BUSN41204 Week 2 Review Session

Simiao Jiao

1/14/23

Schedule

1. Tree Classification Example 1 (Universal Bank)
2. Tree Classification Example 2 (Membership Data)

1 Tree Classification Example 1 (Universal Bank)

A young bank wants to explore ways of converting its liability (deposit) customers to personal loan customers. A campaign the bank ran for liability customers showed a healthy conversion rate of over 9% successes. This has encouraged the retail marketing department to devise smarter campaigns with better target marketing. The goal of our analysis is to model the previous campaign's customer behavior to analyze what combination of factors make a customer more likely to accept a personal loan. This will serve as the basis for the design of a new campaign.

Column Name	Description
ID	Customer ID
Age	Customer's age in completed years
Experience	#years of professional experience
Income	Annual income of the customer ($000)
ZIPCode	Home Address ZIP code.
Family	Family size of the customer
CCAvg	Avg. spending on credit cards per month ($000)
Education	Education Level. 1: Undergrad; 2: Graduate; 3: Advanced/Professional
Mortgage	Value of house mortgage if any. ($000)
Personal Loan	Did this customer accept the personal loan offered in the last campaign?
Securities Account	Does the customer have a securities account with the bank?
CD Account	Does the customer have a certificate of deposit (CD) account with the bank?
Online	Does the customer use internet banking facilities?
CreditCard	Does the customer use a credit card issued by UniversalBank?

Analysis

Loading libraries

```
library(rpart)
library(rpart.plot)
library(caret)    # used for confusionMatrix
```

Loading data

```
bank.df <- read.csv("UniversalBank.csv")
dim(bank.df)
```

```
## [1] 5000   14
```

Drop ID and zip code columns.

```
bank.df <- bank.df[ , -c(1, 5)]
bank.df$Personal.Loan <- as.factor(bank.df$Personal.Loan)
```

Partition 5000 observations randomly into training (3000 records) and validation (2000 records).

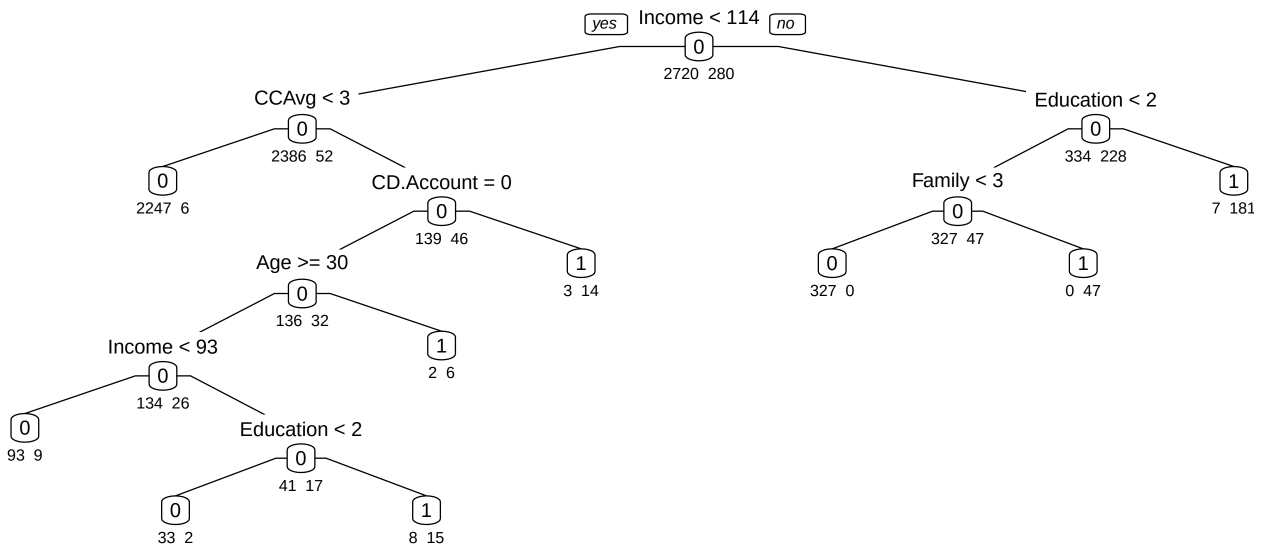
```
set.seed(131)
train.index <- sample(c(1:dim(bank.df)[1]), dim(bank.df)[1]*0.6)
train.df <- bank.df[train.index, ]
valid.df <- bank.df[-train.index, ]
```

Build a classification tree

```
default.ct <- rpart(Personal.Loan ~ ., data = train.df, method = "class")
```

Plot tree

```
prp(default.ct, type = 1, extra = 1, under = TRUE, split.font = 1, varlen = -10)
```



Let us investigate a larger tree

```
deeper.ct <- rpart(Personal.Loan ~ ., data = train.df, method = "class", cp = 0, minsplit = 1)
```

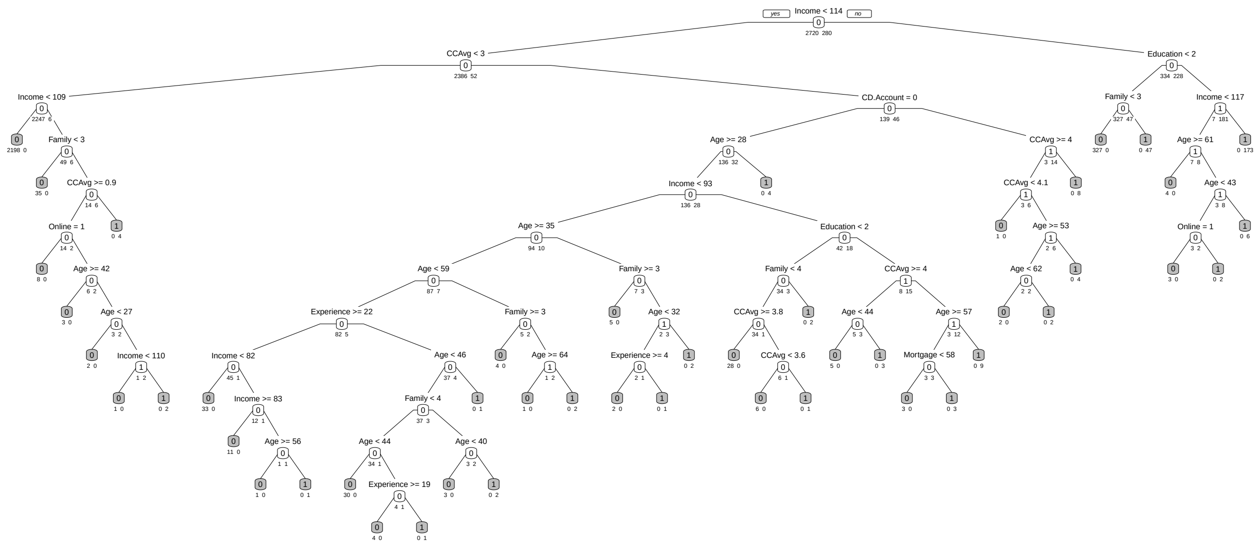
Count number of leaves

```
length(deeper.ct$frame$var[deeper.ct$frame$var == "<leaf>"])
```

```
## [1] 47
```

Plot tree

```
prp(deeper.ct, type = 1, extra = 1, under = TRUE, split.font = 1, varlen = -10,
      box.col=ifelse(deeper.ct$frame$var == "<leaf>", 'gray', 'white'))
```



Compare predictions from both trees on both the training and validation data. We need to set argument `type = "class"` in `predict()` to generate predicted class membership.

Generate confusion matrix for training data

```
default.ct.point.pred.train <- predict(default.ct, train.df, type = "class")
deeper.ct.point.pred.train <- predict(deeper.ct, train.df, type = "class")
cm.default.train <- confusionMatrix(default.ct.point.pred.train, train.df$Personal.Loan)
cm.deeper.train <- confusionMatrix(deeper.ct.point.pred.train, train.df$Personal.Loan)
print(cm.default.train)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 2700   17
##          1   20  263
##
##                Accuracy : 0.9877
##                  95% CI : (0.983, 0.9913)
##     No Information Rate : 0.9067
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9275
##
##  Mcnemar's Test P-Value : 0.7423
##
##             Sensitivity : 0.9926
##             Specificity : 0.9393
##          Pos Pred Value : 0.9937
##          Neg Pred Value : 0.9293
##              Prevalence : 0.9067
##          Detection Rate : 0.9000
##    Detection Prevalence : 0.9057
##       Balanced Accuracy : 0.9660
##
##        'Positive' Class : 0
##
```

```
print(cm.deeper.train)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 2720    0
##          1    0  280
##
##                Accuracy : 1
##                  95% CI : (0.9988, 1)
##     No Information Rate : 0.9067
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0000
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 1.0000
##              Prevalence : 0.9067
##          Detection Rate : 0.9067
##    Detection Prevalence : 0.9067
##       Balanced Accuracy : 1.0000
##
##        'Positive' Class : 0
##
```

Generate confusion matrix for validation data

```
default.ct.point.pred.valid <- predict(default.ct, valid.df, type = "class")
deeper.ct.point.pred.valid <- predict(deeper.ct, valid.df, type = "class")
cm.default.valid <- confusionMatrix(default.ct.point.pred.valid, valid.df$Personal.Loan)
cm.deeper.valid <- confusionMatrix(deeper.ct.point.pred.valid, valid.df$Personal.Loan)
print(cm.default.valid)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1778   14
##          1   22  186
##
##                Accuracy : 0.982
##                  95% CI : (0.9752, 0.9874)
##     No Information Rate : 0.9
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9017
##
##  Mcnemar's Test P-Value : 0.2433
##
##             Sensitivity : 0.9878
##             Specificity : 0.9300
##          Pos Pred Value : 0.9922
##          Neg Pred Value : 0.8942
##              Prevalence : 0.9000
##          Detection Rate : 0.8890
##    Detection Prevalence : 0.8960
##       Balanced Accuracy : 0.9589
##
##        'Positive' Class : 0
##
```

```
print(cm.deeper.valid)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1785   17
##          1   15  183
##
##                Accuracy : 0.984
##                  95% CI : (0.9775, 0.989)
##     No Information Rate : 0.9
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9107
##
##  Mcnemar's Test P-Value : 0.8597
##
##             Sensitivity : 0.9917
##             Specificity : 0.9150
##          Pos Pred Value : 0.9906
##          Neg Pred Value : 0.9242
##              Prevalence : 0.9000
##          Detection Rate : 0.8925
##    Detection Prevalence : 0.9010
##       Balanced Accuracy : 0.9533
##
##        'Positive' Class : 0
##
```

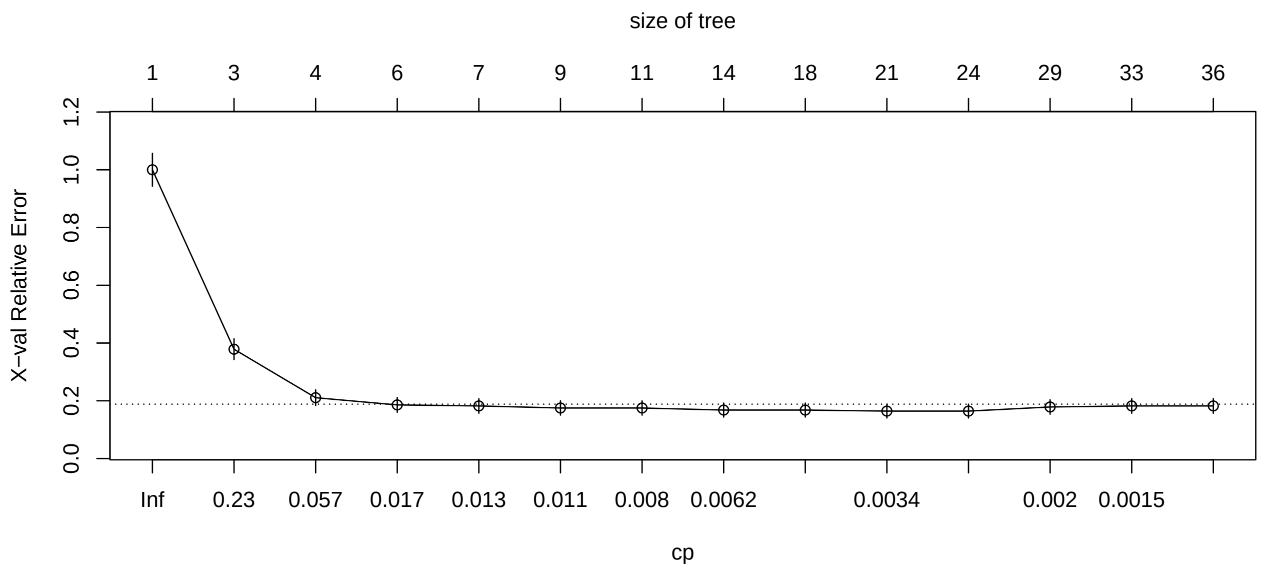
Argument `xval` refers to the number of folds to use in rpart's built-in cross-validation procedure. Argument `cp` sets the smallest value for the complexity parameter.

```
cv.ct <- rpart(Personal.Loan ~ ., data = train.df, method = "class",
            cp = 0.00001, minsplit = 5, xval = 5)
printcp(cv.ct)
```

```
##
## Classification tree:
## rpart(formula = Personal.Loan ~ ., data = train.df, method = "class",
##     cp = 1e-05, minsplit = 5, xval = 5)
##
## Variables actually used in tree construction:
## [1] Age        CCAvg      CD.Account Education  Experience Family     Income
## [8] Mortgage   Online
```

```
##
## Root node error: 280/3000 = 0.093333
##
## n= 3000
##
##           CP nsplit rel error  xerror     xstd
## 1  0.3107143      0  1.000000 1.00000 0.056904
## 2  0.1678571      2  0.378571 0.37857 0.036115
## 3  0.0196429      3  0.210714 0.21071 0.027162
## 4  0.0142857      5  0.171429 0.18571 0.025530
## 5  0.0125000      6  0.157143 0.18214 0.025287
## 6  0.0089286      8  0.132143 0.17500 0.024795
## 7  0.0071429     10  0.114286 0.17500 0.024795
## 8  0.0053571     13  0.092857 0.16786 0.024292
## 9  0.0047619     17  0.071429 0.16786 0.024292
## 10 0.0023810     20  0.057143 0.16429 0.024036
## 11 0.0021429     23  0.050000 0.16429 0.024036
## 12 0.0017857     28  0.039286 0.17857 0.025042
## 13 0.0011905     32  0.032143 0.18214 0.025287
## 14 0.0000100     35  0.028571 0.18214 0.025287
```

```r
plotcp(cv.ct)
```

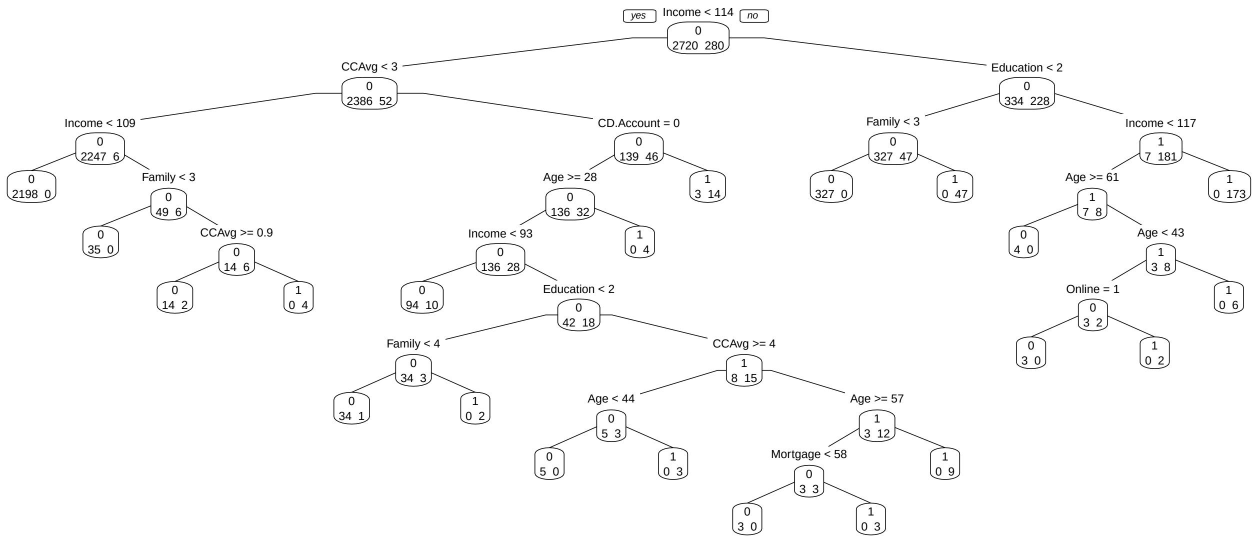


Prune by lower cp

```r
pruned.ct <- prune(cv.ct, cp = cv.ct$cptable[which.min(cv.ct$cptable[,"xerror"]),"CP"])
length(pruned.ct$frame$var[pruned.ct$frame$var == "<leaf>"])
```

```
## [1] 21
```

```r
prp(pruned.ct, type = 1, extra = 1, split.font = 1, varlen = -10)
```



Improved version of pruning

```
# this is the cp parameter with smallest cv-errror
index_cp_min = which.min(cv.ct$cptable[,"xerror"])

# one standard deviation rule
# need to find first cp value for which the xerror is below horizontal line on the plot
(val_h = cv.ct$cptable[index_cp_min, "xerror"] + cv.ct$cptable[index_cp_min, "xstd"])
```

```
## [1] 0.1883219
```

```
(index_cp_std = Position(function(x) x < val_h, cv.ct$cptable[, "xerror"]))
```

```
## [1] 4
```

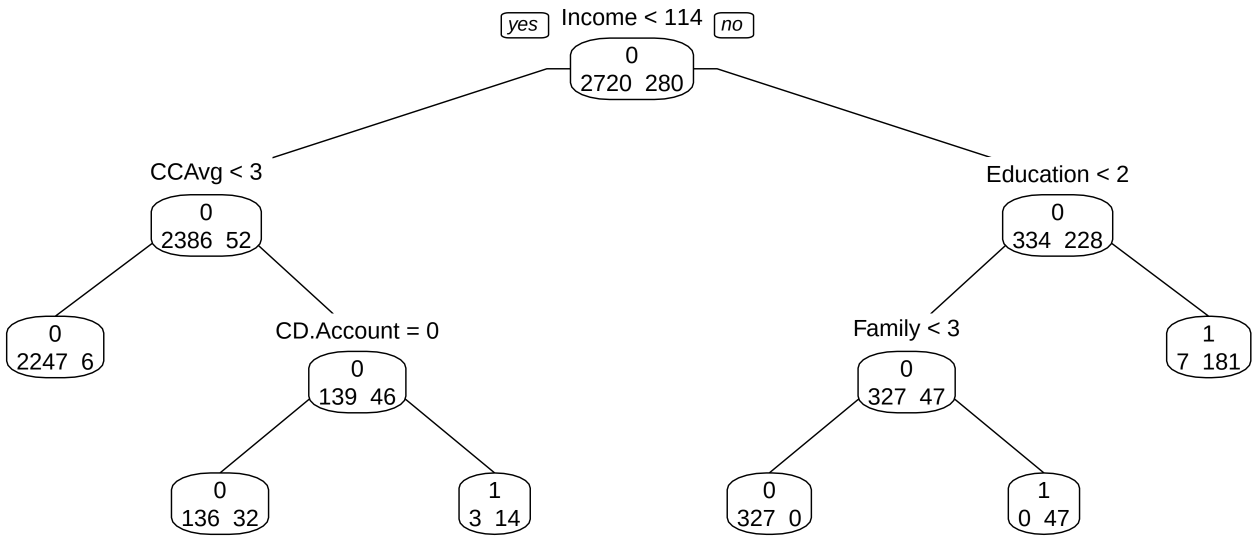
```
(cp_std = cv.ct$cptable[ index_cp_std, "CP" ])
```

```
## [1] 0.01428571
```

```
pruned.ct <- prune(cv.ct, cp = cp_std)
length(pruned.ct$frame$var[pruned.ct$frame$var == "<leaf>"])
```

```
## [1] 6
```

```
prp(pruned.ct, type = 1, extra = 1, split.font = 1, varlen = -10)
```

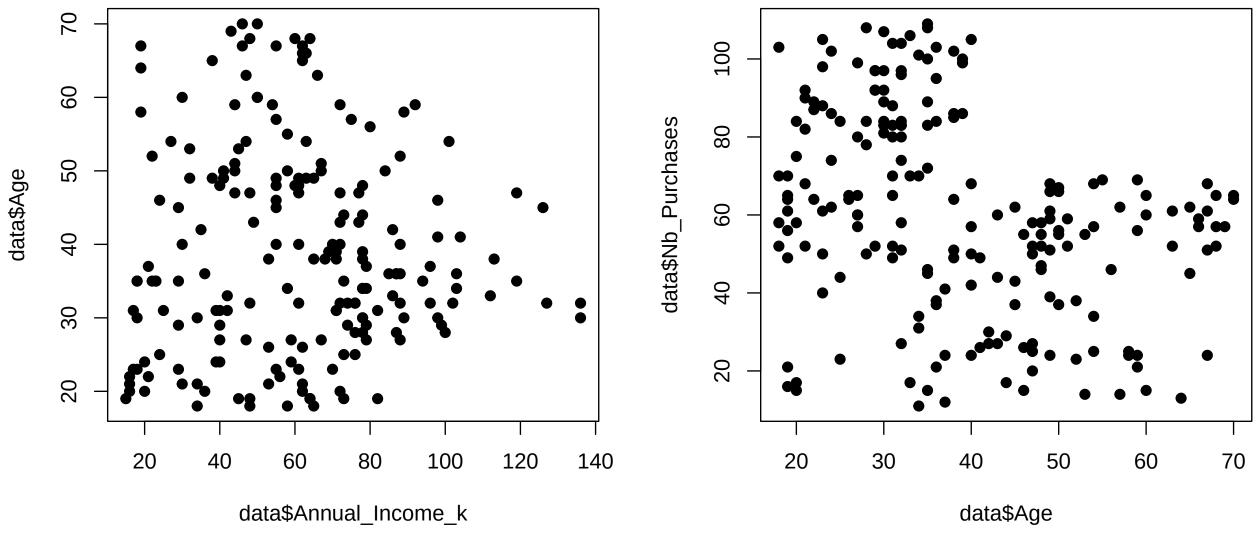


2 Tree Classification Example 2 (Membership Data)

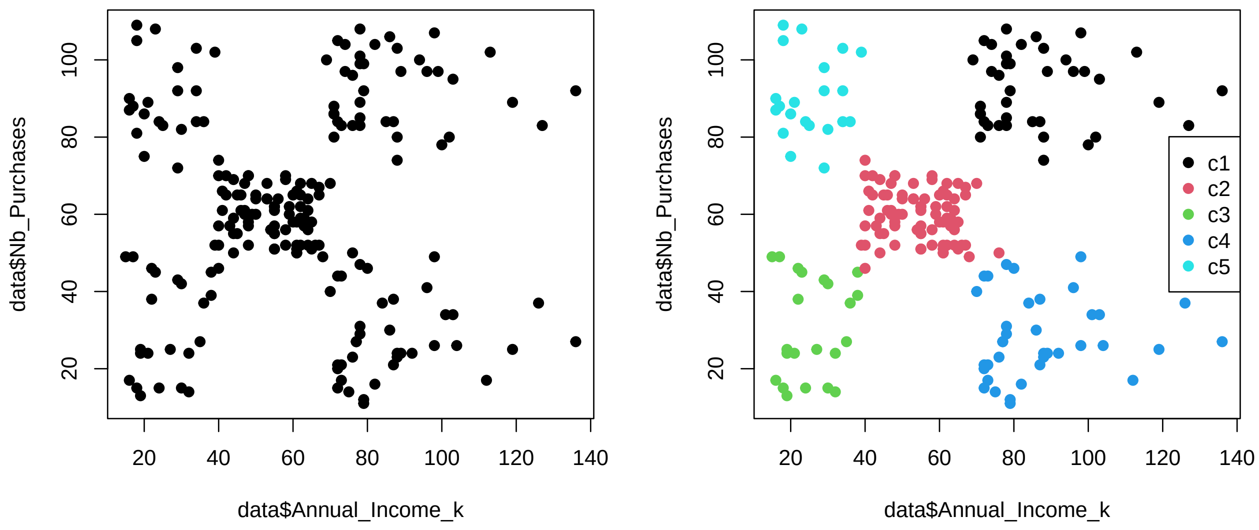
Dataset: Data of customers that have a membership at a specific store. Information available include `Age`, `Sex`, `Annual_Income_k` in k$ and `Nb_Purchases` the number of items purchased in the past month. Goal: Segment the members into different groups to tailor marketing strategies on a per segment basis.

We do not know how many segments there might be in the data.

```r
# load data
data = read.csv('membership_data_c.csv')

# look at the dataframe
# View(data)

# Do some exploratory analysis
# plots
par(mfrow=c(1,2))
plot(data$Annual_Income_k,data$Age,pch=19)
plot(data$Age,data$Nb_Purchases,pch=19)
```



```
plot(data$Annual_Income_k,data$Nb_Purchases,pch=19)
plot(data$Annual_Income_k,data$Nb_Purchases,col=data$clust,pch=19)
legend('right',c("c1","c2","c3","c4","c5"), col=1:5,pch=19)
```



Note:

1. From visual evidence we can see that setting a number of cluster of $k = 5$ seems to be a reasonable choice. (visual clues not always possible and expert knowledge may be needed)
2. A seperation over age is also possible. You may (or may not) want to include age in the clustering. It depends on you business goal and how you want to group people.

Building the tree

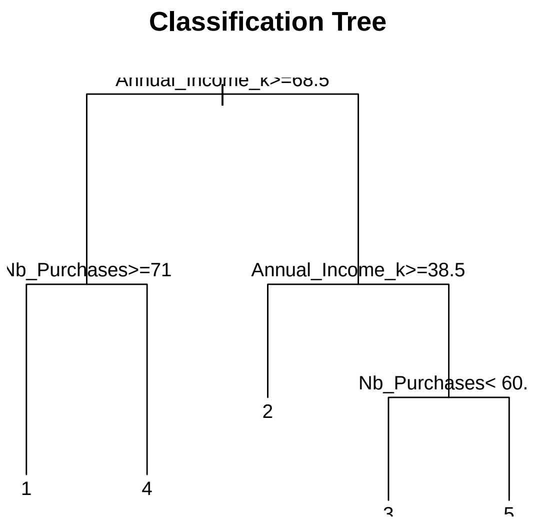
Suppose we want to do the opposite of clustering. We now have data of members that for which we know which cluster they belong to. A new member comes, signs up for the membership card and provides his information: his annual income and an expected number of purchases per month. Now our goal is to predict which segment this new customer should belong to so we can target the marketing strategy.

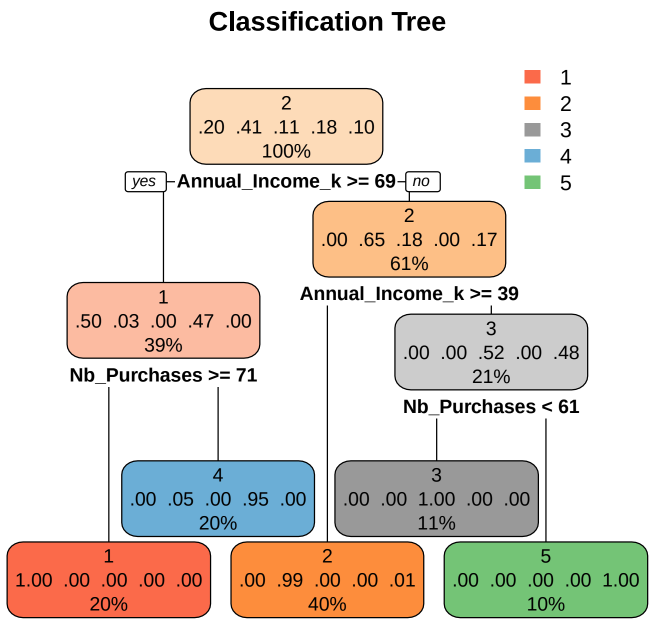
In this section we will:

1. Build a classification model based on decision trees, that takes income and estimated nb of purchases as inputs.
2. Assess the model performance
3. Prune the tree if necessary i.e. too complex by looking at cross validation error.

```
# install.packages("rpart")
library(rpart)
library(rpart.plot)

# Build the tree
t1 = rpart(clust~Annual_Income_k+Nb_Purchases, data, method="class")
# method = "anova" for a regression tree

par(mfrow=c(1,2))
# plot the tree
plot(t1, main="Classification Tree")
```

```
text(t1, cex=0.85)
rpart.plot(t1, main="Classification Tree")
```

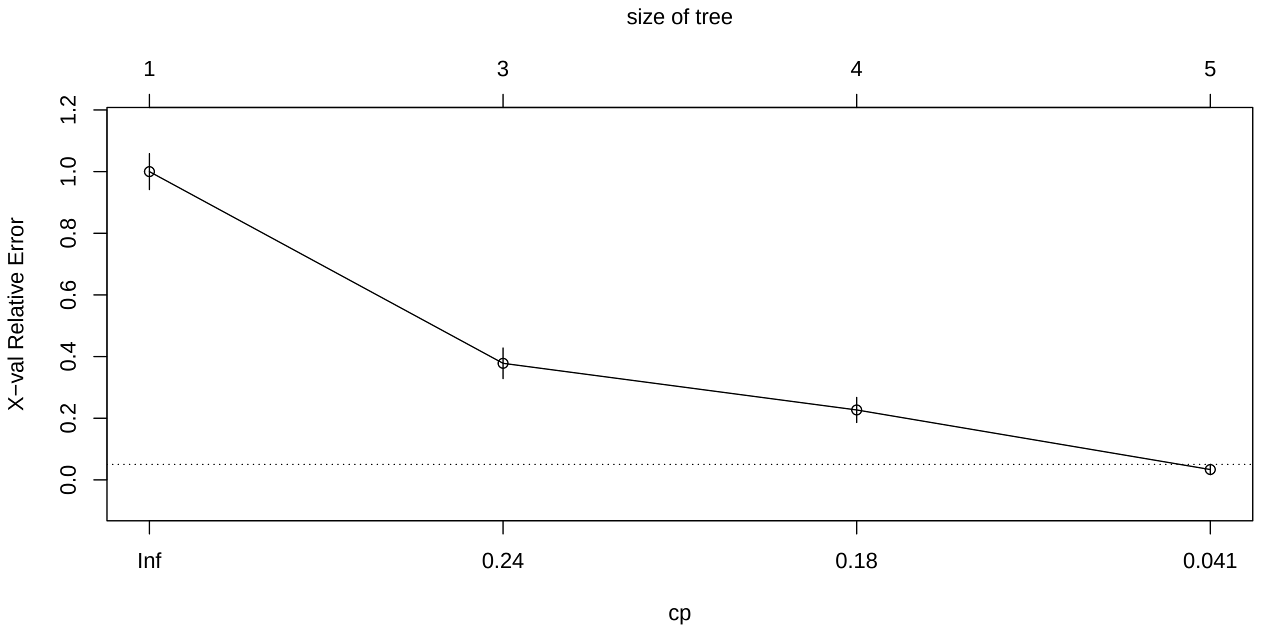




```
par(mfrow=c(1,1))
# look at the fit
printcp(t1)
```

```
##
## Classification tree:
## rpart(formula = clust ~ Annual_Income_k + Nb_Purchases, data = data,
##     method = "class")
##
## Variables actually used in tree construction:
## [1] Annual_Income_k Nb_Purchases
##
## Root node error: 119/200 = 0.595
##
## n= 200
##
##        CP nsplit rel error   xerror     xstd
## 1 0.31092      0   1.00000 1.000000 0.058338
## 2 0.18487      2   0.37815 0.378151 0.049626
## 3 0.16807      3   0.19328 0.226891 0.040611
## 4 0.01000      4   0.02521 0.033613 0.016638
```

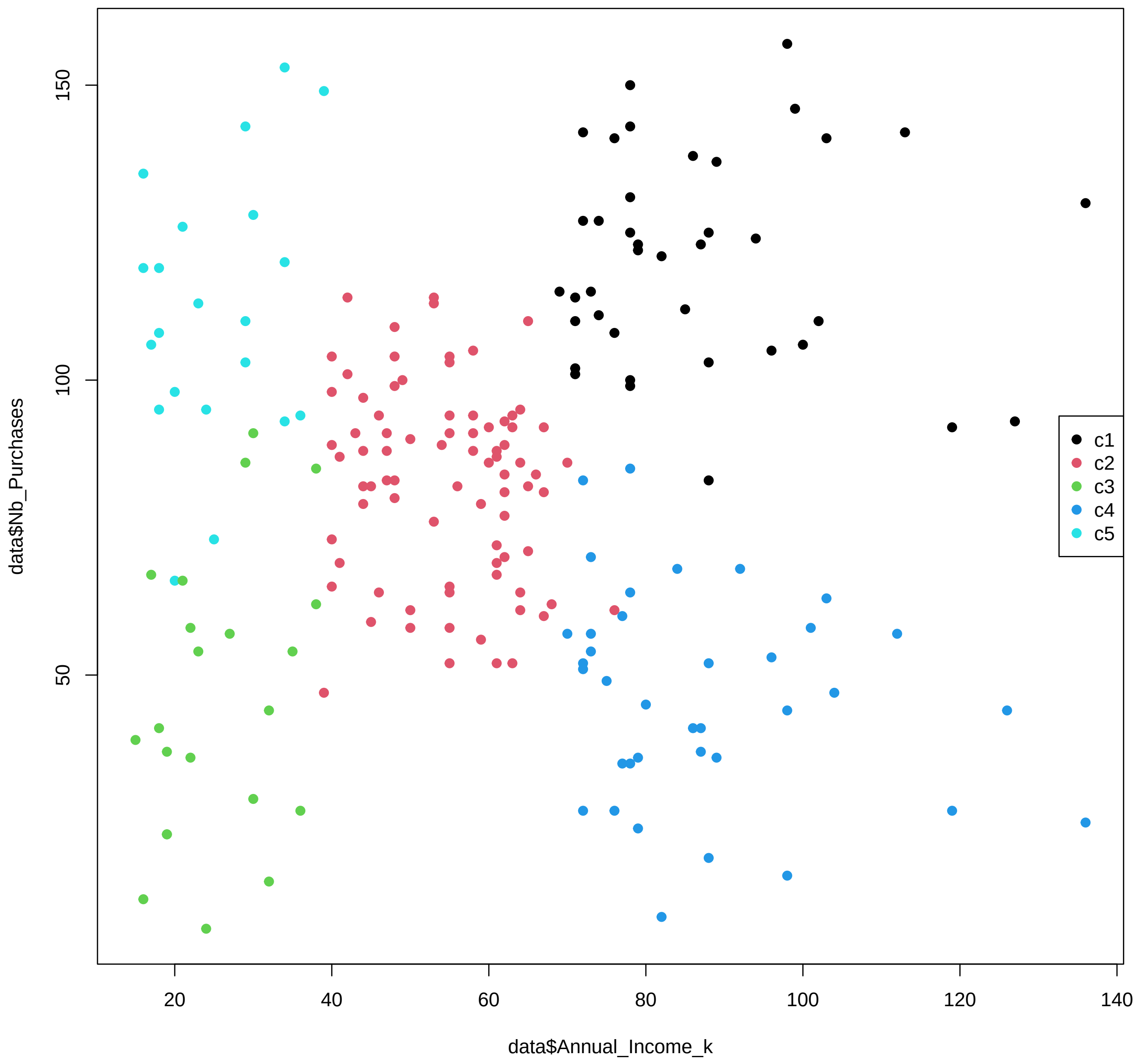
```
plotcp(t1)
```



Add noise to data, build complex model and prune tree

Suppose the clustering and segmentation was performed a few month ago and members have since then changed some of their monthly number of purchases. You're not interested to correct your segmentation of old members but rather build a predictive classification model based on this "noisy" slightly less accurate data.

```
set.seed(1)

#data$clust = clust0$cluster
data$Annual_Income_k = data$Annual_Income_k
data$Nb_Purchases = data$Nb_Purchases+sample(seq(-10,50,1),nrow(data),replace = T)

# plot
par(mfrow=c(1,1))
plot(data$Annual_Income_k,data$Nb_Purchases,col=data$clust,pch=19)
legend('right',c("c1","c2","c3","c4","c5"), col=1:5,pch=19)
```



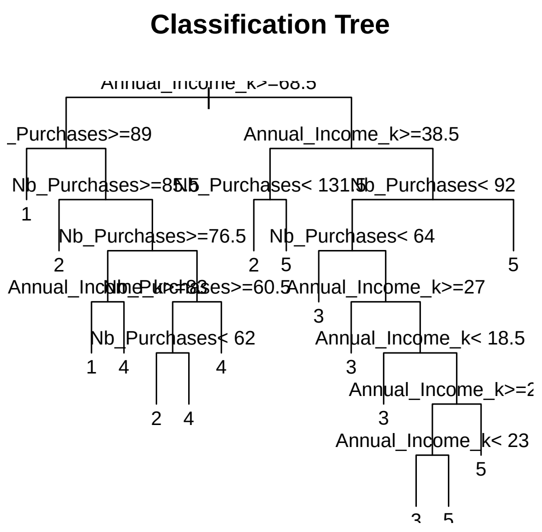
We see now that some (very few) members have moved from one segment to another, while building the model we want to ignore this behavior a make sure the model complexity is not to high to capture those isolated cases. If the model is too complex (i.e too good), then when new data (a new member) signs up the model might misclassify the segment for him.

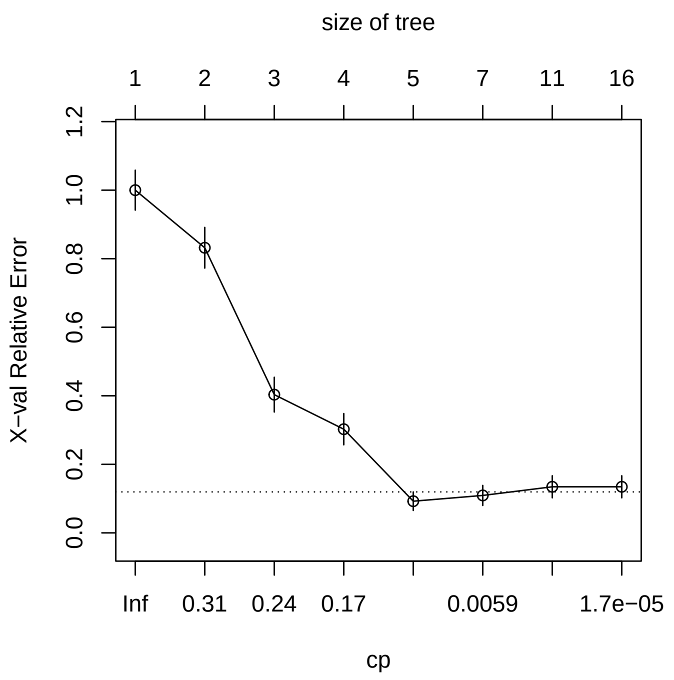
```
# Force to build a complex tree
t_complex = rpart(clust~Annual_Income_k+Nb_Purchases, data, method="class",
               control=rpart.control(minsplit = 1, cp=0.0000001))

# minsplit=1 means the tree keeps splitting as long as there is
# more than 1 sample in a splitting node

# cp (complexity factor) means the tree keeps on splitting as long as
# the delta relative error is more than the specified number

par(mfrow=c(1,2))
# plot the tree
```

```r
plot(t_complex, main="Classification Tree", uniform = T) # very complex tree
text(t_complex, cex=0.85)

# look at the fit
printcp(t_complex)
```

```
##
## Classification tree:
## rpart(formula = clust ~ Annual_Income_k + Nb_Purchases, data = data,
##     method = "class", control = rpart.control(minsplit = 1, cp = 1e-07))
##
## Variables actually used in tree construction:
## [1] Annual_Income_k Nb_Purchases
##
## Root node error: 119/200 = 0.595
##
## n= 200
##
##            CP nsplit rel error   xerror     xstd
## 1 0.3109244      0   1.000000 1.000000 0.058338
## 2 0.3025210      1   0.689076 0.831933 0.059418
## 3 0.1848739      2   0.386555 0.403361 0.050755
## 4 0.1512605      3   0.201681 0.302521 0.045657
## 5 0.0084034      4   0.050420 0.092437 0.027094
## 6 0.0042017      6   0.033613 0.109244 0.029297
## 7 0.0028011     10   0.016807 0.134454 0.032241
## 8 0.0000001     15   0.000000 0.134454 0.032241
```

```r
plotcp(t_complex) # tree overfits the data
```

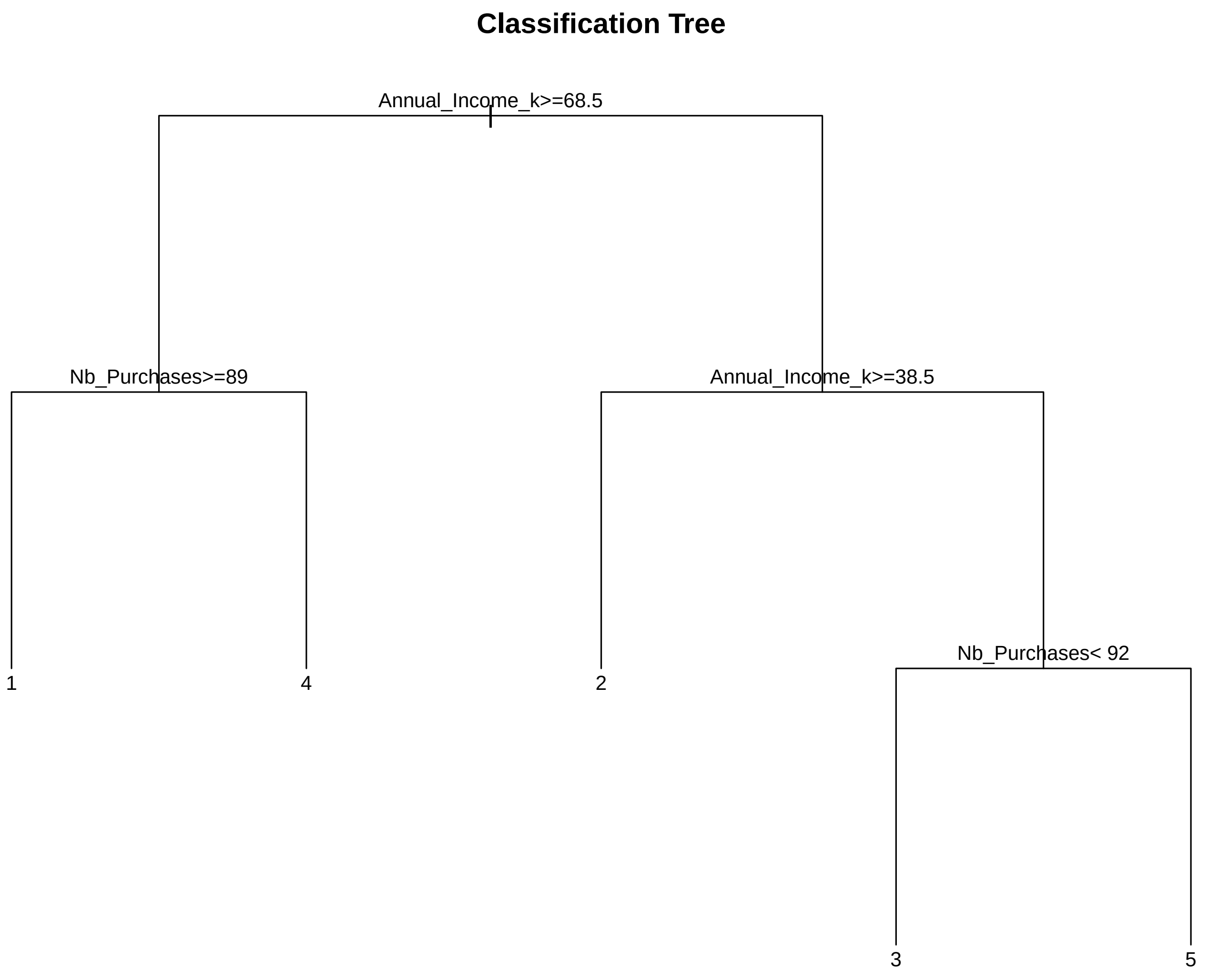




This tree is overfitting the data. It has 0 classification error but the cross validation error goes up after more than 4 splits.

We need to prune the tree.

```r
# find the cp for which xval is minimum (find best complexity)
cp_best = t_complex$cptable[which.min(t_complex$cptable[,"xerror"]),"CP"]

# Rebuild a tree that is not too complex
t_best = rpart(clust~Annual_Income_k+Nb_Purchases, data, method="class",
               control=rpart.control(minsplit = 1, cp=cp_best))

# plot the tree
plot(t_best, main="Classification Tree", uniform = T) # very complex tree
text(t_best, cex=0.85)
```



```r
# make prediction with the best tree
print(newcustomer <- predict(t_best,data.frame(Annual_Income_k=75,Nb_Purchases=60)))
```

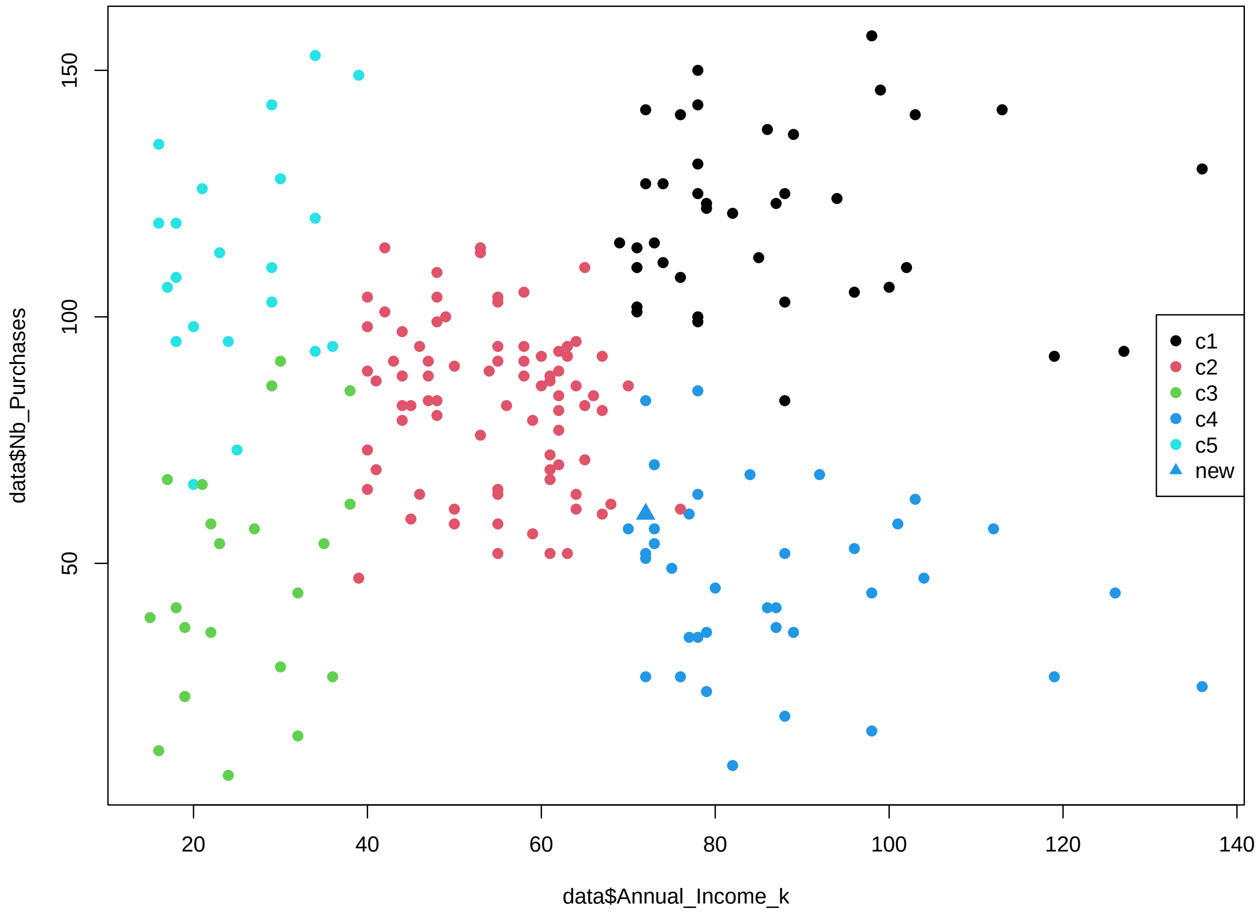
```
##       1    2 3     4 5
## 1 0.025 0.05 0 0.925 0
```

```r
# The predicted for this new member is cluster 4

# See prediction on plot
plot(data$Annual_Income_k,data$Nb_Purchases,col=data$clust,pch=19)
```

```
legend('right',c("c1","c2","c3","c4","c5",'new'), col=c(1:5,4),pch=c(19,19,19,19,19,17))

# find column number with max classification probability
which.max(newcustomer)
```

```
## [1] 4
```

```
# plot point
points(72,60,pch=17,col=which.max(newcustomer),cex=1.5)
```



```
# look at confusion matrix
print(xtabs(~apply(predict(t_best), 1, function(x) which.max(x))+data$clust))
```

```
##                                              data$clust
## apply(predict(t_best), 1, function(x) which.max(x))  1  2  3  4  5
##                                                   1 38  0  0  0  0
##                                                   2  0 79  0  0  1
##                                                   3  0  0 22  0  2
##                                                   4  1  2  0 37  0
##                                                   5  0  0  0  0 18
```

Prediction using the tree model seems to classify the new customer accurately in cluster 4 (although it's in between two noisy red points). This is the advantage of pruning, avoiding overfitting.