Applied Machine Learning by RStudio2017

Koji Mizumura

October 22, 2018

Table of Contents

<https://github.com/topepo/rstudio-conf-2018>

# Getting started

## Course Overview

The session will step through the process of building, visualizing, testing and comparing models that are focused on prediction. The goal of the course is to provide a through workflow in R that can be used with many different regression or classification techniques. Case studies are illustrated functionality.

The goal is to be able to easily build predictive/machine learning models in R using a variety of packages and model types. - “Moldes that are focused on prediction”: what does that mean? - “Machine learning”: so this is deep learning with massive data sets, right?

The course is broken up into sections for regression (predicting numeric outcome) and classification (predicting a category).

## Why R for modeling?

1. R has *cutting edge models*. Machine learning developers in some domains use R as their primary computing environment and their work often results in R packages.
2. It is easy to port or link to other applications. R doesn’t try to be everything to everyone. If you prefer models implemented in C, C++, tensorflow, keras, python, stan, or Weka, you can access these applications without leaving R.
3. R and R packages are built by people who **do** data analysis.
4. The S language is very mature.
5. The machine learning environment in R is extremely rich.

## Downsides to modeling in R

1. R is a data analysis language and is not C or Java. If a high performance deployment is required, R can be treated like a prototyping language.
2. R is s mostly memory-bound. There are plenty of exceptions to this though.
3. The main issue is one of consistency of interface.

For example: - here are two methods for specifying what terms are in a model1. Not all models have both. - 99% of model functions automatically generate dummy variables. - Sparse matrices can be used (unless the can’t).

## Syntax for computing predicted class probabilities

|  |  |  |
| --- | --- | --- |
| **Function** | **Package** | **Code** |
| lda | MASS | predict(obj) |
| glm | stats | predict(obj, type = “response”) |
| gbm | gbm | predict(obj, type = “response”, n.trees) |
| mda | mda | predict(obj, type = “posterior”) |
| rpart | rpart | predict(obj, type = “prob”) |
| Weka | RWeka | predict(obj, type = “probability”) |
| logitboost | LogitBoost | predict(obj, type = “raw”, nIter) |

## Different philosophies used here

There are two main philosophies to data analysis code that will be discussed in this workshop:

The main traditional approach uses high-level syntax and is perhaps the most **untidy** code that you will encounter.

caret is the primary package for untidy predictive modeling: 1. More traditional R coding style. 2. High-level “I do that for you” syntax. 3. More comperehensive (for now) and less modlular. 4. Contains many optimizations and is easily parallelized.

The *tidy* modeling approach espouses the tenets of the tidyverse 1. Reuses existing data structures 2. Compose simple functions with the pipe 3. Embrase functional programming 4. Design for humans

This approach is exemplified by packages such as: modelr, broom, recipes, rsample, yardstick and tidyposterior.

## Example data set - house prices

For regression problems, we will use the Ames IA housing data. There are 2,930 properties in the data.

The sale price was recorded along 81 predictors, including - Location (e.g. neighborhood) and lot information. - House components (garage, fireplace, pool, porch, etc.). - General assessments such as overall quality and condition. - Number of bedrooms, baths, and so on.

More details can be found in De Cock (2011, Journal of Statistics Education).

he raw data are at <http://bit.ly/2whgsQM> but we will use a processed version found in the AmesHousing package.

library(AmesHousing)  
AmesHousing::ames\_raw

## # A tibble: 2,930 x 82  
## Order PID `MS SubClass` `MS Zoning` `Lot Frontage` `Lot Area` Street  
## <int> <chr> <chr> <chr> <int> <int> <chr>   
## 1 1 0526~ 020 RL 141 31770 Pave   
## 2 2 0526~ 020 RH 80 11622 Pave   
## 3 3 0526~ 020 RL 81 14267 Pave   
## 4 4 0526~ 020 RL 93 11160 Pave   
## 5 5 0527~ 060 RL 74 13830 Pave   
## 6 6 0527~ 060 RL 78 9978 Pave   
## 7 7 0527~ 120 RL 41 4920 Pave   
## 8 8 0527~ 120 RL 43 5005 Pave   
## 9 9 0527~ 120 RL 39 5389 Pave   
## 10 10 0527~ 060 RL 60 7500 Pave   
## # ... with 2,920 more rows, and 75 more variables: Alley <chr>, `Lot  
## # Shape` <chr>, `Land Contour` <chr>, Utilities <chr>, `Lot  
## # Config` <chr>, `Land Slope` <chr>, Neighborhood <chr>, `Condition  
## # 1` <chr>, `Condition 2` <chr>, `Bldg Type` <chr>, `House Style` <chr>,  
## # `Overall Qual` <int>, `Overall Cond` <int>, `Year Built` <int>, `Year  
## # Remod/Add` <int>, `Roof Style` <chr>, `Roof Matl` <chr>, `Exterior  
## # 1st` <chr>, `Exterior 2nd` <chr>, `Mas Vnr Type` <chr>, `Mas Vnr  
## # Area` <int>, `Exter Qual` <chr>, `Exter Cond` <chr>, Foundation <chr>,  
## # `Bsmt Qual` <chr>, `Bsmt Cond` <chr>, `Bsmt Exposure` <chr>, `BsmtFin  
## # Type 1` <chr>, `BsmtFin SF 1` <int>, `BsmtFin Type 2` <chr>, `BsmtFin  
## # SF 2` <int>, `Bsmt Unf SF` <int>, `Total Bsmt SF` <int>,  
## # Heating <chr>, `Heating QC` <chr>, `Central Air` <chr>,  
## # Electrical <chr>, `1st Flr SF` <int>, `2nd Flr SF` <int>, `Low Qual  
## # Fin SF` <int>, `Gr Liv Area` <int>, `Bsmt Full Bath` <int>, `Bsmt Half  
## # Bath` <int>, `Full Bath` <int>, `Half Bath` <int>, `Bedroom  
## # AbvGr` <int>, `Kitchen AbvGr` <int>, `Kitchen Qual` <chr>, `TotRms  
## # AbvGrd` <int>, Functional <chr>, Fireplaces <int>, `Fireplace  
## # Qu` <chr>, `Garage Type` <chr>, `Garage Yr Blt` <int>, `Garage  
## # Finish` <chr>, `Garage Cars` <int>, `Garage Area` <int>, `Garage  
## # Qual` <chr>, `Garage Cond` <chr>, `Paved Drive` <chr>, `Wood Deck  
## # SF` <int>, `Open Porch SF` <int>, `Enclosed Porch` <int>, `3Ssn  
## # Porch` <int>, `Screen Porch` <int>, `Pool Area` <int>, `Pool  
## # QC` <chr>, Fence <chr>, `Misc Feature` <chr>, `Misc Val` <int>, `Mo  
## # Sold` <int>, `Yr Sold` <int>, `Sale Type` <chr>, `Sale  
## # Condition` <chr>, SalePrice <int>

library(AmesHousing)  
ames\_geo

## # A tibble: 2,930 x 3  
## PID Longitude Latitude  
## <chr> <dbl> <dbl>  
## 1 0526301100 -93.6 42.1  
## 2 0526350040 -93.6 42.1  
## 3 0526351010 -93.6 42.1  
## 4 0526353030 -93.6 42.1  
## 5 0527105010 -93.6 42.1  
## 6 0527105030 -93.6 42.1  
## 7 0527127150 -93.6 42.1  
## 8 0527145080 -93.6 42.1  
## 9 0527146030 -93.6 42.1  
## 10 0527162130 -93.6 42.1  
## # ... with 2,920 more rows

## Assuming "Longitude" and "Latitude" are longitude and latitude, respectively

## PhantomJS not found. You can install it with webshot::install\_phantomjs(). If it is installed, please make sure the phantomjs executable can be found via the PATH variable.

## Example data set - Fuel economy

The data that are used here are an extended version of the ubiquitous mtcars data set. [fueleconomy.gov](https://www.fueleconomy.gov/feg/download.shtml) was used to obtain fuel efficiency data on cars from 2015-18.

Over this time range, duplicate ratings were eliminated; these occur when the same car is sold for several years in a row. As a result, there are 3294 cars that are listed in the data. The predictors include the automaker and addition information about the cars (e.g. intake valves per cycle, aspiration method, etc).

In our analysis, the data from 2015-2017 are used for training to see if we can predict the 609 cars that were new in 2018.

These data are supplied in the GitHub repo.

## Example data set - Predicting profession

OkCupid is an online data site that serves international users. Kim and Escobedo-Land (2015, Journal of Statistics Education) describe a data set where over 50,000 profiles from the San Fransisco area were made available by the company.

The data contains several types of fields:

* a number of open text essays related to interests and personal descriptions
* single choice type fields, such as profession, diet, gender, body type, etc.
* multiple choice data, including languages spoken, etc.
* **no** usernames or pictures were included.

We will try to predict whether someone has a profession in the STEM fields (science, technology, engineering, and math) using a random sample of the overall dataset.

## Tidyverse syntax

Many tidyverse functions have syntax unlike base R code. For example:

* vectors of variable names are eschewed in favor of *functional programming*. For example:

contains("Sepal")  
  
# instead of  
c("Sepal.Width", "Sepal.Length")

* The *pipe* operator is preferred. For example:

merged <- inner\_join(a, b)  
# is equal to  
merged <- a %>%  
 inner\_join(b)

* Functions are more *modular* than their traditional analogs (dplyr’s filter and select VS base::subset).

## Some example data manipulation code

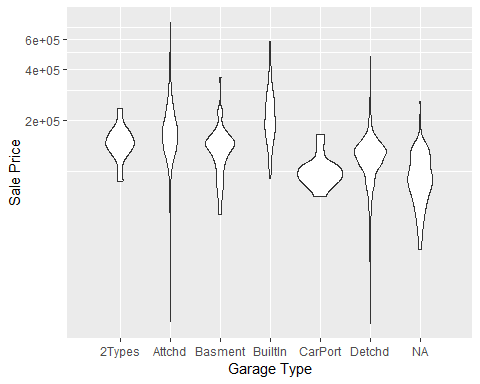
library(tidyverse)  
  
ames <- read\_delim("http://bit.ly/2whgsQM", delim = "\t") %>%  
 rename\_at(vars(contains(' ')), funs(gsub(' ', '\_', .))) %>%  
 rename(Sale\_Price = SalePrice) %>%  
 filter(!is.na(Electrical)) %>%  
 select(-Order, -PID, -Garage\_Yr\_Blt)

library()  
  
ames <- ames\_raw %>%   
 rename\_at(vars(contains(' ')), funs(gsub(' ', '\_', .))) %>%  
 rename(Sale\_Price=SalePrice) %>%   
 filter(!is.na(Electrical)) %>%   
 select(-Order,-PID, -Garage\_Yr\_Blt)  
   
   
ames %>%   
 group\_by(Alley) %>%   
 summarize(mean\_price=mean(Sale\_Price/1000),  
 n=sum(!is.na(Sale\_Price)))

## # A tibble: 3 x 3  
## Alley mean\_price n  
## <chr> <dbl> <int>  
## 1 Grvl 124. 120  
## 2 Pave 177. 78  
## 3 <NA> 183. 2731

## Example ggplot2 code

library(ggplot2)  
  
ggplot(ames,  
 aes(x=Garage\_Type,  
 y=Sale\_Price))+  
 geom\_violin()+  
 coord\_trans(y="log10")+  
 xlab("Garage Type")+  
 ylab("Sale Price")



## Examples of purrr::map\*

library(purrr)  
  
# Summarize via purrr::map  
by\_alley <- split(ames, ames$Alley)  
is\_list(by\_alley)

## [1] TRUE

# glimpse(by\_alley)

map(by\_alley, nrow)

## $Grvl  
## [1] 120  
##   
## $Pave  
## [1] 78

map\_int(by\_alley, nrow)

## Grvl Pave   
## 120 78

# work on no-list vectors too  
ames %>%   
 mutate(Sale\_Price=Sale\_Price %>%   
 map\_dbl(function(x)x/1000)) %>%   
 select(Sale\_Price, Yr\_Sold) %>%   
 head()

## # A tibble: 6 x 2  
## Sale\_Price Yr\_Sold  
## <dbl> <int>  
## 1 215 2010  
## 2 105 2010  
## 3 172 2010  
## 4 244 2010  
## 5 190. 2010  
## 6 196. 2010

## Quick data investigation

To get warmed up, let’s load the Ames data and do some basic investigations into the variables, such as exploratory visualizations or summary statistics. The idea is to get a feel for the data.

library(AmesHousing)  
ames <- make\_ames()

## Where we go from here

**Part 2** Basic Principles - Data Splitting, Models in R, Resampling, Tuning(rsample)

**Part 3** Feature engineering preprocessing - Data treatment (recipes)

**Part 4** Regression Modeling - Measuring Performance, penalized regression, multivariate adaptive regression splines (MARS), ensembles (yardstick, recipes, caret, earth, glmnet, tidyposterior, doParallel)

**Part 5** Classification Modeling - Measuring Performance, trees, ensembles, naive Bayes (yardstick, recipes, caret, rpart, klaR, tidyposterior)

## Resources

<http://www.tidyverse.org/> [R for Data Science](http://r4ds.had.co.nz/) [Jenny’s purrr tutorial](https://jennybc.github.io/purrr-tutorial/) or [Happy R Users Purrr](https://www.rstudio.com/resources/videos/happy-r-users-purrr-tutorial/) [Programming with dplyr vignette](https://cran.r-project.org/web/packages/dplyr/vignettes/programming.html) [Selva Prabhakaran’s ggplot2 tutorial](http://r-statistics.co/Complete-Ggplot2-Tutorial-Part1-With-R-Code.html) [caret package documentation](https://topepo.github.io/caret/) [CRAN Machine Learning Task View](https://cran.r-project.org/web/views/MachineLearning.html)

About these slides…. they were created with Yihui’s xaringan and the stylings are a slightly modified version of Patrick Schratz’s Metropolis theme.

# Part2 Basic principles

## Introduction

In this section, we will introduce concepts that are useful for any type of machine learning model: - modeling versus the model - data splitting - resampling - tuning parameters and overfitting - model tuning

Many of these topics will be put into action in later sections.

### The modeling process

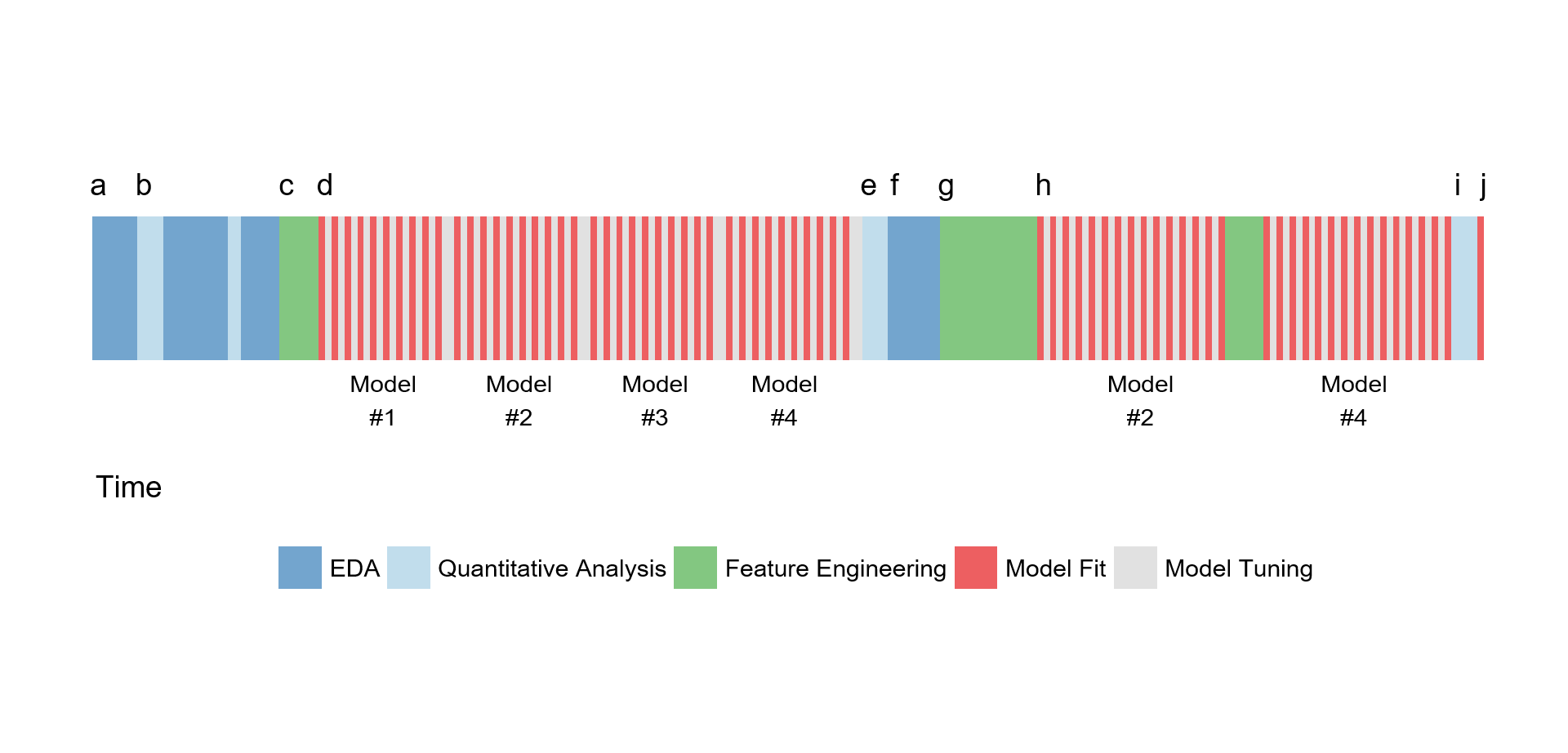
Common steps during model building are:

* estimating model parameters (i.e., training models)
* determining the values of *tuning parameters* that cannot be directly calculated from the data
* model selection (within a model type) and model comparison (between types)
* calculating the performance of the final model that will generalize to new data

Many books and course portray predictive modeling as a short sprint. A better analogy would be a marathon or campaign (depending on how hard the problem is).

### What the modeing process usually look like

knitr::include\_graphics("C:/Users/kojikm.mizumura/Desktop/Data Science/UseR 2018/Applied ML/intro-process-1.png")



## Data usage

### Data Splitting and spending

How do we “spend” the data to find an optimal model? We typically split data into training an test data sets:

* **Training set**: these data are used to estimate model parameters and pick the values of the complexity parameter(s) for the model.
* **Test set**: these data can be used to get an independent assessment of model efficacy. They should not be used during model training.

The more data we spend, the better estimates we’ll get (provided the data is accurate).

Given a fixed amount of data: - too much spent in training won’t allow us to get a good assessment of predictive peformance. We may find a model that fits the training data very well, but is not generalizable (overfitting) - too much spent in testing won’t allow us to get a good assessment of model parameters

Statisticall,y the est course of action would be use all the data for model building and use statistical methods to get estimates of error.

From a non-statistical perspective, many consmers of complex models emphasize the need for untouched set of sampled to evaluate performance.

### Large data set

When a large amount of data are available, it might seem like a good idea to put a large amount into the training set. *Personally*, I think that this causes more trouble than it is worth due to diminishing returns on performance and the added cost and compexity of the required infrastructure.

Alternatively, it is probably a better idea to reserve good percentages of the data for specific parts of the modeling process. For example:

* Save a large chunk of data to perform feature selection prior to model building
* Retain data to calibrarate class probabilities or determine a cutoff via an ROC curve.

Also there may be little need for iterative resampling of the data. A single holdout (aka validation set) may be sufficient in some cases if the data are large enough and the data sampling mechanism is solid.

### Mechanis of data splitting

There are a few different ways to do the split: simple random sampling, stratified sampling based on the outcome, by date, or methods that focus on the distribution of the predictors.

For stratification: - **classification**: this would mean sampling within the classes as to preserve the distribution of the outcome in the training and test sets - **regression**: determine the quartiles of the data set and samples within those artificial groups

### Ames Housing data

ames <- make\_ames()  
dim(ames)

## [1] 2930 81

library(rsample)

##   
## Attaching package: 'rsample'

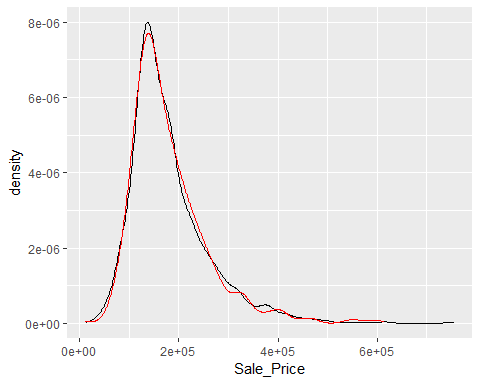
## The following object is masked from 'package:tidyr':  
##   
## fill

# make suret you get the same random numbers  
set.seed(4595)  
  
data\_split <-initial\_split(ames,strata="Sale\_Price")   
  
ames\_train <- training(data\_split)  
ames\_test <- testing(data\_split)  
  
nrow(ames\_train)/nrow(ames)

## [1] 0.7505119

### Outcome distribution

library(ggplot2)  
  
# Do the distribution line-up?  
ggplot(ames\_train,aes(x=Sale\_Price))+  
 geom\_line(stat = "density",   
 trim = TRUE) +   
 geom\_line(data = ames\_test,   
 stat = "density",   
 trim = TRUE, col = "red")



## Creating models in R

### Specifying models in R using formulas

To fit a model to the housing data, the model terms must be specified. Historically, there are two main interfaces for doing this.

The fomula interface using R [fomula rules](https://cran.r-project.org/doc/manuals/r-release/R-intro.html#Formulae-for-statistical-models) to specify a symbolic representation of the terms and variables. For example:

foo(Sale\_Price ~ Neightborhood + Year\_Sold + Neighborhood:Year\_Sold, data=ames\_train)

OR

foo(Sale\_Price~., data=ames\_train)

OR

foo(log10(Sale\_Price)~ns(Longitude, df=3)+ns(Latitude,df=3),data=ames\_train)

This is very convenient but it has some disadvantages.

### Downsides to formulas

* You can’t nest in-line functions such as foo(y ~ pca(scale(x1), scale(x2), scale(x3)), data =dat).
* All the model matrix calculations happen at once and can’t be recycled when used in a model function.
* For very *wide* data sets, the formula method can be extremely inefficient.
* There are limited *roles* that variables can take which has led to several re-impementations of formulas.
* Specifyng multivarite outcomes
* Not all model functions have a formula method.

### Specifying model without formulas

Some modeling function have the non-formula interface. This usually has arguments for the predictors and the outcome(s):

# Usually, the variable must all be numeric  
  
pre\_vars <- c("Year\_Sold","Longitude","Latitude")  
foo(x=ames\_train[,pre\_vars],  
 y=ames\_train$Sale\_Price)

This is inconvenient if you have transformations, factor variables, interactions or any other operations apply prior to modeling.

Overall, it is difficult to predict if a package has one or both of these interfaces. For example, lm only has formulas.

There is a **third interface** using *recipes* that will be discussed later that solve some of these issues.

### A linear regression model

Let’s start by fitting an ordinary linear regression model to the training set. You can choose the model terms for your model but I will use a very simple model:

head(ames\_train)

## # A tibble: 6 x 81  
## MS\_SubClass MS\_Zoning Lot\_Frontage Lot\_Area Street Alley Lot\_Shape  
## <fct> <fct> <dbl> <int> <fct> <fct> <fct>   
## 1 One\_Story\_~ Resident~ 141 31770 Pave No\_A~ Slightly~  
## 2 Two\_Story\_~ Resident~ 74 13830 Pave No\_A~ Slightly~  
## 3 Two\_Story\_~ Resident~ 78 9978 Pave No\_A~ Slightly~  
## 4 One\_Story\_~ Resident~ 43 5005 Pave No\_A~ Slightly~  
## 5 One\_Story\_~ Resident~ 39 5389 Pave No\_A~ Slightly~  
## 6 Two\_Story\_~ Resident~ 60 7500 Pave No\_A~ Regular   
## # ... with 74 more variables: Land\_Contour <fct>, Utilities <fct>,  
## # Lot\_Config <fct>, Land\_Slope <fct>, Neighborhood <fct>,  
## # Condition\_1 <fct>, Condition\_2 <fct>, Bldg\_Type <fct>,  
## # House\_Style <fct>, Overall\_Qual <fct>, Overall\_Cond <fct>,  
## # Year\_Built <int>, Year\_Remod\_Add <int>, Roof\_Style <fct>,  
## # Roof\_Matl <fct>, Exterior\_1st <fct>, Exterior\_2nd <fct>,  
## # Mas\_Vnr\_Type <fct>, Mas\_Vnr\_Area <dbl>, Exter\_Qual <fct>,  
## # Exter\_Cond <fct>, Foundation <fct>, Bsmt\_Qual <fct>, Bsmt\_Cond <fct>,  
## # Bsmt\_Exposure <fct>, BsmtFin\_Type\_1 <fct>, BsmtFin\_SF\_1 <dbl>,  
## # BsmtFin\_Type\_2 <fct>, BsmtFin\_SF\_2 <dbl>, Bsmt\_Unf\_SF <dbl>,  
## # Total\_Bsmt\_SF <dbl>, Heating <fct>, Heating\_QC <fct>,  
## # Central\_Air <fct>, Electrical <fct>, First\_Flr\_SF <int>,  
## # Second\_Flr\_SF <int>, Low\_Qual\_Fin\_SF <int>, Gr\_Liv\_Area <int>,  
## # Bsmt\_Full\_Bath <dbl>, Bsmt\_Half\_Bath <dbl>, Full\_Bath <int>,  
## # Half\_Bath <int>, Bedroom\_AbvGr <int>, Kitchen\_AbvGr <int>,  
## # Kitchen\_Qual <fct>, TotRms\_AbvGrd <int>, Functional <fct>,  
## # Fireplaces <int>, Fireplace\_Qu <fct>, Garage\_Type <fct>,  
## # Garage\_Finish <fct>, Garage\_Cars <dbl>, Garage\_Area <dbl>,  
## # Garage\_Qual <fct>, Garage\_Cond <fct>, Paved\_Drive <fct>,  
## # Wood\_Deck\_SF <int>, Open\_Porch\_SF <int>, Enclosed\_Porch <int>,  
## # Three\_season\_porch <int>, Screen\_Porch <int>, Pool\_Area <int>,  
## # Pool\_QC <fct>, Fence <fct>, Misc\_Feature <fct>, Misc\_Val <int>,  
## # Mo\_Sold <int>, Year\_Sold <int>, Sale\_Type <fct>, Sale\_Condition <fct>,  
## # Sale\_Price <int>, Longitude <dbl>, Latitude <dbl>

simple\_lm <- lm(log10(Sale\_Price)~Longitude+Latitude, data=ames\_train)

Before looking at coefficients, we should do some model checking to see if there is anything obviously wrong with the model.

To get the statistics on the individual points, we willl use the awesome broom package:

library(broom)  
library(magrittr)  
  
simple\_lm\_values <- augment(simple\_lm)  
simple\_lm\_values %>% names()

## [1] "log10.Sale\_Price." "Longitude" "Latitude"   
## [4] ".fitted" ".se.fit" ".resid"   
## [7] ".hat" ".sigma" ".cooksd"   
## [10] ".std.resid"

### Hands-on: some basic diagnostics

From these results, let’s do some visualizations:

* Plot the observed versus fitted values
* Plot the residuals
* Plot the predicted versus resdiduals

Are there any downsides to this approach?

## Model evaluation

### Overall model statistics

If you use the summary method on the lm object, the buttom shows some statistics.

summary(simple\_lm)

##   
## Call:  
## lm(formula = log10(Sale\_Price) ~ Longitude + Latitude, data = ames\_train)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -1.01769 -0.09771 -0.01536 0.10003 0.57637   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) -316.1528 15.0273 -21.04 <2e-16 \*\*\*  
## Longitude -2.0792 0.1346 -15.45 <2e-16 \*\*\*  
## Latitude 3.0135 0.1880 16.03 <2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 0.1614 on 2196 degrees of freedom  
## Multiple R-squared: 0.1808, Adjusted R-squared: 0.1801   
## F-statistic: 242.3 on 2 and 2196 DF, p-value: < 2.2e-16

These statistics are the result of predicting the same data that was used to derive the coefficients. This is problematic because it can lead to optimistic results, especially for models that are extremely flexibile.

The test set is used for assessing performance. **Should we predict the test set** and use those results to estimate these statistics.

**NOPE!**

### Assessing models

Save the test set until the very end when you have one or two models that are your favorite. We’ll need to use the training set but…

For some models, it is possible to get very small residuals by predicting the training set. That’s an issue since we will need to make comparisons between models, create diagnostic plots, etc.

If only we had a method for getting honest performance estimates from the training set..

### Resampling methods

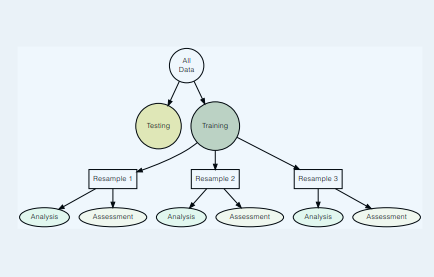
There are additional data splitting schemes that are applied to *training set*. They attempt to stimulate slightly different versions of the traing set. These versions of the original are split into two model subsets.

* The analysis set is used to fit the model (analogous to the training set)
* Performance is determined using the assessment set.

This process is repated many times. There are different flavors or resampling but we will focus on two methods.

### V-Fold Cross-validation

knitr::include\_graphics("C:/Users/kojikm.mizumura/Desktop/Data Science/UseR 2018/Applied ML/resampling methods.PNG")



These are additonal data splitting that are applied to the *training* set. They attempt to simpuate slightly different versions of the *training* set. These versions of the oroginal are split into two model subsets. - The *analysis* set ised o fit the model (analogous to the training set) - Peformance is determined using the *assessment* set.

This process is repeated many times.

There are different flavors or resampling but we will focus on two methods.

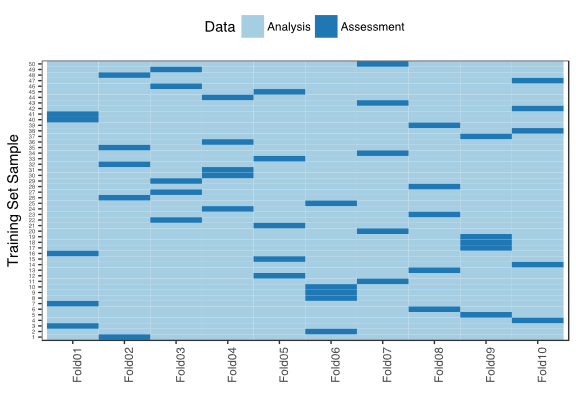
### V-Fold Cross Validation

Here, we randomly split the training data into V distinct blocks of roughly equal size. - We leave out the first block of analysis data and fit a model. - This model is used to predict the held-out block of assessment data. - We continue this process until we’ve predicted all V assessment blocks.

The final performance is based on the hold-out predictions by *averaging* the statistics from the V blocks. V is usually taken to be 5 or 10 and leave one out cross-validation has each sample as a block.

### 10-Fold Cross-Validation with n=50

knitr::include\_graphics("C:/Users/kojikm.mizumura/Desktop/Data Science/UseR 2018/Applied ML/cv-plot-1.png")



### Bootstrapping

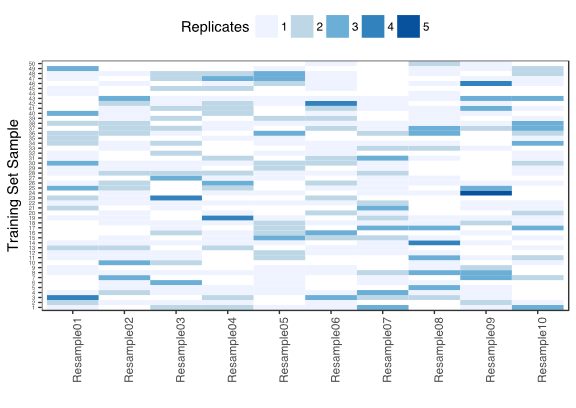
A **boostrap sample** is the same size as the training set but each data point is selected with replacement.

This means that the analysis set will have more than one replicate of a training set instance.

The assessment set contains all samples that were never included in the bootstrap set. It is often called the out-of-bag sample and can vary in size.

On average, 63.1220559% of the training set is contained at least once in the bootstrap sample.

knitr::include\_graphics("C:/Users/kojikm.mizumura/Desktop/Data Science/UseR 2018/Applied ML/boot-plot-1.png")



### Comparing Resampling methods