

Week 5 Practical

Decision Tree


Overview

In this Practical we will learn how to build and interpret a decision tree in R. However, before fitting a predictive model, we will also go through some of the steps we covered in Practicals 2 and 3.

There are several packages in R that can be used to fit a decision tree. We will be working with `rpart`, as it allows us high flexibility and is rather easy to use.

How to use this practical?

The practical provides a set of tasks you should follow to learn a set of skills you will need for your assignments. This is by no means intended to be a complete guide to all the functions and methods available out there for a specific task. It rather focuses on commonly used methods and a minimum you need to understand to be successful in your assignments.

Icon  indicates a challenge you should try to solve. Those challenges are highly recommended.

Objectives

- Load and explore data.
- Build a decision tree.
- Create a fully grown tree.
- Prune the decision tree.
- Make a prediction using the model we built.

Sharing results

While tasks should be self-explanatory, some of the challenges might be more complex. Feel free to share your results or questions in the course discussion forum. Results to all the challenges will be available UPON REQUEST (discussion forum or email).

Datasets

In this practical, we will be using a dataset that was derived from the Kaggle Stroke Prediction Dataset (<https://www.kaggle.com/fedesoriano/stroke-prediction-dataset>). To obtain a dataset for this practical, we applied a function that implements synthetic minority over-sampling technique (SMOTE). We did this in order to obtain a balanced dataset, as the point of this practical is on learning how to build a decision tree, rather than dealing with other issues that will be covered later in the course.

Table 1. Stroke prediction dataset description

Column Name	Description
id	Unique identifier
gender	"Male", "Female" or "Other"
age	Age of the patient
hypertension	0 if the patient doesn't have hypertension, 1 if the patient has hypertension
heart_disease	0 if the patient doesn't have any heart diseases, 1 if the patient has a heart disease
ever_married	"No" or "Yes"
work_type	"children", "Govt_jov", "Never_worked", "Private" or "Self-employed"
residence_type	"Rural" or "Urban"
avg_glucose_level	average glucose level in blood
bmi	body mass index
smoking_status	"formerly smoked", "never smoked", "smokes" or "Unknown"*
stroke	1 if the patient had a stroke or 0 if not

*Note: "Unknown" in smoking_status means that the information is unavailable for this patient

Before getting started

Before going through the tasks, make sure you have a new R Notebook created in RStudio, as shown in Figure 1.

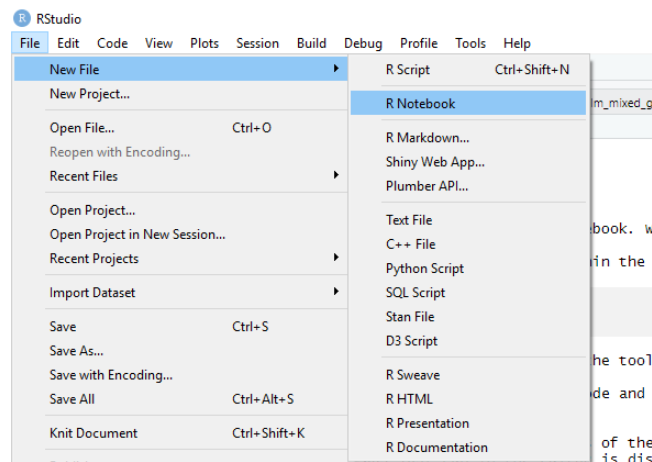


Figure 1. Opening R Notebook in RStudio

Each section of the code (marked by the grey box) should be executed within a single chunk in your Notebook.

Task 1. Data at glance

As we did in previous practicals, as a first step in building a predictive model, we should get familiar with the dataset. Also, it is a good practice to have all your libraries loaded at the beginning of your script (or notebook in this case).

```
# decision tree
library(rpart.plot)

# Data manipulation
library(rgl)
library(rattle)
library(mice)
library(dplyr)
library(GGally)
library(tidyverse)

# Plotting
library(viridis)
library(hrbrthemes)
library(ggplot2)
library(heplots)
```

Once this part is ready, we should load our data.

```
# Load data
data <- read.csv(url("http://bit.ly/infs5100-stroke-data"))
```

If you take a quick look at the loaded dataset, you should find 964 rows:

```
nrow(data)
```

Column names should correspond to those outlined in Table 1 above:

```
names(data)
```

Let's take a quick look at the first several rows:

```
head(data)
```

	id <dbl>	gender <fctr>	age <dbl>	hypertension <int>	heart_disease <int>	ever_married <fctr>	work_type <fctr>	residence_type <fctr>	avg_glucose_level <dbl>
1	62793.00	Male	37.00000	0	0	Yes	Private	Urban	79.56000
2	21162.00	Female	78.00000	0	0	Yes	Self-employed	Rural	81.68000
3	18053.38	Male	74.08813	0	0	Yes	Private	Urban	97.27607
4	28939.00	Male	64.00000	0	0	Yes	Self-employed	Rural	111.98000
5	45277.00	Female	74.00000	0	0	Yes	Private	Rural	231.61000
6	28309.00	Female	67.00000	0	0	Yes	Private	Urban	82.09000

6 rows | 1-10 of 12 columns

We can ignore column `id` as this is just a unique identifier and will not be used in predictive modeling. However, you might notice that `hypertension` and `heart_disease` are represented as `integers`. On the other hand, it might be the case (in your example) that variables such as `work_type` or `residence_type` are loaded as `characters`. To make sure we have all the variables correctly specified, we will set appropriate data types.

```

data$stroke      <- as.factor(data$stroke)
data$gender      <- as.factor(data$gender)
data$hypertension <- as.factor(data$hypertension)
data$heart_disease <- as.factor(data$heart_disease)
data$ever_married <- as.factor(data$ever_married)
data$work_type   <- as.factor(data$work_type)
data$residence_type <- as.factor(data$residence_type)
data$smoking_status <- as.factor(data$smoking_status)
data$bmi         <- as.numeric(data$bmi)

```

Once ready, run a quick summary of your dataset.

```
summary(data)
```

```

      id      gender      age      hypertension heart_disease ever_married      work_type      residence_type avg_glucose_level
Min.   : 121  Female:571  Min.   : 0.48  0:797      0:866      No :254      children   : 89      Rural:457      Min.   : 55.32
1st Qu.:19680  Male :393  1st Qu.:38.00  1:167      1: 98      Yes:710      Govt_job    :123      Urban:507      1st Qu.: 78.28
Median :37716                      Median :57.00                      Never_worked : 1                      Private     :559      Median : 96.22
Mean   :37244                      Mean   :52.60                      Self-employed:192                    Mean   :116.14
3rd Qu.:54927                      3rd Qu.:72.00                      3rd Qu.:142.87
Max.   :72918                      Max.   :82.00                      Max.   :271.74

      bmi      smoking_status      stroke
Min.   :11.30  formerly smoked:191  0:572
1st Qu.:24.50  never smoked :361    1:392
Median :28.50  smokes       :159
Mean   :29.21  unknown      :253
3rd Qu.:32.83
Max.   :60.20
NA's   :66

```

There are several important things you should notice here. The dataset seems to be very close to being balanced, given that we have 572 cases who did not have a stroke and 392 cases with stroke. What is also interesting is that we have 66 missing data points in the `bmi` column, that we will impute like we did in the previous practical.

Task 2. Handling missing values

Like we did in the previous practical, we will use `mice` package to impute missing values.

```

# Impute missing values
data.imputed <- mice(data, m=3, maxit = 50, method = 'pmm', seed = 500)
summary(data.imputed)

```

Please refer to Week 3 Practical for the outline of the arguments used to run this method. Now that we imputed the missing values, we will obtain the final dataset.

```

# Obtain a complete dataset
data.complete <- complete(data.imputed, 1)
head(data.complete)

```

Make sure you run `summary` as well and compare with the summary you run on the original data. You should notice that there are no more missing values in BMI variable. However, mean and median values should be similar to the original dataset.

```
summary(data.complete)
```

Task 3. Data exploration

We will try some of the interesting approaches for plotting and exploring variables in your dataset. In so doing, we will start by exploring numerical variables, before getting into details about categorical features.

This part requires a bit of preprocessing as we want to show histograms for **age**, **avg_glucose_level**, and **bmi** on a single plot. The easiest way to do that is to convert our data from wide to long format (please see http://www.cookbook-r.com/Manipulating_data/Converting_data_between_wide_and_long_format/).

```
data.num.long <- gather(data.complete[, c(1, 3, 9, 10)], metric, value,
  age:bmi, factor_key=TRUE)
```

What happened here?

- We selected required columns – **id**, **age**, **avg_glucose_level**, and **bmi** from the original dataset
- The new dataset - **data.num.long** – will have two new columns – **metric** and **value**.
- The **metric** column will have values “age”, “avg_glucose_level”, and “bmi”. Whereas **value** will contain values for the given row.

Let's see what the new dataset looks like.

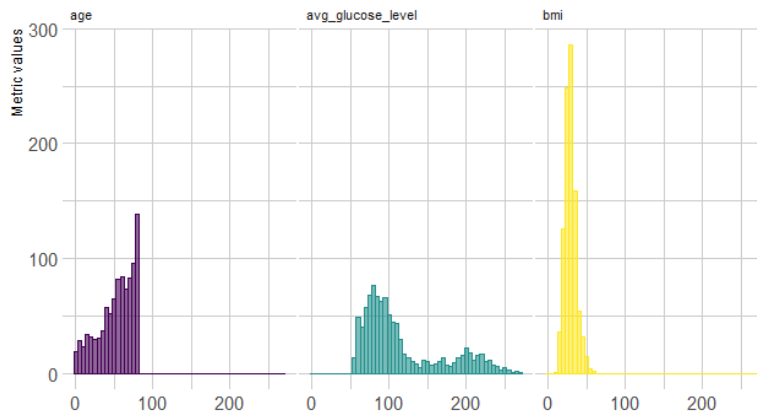
```
View(data.num.long)
```

To plot histograms for those three variables, we can do something as follows:

```
num.plot <- data.num.long %>%
  mutate(text = fct_reorder(metric, value)) %>%
  ggplot(aes(x=value, color=metric, fill=metric)) +
  geom_histogram(alpha=0.6, binwidth = 5) +
  scale_fill_viridis(discrete=TRUE) +
  scale_color_viridis(discrete=TRUE) +
  theme_ipsum() +
  theme(
    legend.position="none",
    panel.spacing = unit(0.1, "lines"),
    strip.text.x = element_text(size = 8)
  ) +
  xlab("") +
  ylab("Metric values") +
  facet_wrap(~metric)

num.plot
```

You should see something as the plot below.



In the practical from last week, we also used `ggpairs` to explore the correlation between numerical variables.

```
ggpairs(data.complete, columns = c(3, 9, 10, 12),
        ggplot2::aes(colour=stroke), progress = FALSE,
        lower=list(combo=wrap("facethist", binwidth=0.5)))
```



Based on the plot above, would you exclude any of the variables (age, average glucose level or bmi) from your predictive model? Why?

One of the great features of `ggpairs` method is that it can handle categorical variables as well. To make sure our plots are clear enough, we will create two visualizations.

```
ggpairs(data.complete, columns = c(2, 4, 5, 6), ggplot2::aes(colour=stroke),
        progress = FALSE)
```

```
ggpairs(data.complete, columns = c(7, 8, 11), ggplot2::aes(colour=stroke),
        progress = FALSE)
```

What can you say about the plots you created above? How would you interpret the association between selected pairs of variables?

Task 4. Building a Decision tree

Before we can build a predictive model, we need to partition our dataset into **training** and **test** sets. It is a common practice to keep 70-80% of the original dataset for training and 20-30% for testing.

Before doing that, we will delete `id` column, as we will not be using it in predictions. Additionally, we will again set seed as partitioning dataset is a random process and we want to be able to replicate our results.

```
data.class <- data.complete[, c(-1)]
set.seed(1000)
```

```
train_index <- sample(1:nrow(data.class), 0.8 * nrow(data.class))
test_index <- setdiff(1:nrow(data.class), train_index)
train <- data.class[train_index,]
test <- data.class[test_index,]
list(train = summary(train), test = summary(test))
```

Once we did all this, fitting a decision tree takes only a single line of code:

```
# Fitting a decision tree
c.tree <- rpart(stroke ~ ., train, method = "class")
```

Here is the outline of the arguments we used:

- `stroke ~ .` tells the method to build a decision tree that will predict `stroke` using all the available features (“.”). You can also specify a subset of features using the following notation `stroke ~ age + bmi`.
- `train` is our training set.
- `method = “class”` is used because we are predicting categorical variable (`stroke`). Other options would be “anova”, “poisson”, or “exp”. If `method` is missing then the routine tries to make an intelligent guess. If `y` is a survival object, then `method = “exp”` is assumed, if `y` has 2 columns then `method = “poisson”` is assumed, if `y` is a factor then, as in our case, `method = “class”` is assumed, otherwise `method = “anova”` is assumed. It is good practice to specify the method directly, especially as more criteria may added to the function in future.
- Recall that we discussed three measures for selecting the best split. Default measure for `rpart` is Gini. We can also set Entropy (Please refer to the documentation for this method).

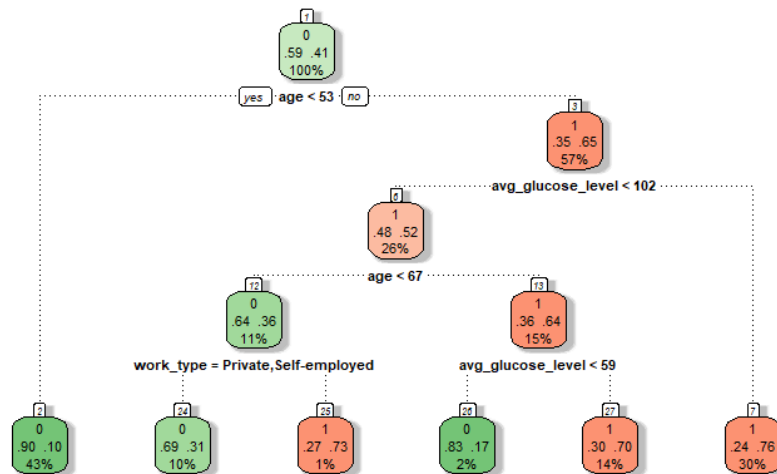
Let’s print the tree:

```
# Printing the tree
print(c.tree)
```

```
n= 771
node), split, n, loss, yval, (yprob)
* denotes terminal node
1) root 771 316 0 (0.59014267 0.40985733)
2) age< 53.40585 334 32 0 (0.90419162 0.09580838) *
3) age>=53.40585 437 153 1 (0.35011442 0.64988558)
6) avg_glucose_level< 102.475 203 97 1 (0.47783251 0.52216749)
12) age< 67.208 86 31 0 (0.63953488 0.36046512)
24) work_type=Private,Self-employed 75 23 0 (0.69333333 0.30666667) *
25) work_type=Govt_job 11 3 1 (0.27272727 0.72727273) *
13) age>=67.208 117 42 1 (0.35897436 0.64102564)
26) avg_glucose_level< 58.99 12 2 0 (0.83333333 0.16666667) *
27) avg_glucose_level>=58.99 105 32 1 (0.30476190 0.69523810) *
7) avg_glucose_level>=102.475 234 56 1 (0.23931624 0.76068376) *
```

Although we can get a lot of useful information from this printout, perhaps it might be easier to plot the tree:

```
# Plot the tree
fancyRpartPlot(c.tree, palettes = c("Greens", "Reds"), sub = "")
```



Few things to note here:

- The splitting rules are created starting with the variables that has the highest association with the response variable – `avg_glucose_level` in this case.
- **Node purity** – each node has two proportions written left and right. The leftmost leaf has .90 and .10, meaning that 90% of the node belongs to the predicted class – 0, i.e., no stroke.
- **Sample proportion** – each node also has a proportion of the sample. For the leftmost node – 43% of the sample belongs to this node.
- **Predicted class** – finally, each node also has a predicted class (0 – no stroke for the leftmost node).

As we discussed in the lecture, there are exponentially many ways to build a decision tree. **The tree we have above is not a fully grown decision tree.** If we look at the documentation for `rpart`, we will see that the default value for the complexity parameter (i.e., `cp`) is 0.05. This ensures that decision tree does not include any split that does not decrease the overall lack of fit by a factor of 5%.

If we change this parameter to 0 – `cp = 0` – we can get a fully grown tree, as follows:

```
# Fit the fully grown tree
c.tree.full <- rpart(stroke ~ ., train, method = "class", cp=0)

fancyRpartPlot(c.tree.full, palettes = c("Greens", "Reds"), sub = "")
```

The fully grown tree adds two all the predictors to the model. Although there is no rule of thumb how to set the complexity parameter, there are two things we should be aware of:

- A large tree is likely to overfit the data – we will talk about this next week.
- A small tree might miss important parameters and thus might lead to a model with high bias.

Task 5. Pruning the decision tree

An optimal tree size can be selected adaptively from the training data. What we usually do is to build a fully-grown decision tree and then extract a nested sub-tree (**prune it**) in a way that gives us the tree that has the minimal node impurities.

Let's select the best complexity parameter:

```
# Select the best complexity parameter
min.cp <-
c.tree.full$cptable[which.min(c.tree.full$cptable[, "xerror"]), "CP"]

# print the best cp
min.cp
```

Finally, let's prune the fully grown decision tree to find the optimal tree for the selected parameter:

```
# Prune the tree fully grown tree
p.tree.full<- prune(c.tree.full, cp=
c.tree.full$cptable[which.min(c.tree.full$cptable[, "xerror"]), "CP"])

fancyRpartPlot(p.tree.full, palettes = c("Greens", "Reds"), sub = "")
```

Task 6. Making a prediction

Once we have our model, we can predict labels for test dataset:

```
# Make a prediction
stroke.predict <- predict(p.tree.full, test, type = "class")

stroke.predicted.data <- cbind(test, stroke.predict)

head(stroke.predicted.data)
```

Try to compare predicted labels with the ones that were already assigned in test dataset. We will talk more about different metrics to evaluate model performance. For this practical, you can calculate average agreement between assigned and predicted labels for our test set.

```
mean(stroke.predict == test$stroke)
```



Challenge 1. We talked in previous weeks about feature selection. Also, we implemented feature selection in the previous practical.

Can you add feature selection to the pipeline outlined in this practical?

That is, before fitting a decision tree, try to apply feature selection, using the method we introduced in the previous practical.