Statistical Modeling in R (part 1)

Chapter 1 - Introduction

Statistical modeling is used for: - Identifying patterns in data~ - Classifying events - Untangling multiple influences - Assessing strength of evidence

A model is a "a representation for a purpose". A statistical model is: - A special type of mathematical model - informed by data - incorporates uncertainty and randomness

The most important blocks for this models using R, are: 1. Function 2. dataframes 3. formulas

> **Notes**: In R, columns are variables and rows are cases os observations.s

To access data contained in a R package you have a few options

1. Use the `data()` function
2. Refer to the package using double - column notation
3. Load the package, then refer to the database by name.

:heavy_exclamation_mark: The `mosaic package` augments simple statistical functions such as `mean()`, `sd()`, `median()`, etc so that they can be used with formulas. For instance the following code.

```
mosaic::mean(wage ~ sex, data = CPS85)
```

will calculate the mean wage for each sex. In contrast, the *builtin* mean()

Chapter 2 - Designing, training and evaluation models

Creating a model is a process. You start with an idea and design a model based on it. You use data to both train and evaluate the model results and design the next steps based on this information.

Steps on creating models: 1. A suitable training data set 2. Specify response and explanatory variables 3. Select a model architecture (*eg: linear regression, probit, recursive partitioning (rpart), etc)* 4. Training a model: automatic process carried out by the computer [*"fits the model to the data"*]

About variables...

We can only have one response variable per model but multiples explanatory variables.

The following functions will be used to train models during this course:

```r
# lm()  for linear models
lm(wage ~ educ + exper, data = CPS85)

# rpart() for recursive partitoning
rpart(wage ~ edu + exper, data = CPS85)

# Note that the inputs in both cases are the same with a formula (~) and a dataframe

# The following code snippet trains a linear model based on a given database [Runners]
# & plots the results
handicap_model_3 <- lm(net ~ age + sex, data = Runners)
fmodel(handicap_model_3)
```

Using the recursive partitioning There are many architectures that can be used to train a model. In this course the recursive partitioning will be used together with the linear model.

The recursive partitioning architectures has a parameter, `cp`, that allows you to dial up or down the complexity of the model being built. [issue addressed during the credit risk model course]

```r
# The syntax is identical to the one used for linear models:
rpart(net ~ sex + age, data=Runners , cp = 0.002 )
```

In the recursive partitioning architecture, the model functions have 'steps'.

New model for a new purpose

Now, let's imagine another possible purpose for a model: **to predict whether or not a person who participated in the race this year will participate next year**.

Predicting whether or not a person will run again next year is a very different purpose than finding a typical running time. For this task the dataframe has changed and now includes only includes observations of people who have run twice during the year. They included a new variable `runs_again` that represents if the person(observation) participated next year (3 years).

Note: As a comparison to the credit risk model i believe that they are analyzing the last year of the run in order to predict the year after (like the current year) .

1. The output of the model will be either TRUE or FALSE
2. The response variable `runs_again` is categorical, not numerical. Since `lm()` is intended for quantitative responses, you'll use only rpart architecture which works well for both numerical and categorical responses.

The following code example creates a rpart model and the output is a plot which showcases the probability of running depending of the age depending on the net

run time.

```
# Create run_again_model
run_again_model <- rpart(runs_again ~ age + sex + net, cp = 0.005, data= Ran_twice)

# Visualize the model (don't change)
fmodel(run_again_model, ~ age + net, data = Ran_twice)
```

Evaluating models

By evaluating, we mean give new data to the newly constructed model and evaluate it. (on the sense of this course, model evaluation will be used to make predictions on data outputs)

The model we choose impacts the conclusions. In a nutshell models work like this:

Figure 1: Esquema modelo

Provide inputs for explanatory variables & calculate the corresponding output.

To make predictions on R we use the function `predict()`. This is a code example for the creation and use of this function

```
new_input <- data.frame (edu=10:14, exper = 5)
predict(model_1, newdata = new_input)
```

We can judge how good the predictions are by using the created model with its own training data. In this case we know the correct answers which we can compare with the models prediction. **For each case we will generate a prediction error. The amount of this errors allows us to compare and evaluate the quality of the model.**

> **Quote:** The traditional function for evaluating a model is `predict()`, which takes as arguments the model and `newdata`, which gets the data frame containing the desired inputs. The result will be the model output(s) corresponding to the specified input(s).

The `statisticaModeling()` package provides an alternative to the `predict()` function called `evaluate_model()`. This function has certain advantages, such as **formatting the output as a data frame alongside the inputs**, and takes two arguments the model and the data argument containing the data frame of model inputs.

Code example for the different output:

```
predict(insurance_cost_model,newdata = example_vals)
363.637

evaluate_model(model=insurance_cost_model, data = example_vals)

  Age Sex Coverage model_output
  60   F      200      363.637
```

Extrapolations based on models

One purpose for evaluating a model is extrapolation: finding the model output for inputs that are outside the range of the data used to train the model. Extrapolation makes sense only for quantitative explanatory variables.

In order to create an extrapolation we can use the `expand.grid ()` function which creates a dataframe from all combinations of factor variables. This can be used to generate random dataframes based on some parameters.

> Quote: Sometimes you want to make a very quick check of what the model output looks like for "typical" inputs. When you use `evaluate_model()` without the data argument, **the function will use the data on which the model was trained to select some typical levels of the inputs**. `evaluate_model()` provides a tabular display of inputs and outputs. The `fmodel()` function works in the same way as `evaluate_model()`, but displays the model graphically. The syntax for `fmodel()` is

```
fmodel(model_object, ~ x_var + color_var + facet_var)
# facet stand for smaller sub-graphs (the same as in excel). Take notice that we are talki

# e.g from the course:
fmodel(insurance_cost_model, ~ Coverage + Age + Sex)

# The coverage variable is the one that will be shown on the y axis. The Age coverage will
```

Chapter 3 - Assessing prediction performance

Choosing explanatory variables

Design choices in statistical models: - the data to use for training - the response variable - the explanatory variable - the model architecture

We regularly use this models for this ends: - Make predictions about an outcome - Run experiments to study relationships between variables - Explore data to identify relationships among variables

Basic choices in model architecture (to be reviewed later) - Categorical response variables. Use rpart() - If numerical will use lm and rpart

Linear relation models is commonly used. *Is good for gradual and proportional relationships. Rpart is good for discontinuous relations.*

When you train and test a model, you use data with values for the explanatory variables as well as the response variable. Training effectively creates a function that will take as inputs values for the explanatory variables and produce as output values corresponding to the response variable.

If the model is good, when provided with the inputs from the testing data, the outputs from the function will give results "close" to the response variable in the testing data. How to measure "close"? **The first step is to subtract the function output from the actual response values in the testing data. The result is called the prediction error and there will be one such error for every case in the testing data.**

> Knowing that the models make different predictions **doesn't tell you which model is better**. You'll compare the **models' predictions to the actual values of the response variable**. The term prediction error or just error is often used, rather than difference. So, rather than speaking of the *mean square difference*, we'll say *mean square error*. **The smaller the mean square error, the closer the model outputs are to the actual response variable.**

In this example, to calculate the differences between predictions and real observation the `with()` function is used. This function *Evaluates an R expression in an environment constructed from data, possibly modifying (a copy of) the original data. Assignments within expr take place in the constructed environment and not in the user's workspace.* It follow the following syntax: `with(data,expr,...)`

```
# In the course the following example is used
base_model_differences <- with(Runners_100, net - base_model_output)
```

> **Quote:** You've seen only part of the technique for using mean square error (MSE) to decide whether to include an explanatory variable in a linear model architecture. The technique isn't yet complete because of a problem: **Whenever you use it you will find that the model with the additional explanatory variable has smaller prediction errors than the base model!**(meand, that by default more information tends to give more explanation. Nonetheless, this is information could be just a duplication of the one we had before) The technique always gives the same indication: include the additional explanatory variable. You'll start to fix this problem so that the technique of comparing MSE becomes useful and meaningful in practice.

Testing models using Cross validation

We start by diving our population used to build our model into two sets: 1. Is a training set which will be used to build our model 2. A testing set accesses model performance

> :heavy_exclamation_mark:_ The term tidy programming refers to a style where variables are always kept as part of a data frame and the functions always take a data frame as an input._

Here is a tidier way to predict and calculate MSE:

```
# As a substitute for predict()
out2 <- evaluate_model(mod, data = Runners)

# with() avoids the use of untidy $
with(data = out2, mean((net - model_output) ^ 2, na.rm = TRUE))
```

The following example showcases a possible code to set a training and testing set on a random fashion:

```
# Generate a random TRUE or FALSE for each case in Runners_100
Runners_100$training_cases <- rnorm(nrow(Runners_100)) > 0

# Build base model net ~ age + sex with training cases
base_model <- lm(net ~ age + sex, data = subset(Runners_100, training_cases))

# Evaluate the model for the testing cases
Preds <- evaluate_model(base_model, data = subset(Runners_100, !training_cases))

# Calculate the MSE on the testing data
with(data = Preds, mean((net - model_output)^2))
```

The first line of code creates a new column with randomly generated true or false values. This is created by setting this column with conditional variables. The `rnorm(nrow(Runners_100))` generates random numbers following a normal distribution using as mean the number of rows in the data frame. By adding `> 0` it creates a conditional expression.

The base_model is them created using a subset of the Runners_100 dataframe where $training_cases = 1(TRUE).

As a general rule, estimates of prediction error based on the training data will be smaller than those based on the testing data. Still, because the division into training and testing sets is done at random, it will happen from time to time that the opposite appears.

Since the result of cross validation varies from trial to trial, it's helpful to run many trials so that you can see how much variation there is. the `cv_pred_error()`

function in the `statisticalModeling` package will carry out this repetitive process.

Example for cross validation with two modes and with t-test calculation:

```
# The base model
base_model <- lm(net ~ age + sex, data = Runners_100)

# An augmented model adding previous as an explanatory variable
aug_model <- lm(net ~ age + sex + previous, data = Runners_100)

# Run cross validation trials on the two models
trials <- cv_pred_error(base_model, aug_model)

# Compare the two sets of cross-validated errors
t.test(mse ~ model, data = trials)
```

> **NOTE ABOUT T-TEST:** The t- test (also called student's T test) compares two means and tells if are different from each other. The t test also tells how significant the differences are, in other words, it lets you know if those differences could have happened by chance.

Chapter 4 - Exploring data with models

Prediction error for categorical variables

Because observation are not numerical we cannot just calculate the mean squared differences.

One workaround is to sum all instances where the prediction and the real observation differ. Therefore, it is similar to assessing performance for quantitative outputs, but we test whether predicted values match actual values and calculate the error rate.

```
# for a model to predict marriage status
with(data=Testing_data, sum(married != mod_a_outputs))

# To get the error rate we apply the mean instead of the sum
with(data=Testing_data, mean(married != mod_a_outputs))
```

Despite what was said before, it turns out that predicting a categorical variable is not the best use for a prediction model. Instead we should arrange information so that the categorical variable is represented in a numerical fashion.

Likelihood: extract the probability that the model assigned to the observed outcome.

```
likelihood_a <- with(res_1, ifelse(actual == "Married", Married, Single))
sum(log(likelihood_a))
```

For dataframe res_1 if value on column actual equals “Married” them the value of likelihood_a is the probability given by the model for Married or otherwise single.

> Generating a null model means creating a variable with no explanation (like a set of 1) that is not random. It is good to calculate the bottom line when comparing models

Exploring data for relationships

`select()` can be used to make a selection of columns. `select(data,columns names, ...)`

> **Recursive partitioning** is a statistical method for multivariable analysis. Recursive partitioning creates a decision tree that strives to correctly classify members of the population by splitting it into sub-populations based on several dichotomous independent variables. The process is termed recursive because each sub-population may in turn be split an indefinite number of times until the splitting process terminates after a particular stopping criterion is reached. Recursive partitioning methods have been developed since the 1980s. Well known methods of recursive partitioning include Ross Quinlan's ID3 algorithm and its successors, C4.5 and C5.0 and Classification and Regression Trees. Ensemble learning methods such as Random Forests help to overcome a common criticism of these methods - their vulnerability to overfitting of the data - by employing different algorithms and combining their output in some way. As compared to regression analysis, which creates a formula that health care providers can use to calculate the probability that a patient has a disease, recursive partition creates a rule such as 'If a patient has finding x, y, or z they probably have disease q'.

The `rpart()` function uses a sensible default for when to stop dividing subgroups. You can exercise some control over this with the cp argument. The function `prp()` plots a rpart model as a tree and the type 3 sets the type of tree.

> **Quote:** One way to explore data is by building models with explanatory variables that you think are important, but in my view this is really confirmation rather than exploration. [Nonetheless this could be an important first step on building a new model. *Include it on our planning for the model*]

One way to explore data is by building models with explanatory variables that you think are important, but in my view this is really confirmation rather than exploration.

Is smoking related to gestation period? Explore using models like gestation ~ . - baby_wt. (This means “explain gestation by all the other variables except baby

weight.")

Chapter 5 - Covariates and effect size

Many times the reason to create a model is to anticipate the outcome of intervention in a system.

Covariates: Explanatory variable that are not themselves of interest to the modeler, but which may shape the response variable

[In the case of our model, the interest rate might be a covariates and therefore should be taken into consideration when describing the model that is "after taking interest in consideration"]

Effect size: How much does the model output change for a given change in the input?

In our model, the inputs *cause* the output. Modeler's interest is often in cause and effect. Doesn't mean the real world system works that way.

Effect sizes carry along the unit of output and inputs. You can quantify effect size as a rate or as a difference. When the explanatory variable is quantitative, it makes sense to use as rate (Change in response/ change in input). For categorical inputs the effect size are those of the response variable.

Effect size is a property of the model and not of the data.

To calculate the effect size one can use the `effect_size()` function as in the following example

```
model <- lm(Cost ~ Age + Sex + Coverage, data = AARP)
effect_size(model,~Coverage)
# Calculates de effect size on cost by changing only Coverage
```

When comparing two different effect sizes, is important to take into consideration that results may come in different units and therefore are incomparable.