PRACTICAL 5

Aim:

The aim of this practical is to illustrate how entropy and information gain can be calculated in R, and to demonstrate how decision tree models can be fit in R (including interpretation of the output of such tree models).

Before you start:

Open RStudio and load the `tidyverse`.

Instructions:

This document contains discussions of some basic concepts, and instructions which you should execute in RStudio. **Text in bold** indicate actions that are to be taken within RStudio. `Text highlighted in grey` shows the code you should enter in RStudio.

Due date:

For Tutorial 5, it will be assumed that you have mastered the basic concepts discussed here.

About the data

In this practical we will consider an example that is discussed on pages 73 – 78 of the textbook. The accompanying dataset is called `churn.csv`, and is available on SUNLearn. The data comes from the cellular phone domain, and the aim is to predict customer churn. *Churn* happens when customers switch from one company to another, and it is something which companies would like to prevent. (Read page 4 of the textbook for more background on customer churn.)

The dataset consists of 20 000 observations; that is, a dataset containing historical information on 20 000 customers. There are 11 attributes in the dataset: the target variable is `LEAVE`, which is an indication of whether a customer stayed or left, and since this is a categorical variable we are dealing with a classification problem.

The goal will be to build a model containing some or all of the other 10 attributes, in order to predict whether customers will stay or leave. This model – based on historical data – could potentially then be used to predict what other / future customers will do. Presumably, if it is predicted that a customer will leave the company, interventions or incentives could be put in place by the company to ensure that the customer will be retained.

The attributes in the dataset are as follows:

Attribute	Description
COLLEGE	**Whether the customer is college educated.** Values are coded as `zero` (customer has no college education) or `one` (customer has a college education).
INCOME	**Customer's annual income**
OVERAGE	**Average overcharges per month**
LEFTOVER	**Average number of leftover minutes per month**
HOUSE	**Estimated value of customer's dwelling**
HANDSET_PRICE	**Cost of phone**
LONG_CALLS_PER_MONTH	**Average number of long calls (>15 minutes in duration) per month**
AVERAGE_CALL_DURATION	**Average duration of calls**
REPORTED_SATISFACTION	**Reported level of satisfaction with the company.** Values are coded as follows: `very_sat` (Very satisfied), `sat, avg, unsat, very_unsat`
REPORTED_USAGE_LEVEL	**Self-reported usage level.** Values coded as follows: `very_high, high, avg, little, very_little`
LEAVE (Target variable)	**Whether the customer stayed or left.** Values coded as `LEAVE` or `STAY`.

Preliminaries

While it is generally recommended that you work in R Markdown, it is not strictly necessary for this practical. I will provide the code instructions in this document, and you can decide whether you want to implement this in R Markdown or whether you prefer to work directly in the console. Solutions to the practical will be provided in script (`.R`) format as well as in R Markdown format (`.Rmd`). Also remember that you will be required to work in R Markdown for your A1 assessment!

1.1 **Load the following packages:**

`DescTools, CORElearn, tree, rpart, rpart.plot`

Note that most of these have not been used before, so if you are working on your own computer you will have to install them first. (Fharga computers should have all of these packages installed already so you will only have to load them.)

1.2 The dataset `churn.csv` is available on SUNLearn.

Download this dataset.

If you will be working directly in the R console (and not in R Markdown), make sure that you download the data file to your R working directory; if not, you will have to enter the full path to the data file when using the `read.csv()` function to import the data.

If you will be using R Markdown, make sure that the data file is saved in the same location where you plan to save your markdown file.

Tip:

Remember that you can use the `getwd()` function to see what your current working directory is, and that you can change it using the `setwd()` function.

1.3 **Import the `churn.csv` dataset into R using one of the methods discussed in Tutorial 4 and store it in an object called `churn`.**

You should pay particular attention to the data types of the attributes when you import the data. If you use the `read.csv()` function, you might find that the categorical attributes are imported as strings instead of factors. You can force R to import these attributes as factors by including the `stringsAsFactors = TRUE` argument in the `read.csv()` function call, as follows:

```
churn <- read.csv("churn.csv", header=TRUE,
stringsAsFactors = TRUE)
```

Explore the data

You should always explore your data before constructing models!

2.1 Recall that a useful function to see the data types of attributes, is `str()`.

Use this function to examine the `churn` data.

```
str(churn)
```

2.2 Summary statistics for a dataset can be accessed by using the `summary()` function.

Type

```
summary(churn)
```

to view summary statistics for the `churn` dataset.

Note that this is a fairly well balanced dataset; in other words, there are almost as many `LEAVE` clients as `STAY` clients.

Tip:

You can create plots to explore the data as well – this will be good revision!

ENTROPY AND INFORMATION GAIN CALCULATIONS

When fitting a decision tree in R, you do not have to explicitly calculate entropy and information gain yourself. This is all done automatically during the model building process. However, since you need to understand what entropy and information gain is, and how it can be used to select informative attributes, we will also explicitly calculate these values in this practical.

3. Remember that the target variable is `LEAVE`, and that it is a binary (i.e. two-class) attribute. If you would like to calculate the entropy for this attribute (i.e. the *parent entropy*), you need the absolute and relative frequencies of the classes (i.e. of `LEAVE` and `STAY`).

3.1 **Calculate the absolute frequencies of the classes in the `LEAVE` attribute and store these frequencies in an object called `leavetable`.**

This can be done using the `table()` function.

```
leavetable <- table(churn$LEAVE)
```

3.2 **Calculate the relative frequencies of the classes in the `LEAVE` attribute and store these in objects `p1` and `p2` respectively (`p1` for `LEAVE` and `p2` for `STAY`).**

```
p1 <- leavetable["LEAVE"] / length(churn$LEAVE)
p2 <- leavetable["STAY"] / length(churn$LEAVE)
```

3.3 **Based on the relative frequencies calculated in (3.2), calculate the entropy for the `LEAVE` attribute. Save this result in an object named `parent_entr` so that you can use it again later on in your information gain calculations.**

```
parent_entr <- (-1 * p1 * log2(p1)) + (-1 * p2 * log2(p2))
```

Note the use of the `log2()` function to calculate base 2 logarithms.

Tip:

Make sure that you are able to calculate such entropy values manually as well (i.e. using pen and peper and a calculator) as this could be required of you in written assessments.

3.4 Entropy can also be calculated by using the `Entropy()` function in the `DescTools` package:

```
Entropy(table(churn$LEAVE), base = 2)
```

---

4. We now know that the parent entropy is 0.999842 and we can use this value in information gain calculations. In this question we will manually calculate the information gain for the `COLLEGE` attribute.

4.1 Before you can calculate information gain for the `COLLEGE` attribute, you need a ***cross-tabulation*** of the `LEAVE` and `COLLEGE` attributes (since `LEAVE` is the target variable). Store this in an object called `mytable`.

[Hint: The `table()` function works for this as well.]

```
mytable <- table(churn$LEAVE, churn$COLLEGE)
```

4.2 Recall that when we are constructing a decision tree, we are trying to determine rules to segment the data. So we started out with a very impure set (parent entropy = 0.999842) and we now need to decide which attribute to use to segment the data into subsets. These subsets will be the *child nodes*, and we want them to be purer than the parent node. Of course there are 10 possible attributes to use to segment the data and we need to decide which one will contribute the most information.

Note that the `COLLEGE` attribute is binary (i.e. can take on only two possible values).

4.2.1 **Calculate the entropy of the child node where `COLLEGE = one` and store it in an object called `child1_entr`.**

```
q1 <- mytable["LEAVE","one"] / (mytable["LEAVE","one"] +
mytable["STAY","one"])
q2 <- mytable["STAY","one"] / (mytable["LEAVE","one"] +
mytable["STAY","one"])
child1_entr <- (-1 * q1 * log2(q1)) + (-1 * q2 * log2(q2))
```

Note how the `mytable` object is subsetted by using the column and row names.

If you calculate this entropy value manually you will see that it is very close to 1, since there are almost exactly the same number of `LEAVE` and `STAY` observations where `COLLEGE = one`.

(Confirm this in the table in (4.1).)

In R, this is printed as 1, so to force R to print more decimal values you can use the following code:

```
sprintf("%.10f",child1_entr)
```

In the `sprintf()` function above, in the argument `"%.10f"` the `f` stands for floating point and `10` represents the number of decimals. The result will therefore be printed as a floating point number with 10 decimal points.

4.2.2 **Calculate the entropy of the child node where `COLLEGE = zero` and store it in an object called `child0_entr`.**

```
r1 <- mytable["LEAVE","zero"] / (mytable["LEAVE","zero"] +
mytable["STAY","zero"])
r2 <- mytable["STAY","zero"] / (mytable["LEAVE","zero"] +
mytable["STAY","zero"])
child0_entr <- (-1 * r1 * log2(r1)) + (-1 * r2 * log2(r2))
```

4.3 **Calculate the information gain for the COLLEGE attribute.**

```
parent_entr - (((mytable["LEAVE","one"] +
mytable["STAY","one"])/nrow(churn)*child1_entr) +
((mytable["LEAVE","zero"] +
mytable["STAY","zero"])/nrow(churn)*child0_entr))
```

5. The manual calculations in R above became very tedious and complex – it is almost easier to calculate it by hand! (Note that I have uploaded a document for you on SUNLearn showing how these calculations should be done manually; this is important to study for written assessments.)

Fortunately, information gain values can be calculated using an R function instead.

5.1 The `attrEval()` function in the `CORElearn` package evaluates the "quality" of attributes, and information gain is one of the many quantities that can be calculated in this way.

The following code should be entered to calculate the information gain values:

```
attrEval(formula = LEAVE~., data = churn, estimator =
"InfGain")
```

The second argument in the function call is easy to understand: it specifies the dataset that should be used (in this case the `churn` data).

The third argument indicates that the quantities that should be calculated are information gain values; other options are also possible but in this module we will restrict our attention to information gain only.

The first argument (`formula = LEAVE~.`) specifies that `LEAVE` is the target variable, and the `.` after the ~ means that all the other attributes in the dataset should be considered as well.

5.2 You might notice that the information gain values obtained in (5.1) are not easy to read on the R output, since many of the values are printed in scientific notation.

We can therefore add some code to force R to use decimal notation, and also to sort the values from high to low:

```
sort(round(attrEval(formula = LEAVE~., data = churn, estimator = "InfGain"), digits = 10), decreasing = TRUE)
```

Firstly note that the calculated information gain value for `COLLEGE` is exactly the same as the value calculated manually in (5.1).

Secondly, if you compare the calculated values to those listed in the textbook on page 76, you will see that they differ for some of the numeric attributes. This is no reason for concern: the difference is due to different discretisation methods for numeric attributes (see page 56 of the textbook for a brief discussion of this).

Lastly – and most importantly – you should note that `HOUSE` has the highest information gain, so when building a decision tree to predict the value of `LEAVE`, `HOUSE` will be the first attribute to split on.

A DECISION TREE FOR THE `churn` DATA

On page 77 in the textbook, a decision tree is fit to the `churn` data. A colour version of this tree plot is shown on the next page. Note the following from the tree:

- `HOUSE` is the first attribute to split on.
- `HOUSE` less than or equal to 600 469 will go along the left branch of the tree and values greater than that will go along the right branch.
- The process of identifying the best attribute to split on now continues, but separately for the left branch and the right branch. Note that for the left branch, the next split is based on the variable `OVERAGE`, whereas for the right branch it is `INCOME`. You can

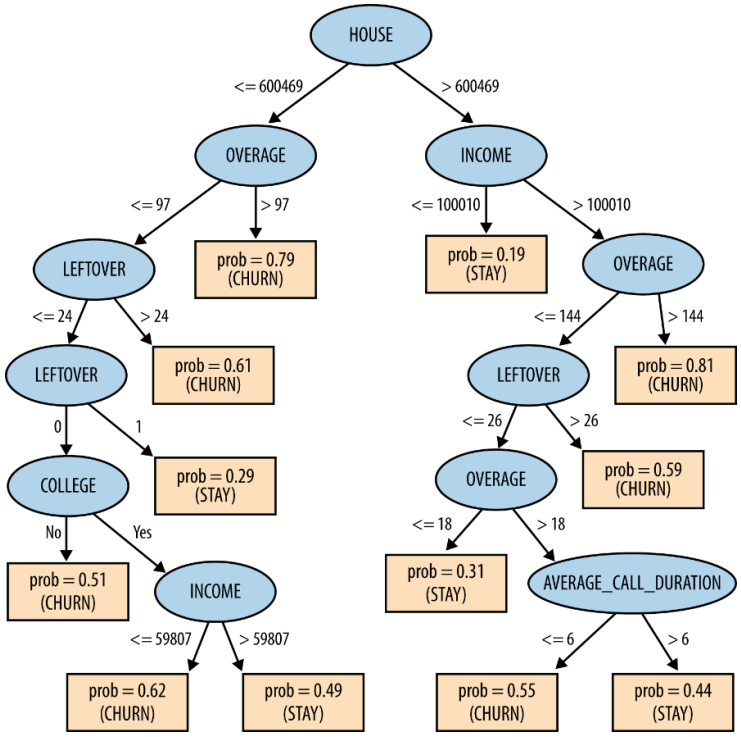
verify this by calculating the information gain values; first for the left branch (that is, where HOUSE <= 600 469):

```
newparent <- churn[which(churn$HOUSE<=600469),]
sort(attrEval(formula = LEAVE~., data = newparent,
estimator = "InfGain"), decreasing = TRUE)
```

Next for the right branch, where HOUSE > 600 469:

```
newparent <- churn[which(churn$HOUSE>600469),]
sort(attrEval(formula = LEAVE~., data = newparent,
estimator = "InfGain"), decreasing = TRUE)
```

- You might have wondered about how numeric attributes are handled when constructing a decision tree; for instance, how the split point of 600 469 was determined for the HOUSE attribute. The process is briefly discussed on page 56 of the textbook but we will not consider it in any further detail in this module; the R functions we use will automatically identify the best split points.



FITTING A DECISION TREE IN R

There are many different functions that can be used to fit a decision tree in R. We will consider just two: `tree()` (in the `tree` package) and `rpart()` (in the `rpart` package).

6. We will first consider the `tree` package.

6.1 Look at the help file for the `tree()` function.

```
?tree::tree
```

Note that since there are different tree functions in R, we can specify that we specifically want to see the help file for the `tree` function from the `tree` package in this way (the package name goes before the double colon and the function name after).

6.2 **Fit a decision tree in R to predict whether customers will churn or not, using the `tree()` function from the `tree` package. Save the tree in an object called `tree.churn`.**

Remember that the target variable is `LEAVE`.

```
tree.churn <- tree(LEAVE~., data = churn)
```

Note that the only two arguments required are the formula specification (first argument) and data specification (second argument). The target variable goes before the ~ sign. The predictors we want to use are listed after the ~ sign, but if we want to use all of the predictors in the dataset we only need to type a full stop.

6.3 We can obtain a summary of the tree model that was fit by using the following code:

```
summary(tree.churn)
```

This produces the following output:

```
## Classification tree:
## tree(formula = LEAVE ~ ., data = churn)
## Variables actually used in tree construction:
## [1] "HOUSE"    "OVERAGE"  "LEFTOVER" "INCOME"
## Number of terminal nodes:  7
## Residual mean deviance:  1.154 = 23060 / 19990
## Misclassification error rate: 0.306 = 6120 / 20000
```

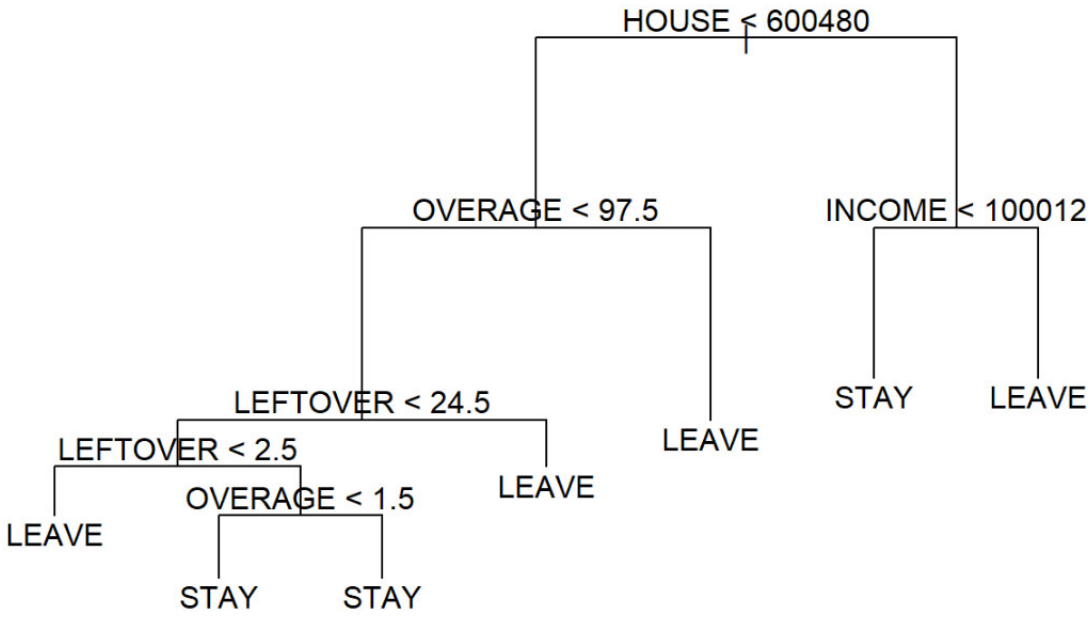
For now we will only focus on the two highlighted parts of the summary output. Firstly, note that only 4 attributes were actually used in the tree model: `HOUSE`, `OVERAGE`, `LEFTOVER` and `INCOME`.

Secondly, the given misclassification error rate of 0.306 means that (1 – 0.306) = 0.694, or 69.4%, of the observations in the `churn` dataset are correctly classified (either as `STAY` or `LEAVE`) by the tree model that was constructed.

6.4 One of the big advantages of trees is that they give us easily interpretable models that are easily visualised as well. You can visualise a tree model object by using the `plot()` function, and include the split rules on your plot by using the `text()` function after the `plot()` function call.

```
plot(tree.churn)
text(tree.churn)
```

The following tree is produced:

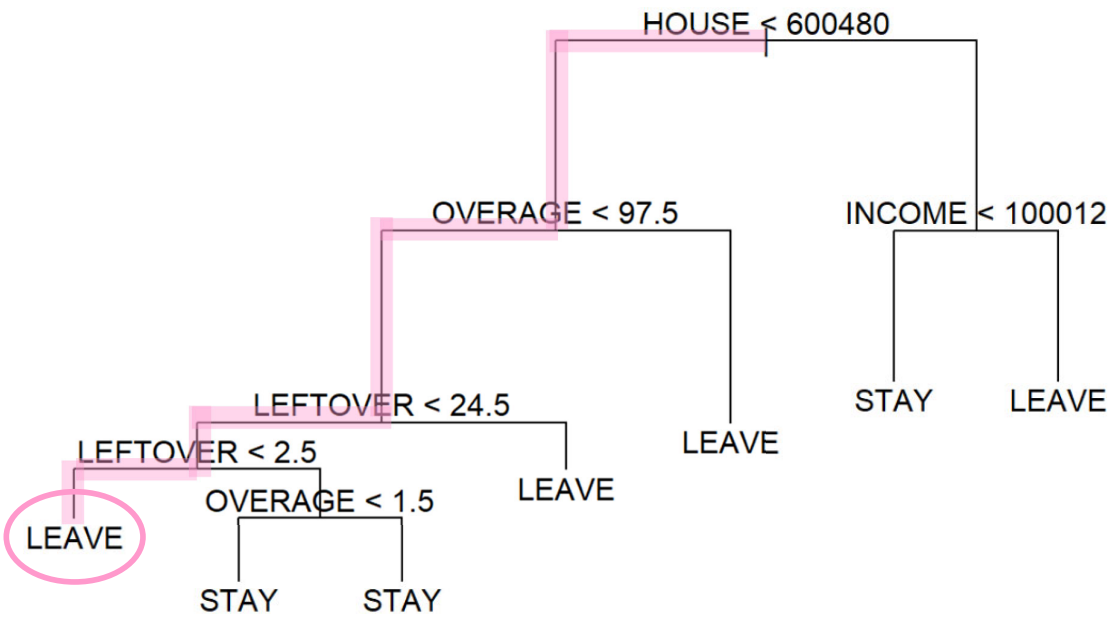


Using this visualisation, it is easy to see how an observation would be classified by the constructed tree model.

Suppose there is a client with the following values for the 4 attributes used in the tree:

HOUSE	OVERAGE	LEFTOVER	INCOME
515 444	63	0	143 501

We can follow the path that the client would take in the tree model, to come up with a final classification. The path is highlighted in the tree plot on the next page:



The classification proceeds as follows:

- Since the customer's `HOUSE` value is 515 444 < 600 480, the left branch is taken.
- The next attribute to split on is `OVERAGE`; the customer has a value of 63 < 97.5 so once again the left branch is taken.
- `LEFTOVER` is considered next; since the customer's `LEFTOVER` value is 0, the left branch is taken once again.
- Lastly, `LEFTOVER` is considered again. Since 0 < 2.5, the left branch is taken.
- The terminal node `LEAVE` is reached, so the customer is classified as `LEAVE`.

6.5 A text summary of the tree rules can also be obtained by just typing the name of the tree model object:

`tree.churn`

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 20000 27720 STAY ( 0.4926 0.5074 )
##    2) HOUSE < 600480 13230 17980 LEAVE ( 0.5822 0.4178 )
##      4) OVERAGE < 97.5 8811 12190 STAY ( 0.4758 0.5242 )
##        8) LEFTOVER < 24.5 5872  7944 STAY ( 0.4089 0.5911 )
##         16) LEFTOVER < 2.5 2931  4056 LEAVE ( 0.5244 0.4756 ) *
##         17) LEFTOVER > 2.5 2941  3562 STAY ( 0.2938 0.7062 )
##           34) OVERAGE < 1.5 1545  1206 STAY ( 0.1320 0.8680 ) *
##           35) OVERAGE > 1.5 1396  1931 STAY ( 0.4728 0.5272 ) *
##        9) LEFTOVER > 24.5 2939  3932 LEAVE ( 0.6094 0.3906 ) *
##      5) OVERAGE > 97.5 4419  4489 LEAVE ( 0.7945 0.2055 ) *
##    3) HOUSE > 600480 6770  8461 STAY ( 0.3174 0.6826 )
##      6) INCOME < 100012 4531  4398 STAY ( 0.1894 0.8106 ) *
##      7) INCOME > 100012 2239  3051 LEAVE ( 0.5766 0.4234 ) *
```

Note that the first row of the output gives the "legend" for interpreting the rules:

`node)`	The node number
`split`	The split rule that was used
`n`	The number of observations in that branch
`deviance`	THIS VALUE CAN BE IGNORED FOR NOW
`yval`	The overall prediction for the branch
`yprob`	The fraction of observations in that branch which take on the values `LEAVE` and `STAY` respectively

For example, the first split rule is HOUSE < 600 480. There are 13 230 observations in this branch. This is not a terminal node (there is not a *), so we are going to ignore the remaining values in that row, but you can note that there are more `LEAVE` observations than `STAY` observations in this branch.

7. We can also use the `rpart()` function (from the `rpart` package) to fit a decision tree.

7.1 **Look at the help file for the `rpart()` function.**

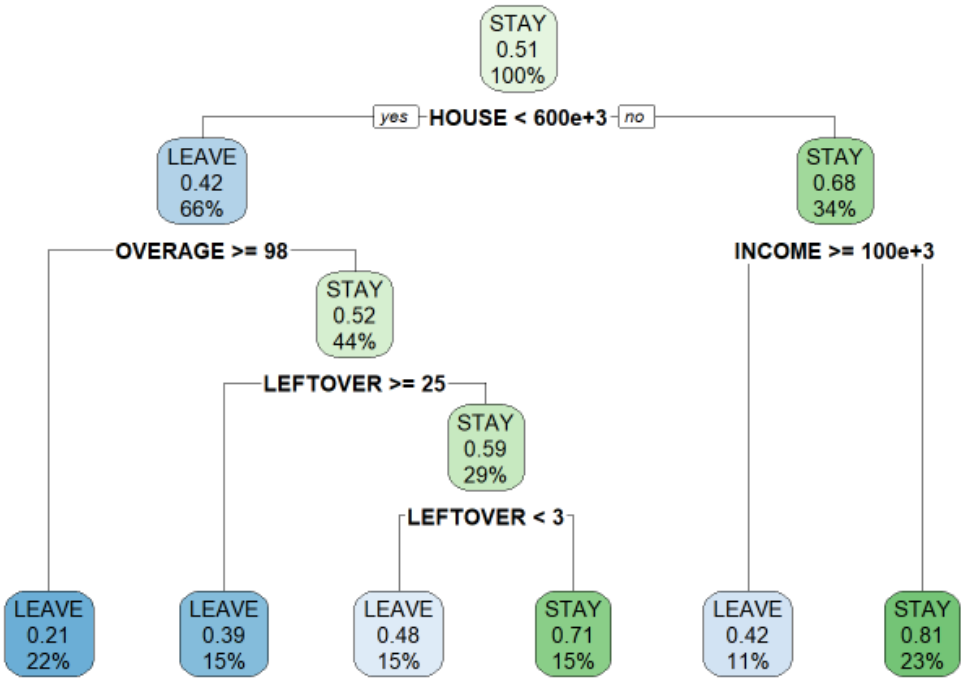
```
?rpart
```

7.2 **Fit a decision tree in R to predict whether customers will churn or not, using the `rpart()` function from the `rpart` package. Save the tree in an object called `tree2.churn`.**

```
tree2.churn <- rpart(LEAVE~., data = churn)
```

7.3 **Visualise the tree in (7.2), using the `rpart.plot()` function, which can be found in the `rpart.plot` package:**

```
rpart.plot(tree2.churn)
```

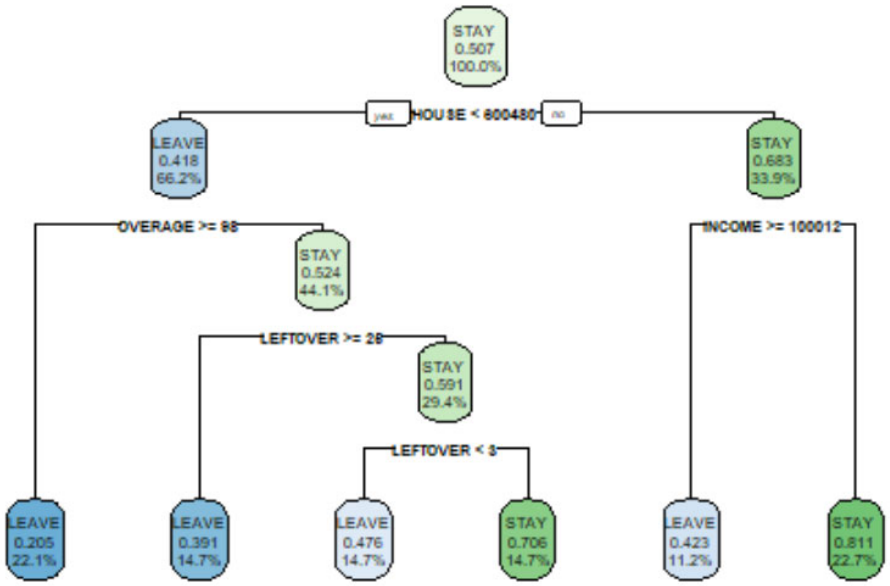


Note that this gives a much "prettier" visualisation that the plot from the `tree()` function.

You will also note that the tree is slightly different from the one obtained in (6.2); this is simply due to different default options for parameters that can be set when constructing a tree.

Lastly, note that R uses scientific notation in some of the split points. If you want to avoid this, you can include the `digits` parameter in your `rpart.plot()` function call, in the following way:

```
rpart.plot(tree2.churn, digits = -5)
```



The exact workings of the `digits` parameter is discussed in the help file for `rpart.plot()`; you should note that setting a negative value means that the standard `format` function is applied.

DISCUSSION OF RESULTS

The trees constructed in (6.2) and (7.2) are not exactly the same, and also differ from the tree in the textbook (page 77) in that there are fewer splits, and the split points differ. There are many different algorithms for constructing trees, and different decisions can be made for example in terms of deciding which attributes to split on (e.g. using information gain or other options), how split points are handled (in the case of numeric variables, or categorical variables with more than 2 categories), when to stop growing a tree, etc. A discussion of these aspects are beyond the scope of this course and will be covered in subsequent Data Science modules.

The misclassification rate of 0.306 for the tree in (6.2) implies that 69.4% of observations were correctly classified by the tree. The tree in the textbook achieved 73% accuracy on its classifications. Note that this does not necessarily mean that the textbook tree is better! These measures only give an indication of how well each tree fared on data it has *seen* (i.e. data that was used to construct the model) – we refer to this as the training error rate. We therefore have no idea which tree will fare better on *unseen* observations. To get a better measure of how well each tree really performs, we should evaluate its performance on unseen data (that is, data that has not been used to construct the tree). This will be discussed towards the end of the semester, but keep this thought in mind while we build different models throughout this course.

You should now have a good grasp of the following:

* How entropy and information gain can be used to help select informative attributes (and what is meant by informative attributes)
* How a decision tree can be constructed based on information gain
* How to fit a decision tree in R and how to interpret its output
* How to evaluate the accuracy of predictions made by the tree
* How to use a tree plot (or text-based rules) to make predictions