,
                         method="class",

                         control=rpart.control(minsplit  = 2, #the minimum number of observatio
                                               minbucket = 4, # the minimum number of observati
                                               xval = 4))     # number of cross-validations.
```


To print the result:


```
In [50]: printcp(cv.ct)
```



```
Classification tree:
rpart(formula = Owner ~ Income + Lot_Size, data = mower.df, method = "class",
    control = rpart.control(minsplit = 2, minbucket = 4, xval = 4))

Variables actually used in tree construction:
[1] Income   Lot_Size

Root node error: 12/24 = 0.5

n= 24

     CP nsplit rel error  xerror    xstd
1 0.500      0      1.00 1.33333 0.19245
2 0.125      1      0.50 1.16667 0.20127
3 0.010      3      0.25 0.91667 0.20341
```

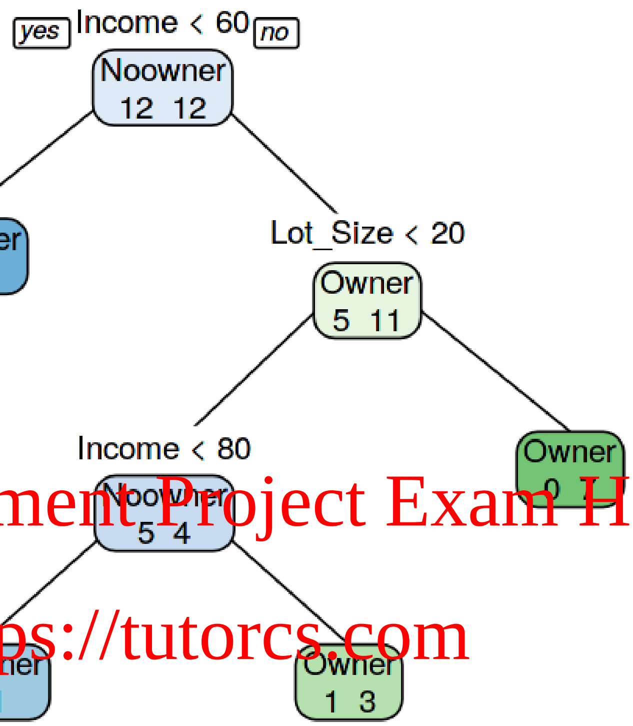
We are looking for the configuration with the lowers cross validation error, column `xerror`.

```
In [53]: pruned.ct <- prune(cv.ct, cp = 0.125)
         pruned.ct
```

```
n= 24

node), split, n, loss, yval, (yprob)
      * denotes terminal node

1) root 24 12 Noowner (0.5000000 0.5000000)
  2) Income< 59.7 8  1 Noowner (0.8750000 0.1250000) *
  3) Income>=59.7 16  5 Owner (0.3125000 0.6875000) *
```

And plot it again:

```
In [52]: rpart.plot(pruned.ct,
                    fallen.leaves = FALSE, # to position the leaf nodes at the bottom of the gra
                    type  = 1, # 1 Label all nodes, not just leaves.
                    extra = 1, # 1 Display the number of observations that fall in the node
                    split.font = 1, # Font for the split labels. 1=normal 2=bold
                    varlen = -10) # Length of variable names in text at the splits (and, for cla
```



There are a lot of packages in R with different cross validation strategies, this is just one of them.

1.0.3 Exercise 3: Validate Clusters

Classification trees can also be used to make sense and check the realibility of clustering algorithm. Perfom the following steps: 1. Load the `"KTC.csv"` as in workshop 8 2. Perform Kmeans clustering with 3 clusters on the numeric columns (don't forgett scaling) 3. Calculate the average values column by cluster as reference 4. Create a new data frame based on the unscaled values and add the cluster number for each row 5. Fit a classification tree with the target the cluster number. (Note: To specify all columns to be used use `k3 ~ .` as formula 6. Interpret the results. How does it differ from the average values ? Discuss with your table

```
In [ ]: # your code here
        fail() # No Answer - remove if you provide an answer
```