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

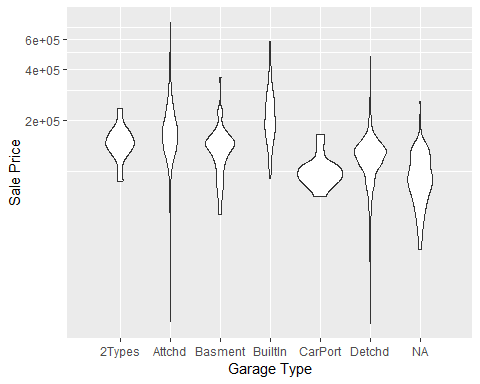
library(tidyverse)  
  
ames <- read\_delim("http://bit.ly/2whgsQM", delim = "\t") %>%  
 rename\_at(vars(contains(' ')), funs(gsub(' ', '\_', .))) %>%  
 rename(Sale\_Price = SalePrice) %>%  
 filter(!is.na(Electrical)) %>%  
 select(-Order, -PID, -Garage\_Yr\_Blt)

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

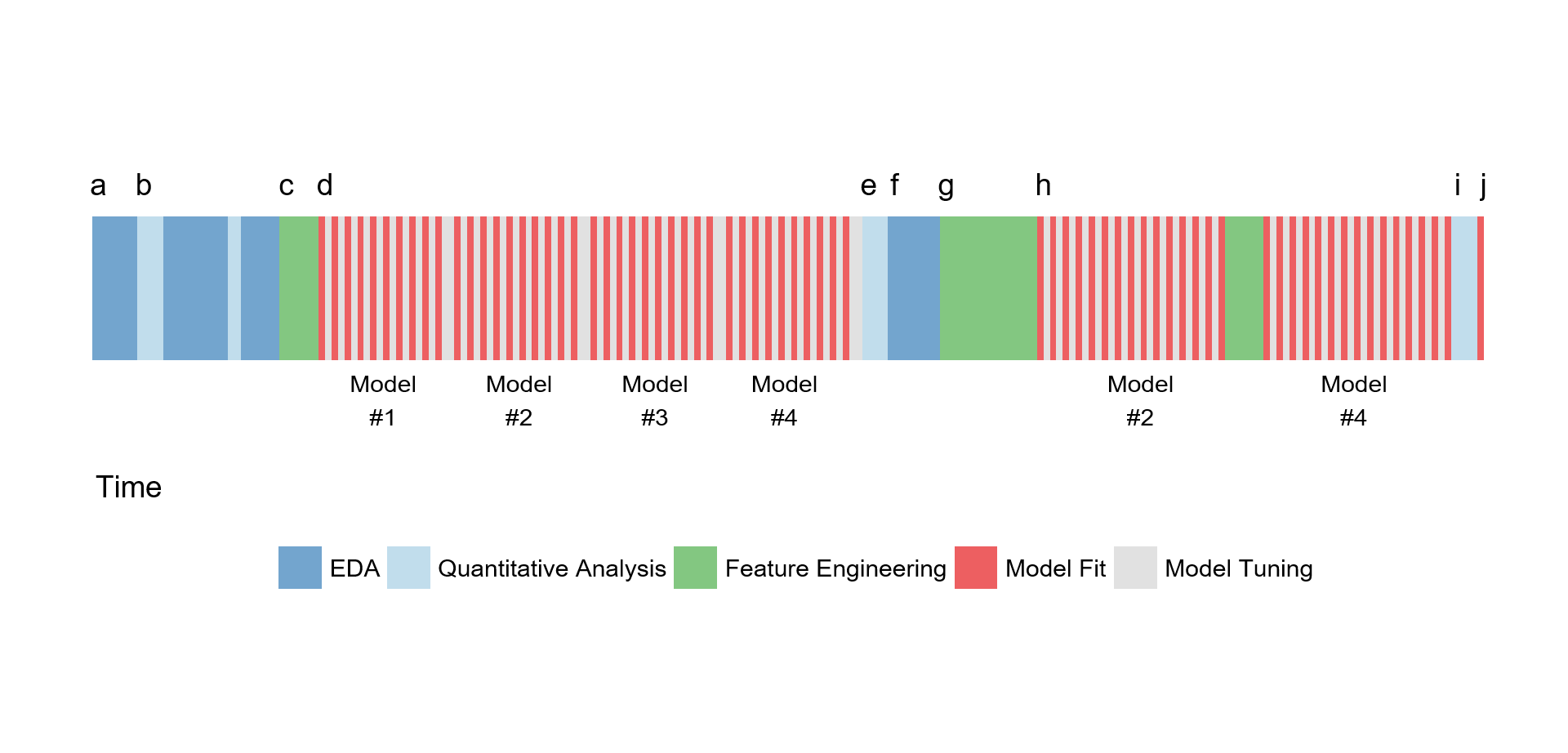
Common steps during model building are:

* 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)

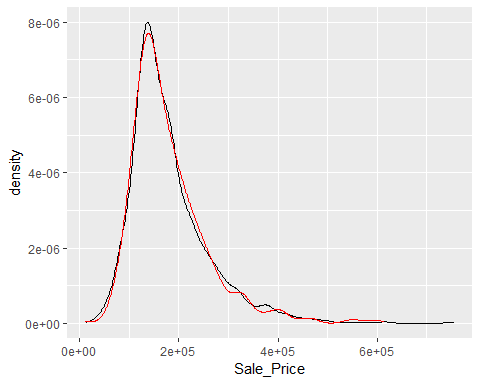
## [1] 2930 81

library(rsample)  
  
# 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

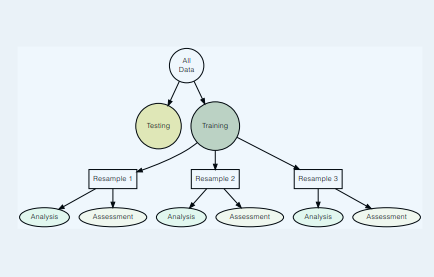
There are additional data splitting schemes that are applied to *training set*. They attempt to stimulate slightly different versions of the traing set. These versions of the original are split into two model subsets.

* 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.

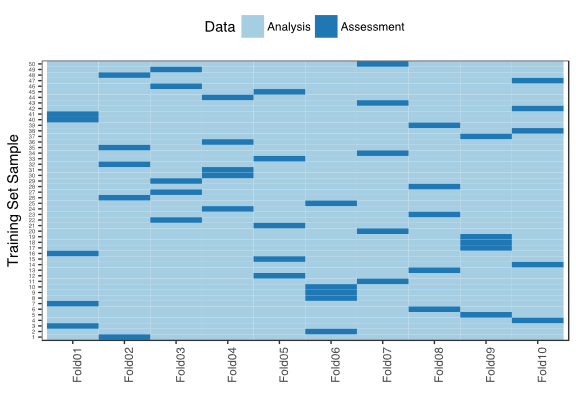
### V-Fold Cross Validation

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

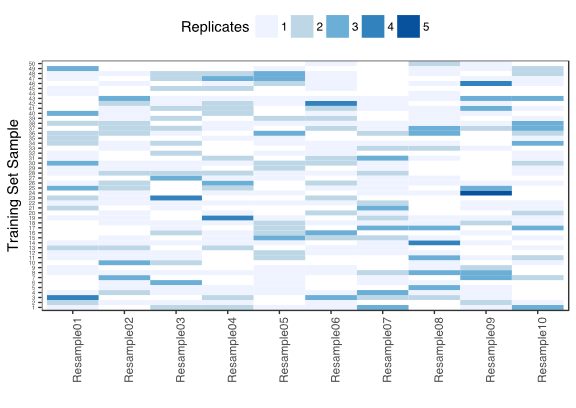
A **boostrap sample** is the same size as the training set but each data point is selected with replacement.

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

If you think of resampling in the same manner as statistical estimators (e.g., maximum likelihood), this becomes a trade-off bias and variance:

* variance is (mostly) driven by the number of resamples (e.g., 5-fold CV larger variance than 10-folds).
* Bias is (mostly) related to how much data is held back. The bootstrap has large bias compared to 10-fold CV.

There are length blog posts about this subject [here](http://appliedpredictivemodeling.com/blog/2014/11/27/vpuig01pqbklmi72b8lcl3ij5hj2qm) and [here](http://appliedpredictivemodeling.com/blog/2014/11/27/08ks7leh0zof45zpf5vqe56d1sahb0).

I tend to favor 5 repeats of 10-fold cross-validation unless the size of the assessment data is “large enough”.

For example, 10% of the Ames training set is 219 properties and this is probably good enough to estimate the and .

### Cross-validating using rsample

library(AmesHousing)  
library(rsample)  
  
set.seed(2433)  
cv\_splits <- vfold\_cv(ames\_train, v=10, strata="Sale\_Price")  
cv\_splits

## # 10-fold cross-validation using stratification   
## # A tibble: 10 x 2  
## splits id   
## <list> <chr>   
## 1 <S3: rsplit> Fold01  
## 2 <S3: rsplit> Fold02  
## 3 <S3: rsplit> Fold03  
## 4 <S3: rsplit> Fold04  
## 5 <S3: rsplit> Fold05  
## 6 <S3: rsplit> Fold06  
## 7 <S3: rsplit> Fold07  
## 8 <S3: rsplit> Fold08  
## 9 <S3: rsplit> Fold09  
## 10 <S3: rsplit> Fold10

The split objects contain the information about the sample size.

cv\_splits$splits[[1]]

## <1977/222/2199>

We use the analysis and assessment functions to get the data.

analysis(cv\_splits$splits[[1]]) %>% dim()

## [1] 1977 81

assessment(cv\_splits$splits[[1]]) %>% dim()

## [1] 222 81

### Resampling the linear model

We Will need to write a function to fit the model to each data set and another to compute performance.

library(yardstick)

## Warning: package 'yardstick' was built under R version 3.5.1

##   
## Attaching package: 'yardstick'

## The following objects are masked from 'package:caret':  
##   
## mnLogLoss, precision, recall

## The following object is masked from 'package:readr':  
##   
## spec

lm\_fit <- function(data\_split, ...){  
 lm(..., data=analysis(data\_split))}  
  
# A formula is also needed for each model:  
form <- as.formula(  
 log10(Sale\_Price)~Longitude+Latitude)

For performance, the first argument should the rsplit objected contained in cv\_splits$splits:

model\_perf <- function(data\_split, mod\_obj) {  
 vars <- rsample::form\_pred(mod\_obj$terms)  
 assess\_dat <- assessment(data\_split) %>%  
 select(!!!vars, Sale\_Price) %>%  
 mutate(  
 pred = predict(  
 mod\_obj,   
 newdata = assessment(data\_split)  
 ),  
 Sale\_Price = log10(Sale\_Price)  
 )  
 rmse <- assess\_dat %>%   
 rmse(truth = Sale\_Price, estimate = pred)  
 rsq <- assess\_dat %>%   
 rsq(truth = Sale\_Price, estimate = pred)  
 data.frame(rmse = rmse, rsq = rsq)  
}

### Resamling the linear model

The purrr package will be used to fit the model to each analysis set. There will be saved in a column called lm\_mod:

library(purrr)  
  
cv\_splits <- cv\_splits %>%   
 mutate(lm\_mod=map(splits, lm\_fit, formula=form))  
cv\_splits

## # 10-fold cross-validation using stratification   
## # A tibble: 10 x 3  
## splits id lm\_mod   
## \* <list> <chr> <list>   
## 1 <S3: rsplit> Fold01 <S3: lm>  
## 2 <S3: rsplit> Fold02 <S3: lm>  
## 3 <S3: rsplit> Fold03 <S3: lm>  
## 4 <S3: rsplit> Fold04 <S3: lm>  
## 5 <S3: rsplit> Fold05 <S3: lm>  
## 6 <S3: rsplit> Fold06 <S3: lm>  
## 7 <S3: rsplit> Fold07 <S3: lm>  
## 8 <S3: rsplit> Fold08 <S3: lm>  
## 9 <S3: rsplit> Fold09 <S3: lm>  
## 10 <S3: rsplit> Fold10 <S3: lm>

### Resampling the linear model (cont.)

Now, let’s compute the two performance measures:

# map2 can be used to move over two objects of equal length  
  
library(dplyr)  
lm\_res <- map2\_df(cv\_splits$splits, cv\_splits$lm\_mod, model\_perf) %>%   
 dplyr::rename(rmse\_simple=rmse, rsq\_simple=rsq)  
  
lm\_res %>% head()

## rmse\_simple rsq\_simple  
## 1 0.1611422 0.1053859  
## 2 0.1577929 0.2393299  
## 3 0.1638518 0.1437850  
## 4 0.1675580 0.1219703  
## 5 0.1531414 0.2037885  
## 6 0.1795991 0.1572545

# Merge in results:  
cv\_splits <- cv\_splits %>% bind\_cols(lm\_res)  
  
# Rename the columns and compute the resampling estimates:  
cv\_splits %>% select(rmse\_simple, rsq\_simple) %>% colMeans

## rmse\_simple rsq\_simple   
## 0.1612427 0.1845147

### What was the ruckus?

Previously, I mentioned that the performance metrics that were naively calculated from the training set could be optimistic. However, this approach estimates the RMSE to be 0.1614, and cross-validation produced an estimate of 0.1613. What was trhe big deal?

Linear regression is a high bias model. This means that it is fairly incapable at beign able to adpt the underlying model function (unless it is linear). For this reason, linear regression is unlikely to overfit to the training set and our two estimates are likely to be the same.

We’ll consider another model shortly that is low bias since it can, theoretically, easily adapt to a wide variety of true model functions.

However, as before, there is also variance to consider. Linear regression is very stable since it leverages all of the data points to estimate parameters. Other methods, such as tree-based models, are not and can drastically change if the training set data is slightly perturbed.

tl;dr: the earlier concern is real but linear regression is less likely to be affected.

### Diagnostics Again

Now let’s look at diagnostics using the predictions from the assessment sets.

get\_assessment <- function(splits, model)  
 augment(model, newdata=assessment(splits)) %>%   
 mutate(.resid=log10(Sale\_Price)-.fitted)  
  
holdout\_results <- map2\_df(cv\_splits$splits, cv\_splits$lm\_mod, get\_assessment)  
holdout\_results %>% dim()

## [1] 2199 84

ames\_train %>% dim()

## [1] 2199 81

### Hands-on partial residual plots

A partial residual plot is used to diagnose what variables should have been in the model.We can plot the hold-out residuals versus different variables to understand if they should have been in the model - If the residuals have no pattern in the data, they are likely to be irrelevant. - If a pattern is seen, it suggests that the variable should have been in the model.

Take 10 min and use ggplot to investigate other predictors using the holdout\_results data frame. geom\_smooth might come in handy.

## Tuning parameters and overfitting

### K-Nearest Neighbors Model

Now let’s consider a more flexible model that is low bas: K-nearest neighbors.The model stores the training set(including the outcome). When a new sample is predicted, training set points are found that are most similar to the new sample being predicted.

The predicted value for the new sample is some summary statistic of the neighbors, usually: - the mean for regression, or - the mode for classification

When is small, the model might be too responsive to underlying data. When is large, it begins to “oversmooth” the neighbors and performance suffers.

Ordinary, since we are computing a **distance**, we would want to center and scale the predictors. Our two predictors are already on the same scale so we can skip this step.

Consider the 2-nearest neighbor model. Would there be a difference in the estimated model performance between re-prediction and cross-validation?

caret has a knnreg function that can be used (the kknn package is another option). IT has a formula method and we’ll use this to illustrate the model:

library(caret)  
  
knn\_train\_mod <- knnreg(log10(Sale\_Price)~Longitude+Latitude,  
 data=ames\_train,  
 k=2)  
  
repredict <- data.frame(price=log10(ames\_train$Sale\_Price)) %>%   
 mutate(pred=predict(knn\_train\_mod, newdata=ames\_train %>% select(Longitude,Latitude)  
 )  
 )  
  
repredict %>% rsq(truth = "price", estimate = "pred") # <- the ruckus is here

## [1] 0.892872

# 0.892872

Thats pretty good, but are we tricking ourselves? One of those two neighbors is always itself. To resample, let’s create another function to fit this model and follow the same resampling process as before:

knn\_fit <- function(data\_split,...)  
 knnreg(..., data=analysis(data\_split))  
  
cv\_splits <- cv\_splits %>%   
 mutate(knn\_mod=map(splits, knn\_fit, formula=form, k=2))  
  
knn\_res <- map2\_df(cv\_splits$splits, cv\_splits$knn\_mod, model\_perf) %>%   
 rename(rmse\_knn=rmse, rsq\_knn=rsq)  
  
# merge in results  
cv\_splits <- cv\_splits %>% bind\_cols(knn\_res)  
  
colMeans(knn\_res)

## rmse\_knn rsq\_knn   
## 0.1012993 0.6869034

### Making formal comparisons

The model appears to be a drastic improvement over simple linear regression, but we are definitely getting highly optimistic results by re-predicting the training set.

We can try to make a more formal assesment of the two current models. Both models used the same resamples, so we have 10 estimates of performance that are matched. Does the matching mean anything?

Most likely **YES**. It is very common to see that there is a resample effect. Similar to repeated measures designs, we can expect a relationship between models and resamples. For example, some resamples will have the worst performance over different models and so on.

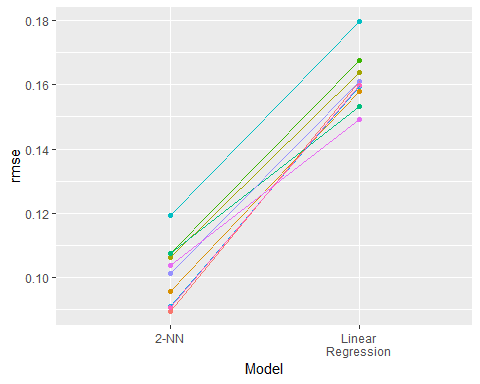
In other words, there is usually a within-resample correlation. For the two models, the estimated correlation in RMSE values is 0.85.

### The resample effect

rs\_comp <- data.frame(  
 rmse = c(cv\_splits$rmse\_simple, cv\_splits$rmse\_knn),  
 Model = rep(c("Linear\nRegression", "2-NN"), each = nrow(cv\_splits)),  
 Resample = cv\_splits$id  
)  
  
rs\_comp

## rmse Model Resample  
## 1 0.16114224 Linear\nRegression Fold01  
## 2 0.15779294 Linear\nRegression Fold02  
## 3 0.16385176 Linear\nRegression Fold03  
## 4 0.16755796 Linear\nRegression Fold04  
## 5 0.15314139 Linear\nRegression Fold05  
## 6 0.17959908 Linear\nRegression Fold06  
## 7 0.15932070 Linear\nRegression Fold07  
## 8 0.16096959 Linear\nRegression Fold08  
## 9 0.14930855 Linear\nRegression Fold09  
## 10 0.15974240 Linear\nRegression Fold10  
## 11 0.08954527 2-NN Fold01  
## 12 0.09583811 2-NN Fold02  
## 13 0.10634629 2-NN Fold03  
## 14 0.10764468 2-NN Fold04  
## 15 0.10760225 2-NN Fold05  
## 16 0.11939967 2-NN Fold06  
## 17 0.09097765 2-NN Fold07  
## 18 0.10116313 2-NN Fold08  
## 19 0.10379151 2-NN Fold09  
## 20 0.09068420 2-NN Fold10

ggplot(rs\_comp, aes(x = Model, y = rmse, group = Resample, col = Resample)) +   
 geom\_point() +   
 geom\_line() +   
 theme(legend.position = "none")



### Model comparison accounting for resampling

With only two models, a paired t-test can be used to estimate the difference in RMSe between the models:

t.test(cv\_splits$rmse\_simple, cv\_splits$rmse\_knn, paired=TRUE)

##   
## Paired t-test  
##   
## data: cv\_splits$rmse\_simple and cv\_splits$rmse\_knn  
## t = 21.265, df = 9, p-value = 5.282e-09  
## alternative hypothesis: true difference in means is not equal to 0  
## 95 percent confidence interval:  
## 0.05356677 0.06632000  
## sample estimates:  
## mean of the differences   
## 0.05994338

Hothorn et al (2012) is the [original paper](https://scholar.google.com/scholar?hl=en&q=analysis+of+benchmark+experiments&btnG=&as_sdt=1%2C7&as_sdtp=) on comparing models using resampling.

We’ll do more extensive analyses with tidyposterior soon.

### Overfitting

Overfitting occurs when a model inappropriately pick up on trends in the training set that do not generalize to new samples. When this occurs, assesment of the model based on the training set can show good performance that does not reproduce in future samples.

Some models have specific “knobs” to control over-fitting - neighborhood size in nearest neighbor models is an example - the number of splits in a tree model

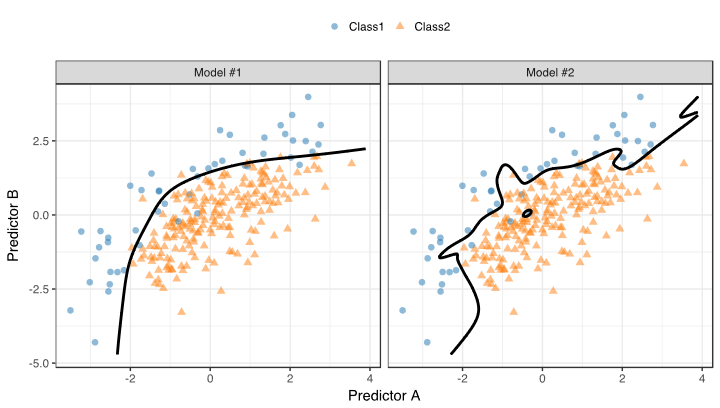
often poor choices for these parameters can result in overfitting

For example, the next slide shows a data set with two predictors. We want to be able to produce a line (i.e., decision boundary) that differentiates two classes of data.

### Two class example,

On the next slide, two classification boundaries are shown for a different model type not yet discussed. The differece in the two panes is solely due to different choices in tuning parameters. One overfits the training data.

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



We usually don’t have two-dimensional data so a quantitative method for under measureing overfitting is needed. **Resampling** fits that description. A simple method for tuning a model is used to grid search:

├── Create a set of candidate tuning parameter values └── For each resample ??? ├── Split the data into analysis and assessment sets ??? ├── [preprocess data] ??? ├── For each tuning parameter value ??? ??? ├── Fit the model using the analysis set ??? ??? └── Compute the performance on the assessment set and save ├── For each tuning parameter value, average the performance over resamples ├── Determine the best tuning parameter value └── Create the final model with the optimal parameter(s) on the training set

**Random search** is a similar technique where the candidate set of parameter values are simulated at random across a wide range. Also, an example of nested resampling can be found [here](http://appliedpredictivemodeling.com/blog/2017/9/2/njdc83d01pzysvvlgik02t5qnaljnd).

### Grid search computations

The bad news is that all of the models (except the final model) are discared. However, as the good news, all of the models (except the final model) can be run in parallel. Let’s look at the Ames K-NN model and evaluate using the same 10-fold cross-validation as before.

We’ll start coding this algorithm from the inside out.

These steps are: ├── Fit the model using the analysis set └── Compute the performance on the assessment set and save

split will be the one of elements of cv\_splits$splits.

knn\_rmse <- function(k, split) {  
 mod <- knnreg(log10(Sale\_Price) ~ Longitude + Latitude,   
 data = analysis(split),   
 k = k)  
 # Extract the names of the predictors  
 preds <- form\_pred(mod$terms)  
 data.frame(Sale\_Price = log10(assessment(split)$Sale\_Price)) %>%  
 mutate(pred = predict(mod, assessment(split) %>% select(!!!preds))) %>%  
 rmse(Sale\_Price, pred)  
}

### Fit the model across values of K

??? ├── For each tuning parameter value ??? ??? └── Run knn\_rmse

knn\_grid <- function(split) {  
 # Create grid  
 tibble(k = 1:20) %>%  
 # Execute grid for this resample  
 mutate(  
 rmse = map\_dbl(k, knn\_rmse, split = split),  
 # Attach the resample indicators using `lables`  
 id = labels(split)[[1]]  
 )  
}

The return values here is a tibble with columns for *k*, the *RMSE*, and the fold ID (e.g., Fold01).

### Top-level iteration over resamples

└── For each resample ??? └── Run knn\_grid

Here, resamp is the resample object cv\_splits

iter\_over\_resamples <-   
 function(resamp)   
 map\_df(resamp$splits, knn\_grid)

### Running the code

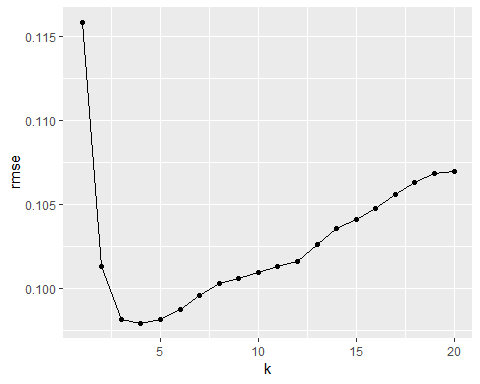
library(rsample)  
knn\_tune\_res <- iter\_over\_resamples(cv\_splits)  
knn\_tune\_res %>% head(15)

## # A tibble: 15 x 3  
## k rmse id   
## <int> <dbl> <chr>   
## 1 1 0.105 Fold01  
## 2 2 0.0895 Fold01  
## 3 3 0.0878 Fold01  
## 4 4 0.0899 Fold01  
## 5 5 0.0912 Fold01  
## 6 6 0.0931 Fold01  
## 7 7 0.0957 Fold01  
## 8 8 0.0963 Fold01  
## 9 9 0.0963 Fold01  
## 10 10 0.0979 Fold01  
## 11 11 0.0981 Fold01  
## 12 12 0.0971 Fold01  
## 13 13 0.0984 Fold01  
## 14 14 0.1000 Fold01  
## 15 15 0.101 Fold01

### The performance profile

To summarize the results for each value of :

library(tidyverse)  
  
rmse\_by\_k <- knn\_tune\_res %>%   
 group\_by(k) %>%   
 summarize(rmse=mean(rmse))  
  
ggplot(rmse\_by\_k, aes(x=k, y=rmse))+  
 geom\_point()+geom\_line()

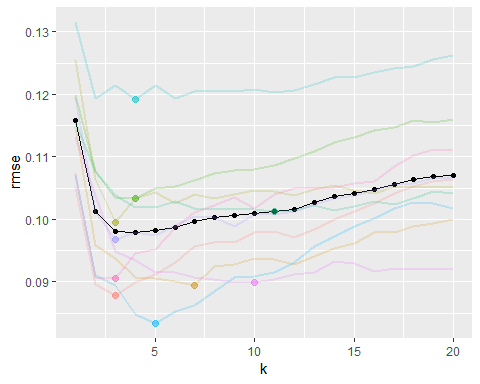


Although it is numerically optimal, we are not required to use a value of 4 neighbors for the final model.

### Resampling variation

How stable is this? We can also plot the individual curves and their minimums.

best\_k <- knn\_tune\_res %>%   
 group\_by(id) %>%   
 summarize(k=k[which.min(rmse)],  
 rmse=rmse[which.min(rmse)])  
  
ggplot(rmse\_by\_k, aes(x=k, y=rmse))+  
 geom\_point()+  
 geom\_line()+  
 geom\_line(data=knn\_tune\_res,  
 aes(group=id, col=id),  
 alpha=0.2,lwd=1)+  
 geom\_point(data=best\_k,  
 aes(col=id),  
 alpha=0.5, cex=2)+  
 theme(legend.position = "none")



### The next Steps

At this point, we would decide on a good value for and then fit the model used going foward:

final\_knn <- knnreg(log10(Sale\_Price)~Longitude+Latitude,  
 data=ames\_train,  
 k=4)

To reiterate: the previous 200 models created during the grid search are not used once K is set. Later, we will look at the high-level API in caret that stremalines almost all of this process for different models.

## Part3: Feature engineering and preprocessing

### Preprocessing and feature engineering

This part mostly concerns what we can do to do our variables to make the models more effective., which is related to the predictors. Operations that we might use are: - transformations of individual predictors or groups of variables, - alternate encodings of a variable, - elimination of predictors(unsupervised) etc.

In statistics, this is generally called preprocessing the data. As usual, the computer science side of modeling has a much flashier name: **feature engineering**.

### Reasons for modifying the data

Some models(KNN, SVMs, PLS, neural networks) require that the predictors variables have the same unit: **centering**, **scaling** the predictors can be used for this purpose.

Other models are very sensitive to correlations between the predictors and **filters** or **PCA signal extraction** can improve the model.

As we’ll see an example, changing the scale of predictors using a **transformation** can lead to a big improvement. In other cases, the data can be encoded **encoded** in a way that maximizes its effect on the model. Representing the data as the day of or the week can be effective for modeling public transportation data.

Many models cannot cope with missing data so **imputation** strategies might be necessary. Development of new *features* that represent something important to the outcome (e,g,m compute distances to public transportation, university buildings, public schools etc.).

### A bivariate example

The plot on the right side shows two predictors from a real *test* set where the object is to predict two classes. The predictors are strongly correlated and each has a right skewed distribution. There appeas to be some class separation but only in the bivariate plot: the individual predictors show poor discrimination of the classes.

Some models might be sensitive to highly correalted and/or skewed predictors. Is there something that we can do to make the predictors easier for the model to use?

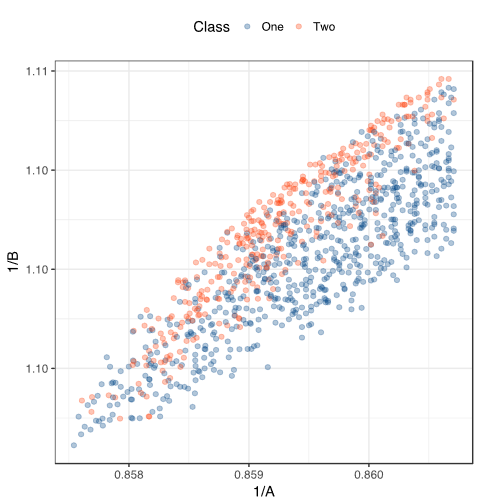
knitr::include\_graphics("C:/Users/kojikm.mizumura/Desktop/Data Science/UseR 2018/Applied ML/bivariate-plot-natural-1.PNG")



We might start by estimating transformations of the predictors to resolve the skewness. The **Box-Cox transformation** is a family of transformations originally designed for the outcomes of models. We can use it here for the predictors.

It uses the data to estimate a wide variety of transformations including the inverse, log, sqrt and polynomial functions. Using each factor in isolation, both predictors were determined to need **inverse transformation** (approximately). The figure on th right shows the data after these transformations have been applied. A logistic regression model shows a substantial improvement in classifying using the altered data.

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



### Resampling and preprocessing

It is important to realize that almost all preprocessing steps that involve estimation sohuld be bundled inside of the resampling process so that the performance estimates are not biased.

* **Bad**: preprocess the data, resample the model
* **Good**: resample the processing and modeling

Also, - avoid information *leakage* by having all operations in the modeling process occur only on the training set. - Do not reestimate anything on the test set. For example, to center new data, the training set mean is used.

## Preprocessing categorical predictors

### Dummy variables

One common procedure for modeling is to create numeric representations of categorical data. This is usually done via *dummy variables*: a set of binary 0/1 variables for different levels of an R factor.

For example, the Ames housing data contains a predictor called Alley with levels: Gravel; No\_Alley\_Access; Paved. Most dummy variable procedure would make two numeric variables from this predictor that are zero for a level and one otherwise.

Data Dummy variables No\_Alley\_Access Paved Gravel 0 0 No\_Alley\_Access 1 0  
Paved 0 1

If there are levels of the factor, only dummy variables are created since the last can be infered from the others. There are different contrast schemes for creating the new variables.

For ordered factors, *polynnomial* contrasts are used. See this [blog post](http://appliedpredictivemodeling.com/blog/2013/10/23/the-basics-of-encoding-categorical-data-for-predictive-models).

How do you create them in R?

The formula method does this for you. Otherwise, the traditional method is to use the model.matrix function to create a matrix. However, there are some caveats to this tha can make things difficult. We’ll show another method for making them shortly.

* Almost always at least. Tree- and rule-based model functions do not. Example are randomforest::randomForest, ranger::ranger, rpart::rpart, C50::C5.0, Cubist::cubist, klaR::NaiveBayes.

### Infrequent levels in categorical factors

One issue is what happens when there are very few values of a level? Consider the Ames training set and the Neighborhood variable. If these data are resampled, what would happen to Landmark and similar locations when dummy variables are created?

A variance predictor that has only a single value (zero) would be the result. For many models (e.g., linear/logistic regression, etc.) would find this numerically problematic and issue a warining and NA values for that coefficient. Trees and similar models would not notice.

There are two main approaches to dealing with this:

1. Run a filter on the training set predictors prior to running the model and remove the zero-variance predictors
2. Recode the factor so that infrequently occuring predictors (and possibly new values) are pooled into an “other” category.

However, model.matrix and the formula method are incapable of doing either of these.

### Recipes

**Recipes** are an alternative method for creating the data frame of predictors for a model. They allow for a sequence of steps that define how data should be handled. Recall that previous part where we used the formula log10(Sale\_Price) ~ Longitude + Latitude? These steps are:

1. Assign Sale\_Price to be the outcome
2. Assign Longitude and Latitude are predictors
3. Log transform the outcome

To start using a recipe, the these steps can be done using

library(recipes)  
  
mod\_rec <- recipe(Sale\_Price ~ Longitude + Latitude, data=ames\_train) %>%   
 step\_log(Sale\_Price, base=10)

This creates the recipe for data preprocessing (but does not execute it yet)/

### Recipes and categorical predictors

To deal with the dummy variable issue, we can expand the recipe with more steps:

mod\_rec <- recipe(Sale\_Price ~ Longitude + Latitude+Neighborhood, data=ames\_train) %>%   
 step\_log(Sale\_Price, base=10) %>%   
   
 # Lump factor levels that occur in <=5% of data as "others"  
 step\_other(Neighborhood, threshold=0.05) %>%   
   
 # create dummy variables for \_any\_ factor variables  
 step\_dummy(all\_nominal())  
  
mod\_rec

## Data Recipe  
##   
## Inputs:  
##   
## role #variables  
## outcome 1  
## predictor 3  
##   
## Operations:  
##   
## Log transformation on Sale\_Price  
## Collapsing factor levels for Neighborhood  
## Dummy variables from all\_nominal()

### Preparing the recipe

Now that we have a preprocessing *specification*, let’s run it on the training set to *prepare* the recipe:

mod\_rec\_trained <- prep(mod\_rec, training=ames\_train, retain=T, verbose=T)

## oper 1 step log [training]   
## oper 2 step other [training]   
## oper 3 step dummy [training]

Here, the training is to determine which factors to pool and to enumerate the factor levels of the Neighborhood variable, retain keeps the processed varion of the training set around so we don’t have to recompute it.

mod\_rec\_trained

## Data Recipe  
##   
## Inputs:  
##   
## role #variables  
## outcome 1  
## predictor 3  
##   
## Training data contained 2199 data points and no missing data.  
##   
## Operations:  
##   
## Log transformation on Sale\_Price [trained]  
## Collapsing factor levels for Neighborhood [trained]  
## Dummy variables from Neighborhood [trained]

### Getting the values

Once the recipe is prepared, it can be applied to any data set using bake:

ames\_test\_dummies <- bake(mod\_rec\_trained, newdata=ames\_test)  
names(ames\_test\_dummies)

## [1] "Sale\_Price" "Longitude"   
## [3] "Latitude" "Neighborhood\_College\_Creek"   
## [5] "Neighborhood\_Old\_Town" "Neighborhood\_Edwards"   
## [7] "Neighborhood\_Somerset" "Neighborhood\_Northridge\_Heights"  
## [9] "Neighborhood\_Gilbert" "Neighborhood\_Sawyer"   
## [11] "Neighborhood\_other"

If retain = TRUE the training set does not need to be “rebaked”. The juice function can return the processed verion of the training data. Selection can be used with bake and the default is everything

### Hands-on: zero-variance filter

Instead of using step\_other, take 10 minutes and research how to eliminate any zero-variance predictors using the recipe [reference site](https://topepo.github.io/recipes/reference/index.html).

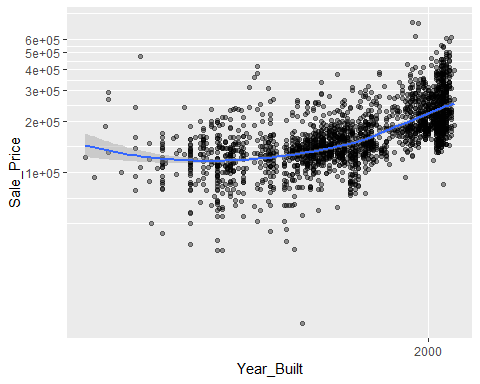
Re-run the recipe with this step. What were the results? Do you prefer either of these approaches to the other?

## Interaction effects

### Interactions

An interaction between two predictors indicates that the relationship between the predictors and the outcome cannnot be described using only one of the variables.For example, let’s look at the relationship between the price of a house and the year in which it was built. The relationship appears to be slightly nonlinear, possible quadratic.

library(ggplot2)  
  
price\_breaks <- (1:6)\*(10^5)  
  
ggplot(ames\_train,   
 aes(x = Year\_Built, y = Sale\_Price))+  
 geom\_point(alpha = 0.4) +  
 scale\_x\_log10() +   
 scale\_y\_continuous(breaks = price\_breaks,  
 trans = "log10")+  
 geom\_smooth(method = "loess")



### Interactions

However, what if we separate this trend based on whether the property has air conditioning (93.4% of the training set) or not (6.6%).

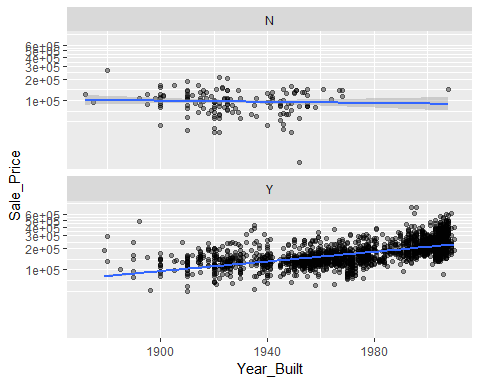
library(MASS)

## Warning: package 'MASS' was built under R version 3.5.1

##   
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':  
##   
## select

# To get rubust linear regression model  
  
ggplot(ames\_train,  
 aes(x=Year\_Built,  
 y=Sale\_Price))+  
 geom\_point(alpha=0.4)+  
 scale\_y\_continuous(breaks=price\_breaks,  
 trans="log10")+  
 facet\_wrap(~Central\_Air, nrow=2)+  
 geom\_smooth(method="rlm")



### Interactions

It appears as though the relationship between the year built and the sale price is somewhat different for the two groups.

* When there is no AC, the trend as perhaps flat or slightly decreasing
* With AC, there is a linear trend or is perhaps slightly quadratic with some outliers at the low end.

mod1 <- lm(log10(Sale\_Price)~Year\_Built+Central\_Air, data=ames\_train)  
mod2 <- lm(log10(Sale\_Price)~Year\_Built+Central\_Air+Year\_Built:Central\_Air, data=ames\_train)  
  
anova(mod1, mod2)

## Analysis of Variance Table  
##   
## Model 1: log10(Sale\_Price) ~ Year\_Built + Central\_Air  
## Model 2: log10(Sale\_Price) ~ Year\_Built + Central\_Air + Year\_Built:Central\_Air  
## Res.Df RSS Df Sum of Sq F Pr(>F)   
## 1 2196 41.349   
## 2 2195 40.239 1 1.1106 60.583 1.078e-14 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

### Interactions in Recipes

We first create the dummy variables for the qualitative predictor (Central\_Air) then use a formula to create the interaction using the **:** operator in addition step:

recipe(Sale\_Price ~ Year\_Built+Central\_Air, data=ames\_train) %>%   
 step\_log(Sale\_Price) %>%   
 step\_dummy(Central\_Air) %>%   
 step\_interact(~starts\_with("Central\_Air"):Year\_Built) %>%   
 prep(training=ames\_train, retain=T) %>%   
 juice %>%   
 # select a few rows with different values  
 slice(153:157)

## # A tibble: 5 x 4  
## Year\_Built Sale\_Price Central\_Air\_Y Central\_Air\_Y\_x\_Year\_Built  
## <int> <dbl> <dbl> <dbl>  
## 1 1912 12.0 1 1912  
## 2 1930 10.9 0 0  
## 3 1900 11.8 1 1900  
## 4 1959 12.1 1 1959  
## 5 1917 11.6 0 0

## Adding Recipes to our rsample worflows

### recipes and rsample

Let’s got back to the Ames housing data and work on building models with recipes using code similar to the previous set of notes. Previously:

library(AmesHousing)  
ames <- make\_ames()  
  
library(rsample)  
set.seed(4595)  
  
data\_split <- initial\_split(ames,strata="Sale\_Price")  
ames\_train <- training(data\_split)  
  
set.seed(2453)  
cv\_splits <- vfold\_cv(ames\_train, v=10, strata="Sale\_Price")

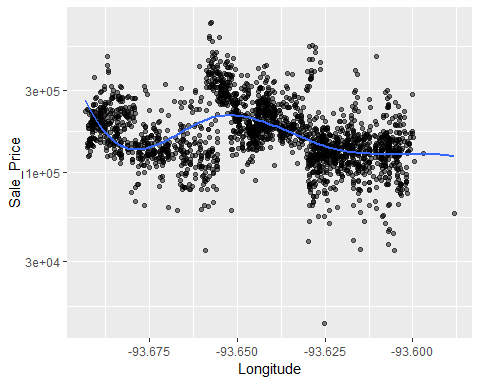
### Linear models again

Let’s add a few extra predictors and some preprocessing - Two numeric predictors are very skewed and could use a transformation(Lot\_Area and Gr\_Liv\_Area). - We’ll add neighborhood in as well and a few other house features. - The K-NN model suggests that the coordinates can be helpful but probably require a nonlinear representation. We can add thewse using **B-splines** with 5 degrees of freedom. To evaluate this, we will create two versions of the recipe to evaluate this hypothesis.

lin\_coords <- recipe(Sale\_Price ~ Bldg\_Type + Neighborhood+Year\_Built+  
 Gr\_Liv\_Area + Full\_Bath + Year\_Sold + Lot\_Area+  
 Central\_Air + Longitude + Latitude,  
 data=ames\_train) %>%   
 step\_log(Sale\_Price, base=10) %>%   
 step\_YeoJohnson(Lot\_Area, Gr\_Liv\_Area) %>%   
 step\_other(Neighborhood, threshold = 0.05) %>%   
 step\_dummy(all\_nominal()) %>%   
 step\_interact(~starts\_with("Central\_Air"):Year\_Built)  
  
coords <- lin\_coords %>%   
 step\_bs(Longitude, Latitude, options=list(df=5))

### Longitude

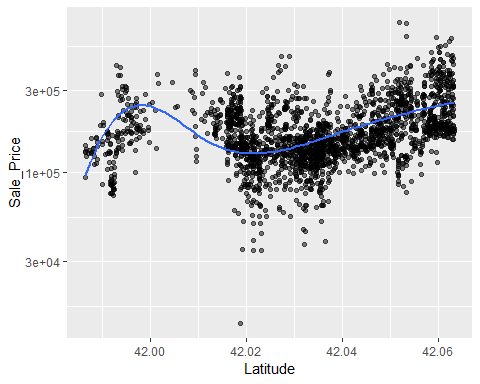
library(ggplot2)  
  
ggplot(ames\_train,  
 aes(x=Longitude, y=Sale\_Price))+  
 geom\_point(alpha=.5)+  
 geom\_smooth(  
 method="lm",  
 formula=y~splines::bs(x,5),  
 se=FALSE)+  
 scale\_y\_log10()



Splines add non-linear versions of the predictor to a linear model to create smooth and flexible relationships between the predictor and outcome. This “basis expansion” technique will be seen against in the regression section of the workshop.

### Latitude

ggplot(ames\_train,  
 aes(x=Latitude, y=Sale\_Price))+  
 geom\_point(alpha=.5)+  
 geom\_smooth(  
 method="lm",  
 formula=y~splines::bs(x,5 ),  
 se=FALSE  
 )+  
 scale\_y\_log10()



### Preparing the Recipes

Our first step is the run prep() on the coords recipe but using each of the analysis sets. rsample() has a function that is a wrapper around prep() that can be used to map over the split objects.

rsample::prepper

## function (split\_obj, recipe, ...)   
## {  
## prep(recipe, training = analysis(split\_obj, recipe = FALSE),   
## ...)  
## }  
## <bytecode: 0x000000001d001790>  
## <environment: namespace:rsample>

library(purrr)  
  
cv\_splits <- cv\_splits %>%   
 mutate(coords = map(splits, prepper, recipe = coords, retain = TRUE))

### Fitting the models

We can use code that is very similar to the previous section. This corde will use the recipe object to get the data. Since each analysis set is used to train the recipe, our previous use of retain=TRUE means that the processed version of the data is withing the recipe.

This can be returned via the juice function.

lm\_fit\_rec <- function(rec\_obj, ...)   
 lm(..., data = juice(rec\_obj))  
cv\_splits <- cv\_splits %>%   
 mutate(fits = map(coords, lm\_fit\_rec, Sale\_Price ~ .))  
  
cv\_splits

## # 10-fold cross-validation using stratification   
## # A tibble: 10 x 4  
## splits id coords fits   
## \* <list> <chr> <list> <list>   
## 1 <S3: rsplit> Fold01 <S3: recipe> <S3: lm>  
## 2 <S3: rsplit> Fold02 <S3: recipe> <S3: lm>  
## 3 <S3: rsplit> Fold03 <S3: recipe> <S3: lm>  
## 4 <S3: rsplit> Fold04 <S3: recipe> <S3: lm>  
## 5 <S3: rsplit> Fold05 <S3: recipe> <S3: lm>  
## 6 <S3: rsplit> Fold06 <S3: recipe> <S3: lm>  
## 7 <S3: rsplit> Fold07 <S3: recipe> <S3: lm>  
## 8 <S3: rsplit> Fold08 <S3: recipe> <S3: lm>  
## 9 <S3: rsplit> Fold09 <S3: recipe> <S3: lm>  
## 10 <S3: rsplit> Fold10 <S3: recipe> <S3: lm>

glance(cv\_splits$fits[[1]])

## # A tibble: 1 x 11  
## r.squared adj.r.squared sigma statistic p.value df logLik AIC  
## \* <dbl> <dbl> <dbl> <dbl> <dbl> <int> <dbl> <dbl>  
## 1 0.805 0.802 0.0794 276. 0 30 2218. -4374.  
## # ... with 3 more variables: BIC <dbl>, deviance <dbl>, df.residual <int>

### Predicting the assessment set

This is alittle more complex. We need three elements contained in our tibble: - the split object (to get the assessment data) - the recipe object (to process the data) - the linear model(for predictions)

The function is not too bad:

assess\_predictions <- function(split\_obj, rec\_obj, mod\_obj){  
 raw\_data <- rsample::assessment(split\_obj)  
 proc\_x <- bake(rec\_obj, newdata=raw\_data, all\_predictors())  
 # now save all of the columns and add predictions  
 bake(rec\_obj, newdata=raw\_data, everything()) %>%   
 mutate(  
 .fitted=predict(mod\_obj, newdata=proc\_x),  
 .resid=Sale\_Price - .fitted, # Sale\_Price is already logged by the recipe  
 # Save the original row number of the data  
 .row=as.integer(split\_obj, data="assessment")  
 )  
}

Since we have three inputs, we will use purrr’s pmap function to walk along all three columns in the tibble.

cv\_splits <- cv\_splits %>%  
 mutate(  
 pred =   
 pmap(  
 lst(split\_obj = cv\_splits$splits, rec\_obj = cv\_splits$coords, mod\_obj = cv\_splits$fits),  
 assess\_predictions   
 )  
 )

## Warning in bs(x = c(42.060872, 42.062303, 42.060977, 42.061193,  
## 42.035333, : some 'x' values beyond boundary knots may cause ill-  
## conditioned bases  
  
## Warning in bs(x = c(42.060872, 42.062303, 42.060977, 42.061193,  
## 42.035333, : some 'x' values beyond boundary knots may cause ill-  
## conditioned bases

## Warning in bs(x = c(-93.6235954, -93.636372, -93.628806, -93.625924,  
## -93.649976, : some 'x' values beyond boundary knots may cause ill-  
## conditioned bases  
  
## Warning in bs(x = c(-93.6235954, -93.636372, -93.628806, -93.625924,  
## -93.649976, : some 'x' values beyond boundary knots may cause ill-  
## conditioned bases

## Warning in bs(x = c(42.060779, 42.059193, 42.059169, 42.060936,  
## 42.046145, : some 'x' values beyond boundary knots may cause ill-  
## conditioned bases  
  
## Warning in bs(x = c(42.060779, 42.059193, 42.059169, 42.060936,  
## 42.046145, : some 'x' values beyond boundary knots may cause ill-  
## conditioned bases

## Warning in bs(x = c(-93.619754, -93.628804, -93.622971, -93.625848,  
## -93.65332, : some 'x' values beyond boundary knots may cause ill-  
## conditioned bases  
  
## Warning in bs(x = c(-93.619754, -93.628804, -93.622971, -93.625848,  
## -93.65332, : some 'x' values beyond boundary knots may cause ill-  
## conditioned bases

We do get soem warnings that the assessment data are outside the range of the analysis set values:

yardstick::metric will compute a small set of summary statistics for metrics model based on the type of outcome (e.g., regression, classification, etc.)

library(yardstick)  
# compute the summary stats  
purrr::map\_df(cv\_splits$pred, metrics, truth=Sale\_Price, estimate=.fitted) %>%   
 colMeans

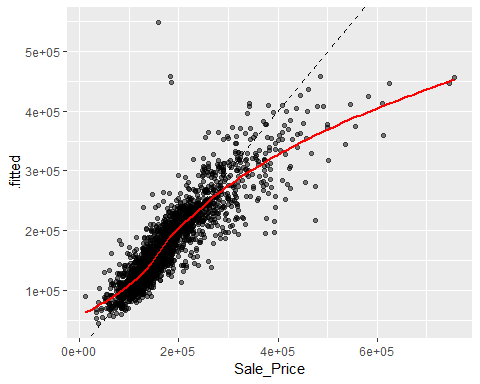
## rmse rsq   
## 0.07976634 0.80050138

These results are better than our K-NN model but “the only way to be comfortable with your results is to never look at them”

### Graphical checks for fit

assess\_pred <- bind\_rows(cv\_splits$pred) %>%   
 mutate(Sale\_Price=10^Sale\_Price,  
 .fitted=10^.fitted)  
  
ggplot(assess\_pred,  
 aes(Sale\_Price, .fitted))+  
 geom\_abline(lty=2)+  
 geom\_point(alpha=.5)+  
 geom\_smooth(se=FALSE, col="red")

## `geom\_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'



The current model is - drastically *over*-predicting the price of three houses - significantly *under*-predicting the price of a number of expensive houses

What would we do next?

1. Try to understand the two big residuals. Are these aberrant houses or does this have something to do with our model? Maybe those extrapolation warning?
2. Find more predictors that differentiate the more expensive houses with the others
3. Try a different model

### About these five houses

From Dmytro Perepolkin via [GitHub/topepo/AmesHousing](https://github.com/topepo/AmesHousing/issues/2#issuecomment-351005269): I did a little bit of research on Kaggle regarding those ve houses (three “partial sale” houses in Edwardsand two upscale in Northridge):

In fact all three of Edwards properties were sold within three months by the same company. The lots are located next to each other in Cochrane Pkwy Ames, IA. I guess the story was that developer was either in trouble or wanted to boost sales, so they were signing sales contracts on half- nished houses in a new development area at a deep discount. Those few houses are likely to have been built rst out of the whole residential block that followed. Two of the upscale houses in Northridge (sold at 745 and 755 thousand, respectively) are located next to each other. Houses are, of course, eyecandies (at least to my taste). They are outliers with regards to size in their own neighborhoods!

Sometimes it really pays off to be a [forensic statistician](https://projecteuclid.org/euclid.aoas/1267453942).

# Regression modeling

* Example data
* Regularized linear models
* Multiavariate Adaptive Regression Splines (MARS)
* Ensembles of MARS models
* Model comparison via Bayesian analysis

## Example data

### Expanded car MPG data

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 2015-2018.

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 additional information about the cars (e.g., intake valves per cycle, asipration 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.

cars <- load("C:/Users/kojikm.mizumura/Desktop/Data Science/UseR 2018/Applied ML/car\_data.RData")  
  
library(dplyr)  
  
# train data  
car\_train <- car\_data %>%  
 filter(model\_year < 2018)  
car\_test <- car\_data %>%  
 filter(model\_year == 2018)

### Some acronyms and codes

* Drive codes: F=2-Wheel Drive, Front R=2-Wheel Drive, Rear 4=4-Wheel Drive A=All Wheel Drive P=Part-time 4-Wheel Drive
* Fuel; DU=Diesel, ultra low sulfur (15 ppm, maximum) G=Gasoline (Regular Unleaded Recommended) GM=Gasoline (Mid Grade Unleaded Recommended) GP=Gasoline (Premium Unleaded Recommended) GPR=Gasoline (Premium Unleaded Required)
* Transmission: A=Auto(A10) SA=Auto(S10) A=Auto(A4) SA=Auto(S4) A=Auto(A5) SA=Auto(S5) A=Auto(A6) SA=Auto(S6) A=Auto(A7) SA=Auto(S7) A=Auto(A8) SA=Auto(S8) A=Auto(A9) SA=Auto(S9) AM=Auto(AM5) SCV=Auto(AV-S10) AM=Auto(AM6) SCV=Auto(AV-S6) AM=Auto(AM7) SCV=Auto(AV-S7) AM=Auto(AM8) SCV=Auto(AV-S8) AMS=Auto(AM-S6) AMS=Auto(AM-S7) AMS=Auto(AM-S8) AMS=Auto(AM-S9) CVT=Auto(AV) gM=Manual(M5) M=Manual(M6) M=Manual(M7)

### Hands-on: explore the car data

As before, let’s take 10 minutes to get familar with the training set using numerical summaries and plots. geom\_smoooth is very helpful here to discover the nature of relationships between the outcome (mpg) abd the potential predictors.

## Linear models

### Linear regression analysis

We’ll start by fitting linear regression models to these data. As a reminder, the “linear” part means that the model is linear in the *parameters*; we can add nonlinear terms to the model (e.g., x^2 or log(x)) without causing issues.

We could start by using lm and the formula method using what we’ve learnt so far.

lm <- lm(mpg ~ . -carline + poly(eng\_displ, 2), data = car\_train)  
summary(lm)

##   
## Call:  
## lm(formula = mpg ~ . - carline + poly(eng\_displ, 2), data = car\_train)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -16.5996 -1.4546 -0.0419 1.2753 28.4056   
##   
## Coefficients: (1 not defined because of singularities)  
## Estimate Std. Error  
## (Intercept) -184.76814 145.26517  
## model\_year 0.12397 0.07212  
## divisionalfa\_romeo -4.06597 1.53035  
## divisionaston\_martin\_lagonda\_ltd -9.67122 1.20866  
## divisionaudi -2.94001 0.69093  
## divisionbentley -6.00950 1.11593  
## divisionbmw -1.88463 0.64749  
## divisionbugatti -9.74320 3.36521  
## divisionbuick -6.55124 0.79228  
## divisioncadillac -4.98706 0.73643  
## divisionchevrolet -4.64447 0.68152  
## divisionchrysler -5.16407 1.04081  
## divisiondodge -5.33181 0.75920  
## divisionferrari\_north\_america,\_inc. -15.64673 1.07492  
## divisionfiat -4.81566 0.90173  
## divisionford -4.44428 0.67692  
## divisiongenesis -4.23496 1.31969  
## divisiongmc -4.74504 0.74730  
## divisionhonda -3.61982 0.71194  
## divisionhyundai\_motor\_company -5.72097 0.68761  
## divisioninfiniti -1.64045 0.70879  
## divisionjaguar -3.87951 0.82620  
## divisionjeep -5.85874 0.74095  
## divisionkia\_motors\_corporation -5.99365 0.69803  
## divisionlamborghini -5.65481 1.25397  
## divisionland\_rover -4.02350 0.95147  
## divisionlexus -2.71416 0.73440  
## divisionlincoln -4.60794 0.76079  
## divisionlotus\_cars\_ltd -5.70854 2.33382  
## divisionmaserati -8.07614 0.99862  
## divisionmazda -0.44896 0.72229  
## divisionmclaren -4.33101 2.30889  
## divisionmclaren\_automotive\_limted -4.78028 3.16994  
## divisionmercedes-benz -1.32635 0.74935  
## divisionmini -3.13743 0.72490  
## divisionmitsubishi\_motors\_corporation -5.41104 0.80221  
## divisionmobility\_ventures\_llc -4.75741 3.20816  
## divisionnissan -4.81842 0.69021  
## divisionpagani\_automobili\_s.p.a. -12.61229 3.28994  
## divisionporsche -4.70810 0.72710  
## divisionram -5.99619 1.00161  
## divisionrolls-royce\_motor\_cars\_limited -6.59392 1.42894  
## divisionroush\_industries,\_inc. -4.34244 1.29094  
## divisionscion -5.39529 1.02082  
## divisionsubaru -4.12297 0.79525  
## divisiontoyota -2.52628 0.69394  
## divisionvolkswagen -2.91235 0.69957  
## divisionvolvo\_cars\_of\_north\_america,\_llc -3.55351 0.89295  
## eng\_displ -4.90885 0.19798  
## cylinders 0.60687 0.14211  
## aspirationSupercharged -2.41968 0.46759  
## aspirationTurbocharged -3.47254 0.21222  
## aspirationTurbocharged+Supercharged -4.08992 1.13763  
## transAM 0.85266 0.46740  
## transAMS -2.79157 0.46527  
## transCVT 9.46500 0.62270  
## transM -6.81218 0.45843  
## transSA -0.21143 0.28064  
## transSCV 2.89371 0.43188  
## gears 0.51314 0.08849  
## lockup\_torque\_converter -6.63098 0.36846  
## drive\_sysA 0.65645 0.30555  
## drive\_sysF 3.79738 0.37381  
## drive\_sysP -0.50388 0.49480  
## drive\_sysR 1.44112 0.31965  
## fuel\_typeG -8.32790 0.46823  
## fuel\_typeGM -8.48769 0.84189  
## fuel\_typeGP -10.12525 0.45674  
## fuel\_typeGPR -9.87975 0.49927  
## intake\_valves\_per\_cyl -5.51848 1.08047  
## exhaust\_valves\_per\_cyl 2.57801 1.00628  
## car\_classlarge\_cars -0.61073 0.30699  
## car\_classmidsize\_cars 0.30697 0.23820  
## car\_classmidsize\_station\_wagons -1.06613 0.72769  
## car\_classminicompact\_cars 0.79189 0.45611  
## car\_classsmall\_pick-up\_trucks\_2wd -4.29810 0.58324  
## car\_classsmall\_pick-up\_trucks\_4wd -4.10765 0.67070  
## car\_classsmall\_station\_wagons -1.24419 0.43120  
## car\_classsmall\_suv\_2wd -4.42255 0.30960  
## car\_classsmall\_suv\_4wd -2.86215 0.31411  
## car\_classspecial\_purpose\_vehicle,\_minivan\_2wd -5.09820 0.70502  
## car\_classspecial\_purpose\_vehicle,\_minivan\_4wd -5.43698 2.21103  
## car\_classspecial\_purpose\_vehicle\_2wd -7.73837 0.59627  
## car\_classspecial\_purpose\_vehicle\_4wd -3.07742 1.47942  
## car\_classspecial\_purpose\_vehicle\_cab\_chassis -4.50972 0.85899  
## car\_classstandard\_pick-up\_trucks\_2wd -2.54663 0.50741  
## car\_classstandard\_pick-up\_trucks\_4wd -2.20227 0.56279  
## car\_classstandard\_suv\_2wd -3.90368 0.43566  
## car\_classstandard\_suv\_4wd -2.64965 0.35771  
## car\_classsubcompact\_cars -0.29372 0.28099  
## car\_classtwo\_seaters -1.06379 0.37377  
## car\_classvans,\_cargo\_types -9.47186 1.59444  
## car\_classvans,\_passenger\_type -8.61742 0.75312  
## poly(eng\_displ, 2)1 NA NA  
## poly(eng\_displ, 2)2 80.84012 4.59918  
## t value Pr(>|t|)   
## (Intercept) -1.272 0.203510   
## model\_year 1.719 0.085727 .   
## divisionalfa\_romeo -2.657 0.007935 \*\*   
## divisionaston\_martin\_lagonda\_ltd -8.002 1.83e-15 \*\*\*  
## divisionaudi -4.255 2.16e-05 \*\*\*  
## divisionbentley -5.385 7.89e-08 \*\*\*  
## divisionbmw -2.911 0.003638 \*\*   
## divisionbugatti -2.895 0.003820 \*\*   
## divisionbuick -8.269 < 2e-16 \*\*\*  
## divisioncadillac -6.772 1.57e-11 \*\*\*  
## divisionchevrolet -6.815 1.17e-11 \*\*\*  
## divisionchrysler -4.962 7.45e-07 \*\*\*  
## divisiondodge -7.023 2.76e-12 \*\*\*  
## divisionferrari\_north\_america,\_inc. -14.556 < 2e-16 \*\*\*  
## divisionfiat -5.340 1.01e-07 \*\*\*  
## divisionford -6.565 6.25e-11 \*\*\*  
## divisiongenesis -3.209 0.001348 \*\*   
## divisiongmc -6.350 2.54e-10 \*\*\*  
## divisionhonda -5.084 3.95e-07 \*\*\*  
## divisionhyundai\_motor\_company -8.320 < 2e-16 \*\*\*  
## divisioninfiniti -2.314 0.020721 \*   
## divisionjaguar -4.696 2.80e-06 \*\*\*  
## divisionjeep -7.907 3.86e-15 \*\*\*  
## divisionkia\_motors\_corporation -8.587 < 2e-16 \*\*\*  
## divisionlamborghini -4.510 6.79e-06 \*\*\*  
## divisionland\_rover -4.229 2.43e-05 \*\*\*  
## divisionlexus -3.696 0.000224 \*\*\*  
## divisionlincoln -6.057 1.59e-09 \*\*\*  
## divisionlotus\_cars\_ltd -2.446 0.014511 \*   
## divisionmaserati -8.087 9.27e-16 \*\*\*  
## divisionmazda -0.622 0.534277   
## divisionmclaren -1.876 0.060796 .   
## divisionmclaren\_automotive\_limted -1.508 0.131676   
## divisionmercedes-benz -1.770 0.076846 .   
## divisionmini -4.328 1.56e-05 \*\*\*  
## divisionmitsubishi\_motors\_corporation -6.745 1.88e-11 \*\*\*  
## divisionmobility\_ventures\_llc -1.483 0.138220   
## divisionnissan -6.981 3.71e-12 \*\*\*  
## divisionpagani\_automobili\_s.p.a. -3.834 0.000129 \*\*\*  
## divisionporsche -6.475 1.13e-10 \*\*\*  
## divisionram -5.987 2.44e-09 \*\*\*  
## divisionrolls-royce\_motor\_cars\_limited -4.615 4.13e-06 \*\*\*  
## divisionroush\_industries,\_inc. -3.364 0.000780 \*\*\*  
## divisionscion -5.285 1.36e-07 \*\*\*  
## divisionsubaru -5.185 2.33e-07 \*\*\*  
## divisiontoyota -3.640 0.000277 \*\*\*  
## divisionvolkswagen -4.163 3.24e-05 \*\*\*  
## divisionvolvo\_cars\_of\_north\_america,\_llc -3.980 7.10e-05 \*\*\*  
## eng\_displ -24.795 < 2e-16 \*\*\*  
## cylinders 4.270 2.02e-05 \*\*\*  
## aspirationSupercharged -5.175 2.46e-07 \*\*\*  
## aspirationTurbocharged -16.363 < 2e-16 \*\*\*  
## aspirationTurbocharged+Supercharged -3.595 0.000330 \*\*\*  
## transAM 1.824 0.068228 .   
## transAMS -6.000 2.25e-09 \*\*\*  
## transCVT 15.200 < 2e-16 \*\*\*  
## transM -14.860 < 2e-16 \*\*\*  
## transSA -0.753 0.451277   
## transSCV 6.700 2.54e-11 \*\*\*  
## gears 5.799 7.48e-09 \*\*\*  
## lockup\_torque\_converter -17.996 < 2e-16 \*\*\*  
## drive\_sysA 2.148 0.031774 \*   
## drive\_sysF 10.159 < 2e-16 \*\*\*  
## drive\_sysP -1.018 0.308612   
## drive\_sysR 4.508 6.82e-06 \*\*\*  
## fuel\_typeG -17.786 < 2e-16 \*\*\*  
## fuel\_typeGM -10.082 < 2e-16 \*\*\*  
## fuel\_typeGP -22.168 < 2e-16 \*\*\*  
## fuel\_typeGPR -19.789 < 2e-16 \*\*\*  
## intake\_valves\_per\_cyl -5.107 3.50e-07 \*\*\*  
## exhaust\_valves\_per\_cyl 2.562 0.010466 \*   
## car\_classlarge\_cars -1.989 0.046762 \*   
## car\_classmidsize\_cars 1.289 0.197617   
## car\_classmidsize\_station\_wagons -1.465 0.143020   
## car\_classminicompact\_cars 1.736 0.082649 .   
## car\_classsmall\_pick-up\_trucks\_2wd -7.369 2.29e-13 \*\*\*  
## car\_classsmall\_pick-up\_trucks\_4wd -6.124 1.05e-09 \*\*\*  
## car\_classsmall\_station\_wagons -2.885 0.003942 \*\*   
## car\_classsmall\_suv\_2wd -14.285 < 2e-16 \*\*\*  
## car\_classsmall\_suv\_4wd -9.112 < 2e-16 \*\*\*  
## car\_classspecial\_purpose\_vehicle,\_minivan\_2wd -7.231 6.27e-13 \*\*\*  
## car\_classspecial\_purpose\_vehicle,\_minivan\_4wd -2.459 0.013996 \*   
## car\_classspecial\_purpose\_vehicle\_2wd -12.978 < 2e-16 \*\*\*  
## car\_classspecial\_purpose\_vehicle\_4wd -2.080 0.037609 \*   
## car\_classspecial\_purpose\_vehicle\_cab\_chassis -5.250 1.64e-07 \*\*\*  
## car\_classstandard\_pick-up\_trucks\_2wd -5.019 5.55e-07 \*\*\*  
## car\_classstandard\_pick-up\_trucks\_4wd -3.913 9.34e-05 \*\*\*  
## car\_classstandard\_suv\_2wd -8.960 < 2e-16 \*\*\*  
## car\_classstandard\_suv\_4wd -7.407 1.74e-13 \*\*\*  
## car\_classsubcompact\_cars -1.045 0.295970   
## car\_classtwo\_seaters -2.846 0.004461 \*\*   
## car\_classvans,\_cargo\_types -5.941 3.22e-09 \*\*\*  
## car\_classvans,\_passenger\_type -11.442 < 2e-16 \*\*\*  
## poly(eng\_displ, 2)1 NA NA   
## poly(eng\_displ, 2)2 17.577 < 2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 3.07 on 2591 degrees of freedom  
## Multiple R-squared: 0.8742, Adjusted R-squared: 0.8697   
## F-statistic: 193.7 on 93 and 2591 DF, p-value: < 2.2e-16

However,

car\_train %>%   
 group\_by(division) %>%   
 count() %>%   
 arrange(n) %>%   
 head(8)

## # A tibble: 8 x 2  
## # Groups: division [8]  
## division n  
## <chr> <int>  
## 1 bugatti 1  
## 2 mclaren\_automotive\_limted 1  
## 3 mobility\_ventures\_llc 1  
## 4 pagani\_automobili\_s.p.a. 1  
## 5 lotus\_cars\_ltd 2  
## 6 mclaren 2  
## 7 alfa\_romeo 5  
## 8 genesis 7

If one of these low occurence car divisions in only in the assessment set, it may cause issues (or errors) in some models.

### Linear regression analysis for the car data

With **recipes**, variables can have different roles, such as case-weights, cluster, censotring indicator inc. We should keep carline in the data so that we can use it to diagnose issues but we don’t want it as a predictor.

The role of this variable will be changed to “car name”.

library(recipes)  
basic\_rec <- recipe(mpg ~ ., data = car\_train) %>%  
 # keep the car name but don't use as a predictor  
 add\_role(carline, new\_role = "car name") %>%  
 # collapse some divisions into "other"  
 step\_other(division, threshold = 0.005) %>%  
 step\_dummy(all\_nominal(), -carline) %>%  
 step\_zv(all\_predictors())

## Warning: Changing role(s) for carline

### Potential issues with linear regression

We’ll look at the car data and examine a few different models to illustrate some more complex models and approaches to optimizing them. We’ll start with linear models.

However, some potential issues with linear methods: - They do not automatically do feature selection and including irrelevant predictors may degreade performance - Linear models are sensitive to situations where the predictors are highly correlated (aka **collinearity**). This isn’t too big of an issue for these data though.

# column division, carline, car\_class has too many classes  
GGally::ggpairs(car\_train[,c(-2,-3,-15)])

To mitigate these two secnarios, **regularization** will be used. This approach adds a penalty to the regression parameters. In order to have a large slope in the model, the predictor will need to have a large impact on the model.

There are different types of regularization methods.

### Effect of collenearity

As an example of collinearity, the longley contains economic data. We can try to predict the number of people employed per year as a function of the population size and the nation’s gross domestic product (GDP).

The issue is that the two predictors have a correlation near 1. What happens when we fit models with both ppredictors versus one-at-a-time?

The coefficients can drastically change depending on what is in the model.

### Regularized linear regression

Now suppose we want to see if regularizing the regression coefficients will result in better fits. The glmnet model can be used to build a linear model using or regularization (or mixture of the two).

* an penalty (penalty is ) can have the effect of setting coefficnets to zero
* regularization () is basically the regression where the magnitude of the coefficients are dampered to avoid overfitting

For a glmnet model, we need to determine the total amount regularization (called lambda) and the mixture of and (called alpha). - alpha=1 is a *lasso model* while alpha=0 is *ridge regression* (aka weight decay)

The predictors require centering and scaling before being used in a glmnet, lasso, ridge regression model.

### Tuning the glmnet model

We have tw otuning parameters now (alpha and lambda). We can extend our previous grid search approach by creating a 2D grid of parameters to test.

* must be between zero and one. A small grid is used for this parameter.
* is not as clear-cut. We consider values on the scale. Usuallty values less than one are sufficient but this is not always true.

We can create combinations of these parameters and store them in a data frame.

glmn\_grid <- expand.grid(alpha = seq(0, 1, by = .25), lambda = 10^seq(-3, -1, length = 20))  
nrow(glmn\_grid)

## [1] 100

Instead of using rsample to tune the model, the train function in the caret package will be introduced.

### Caret

caret was developed to:

* create a unified interface for modeling and predictions to 238 models
* streamline model tuning using resampling
* provide a variety of “helper” functions and classes for day-today model building tasks
* increase computational efficiency using parallel processing
* enable serveral feature selection frameworks

It was originally developed in 2005 and is still very active. There is an extentisve [github.io](https://topepo.github.io/caret/) page and article in [JSS](http://www.jstatsoft.org/v28/i05/paper).

### Caret Basics

train can take a formula method, a recipe object, or non-formula approach (x/y) to specify the model.

train(recipe, data=dataset)  
# or   
train(y~., data=dataset)  
# or   
train(x=predictors, y=outcome)

Another argument, method is used to specify the type of model to fit. This is usually named after the fitting function. We will need to use method="glmnet" for that model. ?models has a list of all possibilities.

One way of listing the submodels should be evaluated is to used the tuneGrid parameter:

train(recipe,   
 data=car\_train,  
 method="glmnet",  
 tuneGrid=glmn\_grid)

Alternatively, the tuneLength argument will let train determine a grid sequence for each parameter.

### The resampling scheme

How much (and how) should we resample these data to create the model?

Previously, cross-validation was discussed. If 10-fold CV was used, the assessment set would consist of 268 cars (on average). That seems like an acceptable amount of data to determine the RMSE for each submodel. train has a control function that can be used to define parameters related to the numerical aspects of the search:

library(caret)  
ctrl <- trainControl(  
 method = "cv",   
 # Save the assessment predictions from the best model  
 savePredictions = "final",  
 # Log the progress of the tuning process  
 verboseIter = TRUE  
 )

### Fitting the model via caret::train

Let’s add some nonlinearity and centering/scaling to the preprocessing and run the search:

glmn\_rec <- basic\_rec %>%  
 step\_poly(eng\_displ) %>%  
 step\_center(all\_predictors()) %>%  
 step\_scale(all\_predictors())  
  
set.seed(3544)  
  
glmn\_mod <- train(  
 glmn\_rec,   
 data = car\_train,  
 method = "glmnet",   
 trControl = ctrl,  
 tuneGrid = glmn\_grid  
 )

## Loading required namespace: glmnet

## Preparing recipe

## Warning: package 'glmnet' was built under R version 3.5.1

## Loading required package: Matrix

##   
## Attaching package: 'Matrix'

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

## Loading required package: foreach

##   
## Attaching package: 'foreach'

## The following objects are masked from 'package:purrr':  
##   
## accumulate, when

## Loaded glmnet 2.0-16

## + Fold01: alpha=0.00, lambda=0.1   
## - Fold01: alpha=0.00, lambda=0.1   
## + Fold01: alpha=0.25, lambda=0.1   
## - Fold01: alpha=0.25, lambda=0.1   
## + Fold01: alpha=0.50, lambda=0.1   
## - Fold01: alpha=0.50, lambda=0.1   
## + Fold01: alpha=0.75, lambda=0.1   
## - Fold01: alpha=0.75, lambda=0.1   
## + Fold01: alpha=1.00, lambda=0.1   
## - Fold01: alpha=1.00, lambda=0.1   
## + Fold02: alpha=0.00, lambda=0.1   
## - Fold02: alpha=0.00, lambda=0.1   
## + Fold02: alpha=0.25, lambda=0.1   
## - Fold02: alpha=0.25, lambda=0.1   
## + Fold02: alpha=0.50, lambda=0.1   
## - Fold02: alpha=0.50, lambda=0.1   
## + Fold02: alpha=0.75, lambda=0.1   
## - Fold02: alpha=0.75, lambda=0.1   
## + Fold02: alpha=1.00, lambda=0.1   
## - Fold02: alpha=1.00, lambda=0.1   
## + Fold03: alpha=0.00, lambda=0.1   
## - Fold03: alpha=0.00, lambda=0.1   
## + Fold03: alpha=0.25, lambda=0.1   
## - Fold03: alpha=0.25, lambda=0.1   
## + Fold03: alpha=0.50, lambda=0.1   
## - Fold03: alpha=0.50, lambda=0.1   
## + Fold03: alpha=0.75, lambda=0.1   
## - Fold03: alpha=0.75, lambda=0.1   
## + Fold03: alpha=1.00, lambda=0.1   
## - Fold03: alpha=1.00, lambda=0.1   
## + Fold04: alpha=0.00, lambda=0.1   
## - Fold04: alpha=0.00, lambda=0.1   
## + Fold04: alpha=0.25, lambda=0.1   
## - Fold04: alpha=0.25, lambda=0.1   
## + Fold04: alpha=0.50, lambda=0.1   
## - Fold04: alpha=0.50, lambda=0.1   
## + Fold04: alpha=0.75, lambda=0.1   
## - Fold04: alpha=0.75, lambda=0.1   
## + Fold04: alpha=1.00, lambda=0.1   
## - Fold04: alpha=1.00, lambda=0.1   
## + Fold05: alpha=0.00, lambda=0.1   
## - Fold05: alpha=0.00, lambda=0.1   
## + Fold05: alpha=0.25, lambda=0.1   
## - Fold05: alpha=0.25, lambda=0.1   
## + Fold05: alpha=0.50, lambda=0.1   
## - Fold05: alpha=0.50, lambda=0.1   
## + Fold05: alpha=0.75, lambda=0.1   
## - Fold05: alpha=0.75, lambda=0.1   
## + Fold05: alpha=1.00, lambda=0.1   
## - Fold05: alpha=1.00, lambda=0.1   
## + Fold06: alpha=0.00, lambda=0.1   
## - Fold06: alpha=0.00, lambda=0.1   
## + Fold06: alpha=0.25, lambda=0.1   
## - Fold06: alpha=0.25, lambda=0.1   
## + Fold06: alpha=0.50, lambda=0.1   
## - Fold06: alpha=0.50, lambda=0.1   
## + Fold06: alpha=0.75, lambda=0.1   
## - Fold06: alpha=0.75, lambda=0.1   
## + Fold06: alpha=1.00, lambda=0.1   
## - Fold06: alpha=1.00, lambda=0.1   
## + Fold07: alpha=0.00, lambda=0.1   
## - Fold07: alpha=0.00, lambda=0.1   
## + Fold07: alpha=0.25, lambda=0.1   
## - Fold07: alpha=0.25, lambda=0.1   
## + Fold07: alpha=0.50, lambda=0.1   
## - Fold07: alpha=0.50, lambda=0.1   
## + Fold07: alpha=0.75, lambda=0.1   
## - Fold07: alpha=0.75, lambda=0.1   
## + Fold07: alpha=1.00, lambda=0.1   
## - Fold07: alpha=1.00, lambda=0.1   
## + Fold08: alpha=0.00, lambda=0.1   
## - Fold08: alpha=0.00, lambda=0.1   
## + Fold08: alpha=0.25, lambda=0.1   
## - Fold08: alpha=0.25, lambda=0.1   
## + Fold08: alpha=0.50, lambda=0.1   
## - Fold08: alpha=0.50, lambda=0.1   
## + Fold08: alpha=0.75, lambda=0.1   
## - Fold08: alpha=0.75, lambda=0.1   
## + Fold08: alpha=1.00, lambda=0.1   
## - Fold08: alpha=1.00, lambda=0.1   
## + Fold09: alpha=0.00, lambda=0.1   
## - Fold09: alpha=0.00, lambda=0.1   
## + Fold09: alpha=0.25, lambda=0.1   
## - Fold09: alpha=0.25, lambda=0.1   
## + Fold09: alpha=0.50, lambda=0.1   
## - Fold09: alpha=0.50, lambda=0.1   
## + Fold09: alpha=0.75, lambda=0.1   
## - Fold09: alpha=0.75, lambda=0.1   
## + Fold09: alpha=1.00, lambda=0.1   
## - Fold09: alpha=1.00, lambda=0.1   
## + Fold10: alpha=0.00, lambda=0.1   
## - Fold10: alpha=0.00, lambda=0.1   
## + Fold10: alpha=0.25, lambda=0.1   
## - Fold10: alpha=0.25, lambda=0.1   
## + Fold10: alpha=0.50, lambda=0.1   
## - Fold10: alpha=0.50, lambda=0.1   
## + Fold10: alpha=0.75, lambda=0.1   
## - Fold10: alpha=0.75, lambda=0.1   
## + Fold10: alpha=1.00, lambda=0.1   
## - Fold10: alpha=1.00, lambda=0.1   
## Aggregating results  
## Selecting tuning parameters  
## Fitting alpha = 0.5, lambda = 0.00127 on full training set

### Aside: submodel trick

Our grid contained 5 values for alpha and 20 values of lambda. Although it might seem like we are fitting 100 models per resample, we are not. For many models, including glmnet, there are some computational shortcuts that can be used.

In this case, for a fixed value of alpha, the glmnet model computes the results for all possible values of lambda. Predictions from any of these models can be obtained from the same objects.

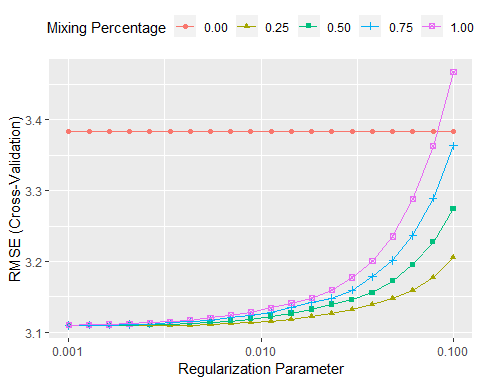
This means that we only need fit 5 models per resample. Trees and otherm odels can often exploit this *submodel trick* and caret automatically does this whenever possible.

### Resampling profile for lambda

summary(glmn\_mod)

## Length Class Mode   
## a0 100 -none- numeric   
## beta 8100 dgCMatrix S4   
## df 100 -none- numeric   
## dim 2 -none- numeric   
## lambda 100 -none- numeric   
## dev.ratio 100 -none- numeric   
## nulldev 1 -none- numeric   
## npasses 1 -none- numeric   
## jerr 1 -none- numeric   
## offset 1 -none- logical   
## call 5 -none- call   
## nobs 1 -none- numeric   
## lambdaOpt 1 -none- numeric   
## xNames 81 -none- character  
## problemType 1 -none- character  
## tuneValue 2 data.frame list   
## obsLevels 1 -none- logical   
## param 0 -none- list

ggplot(glmn\_mod)+scale\_x\_log10()+theme(legend.position="top")



### Model assessment using the assessment set

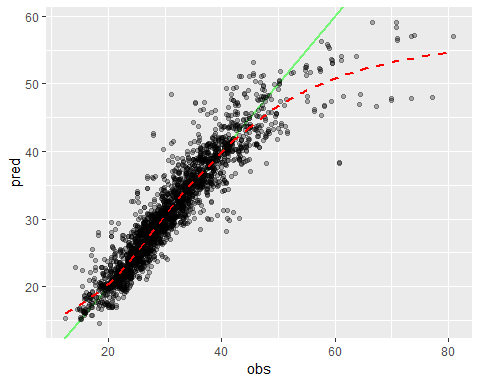
Since we used savePredictions="final", the predictions on the assessment sets are contained in the sub-object glmn\_mod$pred. This can be used to plot the data.

glmn\_mod$pred %>% head(4)

## alpha lambda obs rowIndex pred Resample  
## 1 0.5 0.001274275 30.0196 173 28.48746 Fold03  
## 2 0.5 0.001274275 34.4702 36 31.87873 Fold03  
## 3 0.5 0.001274275 28.9793 13 28.97012 Fold07  
## 4 0.5 0.001274275 21.3382 42 20.99342 Fold03

ggplot(glmn\_mod$pred, aes(x=obs,y=pred))+  
 geom\_abline(col="green",alpha=.5, lwd=1)+  
 geom\_point(alpha=.3)+  
 geom\_smooth(se=FALSE, col="red",  
 lty=2,lwd=1,alpha=.5)

## `geom\_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'



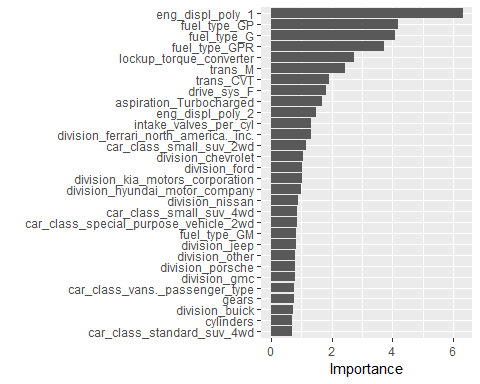
### Variable importance scores

For a **linear model** such as glmnet, we can directly inspect and interpret the model coefficients to understand what is going on.

A more general approach of computing “variable importance scores” can be useful for assessing which predictors are driving the model. These are model-specific.

For this model, we can plot the absolute values of the coefficients. This is another good reason to center and scale the predictors before the model.

reg\_imp <- varImp(glmn\_mod, scale=FALSE)  
ggplot(reg\_imp, top=30)+xlab("")



### Notes on train

Setting the seed just before calling train will ensure the same resamples are used between models. There is also [a help page on reproducibility](https://topepo.github.io/caret/model-training-and-tuning.html#repro).

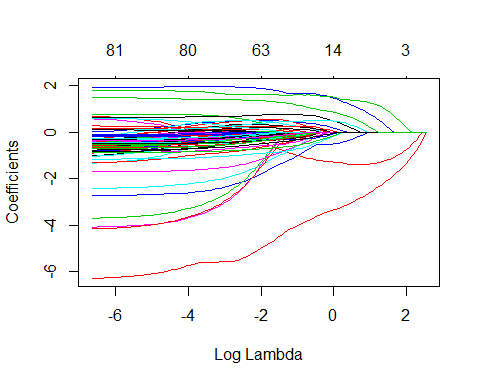
train calls the underlying model (e.g., glmnet) and arguments can be passed to the lower level functions via the …

You can write your own model code ()

### Using glmnet object

The train object saves optimized model that was fit t in the slot finalModel. This can be used as it normally would. The plot on the right is created using

library(glmnet)  
plot(glmn\_mod$finalModel,xvar="lambda")



However, **please don’t predict with it!**. Use the predict method on the object that is produced by train.

## Multivariate Adaptive Regression Splines

### MARS

MARS is a nonlinear machine learning model that develops sequential sets of artificial features that are used in linear models (similar to the previous spline discussion).

The features are “hinge functions” or single knot splines that use the function:

h(x) <- function(x) ifelse(x>0, 0)

The MARS model does a fast search through every predictor and every value of each predictor to find a suitable “split” point for the predictor that results in best features.

Suppose a value x0 is found. The MARS model creates two model terms h(x-x0) and h(x0-x) that are added to the intercept column. This creates a type of segmented regression.

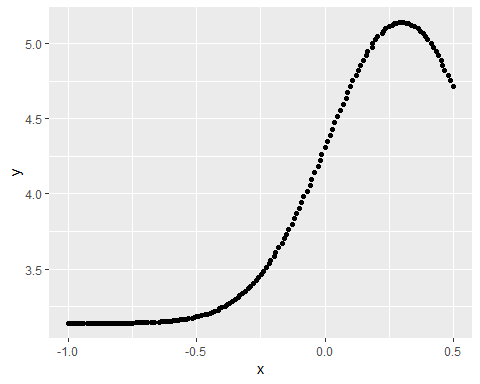
These terms are the same as deep learning rectified linear units [ReLU](https://en.wikipedia.org/wiki/Rectifier_(neural_networks)).

Let’s look at some example data…

### Simulated data

y=2*exp(-6*x(-0.3)^2)+e

x=seq(-1,0.5, by= 0.01)  
# replaced e to 3.14  
y=2\*exp(-6\*(x-0.3)^2)+3.14  
  
bind\_cols(x=x,y=y) %>%   
 ggplot(aes(x=x,y=y))+  
 geom\_jitter()



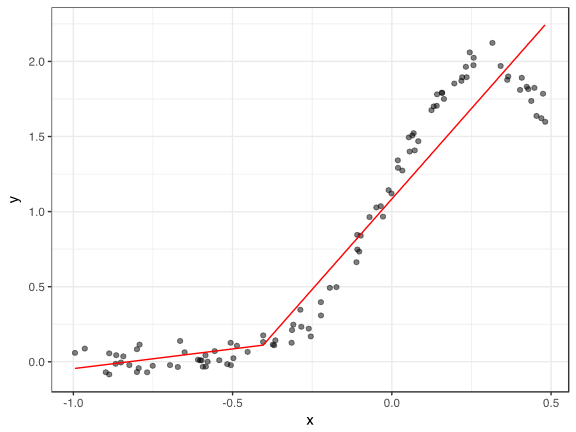
### MARS feature creation - iteration #1

After searching through these data, the model evaluates all possible values of x0 to find the best “cut” of the data. It finally chooses a value of -0.404.

To do this, it creates these two new predictors that isolate different regions of x. If we stop there, these two terms would be added into a linear regression model, yielding:

# y=  
# -.111  
# -0.262\*h(-0.40175-x)  
# +2.41\*h(x-0.40175)

### Fitted model with two features

 ### Growing and Pruning

Similar to tree-based models, MARS starts of with a “growing” stage where it keeps adding new features until it reaches a pre-defined limit. After the first pair is created, the next cut-point is found using another exhaustive search to see which split of a predictor is best conditional on the existing features.

Once all the features are created a **pruning phase** starts where model selection tools are used to eliminate terms that do not contribute meaningfully to the model. Generalized cross-validation [<https://scholar.google.com/scholar?hl=en&as_sdt=0%2C7&q=%22generalized+cross+validation%22&btnG>=] is used to efficiently remove model terms while still providing some protection from overfitting.

### Generalized cross validation with MARS

For linear models, this is a comuptational shortcut that can be used to approximilate leave-one-out cross-validation. GCV can be used since MARS is actually fit using OLS regression. The deafault MARS approach is to let GCV pick how many terms to retain.

I usually don’t use this methodology. Instead of include the number of retained terms as a tuning paramaeter, and use the usual resampling methods to optimize it.

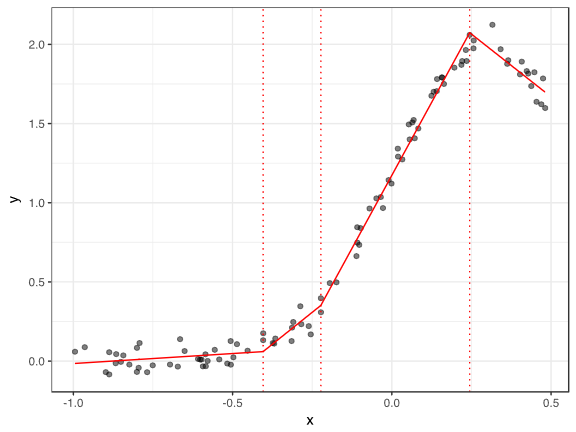
In that way, the relationship between the outcome and the number of terms can be better understood. Also, I’ve found that using GCV with MARS increases the likelihood of overfitting (anecdotally).

### The final model

For the simulated data, the MARS model only requires 4 features to model the data (via GCV).

## y =  
## 0.0599  
## - 0.126 \* h(-0.404175 - x)  
## + 1.61 \* h(x - -0.404175)  
## + 2.08 \* h(x - -0.222918)  
## - 5.27 \* h(x - 0.244406)

The parameters are estimated by adding the MARS features into Ordinary linear regression models using least squares.

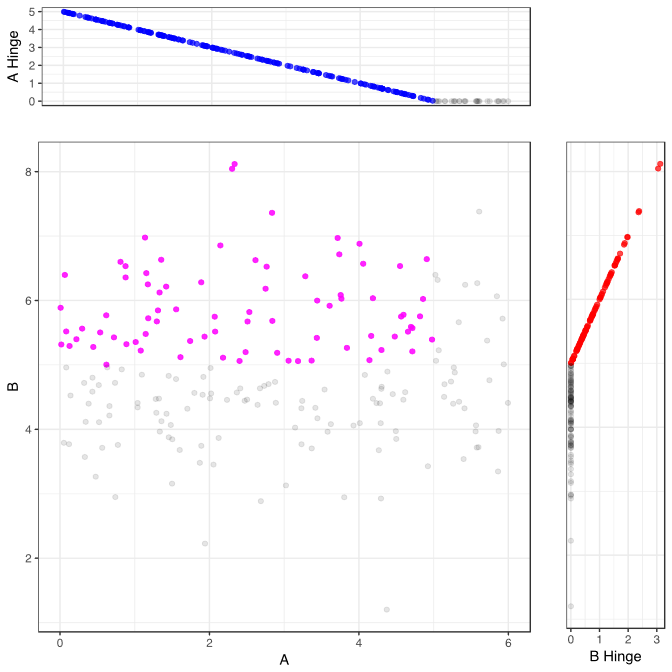


Splitting data into training and test sets.

### Aspects of MARS models

* The model also tests to see if a simple linear term is best (i.e., not split). This is also how dummy variables are evaluated.
* The model automatically conducts *feature selection*; if a predictor is never used in a split, it is functionally independent of the model. This is really good!
* If an additive model is used (as in the previous example), the functional form of each predictor can be determined (and visualized) independently for each predictor.
* A *second degree* MARS model also evaluates interaction of two hinge features (e.g., h(x0~x)}h(y-y0)). This can be useful in isolating regions of bivariate predictor space since it divides two dimensional space into four quadrants.

### Second degree MARS term example



Splitting data into training and test sets.

### MARS in R

The [mda](https://cran.r-project.org/web/packages/mda/index.html) package has a mars function but the [earth](https://cran.r-project.org/web/packages/earth/index.html) package is far superior.

The earth function has both formula and non-formula interfaces. It can also be used with generalized linear models and flexible discriminant analysis.

To use the nominal growing and GCV pruning process, the syntax is

earth(y~., data)  
#or  
earth(x=x, y=y)

The feature creation process can be controlled using the nk, nprune and pmethod parameters although this can be somewhat complex. There is a variable method that tracks the changes in the GCV results as features are added to the model.

### MARS via caret

There are several ways to fit the MARS model using train.

* method="earth" avoids pruning using GCV and uses external resampling to choose the number of retained model terms (using the sub-model trick). The two tuning parameters are nprune (number of retained features) and degree.
* method="gcvEarth" is also available and uses GCV. The degree parameter requires tuning.

I usually use the manual method to better understand the pruning process. For preprocessing, there is *no* need to remove **zero-variance predictors** here (beyond computational efficiency), but dummy variables are required fro qualitative predictors.

Centering and scaling are *not* required.

### Tuning the model

We can reuse much of the glmnet syntax to tune the model.

ctrl$verboseIter <- FALSE  
mars\_grid <- expand.grid(degree = 1:2, nprune = seq(2, 60, by = 2))  
# Using the same seed to obtain the same   
# resamples as the glmnet model.  
set.seed(3544)  
mars\_mod <- train(  
 basic\_rec,   
 data = car\_train,  
 method = "earth",  
 tuneGrid = mars\_grid,  
 trControl = ctrl  
)

## Loading required namespace: earth

## Warning: package 'earth' was built under R version 3.5.1

## Loading required package: plotmo

## Warning: package 'plotmo' was built under R version 3.5.1

## Loading required package: plotrix

## Warning: package 'plotrix' was built under R version 3.5.1

## Loading required package: TeachingDemos

## Warning: package 'TeachingDemos' was built under R version 3.5.1

summary(mars\_grid)

## degree nprune   
## Min. :1.0 Min. : 2   
## 1st Qu.:1.0 1st Qu.:16   
## Median :1.5 Median :31   
## Mean :1.5 Mean :31   
## 3rd Qu.:2.0 3rd Qu.:46   
## Max. :2.0 Max. :60

Running the resampling models (plus the last one), this takes 4.5m.

### While we wait, can I interest you in parallelism?

There is no real barrier to running these in parallel. Can we benefit from splitting the fits up to run on multiple cores? These speed-ups can be very model - and data-dependent but this pattern generally holds?

Note that there is little incremental benefit to using more workers than physical cores on the computer. (A lot more details can be found in [this blog post](http://appliedpredictivemodeling.com/blog/2018/1/17/parallel-processing))

### Running in parallel for caret

To loop through the models and data sets, caret uses the foreach package, which can parallelize for loops.

foreach has a number of parallel **backends** which allow various technologies to be used in conjunction with the package.

On CRAN, these are the “do{X}” packages, such as [doAzureParallel](https://github.com/Azure/doAzureParallel), [doFuture](https://www.rdocumentation.org/packages/doFuture), [doMC](https://www.rdocumentation.org/packages/doMC), [doMPI](https://www.rdocumentation.org/packages/doMPI), [doParallel](https://www.rdocumentation.org/packages/doParallel), [doRedis](https://www.rdocumentation.org/packages/doRedis), and [doSNOW](https://www.rdocumentation.org/packages/doSNOW).

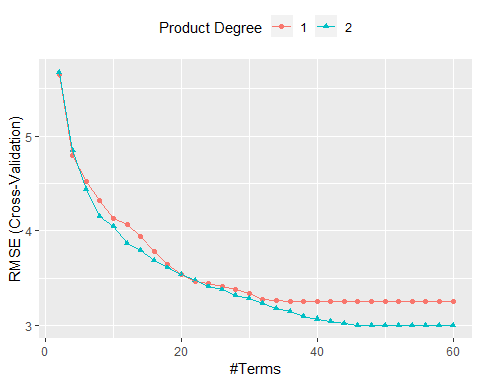
For example, doMC uses the multicore package, which forks processes to split computations (for unix and OS X). doParallel can be used for all operating systems.

To use parallel processing in caret, no changes are needed when calling train. The parallel technology must be registered with foreach prior to calling train:

library(doParallel)  
cl <- makecluster(6)  
registerDoParallel(cl)  
  
# run train...  
stopCluster(cl)  
  
# can be helpful:  
parallel::detectCores(logical=FALSE)

### Resampling profile for MARS

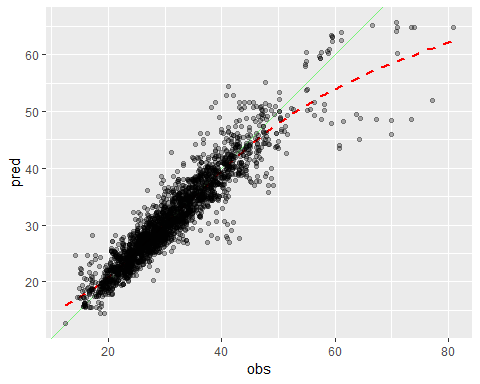
ggplot(mars\_mod)+theme(legend.position = "top")



### Prediction plot

ggplot(mars\_mod$pred, aes(x = obs, y = pred)) +  
 geom\_abline(col = "green", alpha = .5) +   
 geom\_smooth(se = FALSE, col = "red",   
 lty = 2, lwd = 1, alpha = .5) +   
 geom\_point(alpha = .3)

## `geom\_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'



### The underlying model

library(earth)  
mars\_mod$finalModel

## Selected 46 of 49 terms, and 32 of 80 predictors  
## Termination condition: RSq changed by less than 0.001 at 49 terms  
## Importance: eng\_displ, drive\_sys\_F, lockup\_torque\_converter, trans\_M, ...  
## Number of terms at each degree of interaction: 1 14 31  
## GCV 9.012742 RSS 22196.56 GRSq 0.8754532 RSq 0.8856752

### Variable importance scores

Recall that as MARS adds or drops terms from the model, the change in the GCV statistic is used to determine the worth of the terms. earth tracks the changes for each predictor and measures the variable importance based on how much the GCV error *decreases* when the model term is added.

This is cumulative when multiple terms involve the same predictor multiple times.

### Use GCV to tune the model

set.seed(3544)  
  
mars\_gcv\_mod <- train(  
 basic\_rec,  
 data=car\_train,  
 method="gcvEarth",  
 tuneGrid = data.frame(degree=1:2),  
 trControl = ctrl  
)  
  
mars\_gcv\_mod$finalModel

## Selected 46 of 49 terms, and 32 of 80 predictors  
## Termination condition: RSq changed by less than 0.001 at 49 terms  
## Importance: eng\_displ, drive\_sys\_F, lockup\_torque\_converter, trans\_M, ...  
## Number of terms at each degree of interaction: 1 14 31  
## GCV 9.012742 RSS 22196.56 GRSq 0.8754532 RSq 0.8856752

It found the same model using GCV. This is not typical but it is faster (4.1-fold). We will explot this in a moment.

## Ensenbles vs Bagging

### Baggin models

[Baggin](https://scholar.google.com/scholar?cluster=18412826781870444603&hl=en&as_sdt=0,7) is a method of creating ensembles of the model type.

Instead of using the training set, many variations of the data are created that spawn multiple versions of the same model. When predicting a new sample, the individual predictions are generated for each model in the ensemble, and these are blended into a single value. This reduces the variation in the predictions since are averaging pseudo-replicates of the model.

Bagging creates the data sets using a **bootstrap sample** of the training set.

Bagging is most useful when the underlying model has some **instability**