COMP 120: Lab 09 Worksheet

In this lab, we introduce the penultimate component of our data science workflow – modelling. Once you have identified what data is required, acquired that data, and made it clean and tidy, you are now in a position to start using statistical and computational means to extract the underlying stories and patterns embedded in that data. Doing so can provide tools that guide (or possibly even automate) decision making, or may even provide insights that help us understand the behaviour of the phenomena being modelled.

**It must be stressed that this lab is a brief, high-level introduction to modelling.** By the end of this lab, you will be familiar with the generic framework that R provides for modelling, prediction and model description. However, modelling is an involved and complex process (and indeed, each discipline views modelling slightly differently) that cannot be taught in a single lab. There are many other papers at Otago that help you understand modelling in greater detail, and you are encouraged to seek out these courses to help you better understand modelling within your chosen discipline.

Remember to set your working directory and library path (if working on the Student Desktop) at the start of your session. See the Lab 01 for advice on how to do this.

# Setup

Towards the end of the lab, you will be calling a modelling function called randomForest. In order to use this function, you will need the randomForest package installed and loaded in your R environment:

install.packages("randomForest")

Otherwise, there is little needed to perform modelling in R, as the standard R setup offers a comprehensive and powerful modelling framework out of the box. The only package that we will need today is the tidyverse to allow us to easily perform some data manipulation. Therefore, at the start of the lab you should load the following libraries:

library(tidyverse)

library(randomForest)

Later in this lab we will be using random splitting of data to evaluate the models that we build. This will make use of R’s built-in random number generator. To ensure that your results match what is shown in the lab, you should “fix” the seed of the random number generator so that it produces a predictable sequence. The following line achieves this:

set.seed(123) ## FIX THE RANDOM NUMBER GENERATOR FOR REPRODUCIBILITY

Finally, in this lab we will be making use of a subset of data pertaining to (historic) house prices in Boston (USA) from the 1970s. This data was used to analyse and understand the impact that atmospheric levels of nitric oxides (NOx) have on median house prices. This data has been provided on Blackboard and once downloaded, you should load it into R:

housing <- read\_csv("housing.csv")

There are seven variables in the dataset and a brief description of these variables are given below.

medv: Median value of owner-occupied homes in $1000's

chas: Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)

dis: Weighted distances to five Boston employment centres

lstat: Percentage of lower status of the population

nox: Nitric oxides concentration (parts per 10 million)

ptratio: Pupil-teacher ratio by town

rm: Average number of rooms per dwelling

# The Basics of Modelling – the Model Formula

Ultimately, the purpose of modelling is to extract a small program (typically called a function or model) that relates a set of input variables (often called **explanatory** variables or independent variables or features) to a required **response** variable (also called as outcome or dependent variable). Therefore, we need to identify our features and relate these to the response. In R, we do this through defining a **formula**. **Formula** is a special type of object that looks roughly like an algebraic equation. For example, in our housing data, we are trying to relate the response medv (median house value) to the remaining feature variables, so a possible formula for this would be:

f <- medv ~ chas + dis + lstat + nox + ptratio + rm

The code above creates a formula (named f) and can be loosely described in English as follows: our response, medv, is described by some (currently undefined) relationship of the features chas, dis, lstat, nox, ptratio and rm. ***Note that the formula doesn’t describe HOW these variables are related, it only indicates that a relationship is possible.*** It is up to the learning function (we’ll discuss one of these shortly) to characterise these relationships.

Sometimes (in fact, very often) we wish to relate the response to the rest of the features (variables) in the available data. R provides a shortcut (the dot ‘.’ operator) for such cases to make defining formulae easier:

f <- medv ~ .

This formula can be read as: medv is possibly related to ALL the available features in the provided data. We will now use this formula for the remainder of the lab.

# Creating the Model – the Learning Function

The framework for modelling in R provides a clean generic way of building and applying models:

* A learning function (which takes a formula, data, and any required parameters) constructs a model.
* Details of the model can be extracted and explained through the summary() function.
* The model, and a new set of data, are provided to the predict() function to obtain predictions of the response variable for each row in the data.

This framework of “learn” ⭢ summary(...) ⭢ predict(...) provides a simple, flexible interface that greatly simplifies model building and comparison. Most libraries that provide new modelling methods will strongly align with this framework. Here, we will demonstrate this framework, first with one of the simplest learning methods: linear regression. In linear regression, we relate the response to the features by summing weighted features and a constant :

The goal of linear regression is to use our available data to estimate the values for and the coefficients , , …, . This process of estimating the coefficients is often called “training”. For many cases, linear regression is highly effective and often forms the basis of more complex modelling processes. Therefore, it is provided out-of-the-box in R through the lm() function. To use this learning function, we supply a formula to define the model relationships and a data frame with our “training” data:

mdl <- lm(f, housing)

The variable mdl now contains the trained model, with a coefficient for each feature in the model. To extract insights from the model (e.g., which features positively or negatively relate to the response, how confident are we in the estimated values of the coefficients, how accurate is the model, etc.) we can use the summary() function:

summary(mdl)

Call:

lm(formula = f, data = housing)

Residuals:

Min 1Q Median 3Q Max

-15.4630 -2.8751 -0.6255 1.8692 28.1052

Coefficients:

Estimate Std. Error t value Pr(>|t|)

(Intercept) 36.92263 4.55909 8.099 4.29e-15 \*\*\*

chas 3.24430 0.88325 3.673 0.000265 \*\*\*

dis -1.14459 0.16672 -6.865 1.98e-11 \*\*\*

lstat -0.56984 0.04745 -12.010 < 2e-16 \*\*\*

nox -18.74043 3.22732 -5.807 1.13e-08 \*\*\*

ptratio -1.00275 0.11274 -8.895 < 2e-16 \*\*\*

rm 4.11181 0.40722 10.097 < 2e-16 \*\*\*

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 4.933 on 499 degrees of freedom

Multiple R-squared: 0.7158, Adjusted R-squared: 0.7124

F-statistic: 209.4 on 6 and 499 DF, p-value: < 2.2e-16

This summary provides a lot of useful information about the model and provides some important insights into how our input features contribute towards the median house price. For example, a “baseline” median value was around $37000 (the medv variable is in $1000 units), each additional room on a house (as indicated by the rm feature) ***increased*** the value by about $4100, while each unit increase in nitric oxides ***reduced*** the median price by around $19000. The three asterisks at the end of each line for each coefficient suggests a high level in confidence in the value that was estimated for the coefficient (greater than a 99.9% level of confidence). Finally, the second-last line of the summary provides information about the coefficient of determination (R2): one interpretation of this line is that the model accounts for 71.58% of the variation observed in the medv variable.[[1]](#footnote-1)

Once we have the model in place, we can use it to obtain predictions for any new data for which we assume (or pretend for evaluation purposes) that we don’t have a corresponding medv value. We do this through the predict() function:

yhat <- predict(mdl, housing)

Now, we can compare these predictions against any known values, if we have them. For example, we may plot the “residuals” of the model (the difference between the known and predicted values):

housing\_mod <- housing %>% mutate(residual = medv - yhat)

ggplot(housing\_mod, aes(x=medv, y=residual)) +

geom\_point() +

xlim(0, 60) +

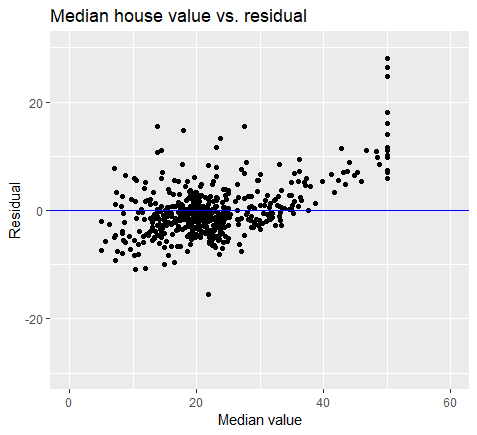
ylim(-30,30) +

geom\_abline(slope = 0, intercept=0, col = "blue") +

ggtitle("Median house value vs. residual") +

xlab("Median value") +

ylab("Residual")



The plot suggests that the model overestimates the lower end of medv, and underestimates the higher end values.

Reflection question: What is the R-squared value for the model you have created above? How can you obtain that value programmatically using the summary() function? Hint: See lecture slides for the answer.

# Model Validation and Comparison

In the previous section, we have built and examined a linear model using the lm(), summary() and predict() functions. However, the validation was a little unrealistic (the data that we used to build the model was also used to evaluate it), so we should adopt a more robust method for evaluating models.[[2]](#footnote-2) A very simple but effective approach is the “train/test split” approach, where we split our available data into two groups, train the model on one of the groups, and validate the model on the other group. We can perform such a split easily using the sample\_frac() and setdiff() functions from the tidyverse. For example, if we wanted a 50-50 split of training and testing data, we use the following:

train <- housing %>% sample\_frac(0.5)

test <- housing %>% setdiff(train)

The sample\_frac() takes two arguments a and b, where a is the dataset and b is an argument whose values is between 0 and 1. The functions returns the corresponding proportion of the dataset indicated by b (i.e., 50% of the rows in dataset a is returned when b is set to 0.5). The setdiff() function, given two arguments a and b, returns the rows in *a* that are not in *b* (i.e., the set difference between a and b).

Now, we can use the train data set in the lm() and summary() processes, and the test data set in the predict() process. That way, the model’s predictions are on data that hasn’t been seen during the learning process, and so represents a more realistic test for the model:

mdl <- lm(f, train)

summary(mdl)

Call:

lm(formula = f, data = train)

Residuals:

Min 1Q Median 3Q Max

-11.8740 -2.7924 -0.7731 1.8464 26.2706

Coefficients:

Estimate Std. Error t value Pr(>|t|)

(Intercept) 36.7769 7.1862 5.118 6.23e-07 \*\*\*

chas 4.9122 1.3985 3.512 0.000528 \*\*\*

dis -1.4675 0.2684 -5.467 1.12e-07 \*\*\*

lstat -0.6379 0.0711 -8.971 < 2e-16 \*\*\*

nox -17.5259 4.9381 -3.549 0.000463 \*\*\*

ptratio -0.9290 0.1833 -5.068 7.91e-07 \*\*\*

rm 4.1877 0.6208 6.745 1.08e-10 \*\*\*

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 5.258 on 246 degrees of freedom

Multiple R-squared: 0.7158, Adjusted R-squared: 0.7089

F-statistic: 103.3 on 6 and 246 DF, p-value: < 2.2e-16

# Use predict function to predict the results for the test data

yhat <- predict(mdl, test)

# Create a new variable called test\_mod that is made up of variables # in test plus the residual variable

test\_mod <- test %>% mutate(residual = medv - yhat)

# Visualise the residual plot

ggplot(test\_mod, aes(x=medv, y=residual)) +

geom\_point() +

xlim(0, 60) +

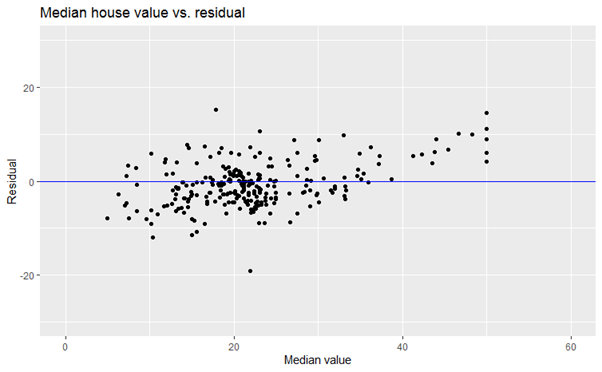
ylim(-30,30) +

geom\_abline(slope = 0, intercept=0, col = "blue") +

ggtitle("Median house value vs. residual") +

xlab("Median value") +

ylab("Residual")



In this particular case, linear modelling is fairly stable, so we don’t see much of a difference in the plot. We can also compute an R2 statistic for the test data in the following way:

rsqr <- function(y, yhat) {

ss.res <- sum((y - yhat)^2)

ss.tot <- sum((y - mean(y))^2)

1 - ss.res / ss.tot

}

rsqr(test$medv, yhat)

[1] 0.6900986

The details of the R2 statistic can be found on Wikipedia ([link](https://en.wikipedia.org/wiki/Coefficient_of_determination#Definitions)). This is the same statistic as computed in the summary() report, only this time it is computed on the previously unseen (in terms of modelling) test data. As we can see, the statistic is lower (0.69 against 0.7158), but this is more likely to be a fairer estimate of the statistic as there is less chance of bias.

Now that we have an established framework for building and evaluating models, we can start comparing between modelling approaches (e.g., to find a model with better predictive power). For example, a random forest model (a famous modelling approach that typically offers very good prediction performance) can be built and compared to linear regression as follows:

# Create a model using randomForest function (using training data)

mdl <- randomForest(f, train)

# Predict the results of the test data

yhat <- predict(mdl, test)

# Display r-squared values

rsqr(test$medv, yhat)

[1] 0.8581553

# Create a new variable that has all the columns of the test dataset # plus the residual variable

test\_mod <- test %>% mutate(residual = medv - yhat)

# Visualise the residual plot

ggplot(test\_mod, aes(x=medv, y=residual)) +

geom\_point() +

xlim(0, 60) +

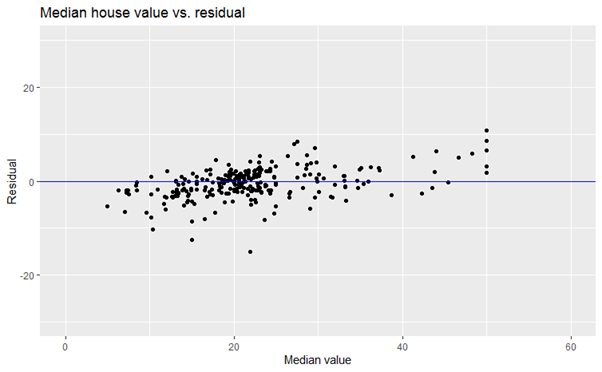
ylim(-30,30) +

geom\_abline(slope = 0, intercept=0, col = "blue") +

ggtitle("Median house value vs. residual") +

xlab("Median value") +

ylab("Residual")



This model offers somewhat better prediction performance (the R2 statistic suggests that 86% of the variance is captured by the model), but at the expense of a very difficult to interpret (and also takes more time to create the model since the model created is complex). What’s important though is that we’ve used the same basic framework to produce both models, so comparing and selecting between them becomes an efficient process.

Alright, this now completes the set of mandatory tasks for this lab. However, there were two other families of models covered in the lectures this week – *classification* and *clustering*. You are expected to understand the code that has been presented in the lectures. Note: each lecture has a corresponding demo R-file which contains code presented in the lectures. Examine the presented code, execute them and understand the outputs. You can pursue this task in this lab or in your own time.

Note that we have only scratched the surface of modelling in this course through the three families of models covered (regression, classification and clustering). Of course, there is lot more to learn about these topics. For example, the high-level intuition behind these models were presented in the lectures without focussing on the mathematical/statistical elements. This is intentional so that students from different academic backgrounds can understand the basics of modelling without getting bogged down with finer details. Those of you who are majoring in data science will learn about modelling in greater depth in the courses to follow. Hopefully, the lectures and the lab on modelling have given you a high-level overview of the models and their utility.

Now, it is time to move on to the mastery tasks for this week ☺

# Mastery Tasks

In lecture 14, we introduced the diamonds dataset containing details of about 54,000 diamonds (with ten variables), which is available as a part of ggplot2 package (which gets loaded automatically as a part of the tidyverse package). After loading the tidyverse package use the ?diamonds command to learn about the ten variables in the dataset. Examine the data in the first 10 rows of the dataset by typing diamonds.It was mentioned in lecture 14 that there appears to be a relationship between two variables *price* and *carat* (i.e., as the value for carat increases, the price increases). For this mastery, we will use the data from the diamonds dataset where the weight of the diamond (i.e., carat variable) is less than or equal to 2.5. Call this dataset as diamonds2.

Using diamonds2 dataset you will create a total of two models – one linear model and one random forest model. The first linear model will be used to predict *price* based on *carat.* Create the first model using the lm() function and call this model as *model1*. Then, create a second model using randomForest() function and call it *model2[[3]](#footnote-3)*. This model has the same explanatory and response variables as model1.

Before creating these two models, you will need to partition the diamonds2 data into training and testing data as you have done in this lab. You will use the same training and testing data for creating and evaluating both the models. The training dataset must comprise 90% of the data in the diamonds2 dataset and the rest of the data should form the testing dataset.

1. Perform steps a-h for the linear model.

[3 marks]

* 1. Write code to construct the model using the training dataset.
  2. Write down (using a comment), the resulting regression equation (for example the regression equation with three explanatory variables might be: Y = 0.895 + .75X1 + 0.86X2 – 0.5X3).
  3. Describe (using a comment) the impact of *carat* on *price*. A sample of such description from the lab example is: “each additional room on a house (as indicated by the rm feature) *increased* the value by about $4100”.
  4. Write code to obtain the R-squared value for the training dataset.
  5. Write a comment showing the value of the R-squared result in question *d* above.
  6. Write code to predict the outcomes for the test dataset.
  7. Write code to obtain the R-squared value for the test dataset. Also, using a comment indicate the R-squared value obtained.
  8. Write code to generate the residual plot for the test dataset.
  9. Describe the result of the residual plot in a few sentences focussing on the pattern you see (i.e., whether you see randomly distributed data points or a specific pattern). What might this pattern mean?

1. Perform the following steps for the random forest model: [2 mark]
   1. Write code to construct the model using the training dataset. To the randomForest function, add an extra argument ntree = 100. This argument sets the number of decision trees constructed to 100. The default value is 500. We will reduce this number to 100 so that it takes less time to create the model. Still, you may have to wait a few minutes before the model gets created.
   2. Write code to obtain the R-squared value for the training dataset. Hint: If the model is called mod2, then printing mod2 will show the result that includes a statement that conveys information along the lines of “% Var explained: XX.YY”. Note this value XX.YY is the mean of the results provided by each of the 100 decision trees in the random forest. You can also use mean(mod2$rsq) to obtain the same value.
   3. Write code to predict the outcomes for the test dataset.
   4. Write code to obtain the R-squared value for the test dataset. Also, using a comment indicate the R-squared value obtained.
   5. Write code to generate the residual plot for the test dataset.
2. Having created two models and answered the questions above, answer the following question:

*Which model out of the two models you have developed might be the best at predicting the outcomes of the test data, and why?* You must clearly explain reason(s) behind reaching your conclusion. [1 mark]

Put all the outputs for the tasks above into a single script provided (mastery-09.R). Place a short comment before each block of code that you use to complete the tasks. When you have completed the tasks, **submit your work on Blackboard before 11pm on Friday the 15th of May**. As usual, not writing good comments for the code you write will attract a penalty of 5% for each question.

1. It is important to remind you here that “accounts for” does not imply “explains” – correlation does not imply causation! Here, we are merely saying that the weighted sum of the input features produces a pattern that recreates about 72% of the observed variation in the medv variable. Further analysis is required to determine if any of these relationships are causal. [↑](#footnote-ref-1)
2. Actually, the method used in the previous section was reasonably acceptable for linear models, but certainly not acceptable for all modelling approaches. [↑](#footnote-ref-2)
3. Note: the code for generating the model using the Random Forest algorithm is likely to take time. This is because out dataset contains a large number of samples and the algorithm performs some heavy-duty computation in creating many decision-trees and then averaging the results from those into one result. So, be patient. The result should appear in a few minutes (hopefully!). [↑](#footnote-ref-3)