UC business Analytics R programming Guide

Koji Mizumura

2019-06-05

Table of Contents

This is a practice of [UC business analytics R programming guide](http://uc-r.github.io/).

# Predictive analytics

## Machine Learning

### Preparing for regression problems

Machine learning is a very iterative process. If performed and interpreted correctly, we can have great confidence in our outcomes. If not, the results will be useless. Approaching machine learning correctly means approaching it strategically by spending our data wisely on learning and validation procedures, properly pre-processing variables, minimizing data leakage, tuning parameters and assessing model performance.

Before introducing specific algorithms, this tutorial introduces concepts that are commonly required in the supervised machine learning process and that you will see briskly covered in tutorieals that follow. This tutorial will prepare you with the fundamentals needed prior to applying supervised machine learning algorithms.

#### tl;dr

Before introducing specific algorithms, this tutorial introduces concepts that you will see briskly covered in each chapter and are necessary for any type of supervised machine learning model:

1. Prerequisites: what you will need to reproduce the analysis in this tutorial

##### Prerequisites

This tutorial leverages the following packages.

library(rsample)  
library(caret)  
library(h2o)  
library(dplyr)  
  
# turn off progress bars  
h2o.no\_progress()  
  
# launch h2o  
h2o.init()  
##   
## H2O is not running yet, starting it now...  
##   
## Note: In case of errors look at the following log files:  
## C:\Users\KOJIKM~1.MIZ\AppData\Local\Temp\RtmpOmYaQa/h2o\_KojiKM\_Mizumura\_started\_from\_r.out  
## C:\Users\KOJIKM~1.MIZ\AppData\Local\Temp\RtmpOmYaQa/h2o\_KojiKM\_Mizumura\_started\_from\_r.err  
##   
##   
## Starting H2O JVM and connecting: . Connection successful!  
##   
## R is connected to the H2O cluster:   
## H2O cluster uptime: 5 seconds 628 milliseconds   
## H2O cluster timezone: Asia/Tokyo   
## H2O data parsing timezone: UTC   
## H2O cluster version: 3.22.1.1   
## H2O cluster version age: 5 months !!!   
## H2O cluster name: H2O\_started\_from\_R\_KojiKM.Mizumura\_tty566   
## H2O cluster total nodes: 1   
## H2O cluster total memory: 1.96 GB   
## H2O cluster total cores: 4   
## H2O cluster allowed cores: 4   
## H2O cluster healthy: TRUE   
## H2O Connection ip: localhost   
## H2O Connection port: 54321   
## H2O Connection proxy: NA   
## H2O Internal Security: FALSE   
## H2O API Extensions: Algos, AutoML, Core V3, Core V4   
## R Version: R version 3.6.0 (2019-04-26)

To illustrate some of concepts we will use the Ames Housing data that has been included in the AmesHousing package and the employee attrition data that has been included in the rsample package. The housing data represents a continuous response variable (Sale\_Price) along with 80 features (predictor variables) for 2930 homes in Ames, IA. Read more about this data [here](https://cran.r-project.org/web/packages/AmesHousing/AmesHousing.pdf). The attrition data represents a classification response variable (Attrition) with 30 features for 1470 employees. Read more about this data [here](https://www.ibm.com/communities/analytics/watson-analytics-blog/hr-employee-attrition/)

Throughout this tutorial, we will demonstrate approaches with the regular df data frame. However, since many of the supervised machine learning tutorials that we provide leverage h2o, we also show how to do some of the things with h2o. This requires your data to be in H2O object, which you can convert any data friame easily with as.h2o.

# ames data  
ames <- AmesHousing::make\_ames()  
ames.h2o <- as.h2o(ames)  
  
# attrition data  
churn <- rsample::attrition %>%   
 mutate\_if(is.ordered, factor, ordered=FALSE)  
churn.h2o <- as.h2o(churn)

#### Data splitting

##### Spending our data wisely

A major goal of the machine learning process is to find an algorithm that most accurately predicts future values based on a set of inputs . In other words, we want an algorithm that not only fits well to our past data, but more importantly, one that predicts a future outcome accurately. This is called the generalizability of our algorithm. How we 窶徭pend窶? our data will help us understand how well our algorithm generalizes to unseen data.

To provide an accurate understanding of the generalizability of our fina optimal model, we split our data into training and test data sets.

* **Training set**: these data are used to train our algorithms and tune hyper-parameters.
* **Test set**: having chosen a final model, these data are used to estimate its prediction error (generalization error). These data should not be used *during model training*

Given a fixed amount of data, typical recommendations for splitting your data into training-testing splits include 60% (training) - 40% (testing), 70%-30%, or 80%-20%. Generally speaking, these are appropriate guidelines to follow; however, it is good to keep in mind that as your overall data set gets smaller,

* Spending too much in training () won’t allow us to get a good assessment of predictive performance. We may find a model that fits the training data very well, but is not generalizable (overfitting),
* sometimes too much spent in testing () won’t allow us to get a good assessment of model parameters.

Typically, we are lacking in the size of our data here, so a 70-30 split is often sufficinet. The two most common ways of splittin data include **simple random sampling** and **stratified sampling**.

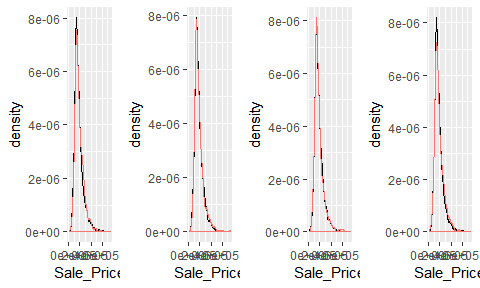
##### Simple random sampling

The simplest way to split the data into training and test sets is to take a simple random sample. This does not control for any data attributes, such as the percentage of data in the quantiles in your response variable (). There are multiple ways to split our data. Here we show four options to produce a 70-30 split (note that setting the seed value allows you to reproduce your randomized splits):

# base R  
df <- ames  
df.h2o <- ames.h2o  
  
set.seed(123)  
df  
## # A tibble: 2,930 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 One\_Story\_~ Resident~ 80 11622 Pave No\_A~ Regular   
## 3 One\_Story\_~ Resident~ 81 14267 Pave No\_A~ Slightly~  
## 4 One\_Story\_~ Resident~ 93 11160 Pave No\_A~ Regular   
## 5 Two\_Story\_~ Resident~ 74 13830 Pave No\_A~ Slightly~  
## 6 Two\_Story\_~ Resident~ 78 9978 Pave No\_A~ Slightly~  
## 7 One\_Story\_~ Resident~ 41 4920 Pave No\_A~ Regular   
## 8 One\_Story\_~ Resident~ 43 5005 Pave No\_A~ Slightly~  
## 9 One\_Story\_~ Resident~ 39 5389 Pave No\_A~ Slightly~  
## 10 Two\_Story\_~ Resident~ 60 7500 Pave No\_A~ Regular   
## # ... with 2,920 more rows, and 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>  
index <- sample(1:nrow(df), round(nrow(df) \* 0.7))  
train\_1 <- df[index, ]  
test\_1 <- df[-index, ]  
  
# caret package  
set.seed(123)  
index2 <- createDataPartition(df$Sale\_Price, p = 0.7, list = FALSE)  
train\_2 <- df[index2, ]  
test\_2 <- df[-index2, ]  
  
# rsample package  
set.seed(123)  
split\_1 <- initial\_split(df, prop = 0.7)  
train\_3 <- training(split\_1)  
test\_3 <- testing(split\_1)  
  
# h2o package  
split\_2 <- h2o.splitFrame(df.h2o, ratios = 0.7, seed = 123)  
train\_4 <- split\_2[[1]]  
test\_4 <- split\_2[[2]]

Since this sampling approach will randomly sample across the distribution of (Sale\_Price), you will typically result in a similar distribution betwen your training and test sets as iilustrated below.

# base R  
p1 <- ggplot()+  
 geom\_density(data=train\_1, aes(Sale\_Price), show.legend = FALSE)+  
 geom\_density(data=test\_1, aes(x=Sale\_Price, col="red"),show.legend = FALSE)  
   
# caret   
p2 <- ggplot()+  
 geom\_density(data=train\_2, aes(Sale\_Price),show.legend = FALSE)+  
 geom\_density(data=test\_2, aes(x=Sale\_Price, col="red"),show.legend = FALSE)  
  
# sample  
p3 <- ggplot()+  
 geom\_density(data=train\_3, aes(Sale\_Price),show.legend = FALSE)+  
 geom\_density(data=test\_3, aes(x=Sale\_Price, col="red"),show.legend = FALSE)  
  
# h2o  
  
p4 <- ggplot()+  
 geom\_density(data=as\_tibble(train\_4), aes(Sale\_Price),show.legend = FALSE)+  
 geom\_density(data=as\_tibble(test\_4), aes(x=Sale\_Price, col="red"),show.legend = FALSE)  
  
gridExtra::grid.arrange(p1,p2,p3,p4,  
 nrow=1)



##### Stratified sampling

However, if we want to explictly control our sampling so that our training and test sets have similar distributions, we can use **stratified sampling**. This is more common with classification problems where the response variable may be imbalanced (90% of observations with response “Yes” and 10% with response “No”). However, we can also apply to regression problems for data sets that have a small sample size and where the response variable deviates strongly from *normality*.

With a continuous response variable, stratified sampling will break y down into quantiles and randomly sample from each quantile. Consequently, this will help ensure a balanced representation of the response distribution in both the training and test sets.

The easiest way to perform stratified sampling on a response variable is to use the rsample package, where you specify the response variable to strata fy. The following illustrates that in our original employee attrition data we have an imbalanced response (No: 84%, Yes:16%). By enforcing stratified sampling both our training and testing sets have approximiately equal response distributions.

# original response distribution  
table(churn$Attrition) %>% prop.table()  
##   
## No Yes   
## 0.8387755 0.1612245  
  
# stratified sampling with the rsample package  
set.seed(123)  
split\_strat <- initial\_split(churn, prop=0.7, strata = "Attrition")  
train\_strat <- training(split\_strat)  
test\_strat <- testing(split\_strat)  
  
# consistent response ratio between train & test  
table(train\_strat$Attrition) %>% prop.table()  
##   
## No Yes   
## 0.838835 0.161165  
table(test\_strat$Attrition) %>% prop.table()  
##   
## No Yes   
## 0.8386364 0.1613636

#### Feature engineering

**Feature engineering** generally refers to the process of adding, deleting and transforming the variables to be applied to your machine learning algorithms.

Feature engineering is a siginificant process and requires you to spend substantial time understanding your data… or as Leo Breiman said “live with your data before you plunge into modeling”

Although this guide is primarily concerned with machine learning algorithms, feature engineering can make or break an algorithm窶冱 predictive ability. We will not cover all the potential ways of implementing feature engineering; however, we will cover a few fundamental pre-processing tasks that can significantly improve modeling performance.

1. One-hot encoding

Many models require all variables to be numeric. Consequently, we need to transform any categorical variables into numeric representation so that these algorithms can compute. Some packages automate this process (i.e., h2o, glm, caret) while others do not (i.e., glmnet, keras). Furthermore, there are many ways to encode categorical variables as numeric representation (i.e., one-hot, orinal, binary, sum, Helmert).

The most common is refered to as one-hot encoding, where we transpose our categorical variables so that each level of the feature is represented as a boolean value. For example, one-hot encoding variable x in the following:

sample <- tibble::tribble(  
 ~id, ~x,  
 1,"a",  
 2,"c",  
 3,"b",  
 4,"c",  
 5,"c",  
 6,"a",  
 7,"b",  
 8,"c"  
)  
  
sample %>%   
 mutate(x=as.factor(x))  
## # A tibble: 8 x 2  
## id x   
## <dbl> <fct>  
## 1 1 a   
## 2 2 c   
## 3 3 b   
## 4 4 c   
## 5 5 c   
## 6 6 a   
## 7 7 b   
## 8 8 c

results in the following representation:

If you need to manually implement one-hot encoding yourself, you can do that with caret::dummyVars. Sometimes you many have a feature level with very few observations and all these observations show up in the test set but not the training set. The benefit of using dummyVars on the full data set and then applying the result to both the train and test data sets is that it will guarantee that the same features are represented in both the train and test data.

# full rank one-hot encode - recommended for generalized linear models and neural networks.  
  
library(caret)  
full\_rank <- dummyVars(~., data = df, fullRank = TRUE)  
train\_oh <- predict(full\_rank, train\_1)  
test\_oh <- predict(full\_rank, test\_1)  
  
train\_1 %>% head()  
## # 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\_~ Floating~ 81 11216 Pave No\_A~ Regular   
## 2 Two\_Story\_~ Floating~ 0 2998 Pave No\_A~ Regular   
## 3 One\_Story\_~ Resident~ 0 17871 Pave No\_A~ Moderate~  
## 4 Two\_Story\_~ Floating~ 85 10574 Pave No\_A~ Regular   
## 5 One\_and\_Ha~ Resident~ 50 6000 Pave No\_A~ Regular   
## 6 Two\_Story\_~ Floating~ 35 4251 Pave Paved Slightly~  
## # ... 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>  
train\_oh %>% as\_tibble() %>% head()  
## # A tibble: 6 x 308  
## MS\_SubClass.One~ MS\_SubClass.One~ MS\_SubClass.One~ MS\_SubClass.One~  
## <dbl> <dbl> <dbl> <dbl>  
## 1 0 0 0 0  
## 2 0 0 0 0  
## 3 0 0 0 0  
## 4 0 0 0 0  
## 5 0 0 0 1  
## 6 0 0 0 0  
## # ... with 304 more variables: MS\_SubClass.Two\_Story\_1946\_and\_Newer <dbl>,  
## # MS\_SubClass.Two\_Story\_1945\_and\_Older <dbl>,  
## # MS\_SubClass.Two\_and\_Half\_Story\_All\_Ages <dbl>,  
## # MS\_SubClass.Split\_or\_Multilevel <dbl>, MS\_SubClass.Split\_Foyer <dbl>,  
## # MS\_SubClass.Duplex\_All\_Styles\_and\_Ages <dbl>,  
## # MS\_SubClass.One\_Story\_PUD\_1946\_and\_Newer <dbl>,  
## # MS\_SubClass.One\_and\_Half\_Story\_PUD\_All\_Ages <dbl>,  
## # MS\_SubClass.Two\_Story\_PUD\_1946\_and\_Newer <dbl>,  
## # MS\_SubClass.PUD\_Multilevel\_Split\_Level\_Foyer <dbl>,  
## # MS\_SubClass.Two\_Family\_conversion\_All\_Styles\_and\_Ages <dbl>,  
## # MS\_Zoning.Residential\_High\_Density <dbl>,  
## # MS\_Zoning.Residential\_Low\_Density <dbl>,  
## # MS\_Zoning.Residential\_Medium\_Density <dbl>, MS\_Zoning.A\_agr <dbl>,  
## # MS\_Zoning.C\_all <dbl>, MS\_Zoning.I\_all <dbl>, Lot\_Frontage <dbl>,  
## # Lot\_Area <dbl>, Street.Pave <dbl>, Alley.No\_Alley\_Access <dbl>,  
## # Alley.Paved <dbl>, Lot\_Shape.Slightly\_Irregular <dbl>,  
## # Lot\_Shape.Moderately\_Irregular <dbl>, Lot\_Shape.Irregular <dbl>,  
## # Land\_Contour.HLS <dbl>, Land\_Contour.Low <dbl>,  
## # Land\_Contour.Lvl <dbl>, Utilities.NoSeWa <dbl>,  
## # Utilities.NoSewr <dbl>, Lot\_Config.CulDSac <dbl>,  
## # Lot\_Config.FR2 <dbl>, Lot\_Config.FR3 <dbl>, Lot\_Config.Inside <dbl>,  
## # Land\_Slope.Mod <dbl>, Land\_Slope.Sev <dbl>,  
## # Neighborhood.College\_Creek <dbl>, Neighborhood.Old\_Town <dbl>,  
## # Neighborhood.Edwards <dbl>, Neighborhood.Somerset <dbl>,  
## # Neighborhood.Northridge\_Heights <dbl>, Neighborhood.Gilbert <dbl>,  
## # Neighborhood.Sawyer <dbl>, Neighborhood.Northwest\_Ames <dbl>,  
## # Neighborhood.Sawyer\_West <dbl>, Neighborhood.Mitchell <dbl>,  
## # Neighborhood.Brookside <dbl>, Neighborhood.Crawford <dbl>,  
## # Neighborhood.Iowa\_DOT\_and\_Rail\_Road <dbl>,  
## # Neighborhood.Timberland <dbl>, Neighborhood.Northridge <dbl>,  
## # Neighborhood.Stone\_Brook <dbl>,  
## # Neighborhood.South\_and\_West\_of\_Iowa\_State\_University <dbl>,  
## # Neighborhood.Clear\_Creek <dbl>, Neighborhood.Meadow\_Village <dbl>,  
## # Neighborhood.Briardale <dbl>, Neighborhood.Bloomington\_Heights <dbl>,  
## # Neighborhood.Veenker <dbl>, Neighborhood.Northpark\_Villa <dbl>,  
## # Neighborhood.Blueste <dbl>, Neighborhood.Greens <dbl>,  
## # Neighborhood.Green\_Hills <dbl>, Neighborhood.Landmark <dbl>,  
## # Condition\_1.Feedr <dbl>, Condition\_1.Norm <dbl>,  
## # Condition\_1.PosA <dbl>, Condition\_1.PosN <dbl>,  
## # Condition\_1.RRAe <dbl>, Condition\_1.RRAn <dbl>,  
## # Condition\_1.RRNe <dbl>, Condition\_1.RRNn <dbl>,  
## # Condition\_2.Feedr <dbl>, Condition\_2.Norm <dbl>,  
## # Condition\_2.PosA <dbl>, Condition\_2.PosN <dbl>,  
## # Condition\_2.RRAe <dbl>, Condition\_2.RRAn <dbl>,  
## # Condition\_2.RRNn <dbl>, Bldg\_Type.TwoFmCon <dbl>,  
## # Bldg\_Type.Duplex <dbl>, Bldg\_Type.Twnhs <dbl>, Bldg\_Type.TwnhsE <dbl>,  
## # House\_Style.One\_and\_Half\_Unf <dbl>, House\_Style.One\_Story <dbl>,  
## # House\_Style.SFoyer <dbl>, House\_Style.SLvl <dbl>,  
## # House\_Style.Two\_and\_Half\_Fin <dbl>,  
## # House\_Style.Two\_and\_Half\_Unf <dbl>, House\_Style.Two\_Story <dbl>,  
## # Overall\_Qual.Poor <dbl>, Overall\_Qual.Fair <dbl>,  
## # Overall\_Qual.Below\_Average <dbl>, Overall\_Qual.Average <dbl>,  
## # Overall\_Qual.Above\_Average <dbl>, Overall\_Qual.Good <dbl>,  
## # Overall\_Qual.Very\_Good <dbl>, Overall\_Qual.Excellent <dbl>,  
## # Overall\_Qual.Very\_Excellent <dbl>, Overall\_Cond.Poor <dbl>,  
## # Overall\_Cond.Fair <dbl>, ...  
  
# less than full rank  
dummy <- dummyVars(~., data = df, fullRank=FALSE)  
train\_oh <- predict(dummy, train\_1)  
test\_oh <- predict(dummy, test\_1)

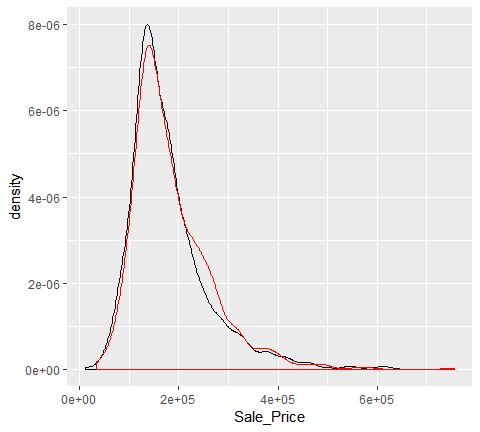
Two things to note:

* Since one-hot encoding adds new features it can significantly increase the dimensionality of our data. If you have a dataset with many categorical variables and those categorical variables in turn have many unique levels, the number of features can explode. In these cases you may want to explore ordinal encoding of your data.
* if using h2o you do not need to explictly encode your categorical variables but you can override the default encoding. This can be considered a tuning parameter as some encoding will improve modeling accuracy over other encodings. See the encoding options for h2o [here](http://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/algo-params/categorical_encoding.html).

##### Response Transformation

Although not a requirement, normalizing the distribution of the response variable by using a *transformation* can lead to a big improvement, especially for parametric models. As we saw in the data splitting section, our response variable Sale\_Price is right skewed.

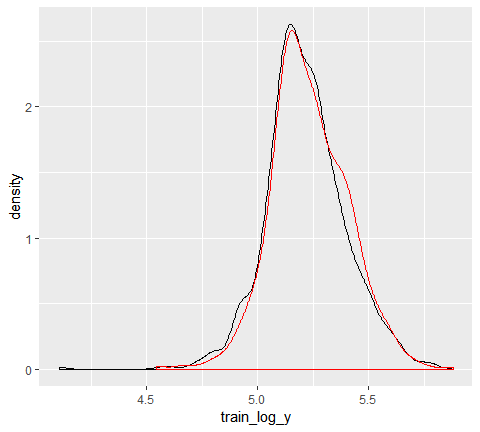
ggplot(train\_1,aes(x=Sale\_Price))+  
 geom\_density(trim=TRUE)+  
 geom\_density(data=test\_1, trim=T, col="red")



To normalize, we have two options:

**Option 1**: normalize with a \_\_log transformation\_.\_ This will transform most right skewed distributions to be approximiately normal.

# log transformation  
train\_log\_y <- log10(train\_1$Sale\_Price)  
test\_log\_y <- log10(test\_1$Sale\_Price)  
  
log\_transform <- ggplot(data = tibble(train\_log\_y), aes(train\_log\_y))+  
 geom\_density(trim=TRUE)+  
 geom\_density(data=tibble(test\_log\_y),aes(test\_log\_y), trim=T, col="red")  
log\_transform



**Option 2**: use a **Box Cox transformation**.

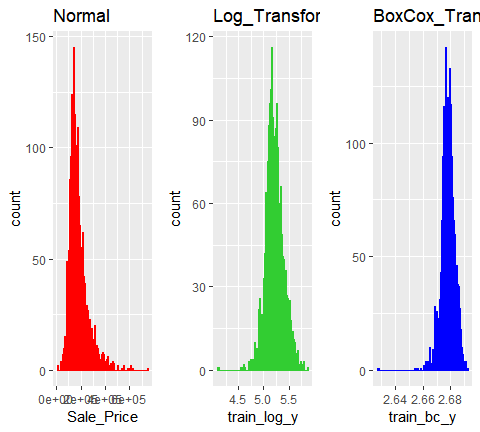
A Box Cox transformation is more flexible and will find the transformation from a family of power transformations that will transform the variable as close as possible to a normal distribution.

**Important notes**: be sure to compute the lambda on the training set and apply that same lambda to both the training and test set to minimize data leakage.

# Box cox transformation  
lambda <- forecast::BoxCox.lambda(train\_1$Sale\_Price)  
train\_bc\_y <- forecast::BoxCox(train\_1$Sale\_Price, lambda)  
test\_bc\_y <- forecast::BoxCox(test\_1$Sale\_Price, lambda)

We can see that in this example, the log transformation and Box Cox transformation both do about equally well in transforming our response variable to be normally distributed.

p1 <- ggplot(data=train\_1, aes(Sale\_Price))+  
 geom\_histogram(bins=100, col="red")+  
 ggtitle("Normal")  
  
p2 <- ggplot(data = tibble(train\_log\_y), aes(train\_log\_y))+  
 geom\_histogram(bins=100, col="lime green")+  
 ggtitle("Log\_Transform")  
  
p3 <- ggplot(data=tibble(train\_bc\_y), aes(train\_bc\_y))+  
 geom\_histogram(bins=100,col="blue")+  
 ggtitle("BoxCox\_Transform")  
  
gridExtra::grid.arrange(p1,p2,p3, nrow=1)



Note that when you model with a transformed response variable, your predictions will also be in the transformed value. You will likely want to re-transform your predicted values back to their normal state so that decision-makers can interpret the results. The following code can do this for you:

# log transform a value  
y <- log(10)  
  
# re-transforming the log-transformed value  
exp(y)  
## [1] 10  
  
# Box Cox transform a value  
y <- forecast::BoxCox(10, lambda)  
y  
## [1] 1.55142  
## attr(,"lambda")  
## [1] -0.3689351  
# forecast::BoxCox  
  
# inverse Box Cox function  
inv\_box\_cox <- function(x,lambda){  
 if (lambda==0) exp(x) else (lambda\*x+1)^(1/lambda)  
}  
  
# re-transfrming the Box Cox transform value  
inv\_box\_cox(y, lambda)  
## [1] 10  
## attr(,"lambda")  
## [1] -0.3689351

##### Predictor transformation

Some models such as K-NN, SVMs, PLS, neural networks require that the features have the same units. **Centering** and **scaling** can be used for this purpose and is often refered to as **standardizing** the features. Standardizing numeric variables results in zero mean and unit variance, which provides a common comparable unit of measure across all the variables.

Some packages have built^in arguments (i.e., h20, caret) to standardize and some do not (ie., glm, keras). IF you need to manually standardize your variables you can use the preProcess function provided by the caret package.

For example, here we center and scale our predictor variables. Note, it is important you standardize the test data based on the training mean and variance values of each feature. This minimizes data leakage.

# identify only the predictor variables  
features <- setdiff(names(train\_1), "Sale\_Price")  
  
# pre-process estimation based on training features   
pre\_process <- caret::preProcess(  
 x = train\_1[,features],  
 method = c("center", "scale")  
)  
  
# apply to both training & test  
train\_x <- predict(pre\_process, train\_1[, features])  
test\_x <- predict(pre\_process, test\_1[,features])

##### Alternative feature transformation

There are some alternative transformations that you can perform:

* Normalizing the predictor variables with a *Box Cox transformation* can improve parametric model performance.
* Collapsing highly correlated variables with *PCA* can reduce the number of features and increase the stability of generalize linear models. However, this reduces the amount of information at your disposal and future tutorials show you how to use regularization as a better alternative to PCA.
* Removing *near-zero* or *zero variance variables*. Variables with vary little variance tend to not improve model performance and can be removed.
* preProcess provides other options which you can read more about [here](https://topepo.github.io/caret/pre-processing.html).

# identify only the predictor variables  
features <- setdiff(names(train\_1), "Sale\_Price")  
  
# pre-process estimation based on training features  
pre\_process <- preProcess(  
 x = train\_1[, features],  
 method = c("center", "scale", "pca", "nzv")   
 )  
  
# apply to both training & test  
train\_x <- predict(pre\_process, train\_1[,features])  
test\_x <- predict(pre\_process, test\_1[, features])

#### Basic mddel formulation

There are **many** packages to perform machine learning and there are always more than one to perform each algorithm (i.e., there are over 20 packages to perform random forests). There are pros/cons to each package; some nay be more computationally efficient while other may have more hyperparameter tuning options.

Future tutorials will expose you to several packages; some that have become “the standard” and others that are new and may be considered “maturing”. Just realize there are more ways than one to skin.

For example, these three functions will all produce the samme linear regression model output:

lm.lm <- lm(Sale\_Price~., data = train\_1)   
lm.glm <- glm(Sale\_Price~., data=train\_1, family=gaussian)  
lm.caret <- caret::train(Sale\_Price ~., data=train\_1, method = "lm")  
  
lm\_multiple\_package = tibble(  
 model = c("lm", "glm", "caret"),  
 outcome = list(lm.lm, lm.glm, lm.caret)  
)  
  
# test <- "abcde"  
# stringr::str\_sub(test, 1,3)  
  
lm\_multiple\_package$outcome[[1]] %>%   
 broom::tidy() %>%   
 ggplot(aes(stringr::str\_sub(term, 1, 3), estimate, col=p.value<0.01))+  
 geom\_point()+  
 coord\_flip()  
  
# lm\_multiple\_package %>%   
# mutate( outcome\_tidy = purrr::map(outcome, broom::tidy))

One thing you will notice throughout future tutotiral s is that we can specify our model formulation in different ways. In the above examples, we use the model formulation (Sale\_Price, which says explain Sale\_Price based on all features) approach. Alternative approaches include the matrix formulation and variable name specification approaches.

*Matrix formulation* requires that we separate our response variable from our features. For example, in the regularization tutorial we will use glmnet which requires our features x and response y to be specified separately:

# get feature names  
features <- setdiff(names(train\_1), "Sale\_Price")  
  
# create feature and response set  
train\_x <- train\_1[, features]  
train\_y <- train\_1$Sale\_Price  
  
# example of matrix formulation  
library(glmnet)  
glmnet.m1 <- glmnet(x = train\_x, y = train\_y)

Alternatively, h2o uses *variable name specification* where we provide all the data combined in one training\_frame but we specify the features and response with character strings:

# create variable names and h2o training frame  
h2o.init()  
## Connection successful!  
##   
## R is connected to the H2O cluster:   
## H2O cluster uptime: 31 seconds 746 milliseconds   
## H2O cluster timezone: Asia/Tokyo   
## H2O data parsing timezone: UTC   
## H2O cluster version: 3.22.1.1   
## H2O cluster version age: 5 months !!!   
## H2O cluster name: H2O\_started\_from\_R\_KojiKM.Mizumura\_tty566   
## H2O cluster total nodes: 1   
## H2O cluster total memory: 1.96 GB   
## H2O cluster total cores: 4   
## H2O cluster allowed cores: 4   
## H2O cluster healthy: TRUE   
## H2O Connection ip: localhost   
## H2O Connection port: 54321   
## H2O Connection proxy: NA   
## H2O Internal Security: FALSE   
## H2O API Extensions: Algos, AutoML, Core V3, Core V4   
## R Version: R version 3.6.0 (2019-04-26)  
y <- "Sale\_Price"  
x <- setdiff(names(train\_1), y)  
train.h2o <- as.h2o(train\_1)  
  
# example of variable name specification  
h20.m1 <- h2o.glm(x=x, y=y, training\_frame = train.h2o)

##### Model tuning

Hyperparameters control the level of model complexity. Some algorithms have many tuning parameters while others have only one or two. Tuning can be a good thing as it allows us to transform our model to better align with pattersn within our data For example the simple illustration below shows how the more flexible model aligns more closely to the data than fixed linear model.

However, highly tunable models can also be dangerous because they allow us to overfit our model to the training data, which will not generalize well to future unseen data.

Throughout the future tutorial, we will demonstrate how to tune the different parameters for each model. However, we bring up this point because it feeds into the next section nicely.

##### Cross validation for generalization

Our goal is to not only find a model that performs well on training data, but to find one that performs well on *future unseen data*. So although we can tune our model to reduce some error metric to near zero on our training data, this may not generalize well to future unseen data. Consequently, our goal is to find a model and its hyperparameters that will minimize error on hold-out data.

The model on the left is considered rigid and consistent. If we provided it a new training sample with slightly different values, the model would not change much, if at all. Although it is consistent, the models does not accurately capture the underlying relationship. This is considered a model with high *bias*.

The model on the right is far more inconsistent. Even with small changes to our training sample, this model would likely change significantly. This is considered a model with high *variance*.

The model in the middle balances the two and likely will minimize the error on future unseen data compared to the high bias and high variance models. This is our goal.

knitr::include\_graphics("")

To find the model that balances the *bias-variance tradeoff*, we search for a model that minimizes a *k*-fold cross-validation error metric (you will also be introduced to what’s called an *out of bag error* which provides a similar form of evaluation). *k*-fold cross-validation is a resampling method that randomly divides the training data into *k* groups (aka folds) of approximately same size. The model is fit on folds and then held-out validation fods is used to compute the error.

This process results in *k* estimates of the test error (). Thus, the *k*-fold CV estimate is computed by averagin these values, which provides us with an approximation of the error to expect on unseen data.

knitr::include\_graphics("")

Most algorithms and packages we cover in future tutorials have built-in cross-validation capabilities. One typically uses a 5 or 10 fold CV ( *k*=5 or *k*=10 ). For example, h2o implements CV with the nfolds argument:

# example of 10 fold CV in h2o  
h2o.cv <- h2o.gbm(  
 x=x,  
 y=y,  
 training\_frame = train.h2o,  
 nfolds =10  
)

#### Model evaulation

This leads us to our final topic, error metrics to evaluate performance. There are several metrics we can choose from to assess the error of a supervised machine learning model. The most common include:

###### Regression models

* **MSE**: Mean squared error is the average of the squared eorr (). The squared component results in larger errors having larger penalties. This (along with RMSE) is the most common error metric to use. Objective: **minimize**
* **RMSE**: Root Root mean squared error. This simply takes the square root of the MSE metric (RMSE=$ $) so that your error is in the same units as your response variable. If your response variable units are dollars, the units of MSE are dollars-squared, but the RMSE will be in dollars. Objective: **minimize**
* **Deviance**:Short for mean residual deviance. In essence, it provides a measure of goodness-of-fit of the model being evaluated when compared to the null model (intercept only). If the response variable distribution is gaussian, then it is equal to MSE. When not, it usually gives a more useful estimate of error. **Objective: minimize**
* **MAE**:ean absolute error. Similar to MSE but rather than squaring, it just takes the mean absolute difference between the actual and predicted values () **Objective: minimize**
* **RMSLE**:Root mean squared logarithmic error. Similiar to RMSE but it performs a log() on the actual and predicted values prior to computing the difference ($ RMSLE = \sqrt{\_{i=1}^n (log(y\_i+1)-log())$ ) When your response variable has a wide range of values, large response values with large errors can dominate the MSE/RMSE metric. RMSLE minimizes this impact so that small response values with large errors can have just as meaningful of an impact as large response values with large errors. **Objective: minimize**
* : This is a popular metric that represents the proportion of the variance in the dependent variable that is predictable from the independent variable. Unfortunately, it has several limitations. For example, two models built from two different data sets could have the exact same RMSE but if one has less variability in the response variable then it would have a lower than the other. You should not place too much emphasis on this metric.

Most models we assess in future tutorials will report most, if not all, of these metrics. We will often emphasize and RMSE but its good to realize that certain situations warrant emphasis on some more than others.

##### Classification models

* **Misclassification**: This is the overall error. For example, say you are predicting 3 classes ( high, medium, low ) and each class has 25, 30, 35 observations respectively (90 observations total). If you misclassify 3 observations of class high, 6 of class medium, and 4 of class low, then you misclassified 13 out of 90 observations resulting in a 14% misclassification rate. Objective: minimize
* **Mean per class error**: This is the average error rate for each class. For the above example, this would be the mean of $, , $ , which is 12%. If your classes are balanced this will be identical to misclassification. **Objective: minimize**
* **MSE**: Mean squared error. Computes the distance from 1.0 to the probability suggested. So, say we have three classes, A, B, and C, and your model predicts a probabilty of 0.91 for A, 0.07 for B, and 0.02 for C. If the correct answer was A the , if it is B , if it is C . The squared component results in large differences in probabilities for the true class having larger penalties. **Objective: minimize**
* **Cross-entropy (aka Log Loss or Deviance)**: Similar to MSE but it incorporates a log of the predicted probability multiplied by the true class. Consequently, this metric disproportionately punishes predictions where we predict a small probability for the true class, which is another way of saying having high confidence in the wrong answer is really bad. **Objective: minimize**
* **Gini index**: Mainly used with tree-based methods and commonly referred to as a measure of purity where a small value indicates that a node contains predominantly observations from a single class. **Objective: minimize**

When applying classification models, we often use a *confusion matrix* to evaluate certain performance measures.A confusion matrix is simply a matrix that compares actual categorical levels (or events) to the predicted categorical levels. When we predict the right level, we refer to this as a true positive. However, if we predict a level or event that did not happen this is called a false positive (i.e. we predicted a customer would redeem a coupon and they did not). Alternatively, when we do not predict a level or event and it does happen that this is called a false negative (i.e. a customer that we did not predict to redeem a coupon does).

We can extract different levels of performance from these measures. For example, given the classification matrix below, we can assess the following:

* **Accuracy**: Overall, how often is the classifier correct? Opposite of misclassification above. Example: . **Objective: maximize**
* **Precision**: How accurately does the classifier predict events? This metric is concerned with maximizing the true positives to false positive ratio. In other words, for the number of predictions that we made, how many were correct? Example: . **Objective: maximize**
* **Sensitivity (aka recall)**: How accurately does the clasifier classify actual events? This metric is concerned with maximizing the true positives to false negative ratio. In other words, for the events that occured, how many did we predict? Example: . **Objective: maximize**
* **Specificity**: How accurately does the classifier classify actual events? This metric is concerned with maximizing the true positives to false negatives ratio. In other words, for the events that occurred, how many did we predict? Example: . **Objective: maximize**
* **AUC**: Area under the curve. A good classifier will have high precision and sensitivity. This means the classifier does well when it predicts an event will and will not occur, which minimizes false positives and false negatives. To capture this balance, we often use a ROC curve that plots the false positive rate along the x-axis and the true positive rate along the y-axis. A line that is diagonal from the lower left corner to the upper right corner represents a random guess. The higher the line is in the upper left-hand corner, the better. AUC computes the area under this curve. Objective: maximize

### Linear regression

Linear regression is a very simple approach for supervised learning.In particular, linear regression is a useful tool for predicting a quantitative response. Linear regression has been around for a long time and is the topic of innumerable textbooks. Though it may seem somewhat dull compared to some of the more modern statistical learning approaches described in later tutorials, linear regression is still a useful and widely used statistical learning method. Moreover, it serves as a good jumping-off point for newer approaches: as we will see in later tutorials, many fancy statistical learning approaches can be seen as generalizations or extensions of linear regression. Consequently, the importance of having a good understanding of linear regression before studying more complex learning methods cannot be overstated.

#### tl;dr

This tutorial serves asa an introduction to linear regression 1. [Replication requirements](#RR): What you will need to reproduce 2. Preparing our data: Prepare our data for modeling 3. Simple linear regression: Predicting a quantitative response with a single predictor variable 4. Multiple linear regression: Predicting a quantitative response with multiple predictor variables 5. Incorporating interactions: Removing the additive assumption 6. Additional considerations: A few other considerations to know about

##### Replication requirements

This tutorial primarily leverages this [advertising data](http://www-bcf.usc.edu/~gareth/ISL/Advertising.csv) provided by the authors of [an Introduction to Statistical Learning](http://www-bcf.usc.edu/~gareth/ISL/index.html). This is a simple data set that contains, in thousands of dolloars, TV, Radio, and Newspaper budgets for 200 different markets along with the Sales, in thousands of units, for each market. We will also use a few package that provide data manipulation, visualization, pipeline modeling functions, and model output tidying functions.

# Packages  
library(tidyverse) # data manipulation and visualizations  
library(modelr)# provides easy pipeline modeling functions  
library(broom) # helps to tidy up model outputs  
  
advertising <- read\_csv("http://www-bcf.usc.edu/~gareth/ISL/Advertising.csv") %>%  
 select(-X1)  
  
advertising  
## # A tibble: 200 x 4  
## TV radio newspaper sales  
## <dbl> <dbl> <dbl> <dbl>  
## 1 230. 37.8 69.2 22.1  
## 2 44.5 39.3 45.1 10.4  
## 3 17.2 45.9 69.3 9.3  
## 4 152. 41.3 58.5 18.5  
## 5 181. 10.8 58.4 12.9  
## 6 8.7 48.9 75 7.2  
## 7 57.5 32.8 23.5 11.8  
## 8 120. 19.6 11.6 13.2  
## 9 8.6 2.1 1 4.8  
## 10 200. 2.6 21.2 10.6  
## # ... with 190 more rows

##### Preparing our data

Initial descovery of relationships is usually done with a training set while a test set is used for evaluating whether the discovered relationships hold.More formally, a training set is a set of data used to discover potentially predictive relationships. A test set is a set of data used to assess the strength and utility of a predictive relationship. In a later tutorial we will cover more sophisticated ways for training, validating, and testing predictive models but for the time being we窶冤l use a conventional 60% / 40% split where we training our model on 60% of the data and then test the model performance on 40% of the data that is withheld.

set.seed(123)  
sample <- sample(c(TRUE, FALSE), nrow(advertising), replace=TRUE, prob=c(0.6,0.4))  
train <- advertising[sample, ]  
test <- advertising[!sample, ]

#### Simple linear regression

*Simple linear regression* lives up to its name; it is a very straightfoward approach for prediting a quantitative response on the basis of a single predictor variable . It assumes that there is approximately a linear relationship between and . Using our advertising data, suppose we wish to model the linear relationship between the TV budget and sales. We can write this as:

where; - represents sale - represents *TV adversiting budget* - is the intercept - is the coefficient representing the linear relationship - is mean-zero random error term

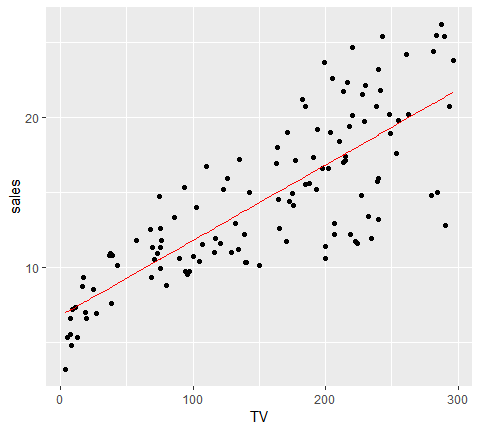
##### Model building

To build this model in R we use the formula notation of .

model1 <- lm(sales ~ TV, data = train)  
summary(model1)  
##   
## Call:  
## lm(formula = sales ~ TV, data = train)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -8.5816 -1.7845 -0.2533 2.1715 6.9345   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 6.764098 0.607592 11.13 <2e-16 \*\*\*  
## TV 0.050284 0.003463 14.52 <2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 3.204 on 120 degrees of freedom  
## Multiple R-squared: 0.6373, Adjusted R-squared: 0.6342   
## F-statistic: 210.8 on 1 and 120 DF, p-value: < 2.2e-16

In the background the lm, which stands for 窶徑inear model窶?, is producing the best-fit linear relationship by minimizing the least squares criterion (alternative approaches will be considered in later tutorials). This fit can be visualized in the following illustration where the 窶彙est-fit窶? line is found by minimizing the sum of squared errors (the errors are represented by the vertical black line segments).

ggplot(data = train, aes(TV, sales))+  
 geom\_point()+  
 geom\_line(aes(y=predict(model1,train)), col="red")



For initial assessment of our model we can use summary. This provides us with a host of information about our model, which we窶冤l walk through. Alternatively, you can also use glance(model1) to get a 窶徼idy窶? result output.

library(broom)  
model\_results <-   
 list(  
 tidy = broom::tidy(model1),  
 glance = glance(model1)  
 )  
  
model\_results[[1]]  
## # A tibble: 2 x 5  
## term estimate std.error statistic p.value  
## <chr> <dbl> <dbl> <dbl> <dbl>  
## 1 (Intercept) 6.76 0.608 11.1 3.31e-20  
## 2 TV 0.0503 0.00346 14.5 3.41e-28  
model\_results[[2]]  
## # A tibble: 1 x 11  
## r.squared adj.r.squared sigma statistic p.value df logLik AIC BIC  
## <dbl> <dbl> <dbl> <dbl> <dbl> <int> <dbl> <dbl> <dbl>  
## 1 0.637 0.634 3.20 211. 3.41e-28 2 -314. 634. 643.  
## # ... with 2 more variables: deviance <dbl>, df.residual <int>

##### Asessing coeficcients

Our original formula in Eq. (1) includes $\ﾎｲ\_0$ for our intercept coefficent and for our slope coefficient. If we look at our model results (here we use tidy to just print out a tidy version of our coefficent results) we see that our model takes the form of

#broom::tidy()  
model\_results[[1]]  
## # A tibble: 2 x 5  
## term estimate std.error statistic p.value  
## <chr> <dbl> <dbl> <dbl> <dbl>  
## 1 (Intercept) 6.76 0.608 11.1 3.31e-20  
## 2 TV 0.0503 0.00346 14.5 3.41e-28

In other words, our intercept estiamte is so when the TV advertising budget is zero, we can expect sales to be (remember we are operating in units of ). And for every increase in the TV advertising budget we expect the average increase in sales to be 50 units.

It窶冱 also important to understand if the these coefficients are statistically significant. In other words, can we state these coefficients are statistically different then 0? To do that we can start by assessing the standard error (SE). The SE for and are computed with:

where . We see that our model results provide the SE (noted as Std.error). We can use the SE to compute the 95% confidence internal for the coefficients.

To get this information in R, we can simply use:

confint(model1)  
## 2.5 % 97.5 %  
## (Intercept) 5.56110868 7.96708701  
## TV 0.04342678 0.05714057

Our results show us that our 95% confidence interval for (TV) is [.043, .057]. Thus, since zero is not in this interval we can conclude that as the TV advertising budget increases by $1,000 we can expect the sales to increase by 43-57 units. This is also supported by the *t-statistic* provided by our results, which are computed by

which measures he number of standard deviations that is away from 0. Thus a large t-statistic such as ours will produe a small p-value (a small p-value indicates that it is unlikely to observe such a substantial association between the predictor variable and the response due to chance). Thus, we can conclude that a relationship between TV advertising budget and sales exists.

##### Assessing model accuracy

Next, we want to understand the extent to which the model fits the data. This is typically referred to as the *goodness-of-fit*. We can measure this quantitatively by assessing three things:

1. Residual standard error
2. R squared()
3. F-statistic

The RSE is an estimate of the standard deviation of . Roughly speaking, it is the average amount that the response will deviate from the true regression line. It is computed by:

We get the RSE at the bottom of summary(model1), we can also get it directly with:

sigma(model1)  
## [1] 3.204129

An RSE value of means the actual sales in each market will deviate from the true regression line by approximately units, on average. Is this significant? Well, that窶冱 subjective but when compared to the average value of sales over all markets the percentage error is :

sigma(model1)/mean(train$sales)  
## [1] 0.2207373

The RSE provides an absolute measure of lack of fit of our model to the data. But since it is measured in the units of , it is not always clear what constitutes a good RSE. The statistic provides an alternative measure of fit. It represents the proportion of variance explained and so it always takes on a value between 0 and 1, and is independent of the scale of . is simply a function of residual sum of squares (RSS) and total sum of squares (TSS):

Similar to RSE the can be found at the bottom of summary(model1) but we can also generate it directly with rsquare. The result suggests that TV advertising budget can explain 64% of the variability in our sales data.

rsquare(model1, data = train)  
## [1] 0.6372581

As a side note, in a simple linear regression model the value will equal the squared correlation between and :

cor(train$TV, train$sales)^2  
## [1] 0.6372581

Lastly, the *F-statistic* tests to see if at least one predictor variable has a non-zero coefficient. This becomes more important once we start using multiple predictors as in multiple linear regression; however, we窶冤l introduce it here. The *F-statistic* is computed as:

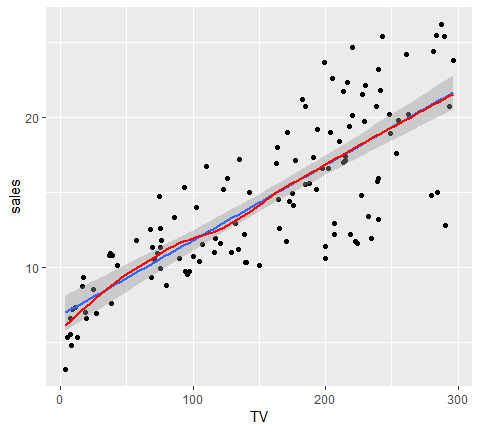
Hence, a larger F-statistic will produce a significant p-value (<0.05). In ou case, we see at the bottom of our summary statement that the F-statistic is 210.8 producing a p-value of .

Combined, our RSE, , and F-statistic results suggest that our model has an ok fit, but we could likely do better.

##### Asessing our model visually

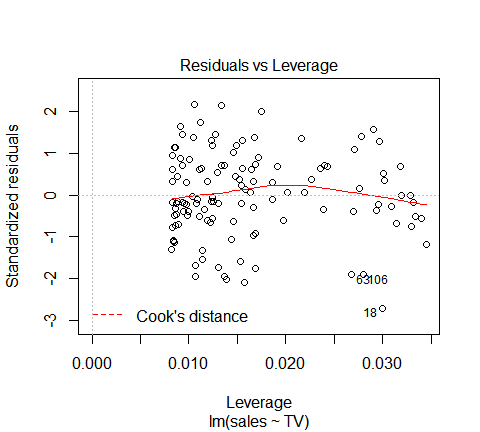
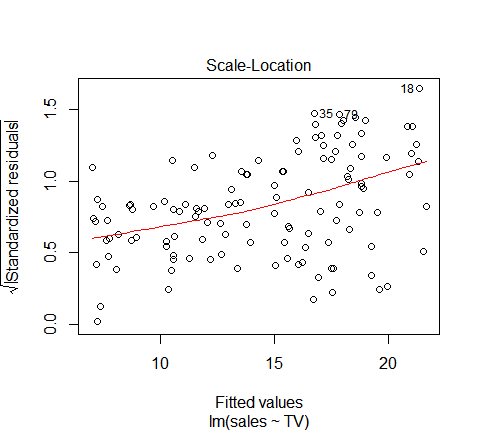
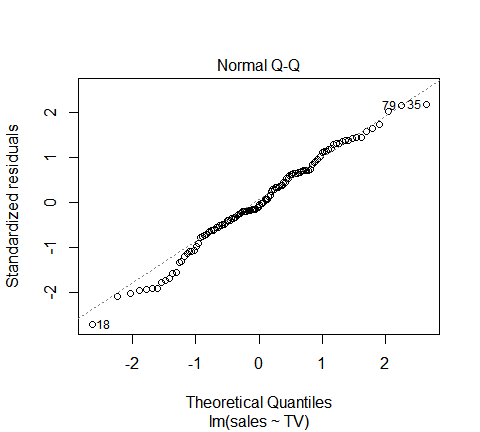
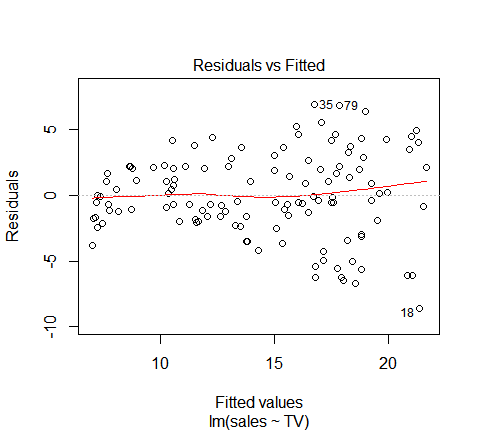
Not only is it important to to understand quantitative measures regarding our coefficient and model accuracy but we should also understand visual approaches to assess our model. First, we should always visualize our model within our data when possible. For simple linear regression this is quite simple. Here we use geom\_smooth(method = "lm") followed by geom\_smooth(). This allows us to compare the linearity of our model (blue line with the 95% confidence interval in shaded region) with a non-linear **LOESS model**. Considering the LOESS smoother remains within the confidence interval we can assume the linear trend fits the essence of this relationship. However, we do note that as the TV advertising budget gets closer to 0 there is a stronger reduction in sales beyond what the linear trend follows.

ggplot(train, aes(TV, sales))+  
 geom\_point()+  
 geom\_smooth(method = "lm")+  
 geom\_smooth(se=FALSE, color="red")



An important part of assessing regression models is visualizing residuals. If you use plot(model1) four residual plots will be produced that provide some insights. Here I will walk through creating each of these plots within ggplot and explain their insights.

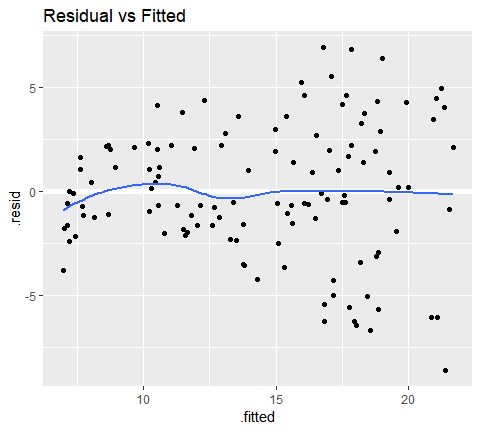
plot(model1)



First is a plot of residual versus fitter values. This will signal two important concerns:

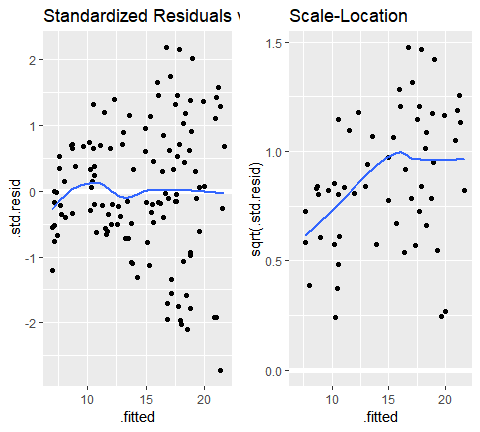
1. Non-linearity: if a discernible pattern (blue line) exists then this suggests either non-linearity or that other attributes have not been adequately captured. Our plot indicates that the assumption of linearity is fair.
2. Heteroskedasticity: an important assumption of linear regression is that the error terms have a constant variance, . If there is a funnel shape with our residuals, as in our plot, then we have violated this assumption. Sometimes this can be resolved with a log or square root transformation of in our model.

# add model diagnostics to our training data  
model1\_results <- augment(model1, train)  
model1\_results  
## # A tibble: 122 x 11  
## TV radio newspaper sales .fitted .se.fit .resid .hat .sigma  
## <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 230. 37.8 69.2 22.1 18.3 0.392 3.77 0.0149 3.20  
## 2 17.2 45.9 69.3 9.3 7.63 0.556 1.67 0.0301 3.21  
## 3 8.7 48.9 75 7.2 7.20 0.581 -0.00157 0.0329 3.22  
## 4 57.5 32.8 23.5 11.8 9.66 0.443 2.14 0.0191 3.21  
## 5 8.6 2.1 1 4.8 7.20 0.582 -2.40 0.0329 3.21  
## 6 200. 2.6 21.2 10.6 16.8 0.330 -6.21 0.0106 3.17  
## 7 215. 24 4 17.4 17.6 0.358 -0.160 0.0125 3.22  
## 8 97.5 7.6 7.2 9.7 11.7 0.350 -1.97 0.0119 3.21  
## 9 204. 32.9 46 19 17.0 0.338 1.97 0.0111 3.21  
## 10 67.8 36.6 114 12.5 10.2 0.417 2.33 0.0169 3.21  
## # ... with 112 more rows, and 2 more variables: .cooksd <dbl>,  
## # .std.resid <dbl>  
  
ggplot(model1\_results, aes(.fitted, .resid))+  
 geom\_ref\_line(h=0)+  
 geom\_point()+  
 geom\_smooth(se=FALSE)+  
 ggtitle("Residual vs Fitted")



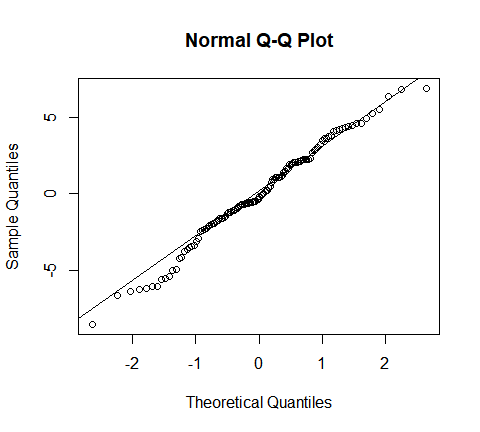
We can get this same kind of information with a couple other plots which you will see when using plot(model1). The first is comparing standardized residuals versus fitted values. This is the same plot as above but with the residuals standardized to show where residuals deviate by 1, 2, 3+ standard deviations. This helps us to identify outliers that exceed 3 standard deviations. The second is the scale-location plot. This plot shows if residuals are spread equally along the ranges of predictors. This is how you can check the assumption of equal variance (homoscedasticity). It窶冱 good if you see a horizontal line with equally (randomly) spread points.

p1 <- ggplot(model1\_results, aes(.fitted, .std.resid))+  
 geom\_ref\_line(h=0)+  
 geom\_point()+  
 geom\_smooth(se = FALSE) +  
 ggtitle("Standardized Residuals vs Fitted")  
  
p2 <- ggplot(model1\_results, aes(.fitted, sqrt(.std.resid))) +  
 geom\_ref\_line(h = 0) +  
 geom\_point() +  
 geom\_smooth(se = FALSE) +  
 ggtitle("Scale-Location")  
  
gridExtra::grid.arrange(p1, p2, nrow = 1)



The next plot assess the normality of our residuals. A Q-Q plot plots the distribution of our residuals against the theoretical normal distribution. The closer the points are to falling directly on the diagonal line then the more we can interpret the residuals as normally distributed. If there is strong snaking or deviations from the diagonal line then we should consider our residuals non-normally distributed. In our case we have a little deviation in the bottom left-hand side which likely is the concern we mentioned earlier that as the TV advertising budget approaches 0 the relationship with sales appears to start veering away from a linear relationship.

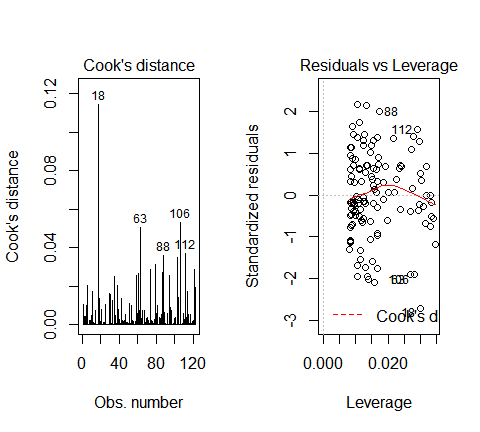
qq\_plot <- qqnorm(model1\_results$.resid)  
qq\_plot <- qqline(model1\_results$.resid)



Last are the Cook窶冱 Distance and residuals versus leverage plot. These plot helps us to find influential cases (i.e., subjects) if any. Not all outliers are influential in linear regression analysis. Even though data have extreme values, they might not be influential to determine a regression line. That means, the results wouldn窶冲 be much different if we either include or exclude them from analysis. They follow the trend in the majority of cases and they don窶冲 really matter; they are not influential. On the other hand, some cases could be very influential even if they look to be within a reasonable range of the values. They could be extreme cases against a regression line and can alter the results if we exclude them from analysis. Another way to put it is that they don窶冲 get along with the trend in the majority of the cases.

Here we are looking for outlying values (we can select the top n outliers to report with id.n. The identified (labeled) points represent those splots where cases can be influential against a regression line. When cases have high Cook窶冱 distance scores and are to the upper or lower right of our leverage plot they have leverage meaning they are influential to the regression results. The regression results will be altered if we exclude those cases.

par(mfrow=c(1, 2))  
  
plot(model1, which = 4, id.n = 5)  
plot(model1, which = 5, id.n = 5)



If you want to look at these top 5 observations with the highest Cook窶冱 distance in case you want to assess them further you can use the following.

model1\_results %>%   
 top\_n(5, wt=.cooksd)  
## # A tibble: 5 x 11  
## TV radio newspaper sales .fitted .se.fit .resid .hat .sigma .cooksd  
## <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 291. 4.1 8.5 12.8 21.4 0.555 -8.58 0.0300 3.12 0.114   
## 2 280. 10.1 21.4 14.8 20.9 0.524 -6.05 0.0268 3.17 0.0504  
## 3 243. 49 44.3 25.4 19.0 0.423 6.41 0.0175 3.16 0.0362  
## 4 284. 10.6 6.4 15 21.1 0.536 -6.06 0.0280 3.17 0.0530  
## 5 288. 43 71.8 26.2 21.2 0.546 4.97 0.0290 3.18 0.0371  
## # ... with 1 more variable: .std.resid <dbl>

So, what does having patterns in residuals mean to your research? It窶冱 not just a go-or-stop sign. It tells you about your model and data. Your current model might not be the best way to understand your data if there窶冱 so much good stuff left in the data.

In that case, you may want to go back to your theory and hypotheses. Is it really a linear relationship between the predictors and the outcome? You may want to include a quadratic term, for example. A log transformation may better represent the phenomena that you窶囘 like to model. Or, is there any important variable that you left out from your model? Other variables you didn窶冲 include (e.g., Radio or Newspaper advertising budgets) may play an important role in your model and data. Or, maybe, your data were systematically biased when collecting data. You may want to redesign data collection methods.

Checking residuals is a way to discover new insights in your model and data!

##### Making predictions

Often the goal with regression approaches is to make predictions on new data. To assess how well our model do in this endeavor we need to assess how it does in making predictions against our test data set. This informs us how well our model generalizes to data outside our training set. We can use our model to predict Sales values for our test data by using add\_predictions.

test <- test %>%   
 add\_predictions(model1)  
test  
## # A tibble: 78 x 5  
## TV radio newspaper sales pred  
## <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 44.5 39.3 45.1 10.4 9.00  
## 2 152. 41.3 58.5 18.5 14.4   
## 3 181. 10.8 58.4 12.9 15.9   
## 4 120. 19.6 11.6 13.2 12.8   
## 5 66.1 5.8 24.2 8.6 10.1   
## 6 23.8 35.1 65.9 9.2 7.96  
## 7 195. 47.7 52.9 22.4 16.6   
## 8 147. 23.9 19.1 14.6 14.2   
## 9 218. 27.7 53.4 18 17.7   
## 10 237. 5.1 23.5 12.5 18.7   
## # ... with 68 more rows

The primary concern is to assess if the **out-of-sample** mean squared error (MSE), also known as the mean squared prediction error, is substantially higher than the in-sample mean square error, as this is a sign of deficiency in the model. The MSE is computed as

We can easily compare the test sample MSE to the training sample MSE with the following.The difference is not that significant. However, this practice becomes more powerful when you are comparing multiple models. For example, if you developed a simple linear model with just the Radio advertising budget as the predictor variable, you could then compare our two different simple linear models and the one that produces the lowest test sample MSE is the preferred model.

# test MSE  
test %>%   
 add\_predictions(model1) %>%   
 summarise(MSE = mean((sales-pred)^2))  
## # A tibble: 1 x 1  
## MSE  
## <dbl>  
## 1 11.3  
  
# training MSE  
train %>%   
 add\_predictions(model1) %>%  
 summarise(MSE = mean((sales - pred)^2))  
## # A tibble: 1 x 1  
## MSE  
## <dbl>  
## 1 10.1  
## # A tibble: 1 ﾃ? 1  
## MSE  
## <dbl>  
## 1 10.09814

In the next tutorial we will look at how we can extend a simple linear regression model into a multiple regression.

#### Multiple regression

Simple linear regression is a useful approach for predicting a response on the basis of a single predictor variable. However, in practice we often have more than one predictor. For example, in the Advertising data, we have examined the relationship between sales and TV advertising. We also have data for the amount of money spent advertising on the radio and in newspapers, and we may want to know whether either of these two media is associated with sales. How can we extend our analysis of the advertising data in order to accommodate these two additional predictors?

We can extend the simple linear regression model so that it can directly accommodate multiple predictors. We can do this by giving each predictor a separate slope coefficient in a single model. In general, suppose that we have p distinct predictors. Then the multiple linear regression model takes the form

##### Model building

If we want to run a model that uses TV, Radio, and Newspaper to predict Sales then we build this model in R using a similar approach introduced in the Simple Linear Regression tutorial.

model2 <- lm(sales ~ TV + radio + newspaper, data=train)

We can also assess this model as before:

summary(model2)  
##   
## Call:  
## lm(formula = sales ~ TV + radio + newspaper, data = train)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -4.8426 -0.6466 0.2165 1.0640 2.6804   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 2.822206 0.369369 7.641 6.29e-12 \*\*\*  
## TV 0.047362 0.001657 28.577 < 2e-16 \*\*\*  
## radio 0.196375 0.010347 18.979 < 2e-16 \*\*\*  
## newspaper -0.010593 0.006460 -1.640 0.104   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 1.527 on 118 degrees of freedom  
## Multiple R-squared: 0.9189, Adjusted R-squared: 0.9169   
## F-statistic: 445.9 on 3 and 118 DF, p-value: < 2.2e-16

##### Assessing coefficients

The interpreation of our coefficients is the same as in a linear regression model. First, we see that our coefficients for TV and Radio advertising budget are statistically significant (p-value < 0.05) while the coefficient for Newspaper is not. Thus, changes in Newspaper budget do not appear to have a relationship with changes in sales. However, for TV our coefficent suggests that for every increase in TV advertising budget, holding all other predictors constant, we can expect an increase of 47 sales units, on average (this is similar to what we found in the simple linear regression). The Radio coefficient suggests that for every increase in Radio advertising budget, holding all other predictors constant, we can expect an increase of 196 sales units, on average.

broom::tidy(model2)  
## # A tibble: 4 x 5  
## term estimate std.error statistic p.value  
## <chr> <dbl> <dbl> <dbl> <dbl>  
## 1 (Intercept) 2.82 0.369 7.64 6.29e-12  
## 2 TV 0.0474 0.00166 28.6 7.36e-55  
## 3 radio 0.196 0.0103 19.0 1.17e-37  
## 4 newspaper -0.0106 0.00646 -1.64 1.04e- 1

We can also get our confidence intervals around these coefficient estimates as we did before. Here we see how the confidence interval for Newspaper includes 0 which suggests that we cannot assume the coefficient estimate of -0.0106 is different than 0.

confint(model2)  
## 2.5 % 97.5 %  
## (Intercept) 2.09075443 3.553657581  
## TV 0.04408008 0.050644109  
## radio 0.17588568 0.216864467  
## newspaper -0.02338577 0.002200661

##### Assessing model accuracy

Assessing model accuracy is very similar as when assessing simple linear regression models. Rather than repeat the discussion, here I will highlight a few key considerations. First, multiple regression is when the F-statistic becomes more important as this statistic is testing to see if at least one of the coefficients is non-zero. When there is no relationship between the response and predictors, we expect the F-statistic to take on a value close to 1. On the other hand, if at least predictor has a relationship then we expect . In our summary print out above for model 2 we saw that with suggesting that at least one of the advertising media must be related to sales.

In addition, if we compare the results from our simple linea regression model (model1) and our multiple regression model (model2) we can make some important comparisons:

list(  
 model1 = broom::glance(model1),  
 model2 = broom::glance(model2)  
)  
## $model1  
## # A tibble: 1 x 11  
## r.squared adj.r.squared sigma statistic p.value df logLik AIC BIC  
## <dbl> <dbl> <dbl> <dbl> <dbl> <int> <dbl> <dbl> <dbl>  
## 1 0.637 0.634 3.20 211. 3.41e-28 2 -314. 634. 643.  
## # ... with 2 more variables: deviance <dbl>, df.residual <int>  
##   
## $model2  
## # A tibble: 1 x 11  
## r.squared adj.r.squared sigma statistic p.value df logLik AIC BIC  
## <dbl> <dbl> <dbl> <dbl> <dbl> <int> <dbl> <dbl> <dbl>  
## 1 0.919 0.917 1.53 446. 3.49e-64 4 -223. 456. 470.  
## # ... with 2 more variables: deviance <dbl>, df.residual <int>

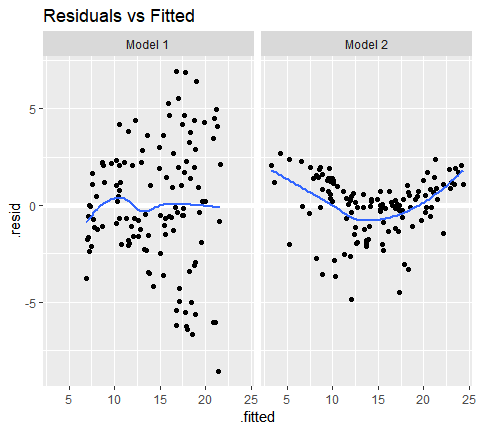
1. : Model 2窶冱 is substantially higher than model 1 suggesting that model 2 does a better job explaining the variance in sales. It窶冱 also important to consider the adjusted . The adjusted is a modified version of that has been adjusted for the number of predictors in the model. The adjusted increases only if the new term improves the model more than would be expected by chance. Thus, since model 2窶冱 adjusted is also substantially higher than model 1 we confirm that the additional predictors are improving the model窶冱 performance.
2. **RSE**: Model 2窶冱 RSE (sigma) is lower than model 1. This shows that model 2 reduces the variance of our parameter which corroborates our conclusion that model 2 does a better job modeling sales.
3. **F-statistic**: the F-statistic (statistic) in model 2 is larger than model 1. Here larger is better and suggests that model 2 provides a better 窶徃oodness-of-fit窶?.
4. **Other**: We can also use other various statistics to compare the quality of our models. These include Akaike information criterion (AIC) and Bayesian information criterion (BIC), which we see in our results, among others. We窶冤l go into more details regarding these statistics in the Linear Model Selection tutorial but for now just know that models with lower AIC and BIC values are considered of better quality than models with higher values.

So we understand that quantitative attributes of our second model suggest it is a better fit, how about visually?

##### Assessing our model visually

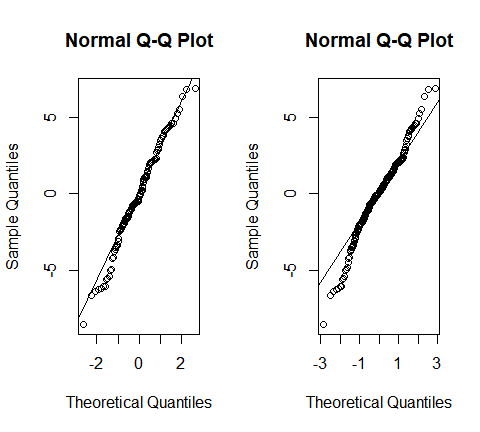
Our main focus is to assess and compare residual behavior with our models. First, if we compare model 2窶冱 residuals versus fitted values we see that model 2 has reduced concerns with heteroskedasticity; however, we now have discernible patter suggesting concerns of linearity. We窶冤l see one way to address this in the next section.

# add model diagnostics to our training data  
model1\_results <- model1\_results %>%  
 mutate(Model = "Model 1")  
  
model2\_results <- augment(model2, train) %>%  
 mutate(Model = "Model 2") %>%  
 rbind(model1\_results)  
  
ggplot(model2\_results, aes(.fitted, .resid)) +  
 geom\_ref\_line(h = 0) +  
 geom\_point() +  
 geom\_smooth(se = FALSE) +  
 facet\_wrap(~ Model) +  
 ggtitle("Residuals vs Fitted")



This concern with normality is supported when we compare the Q-Q plots. So although our model is performing better numerically, we now have a greater concern with normality then we did before! This is why we must always assess models numerically and visually!

par(mfrow=c(1, 2))  
  
# Left: model 1  
qqnorm(model1\_results$.resid); qqline(model1\_results$.resid)  
  
# Right: model 2  
qqnorm(model2\_results$.resid); qqline(model2\_results$.resid)



##### Making predictions

To see how our models compare when making predictions on an out-of-sample data set we窶冤l compare MSE. Here we can use gather\_predictions to predict on our test data with both models and then, as before, compute the MSE. Here we see that model 2 drastically reduces MSE on the out-of-sample. So although we still have lingering concerns over residual normality model 2 is still the preferred model so far.

test %>%  
 gather\_predictions(model1, model2) %>%  
 group\_by(model) %>%  
 summarise(MSE = mean((sales-pred)^2))  
## # A tibble: 2 x 2  
## model MSE  
## <chr> <dbl>  
## 1 model1 11.3   
## 2 model2 3.75  
## # A tibble: 2 ﾃ? 2  
## model MSE  
## <chr> <dbl>  
## 1 model1 11.34993  
## 2 model2 3.75494

#### Incorporating interactions

In our previous analysis of the Advertising data, we concluded that both TV and radio seem to be associated with sales. The linear models that formed the basis for this conclusion assumed that the effect on sales of increasing one advertising medium is independent of the amount spent on the other media. For example, the linear model (Eq. 10) states that the average effect on sales of a one-unit increase in TV is always , regardless of the amount spent on radio.

However, this simple model may be incorrect. Suppose that spending money on radio advertising actually increases the effectiveness of TV advertising, so that the slope term for TV should increase as radio increases. In this situation, given a fixed budget of $100,000, spending half on radio and half on TV may increase sales more than allocating the entire amount to either TV or to radio. In marketing, this is known as a synergy effect, and in statistics it is referred to as an interaction effect. One way of extending our model 2 to allow for interaction effects is to include a third predictor, called an interaction term, which is constructed by computing the product of $X\_1, and (here we窶冤l drop the Newspaper variable). This results in the model

# Option 1  
model3 <- lm(sales~TV+radio+TV\*radio, data=train)  
  
# Option 2  
model3 <- lm(sales ~ TV \* radio, data = train)

##### Asessing coefficients

We see that all our coefficients are statistically significant. Now we can interpret this as an increase in TV advertising of \_1\_3$*radio)*  1000 = 21 + 1\*radio. And an increase in radio advertising of $1,000 will be associated with an increase in sales of

broom::tidy(model3)  
## # A tibble: 4 x 5  
## term estimate std.error statistic p.value  
## <chr> <dbl> <dbl> <dbl> <dbl>  
## 1 (Intercept) 6.50 0.308 21.1 7.25e-42  
## 2 TV 0.0208 0.00188 11.1 5.25e-20  
## 3 radio 0.0390 0.0106 3.69 3.44e- 4  
## 4 TV:radio 0.00101 0.0000621 16.3 4.47e-32

##### Assessing model accuracy

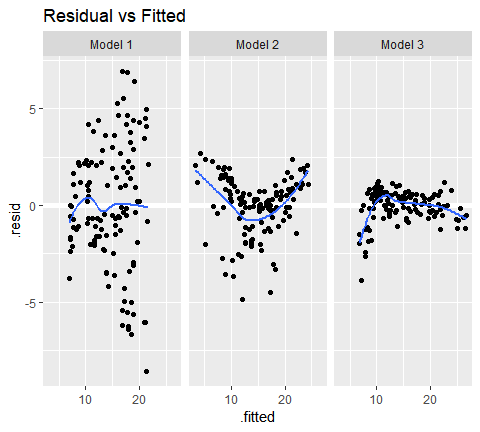
We can compare our model results across all three models. We see that our adjusted and F-statistic are highest with model 3 and our RSE, AIC and BIC are the lowest with model 3; suggesting the model 3 outperforms the other models.

list(model1 = broom::glance(model1),  
 model2 = broom::glance(model2),  
 model3 = broom::glance(model3))   
## $model1  
## # A tibble: 1 x 11  
## r.squared adj.r.squared sigma statistic p.value df logLik AIC BIC  
## <dbl> <dbl> <dbl> <dbl> <dbl> <int> <dbl> <dbl> <dbl>  
## 1 0.637 0.634 3.20 211. 3.41e-28 2 -314. 634. 643.  
## # ... with 2 more variables: deviance <dbl>, df.residual <int>  
##   
## $model2  
## # A tibble: 1 x 11  
## r.squared adj.r.squared sigma statistic p.value df logLik AIC BIC  
## <dbl> <dbl> <dbl> <dbl> <dbl> <int> <dbl> <dbl> <dbl>  
## 1 0.919 0.917 1.53 446. 3.49e-64 4 -223. 456. 470.  
## # ... with 2 more variables: deviance <dbl>, df.residual <int>  
##   
## $model3  
## # A tibble: 1 x 11  
## r.squared adj.r.squared sigma statistic p.value df logLik AIC BIC  
## <dbl> <dbl> <dbl> <dbl> <dbl> <int> <dbl> <dbl> <dbl>  
## 1 0.975 0.974 0.855 1508. 6.91e-94 4 -152. 314. 328.  
## # ... with 2 more variables: deviance <dbl>, df.residual <int>

##### Assessing our model visually

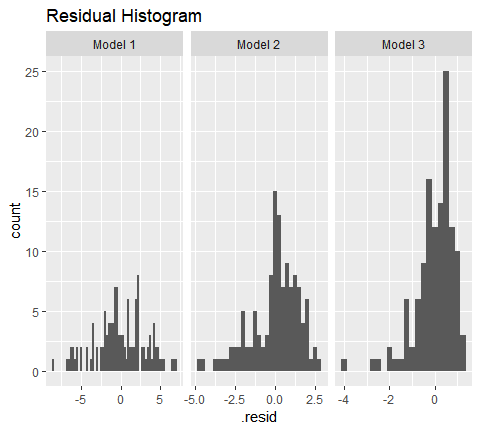
Visually assessing our residuals versus fitted values we see that model three does a better job with constant variance and, with the exception of the far left side, does not have any major signs of non-normality.

# add model diagnostics to our training data  
  
model3\_results <- augment(model3, train) %>%   
 mutate(Model = "Model 3") %>%   
 rbind(model2\_results)  
  
ggplot(model3\_results, aes(.fitted, .resid))+  
 geom\_ref\_line(h=0)+  
 geom\_point()+  
 geom\_smooth(se=FALSE)+  
 facet\_wrap(~Model)+  
 ggtitle("Residual vs Fitted")



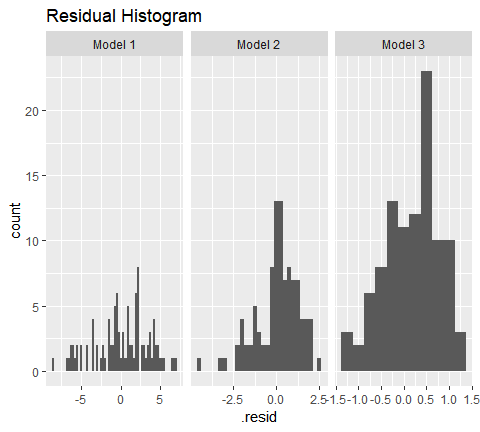
As an alternative to the Q-Q plot we can also look at residual histograms for each model. Here we see that model 3 has a couple large left tail residuals. These are related to the left tail dip we saw in the above plots.

ggplot(model3\_results, aes(.resid))+  
 geom\_histogram(binwidth = .25)+  
 facet\_wrap(~Model, scales="free\_x")+  
 ggtitle("Residual Histogram")



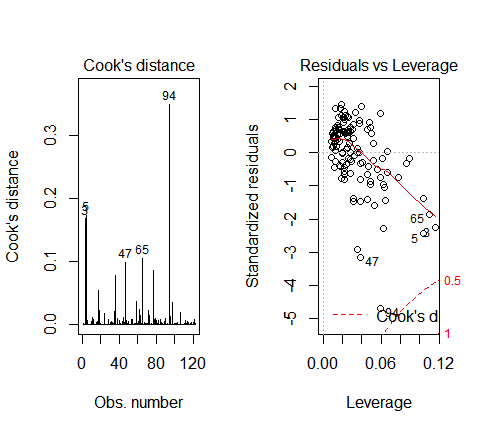
These residuals can be tied back to when our model is trying to predict low levels of sales (< 10,000). If we remove these sales our residuals are more normally distributed. What does this mean? Basically our linear model does a good job predicting sales over 10,000 units based on TV and Radio advertising budgets; however, the performance deteriates when trying to predict sales less than 10,000 because our linear assumption does not hold for this segment of our data.

model3\_results %>%  
 filter(sales > 10) %>%  
 ggplot(aes(.resid)) +  
 geom\_histogram(binwidth = .25) +  
 facet\_wrap(~ Model, scales = "free\_x") +  
 ggtitle("Residual Histogram")



This can be corroborated by looking at the Cook窶冱 Distance and Leverage plots. Both of them highlight observations 3, 5, 47, 65, and 94 as the top 5 influential observations.

par(mfrow=c(1, 2))  
  
plot(model3, which = 4, id.n = 5)  
plot(model3, which = 5, id.n = 5)



If we look at these observations we see that they all have low Sales levels.

train[c(3,5,47,65,94),]  
## # A tibble: 5 x 4  
## TV radio newspaper sales  
## <dbl> <dbl> <dbl> <dbl>  
## 1 8.7 48.9 75 7.2  
## 2 8.6 2.1 1 4.8  
## 3 5.4 29.9 9.4 5.3  
## 4 13.1 0.4 25.6 5.3  
## 5 4.1 11.6 5.7 3.2

##### Making prediction

Again, to see how our models compare when making predictions on an out-of-sample data set we窶冤l compare the MSEs across all our models. Here we see that model 3 has the lowest out-of-sample MSE, further supporting the case that it is the best model and has not overfit our data.

test %>%   
 gather\_predictions(model1, model2,model3) %>%   
 group\_by(model) %>%   
 summarise(MSE = mean((sales-pred)^2))  
## # A tibble: 3 x 2  
## model MSE  
## <chr> <dbl>  
## 1 model1 11.3   
## 2 model2 3.75  
## 3 model3 1.15

#### Additional consideration

##### Qualitative predictors

In our discussion so far, we have assumed that all variables in our linear regression model are *quantitative*. But in practice, this is not necessarily the case; often some predictors are *qualitative*.

For example, the [Credit data set](http://www-bcf.usc.edu/~gareth/ISL/Credit.csv) records balance (average credit card debt for a number of individuals) as well as several quantitative predictors: age, cards (number of credit cards), education (years of education), income (in thousands of dollars), limit (credit limit), and rating (credit rating).

Suppose that we wish to investigate differences in credit card balance between males and females, ignoring the other variables for the moment. If a qualitative predictor (also known as a factor) only has two levels, or possible values, then incorporating it into a regression model is very simple. We simply create an indicator or dummy variable that takes on two possible numerical values. For example, based on the gender, we can create a new variable that takes the form

and use this variable as a predictor in the regression equation. This results in the model

Now can be interpreted as the average credit card balance among males, as the average credit card balance among females, and as the average difference in credit card balance between females and males. We can produce this model in R using the same syntax as we saw earlier:

# credit <- read\_csv("http://www-bcf.usc.edu/~gareth/ISL/Credit.csv")  
credit <- ISLR::Credit  
credit %>% head()  
## ID Income Limit Rating Cards Age Education Gender Student Married  
## 1 1 14.891 3606 283 2 34 11 Male No Yes  
## 2 2 106.025 6645 483 3 82 15 Female Yes Yes  
## 3 3 104.593 7075 514 4 71 11 Male No No  
## 4 4 148.924 9504 681 3 36 11 Female No No  
## 5 5 55.882 4897 357 2 68 16 Male No Yes  
## 6 6 80.180 8047 569 4 77 10 Male No No  
## Ethnicity Balance  
## 1 Caucasian 333  
## 2 Asian 903  
## 3 Asian 580  
## 4 Asian 964  
## 5 Caucasian 331  
## 6 Caucasian 1151  
model4 <- lm(Balance ~ Gender, data=credit)

The results below suggest that females are estimated to carry in credit card debt where males carry

broom::tidy(model4)  
## # A tibble: 2 x 5  
## term estimate std.error statistic p.value  
## <chr> <dbl> <dbl> <dbl> <dbl>  
## 1 (Intercept) 510. 33.1 15.4 2.91e-42  
## 2 GenderFemale 19.7 46.1 0.429 6.69e- 1

The decision to code females as 0 and makes as 1 in is arbitrary, and has no effect on the regression fit, but does alter the interpretation of the coefficients. If we want to change the reference variable (the variable coded as 0) we can change the factor levels.

# credit$Gender <- factor(credit$Gender, levels = c("Male", "Female"))  
  
lm(Balance ~ Gender, data = credit) %>%  
 tidy()  
## # A tibble: 2 x 5  
## term estimate std.error statistic p.value  
## <chr> <dbl> <dbl> <dbl> <dbl>  
## 1 (Intercept) 510. 33.1 15.4 2.91e-42  
## 2 GenderFemale 19.7 46.1 0.429 6.69e- 1

A similar process ensues for qualitative predictor categories with more than two levels. For instance, if we want to assess the impact that ethnicity has on credit balance we can run the following model. Ethnicity has three levels: *African American, Asian, Caucasian*. We interpret the coefficients much the same way. In this case we see that the estimated balance for the baseline, African American, is $531.00. It is estimated that the Asian category will have $18.69 less debt than the African American category, and that the Caucasian category will have $12.50 less debt than the African American category. However, the p-values associated with the coefficient estimates for the two dummy variables are very large, suggesting no statistical evidence of a real difference in credit card balance between the ethnicities.

lm(Balance ~ Ethnicity, data = credit) %>%   
 broom::tidy()  
## # A tibble: 3 x 5  
## term estimate std.error statistic p.value  
## <chr> <dbl> <dbl> <dbl> <dbl>  
## 1 (Intercept) 531. 46.3 11.5 1.77e-26  
## 2 EthnicityAsian -18.7 65.0 -0.287 7.74e- 1  
## 3 EthnicityCaucasian -12.5 56.7 -0.221 8.26e- 1

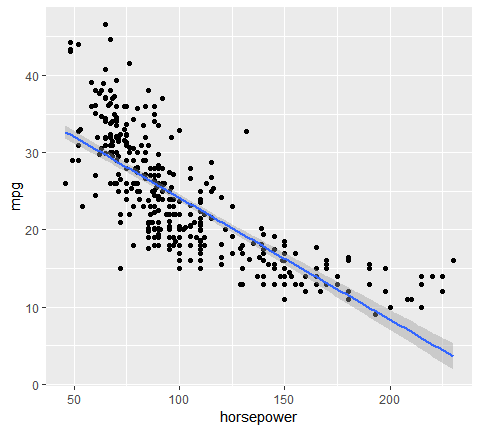
The process for assessing model accuracy, both numerically and visually along with measuring predictions can follow the same process as outlined for qualitative predictor variables.

##### Transformations

Linear regression models assume a linear relationship between the response and predictors. But in some cases, the true relationship between the response and the predictors may be non-linear. We can accomodate certain non-linear relationships by transforming variables (i.e. log(x), sqrt(x)) or using polynomial regression.

As an example consider the Auto data set. We can see that a linear trend does not fit the relationship between mpg and horsepower.

auto <- ISLR::Auto  
ggplot(auto, aes(horsepower, mpg)) +  
 geom\_point() +  
 geom\_smooth(method = "lm")



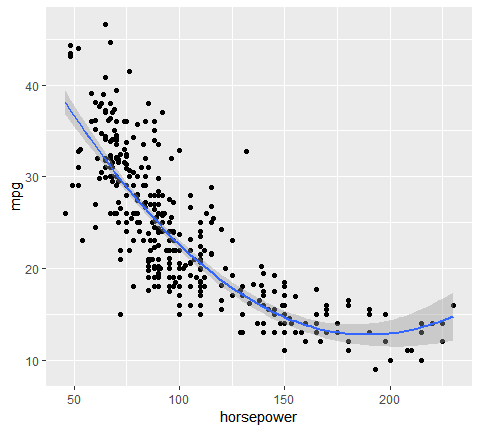
We can try to address the non-linear relationship with a quadratic relationship, which takes the form of:

We can fit this model in R with:

model5 <- lm(mpg ~ horsepower + I(horsepower^2), data = auto)  
broom::tidy(model5)  
## # A tibble: 3 x 5  
## term estimate std.error statistic p.value  
## <chr> <dbl> <dbl> <dbl> <dbl>  
## 1 (Intercept) 56.9 1.80 31.6 1.74e-109  
## 2 horsepower -0.466 0.0311 -15.0 2.29e- 40  
## 3 I(horsepower^2) 0.00123 0.000122 10.1 2.20e- 21

Does this fit our relationship better? We can visualize it with:

ggplot(auto, aes(horsepower, mpg)) +  
 geom\_point() +  
 geom\_smooth(method = "lm", formula = y ~ x + I(x^2))

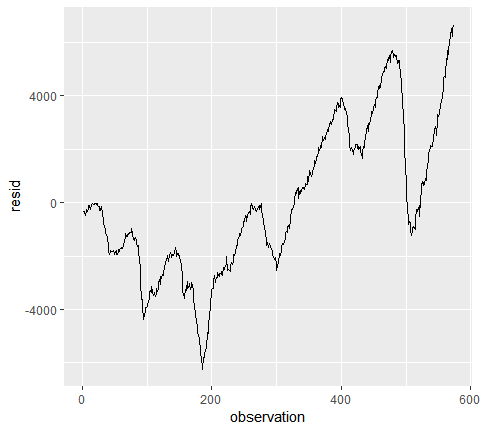


##### Correlation of error terms

An important assumption of the linear regression model is that the error terms, , are uncorrelated. Correlated residuals frequently occur in the context of time series data, which consists of observations for which measurements are obtained at discrete points in time. In many cases, observations that are obtained at adjacent time points will have positively correlated errors. This will result in biased standard errors and incorrect inference of model results.

To illustrate, we窶冤l create a model that uses the number of unemployed to predict personal consumption expenditures (using the economics data frame provided by ggplot2). The assumption is that as more people become unemployed personal consumption is likely to reduce. However, if we look at our model窶冱 residuals we see that adjacent residuals tend to take on similar values. In fact, these residuals have a .998 autocorrelation. This is a clear violation of our assumption. We窶冤l learn how to deal with correlated residuals in future tutorials.

df <- economics %>%   
 mutate(observation = 1:n())  
  
model6 <- lm(pce ~ unemploy, data = df)  
  
df %>%   
 add\_residuals(model6) %>%   
 ggplot(aes(observation, resid))+  
 geom\_line()



##### Collinearity

*Collinearity* refers to the situation in which two or more predictor variables are closely related to one another. The presence of collinearity can pose problems in the regression context, since it can be difficult to separate out the individual effects of collinear variables on the response. In fact, collinearity can cause predictor variables to appear as statistically insignificant when in fact they are significant.

For example, compares the coefficient estimates obtained from two separate multiple regression models. The first is a regression of balance on age and limit, and the second is a regression of balance on rating and limit. In the first regression, both age and limit are highly significant with very small p- values. In the second, the collinearity between limit and rating has caused the standard error for the limit coefficient estimate to increase by a factor of 12 and the p-value to increase to 0.701. In other words, the importance of the limit variable has been masked due to the presence of collinearity.

model7 <- lm(Balance ~ Age + Limit, data = credit)  
model8 <- lm(Balance ~ Rating + Limit, data = credit)  
  
list(`Model1` = broom::tidy(model7),  
 `Model2` = broom::tidy(model8))  
## $Model1  
## # A tibble: 3 x 5  
## term estimate std.error statistic p.value  
## <chr> <dbl> <dbl> <dbl> <dbl>  
## 1 (Intercept) -173. 43.8 -3.96 9.01e- 5  
## 2 Age -2.29 0.672 -3.41 7.23e- 4  
## 3 Limit 0.173 0.00503 34.5 1.63e-121  
##   
## $Model2  
## # A tibble: 3 x 5  
## term estimate std.error statistic p.value  
## <chr> <dbl> <dbl> <dbl> <dbl>  
## 1 (Intercept) -378. 45.3 -8.34 1.21e-15  
## 2 Rating 2.20 0.952 2.31 2.13e- 2  
## 3 Limit 0.0245 0.0638 0.384 7.01e- 1

A simple way to detect collinearity is to look at the correlation matrix of the predictors. An element of this matrix that is large in absolute value indicates a pair of highly correlated variables, and therefore a collinearity problem in the data. Unfortunately, not all collinearity problems can be detected by inspection of the correlation matrix: it is possible for collinear- ity to exist between three or more variables even if no pair of variables has a particularly high correlation. We call this situation multicollinearity.

Instead of inspecting the correlation matrix, a better way to assess multi- collinearity is to compute the *variance inflation factor (VIF)*. The VIF is the ratio of the variance of when fitting the full model divided by the variance of if fit on its own. The smallest possible value for VIF is 1, which indicates the complete absence of collinearity. Typically in practice there is a small amount of collinearity among the predictors. As a rule of thumb, a VIF value that exceeds 5 or 10 indicates a problematic amount of collinearity. The VIF for each variable can be computed using the formula

where is the from a regression of onto all of the other predictors. We can use the vif function from the car package to compute the VIF. As we see below model 7 is near the smallest possible VIF value where model 8 has obvious concerns.

car::vif(model7)  
## Age Limit   
## 1.010283 1.010283  
## Age Limit   
## 1.010283 1.010283  
car::vif(model8)  
## Rating Limit   
## 160.4933 160.4933  
## Rating Limit   
## 160.4933 160.4933

### Linear model selection

It is often the case that some or many of the variables used in a multiple regression model are in fact not associated with the response variable. Including such irrelevant variables leads to unnecessary complexity in the resulting model. Unfortunately, manually filtering through and comparing regression models can be tedious. Luckily, several approaches exist for automatically performing feature selection or variable selection 窶? that is, for identifying those variables that result in superior regression results. This tutorial will cover a traditional approach known as model selection.

#### tl;dr

This tutorial serves as an introduction to linear model selection and covers:

1. [Replication requrements: What you will need to reproduce the analysis in this tutorial](#replication_requirements)
2. Best subset selection: Finding the best combination of the p predictors
3. Stepwise selection: Computationally efficient approach for feature selection
4. Comparing models: Determining which model is best
5. Additional resources: additional resources to help you learn more

#### Replication requirements

This tutorial primary leverages the Hitters data provided by the ISLR package. This is a data set that contains number of hits, homeruns, RBIs, and other information for 322 major league baseball players. We窶冤l also use tidyverse for some basic data manipulation and visualization. Most importantly, we窶冤l use the leaps package to illustrate subset selection methods.

# package  
library(leaps) # model selection functions  
  
# load data and remove rows with missing data  
(hitters <- na.omit(ISLR::Hitters) %>%   
 as\_tibble())  
## # A tibble: 263 x 20  
## AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun CRuns  
## <int> <int> <int> <int> <int> <int> <int> <int> <int> <int> <int>  
## 1 315 81 7 24 38 39 14 3449 835 69 321  
## 2 479 130 18 66 72 76 3 1624 457 63 224  
## 3 496 141 20 65 78 37 11 5628 1575 225 828  
## 4 321 87 10 39 42 30 2 396 101 12 48  
## 5 594 169 4 74 51 35 11 4408 1133 19 501  
## 6 185 37 1 23 8 21 2 214 42 1 30  
## 7 298 73 0 24 24 7 3 509 108 0 41  
## 8 323 81 6 26 32 8 2 341 86 6 32  
## 9 401 92 17 49 66 65 13 5206 1332 253 784  
## 10 574 159 21 107 75 59 10 4631 1300 90 702  
## # ... with 253 more rows, and 9 more variables: CRBI <int>, CWalks <int>,  
## # League <fct>, Division <fct>, PutOuts <int>, Assists <int>,  
## # Errors <int>, Salary <dbl>, NewLeague <fct>

#### Best subset selection

To perform best subset selection, we fit a separate least squares regression for each possible combination of the *p* predictors. That is, we fit all *p* models that contain exactly one predictor, all

= p(p-1)/2 models that contain exactly two predictors, and so forth.

We then look at all of the resulting models, with the goal of identifying the one that is best.

The three-stage process of perfoming best subset selection includes:

**step1**: LEt denote the null model, which contains no predictors. This model simply predicts the sample mean for each observation.

**Step2**: For k = 1,2,…,p:

* Fit all (p k) models that contain exactly k predictors
* Pick the best among these (p k) models, and call it . Here best is defined as having the smallest RSS, or equivalently largest

**Step3**: Select a single best model from among using cross-validated prediction error, , AIC, BIC or adjusted .

Let’s illustrate with our data. We can perform a best subset search using regsubsets (part of the leaps library), which identifies the best model for a given number of *k* predictors, where best is quantified using RSS.

The syntax is the same as the lm function. By default, regsubsets only reports results up to the best eight-variable model. But the nvmax option can be used in order to return as many variables as are desired. Here we fit up to a 19-variable model.

best\_subset <- regsubsets(Salary ~ ., hitters, nvmax = 19)

The resubsets function returns a list-object with lots of information. Initially, we can use the summary command to assess the best set of variables for each model size. So, for a model with 1 variable we see that CRBI has an asterisk signalling that a regression model with *Salary ~ CRBI* is the best single variable model. The best 2 variable model is *Salary ~ CRBI + Hits*. The best 3 variable model is *Salary ~ CRBI + Hits + PutOuts*. An so forth.

summary(best\_subset)  
## Subset selection object  
## Call: regsubsets.formula(Salary ~ ., hitters, nvmax = 19)  
## 19 Variables (and intercept)  
## Forced in Forced out  
## AtBat FALSE FALSE  
## Hits FALSE FALSE  
## HmRun FALSE FALSE  
## Runs FALSE FALSE  
## RBI FALSE FALSE  
## Walks FALSE FALSE  
## Years FALSE FALSE  
## CAtBat FALSE FALSE  
## CHits FALSE FALSE  
## CHmRun FALSE FALSE  
## CRuns FALSE FALSE  
## CRBI FALSE FALSE  
## CWalks FALSE FALSE  
## LeagueN FALSE FALSE  
## DivisionW FALSE FALSE  
## PutOuts FALSE FALSE  
## Assists FALSE FALSE  
## Errors FALSE FALSE  
## NewLeagueN FALSE FALSE  
## 1 subsets of each size up to 19  
## Selection Algorithm: exhaustive  
## AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun CRuns  
## 1 ( 1 ) " " " " " " " " " " " " " " " " " " " " " "   
## 2 ( 1 ) " " "\*" " " " " " " " " " " " " " " " " " "   
## 3 ( 1 ) " " "\*" " " " " " " " " " " " " " " " " " "   
## 4 ( 1 ) " " "\*" " " " " " " " " " " " " " " " " " "   
## 5 ( 1 ) "\*" "\*" " " " " " " " " " " " " " " " " " "   
## 6 ( 1 ) "\*" "\*" " " " " " " "\*" " " " " " " " " " "   
## 7 ( 1 ) " " "\*" " " " " " " "\*" " " "\*" "\*" "\*" " "   
## 8 ( 1 ) "\*" "\*" " " " " " " "\*" " " " " " " "\*" "\*"   
## 9 ( 1 ) "\*" "\*" " " " " " " "\*" " " "\*" " " " " "\*"   
## 10 ( 1 ) "\*" "\*" " " " " " " "\*" " " "\*" " " " " "\*"   
## 11 ( 1 ) "\*" "\*" " " " " " " "\*" " " "\*" " " " " "\*"   
## 12 ( 1 ) "\*" "\*" " " "\*" " " "\*" " " "\*" " " " " "\*"   
## 13 ( 1 ) "\*" "\*" " " "\*" " " "\*" " " "\*" " " " " "\*"   
## 14 ( 1 ) "\*" "\*" "\*" "\*" " " "\*" " " "\*" " " " " "\*"   
## 15 ( 1 ) "\*" "\*" "\*" "\*" " " "\*" " " "\*" "\*" " " "\*"   
## 16 ( 1 ) "\*" "\*" "\*" "\*" "\*" "\*" " " "\*" "\*" " " "\*"   
## 17 ( 1 ) "\*" "\*" "\*" "\*" "\*" "\*" " " "\*" "\*" " " "\*"   
## 18 ( 1 ) "\*" "\*" "\*" "\*" "\*" "\*" "\*" "\*" "\*" " " "\*"   
## 19 ( 1 ) "\*" "\*" "\*" "\*" "\*" "\*" "\*" "\*" "\*" "\*" "\*"   
## CRBI CWalks LeagueN DivisionW PutOuts Assists Errors NewLeagueN  
## 1 ( 1 ) "\*" " " " " " " " " " " " " " "   
## 2 ( 1 ) "\*" " " " " " " " " " " " " " "   
## 3 ( 1 ) "\*" " " " " " " "\*" " " " " " "   
## 4 ( 1 ) "\*" " " " " "\*" "\*" " " " " " "   
## 5 ( 1 ) "\*" " " " " "\*" "\*" " " " " " "   
## 6 ( 1 ) "\*" " " " " "\*" "\*" " " " " " "   
## 7 ( 1 ) " " " " " " "\*" "\*" " " " " " "   
## 8 ( 1 ) " " "\*" " " "\*" "\*" " " " " " "   
## 9 ( 1 ) "\*" "\*" " " "\*" "\*" " " " " " "   
## 10 ( 1 ) "\*" "\*" " " "\*" "\*" "\*" " " " "   
## 11 ( 1 ) "\*" "\*" "\*" "\*" "\*" "\*" " " " "   
## 12 ( 1 ) "\*" "\*" "\*" "\*" "\*" "\*" " " " "   
## 13 ( 1 ) "\*" "\*" "\*" "\*" "\*" "\*" "\*" " "   
## 14 ( 1 ) "\*" "\*" "\*" "\*" "\*" "\*" "\*" " "   
## 15 ( 1 ) "\*" "\*" "\*" "\*" "\*" "\*" "\*" " "   
## 16 ( 1 ) "\*" "\*" "\*" "\*" "\*" "\*" "\*" " "   
## 17 ( 1 ) "\*" "\*" "\*" "\*" "\*" "\*" "\*" "\*"   
## 18 ( 1 ) "\*" "\*" "\*" "\*" "\*" "\*" "\*" "\*"   
## 19 ( 1 ) "\*" "\*" "\*" "\*" "\*" "\*" "\*" "\*"

We can also get the RSS, , adjusted , and BIC from the results which help us to assess the best overall model; however, we will illustrate this in the [comparing models](#comparing-models) section. First, let’s look at how to perform stepwise selection.

#### Stepwise selection

For computational reasons, best subset selection cannot be applied when the number of *p* predictor variable is large. est subset selection may also suffer from statistical problems when p is large. The larger the search space, the higher the chance of finding models that look good on the training data, even though they might not have any predictive power on future data. Thus an enormous search space can lead to overfitting and high variance of the coefficient estimates. For both of these reasons, stepwise methods, which explore a far more restricted set of models, are attractive alternatives to best subset selection.

##### Foward stepwise

*Forward stepwise* selection begins with a model containing no predictors, and then adds predictors to the model, one-at-a-time, until all of the predictors are in the model. In particular, at each step the variable that gives the greatest additional improvement to the fit is added to the model.

The three-state process of performing foward stepwise selection includes:

**Step1**:Let denote the null model, which contains no predictors. This model simply predicts the sample mean for each observation.

**step2**: For - Consider all *p-k* models that augument the predictors in with one additional predictor - Choose the best among these *p-k* models, and call it . Here best is defined as having smallest RSS or highest

**Step3:** Select a single best model from among using cross-validated prediction error , AIC, BIC or adjusted

We can perform backward stepwise using regsubsets by setting method = "backward":

backward <- regsubsets(Salary ~ ., hitters,  
 nvmax = 19, method = "backward")

#### Comparing models

So far, I’ve illustrated how to perform the best subset and stepwise procedures. Now let’s discuss how to compare all the models that these approaches output in order to identify the *best* model. That is, let’s perform step 3 discussed in each of the 3-state processes outlined approaches.

1. We can indirectly estimate test error by making an adjustment to the training error to account for the bias due to ovefitting.
2. We can directly estimate the test error, using either a validation set approach or a cross-validation approach.

We consider both of these approaches below.

##### Indirectly estimating test error with , AIC, BIC and adjusted

When performing the best subset or stepwise approaches, the models are selected based on the fact that they minimize the training set mean square error (MSE). Because of this and the fact that using the training MSE and will bias our results we should not use these statstics to determine which of the models is “the best”.

However, a number of technique for adjusting the training error for the model size are available. These approaches can be used to select among a set of models with different number of variables These include:

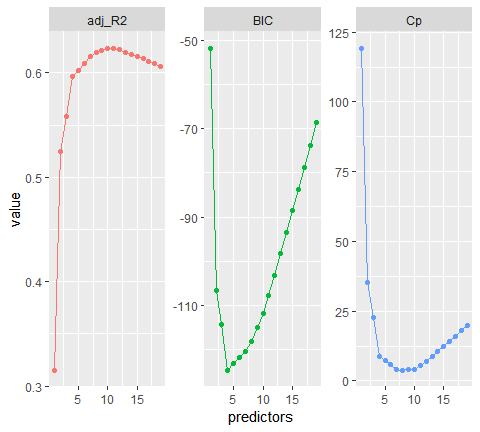
**Objective: Minimize** - : - : -

**Objective: Maximize** - adjusted : adj = 1 -

where, is the number of predictors and is an estimate of the variance of the error associated with each response measurement in a regression model. Each of these statistics adds a penalty to the training RSS in order to adjust for the fact that the training error tends to underestimate the test error. Clearly, the penalty increases as the number of predictors in the model increases.

Therefore, these statistics provide an unbiased estimate of test MSE. If we perform our model using a training vs. testing validation approach we can use these statistics to determine the preferred model. These statistics are contained in the output provided by the regsubsets function. Let窶冱 extract this information and plot them.

# create training - testing data  
set.seed(1)  
sample <- sample(c(T,F), nrow(hitters), replace=T, prob = c(0.6, 0.4))  
train <- hitters[sample,]  
test <- hitters[!sample,]  
  
# perform best subset selection  
best\_subset <- regsubsets(Salary~., train, nvmax = 19)  
results <- summary(best\_subset)  
  
# extract and plot results  
tibble(predictors = 1:19,  
 adj\_R2 = results$adjr2,  
 Cp = results$cp,  
 BIC = results$bic) %>%   
 gather(statistic, value, -predictors) %>%   
 ggplot(aes(predictors, value, color=statistic))+  
 geom\_line(show.legend = F)+  
 geom\_point(show.legend = F)+  
 facet\_wrap(~statistic, scales="free")



Here we see that our results identify slightly different models that are considered the best. The adjusted statistic suggests the 10 variable model is preferred, the BIC statistic suggests the 4 variable model, and the suggests the 8 variable model.

which.max(results$adjr2)  
## [1] 10  
which.min(results$bic)  
## [1] 4  
min(results$bic)  
## [1] -124.6282  
which.min(results$cp)  
## [1] 8

We can compare the variables and coefficinets that these models include using the coef function.

# 10 variable model  
coef(best\_subset, 10)  
## (Intercept) AtBat Hits Walks CAtBat CHits   
## -47.3715661 -1.3695666 6.3013473 4.5757613 -0.3118794 1.4799307   
## CHmRun CWalks LeagueN DivisionW PutOuts   
## 1.2971405 -0.5026157 62.5613310 -62.3548737 0.2527181  
  
# 4 variable model  
coef(best\_subset, 4)  
## (Intercept) Runs CAtBat CHits PutOuts   
## -83.1199265 5.5530883 -0.4741822 2.0560595 0.3118252  
  
# 8 variable model  
coef(best\_subset, 8)  
## (Intercept) AtBat Hits Walks CAtBat CHits   
## -59.2371674 -1.4744877 6.6802515 4.4777879 -0.3203862 1.5160882   
## CHmRun CWalks PutOuts   
## 1.1861142 -0.4714870 0.2748103

We could perform the same process using foward and backward stepwise selection and obtain even more options for optimal models. For example, if I assess the optimal for foward/backward stepwise, we see that they suggest that an 8 variable model minimizes the statsitic, similar to the best subset approach above.

forward <- regsubsets(Salary ~ ., train, nvmax = 19, method = "forward")  
backward <- regsubsets(Salary ~ ., train, nvmax = 19, method = "backward")  
  
# which models minimize Cp?  
which.min(summary(forward)$cp)  
## [1] 8  
## [1] 8  
which.min(summary(backward)$cp)  
## [1] 8  
## [1] 8

However, when we assess these models we see that the 8 variable models include different predictors. Although, all models include AtBat, Hits, Walks, CWalks, and PutOuts, there are unique variables in each model.

coef(best\_subset, 8)  
## (Intercept) AtBat Hits Walks CAtBat CHits   
## -59.2371674 -1.4744877 6.6802515 4.4777879 -0.3203862 1.5160882   
## CHmRun CWalks PutOuts   
## 1.1861142 -0.4714870 0.2748103  
## (Intercept) AtBat Hits Walks CAtBat CHits   
## -59.2371674 -1.4744877 6.6802515 4.4777879 -0.3203862 1.5160882   
## CHmRun CWalks PutOuts   
## 1.1861142 -0.4714870 0.2748103  
coef(forward, 8)  
## (Intercept) AtBat Hits Walks CRuns   
## -112.7724200 -2.1421859 8.8914064 5.4283843 0.8555089   
## CRBI CWalks LeagueN PutOuts   
## 0.4866528 -0.9672115 64.1628445 0.2767328  
## (Intercept) AtBat Hits Walks CRuns   
## -112.7724200 -2.1421859 8.8914064 5.4283843 0.8555089   
## CRBI CWalks LeagueN PutOuts   
## 0.4866528 -0.9672115 64.1628445 0.2767328  
coef(backward, 8)  
## (Intercept) AtBat Hits Walks CAtBat CHits   
## -59.2371674 -1.4744877 6.6802515 4.4777879 -0.3203862 1.5160882   
## CHmRun CWalks PutOuts   
## 1.1861142 -0.4714870 0.2748103  
## (Intercept) AtBat Hits Walks CAtBat CHits   
## -59.2371674 -1.4744877 6.6802515 4.4777879 -0.3203862 1.5160882   
## CHmRun CWalks PutOuts   
## 1.1861142 -0.4714870 0.2748103

This highlights two important findings:

1. Different subsetting procedures (best subset vs. forward stepwise vs. backward stepwise) will likely identify different 窶彙est窶? models.
2. Different indirect error test estimate statistics () will likely identify different best models.

This is why it is important to always perform validation; that is to always estiamte the test error directly either by using a validation set or using cross-validation.

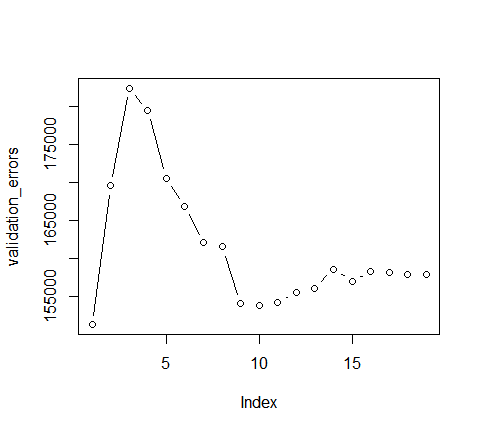
##### Directly estimating test error

We now compute the validation set error for the best model of each model size. We first make a model matrix from the test data. The model.matrix function is used in many regression packages for build- ing an 窶弭窶? matrix from data.

test\_m <- model.matrix(Salary~., data = test)

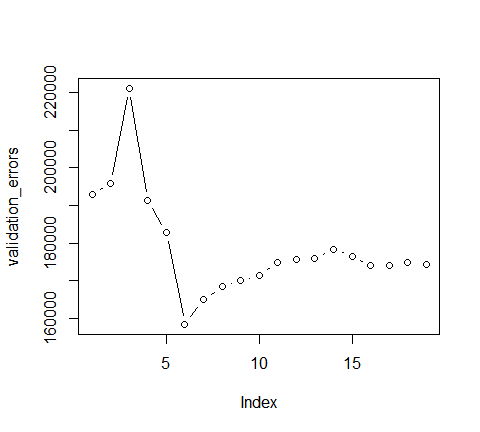
Now we can loop through each moel size (i.e., 1 variable, 2 variables, … 19 variables) and extract the coefficient for the best model of that size, multiply them into the appropriate coluns of the test moel matrix to form the predictors and compute the test MSE.

# create empty vector to fill with erro values  
validation\_errors <- vector("double", length=19)  
  
for(i in 1:19) {  
 coef\_x <- coef(best\_subset, id = i) # extract coefficients for model size i  
 pred\_x <- test\_m[ , names(coef\_x)] %\*% coef\_x # predict salary using matrix algebra  
 validation\_errors[i] <- mean((test$Salary - pred\_x)^2) # compute test error btwn actual & predicted salary  
}  
  
# plot validation errors  
plot(validation\_errors, type = "b")



Here, we actually see that the 1 variable model produced by the best subset approach produces the lowest test MSE! If we repeat this using a different random value seed, we will get a slightly different model that is the 窶彙est窶?. However, if you recall from the [Resampling Methods](http://uc-r.github.io/resampling_methods) tutorial, this is to be expected when using a training vs. testing validation approach.

# create training - testing data  
set.seed(5)  
sample <- sample(c(TRUE, FALSE), nrow(hitters), replace = T, prob = c(0.6,0.4))  
train <- hitters[sample, ]  
test <- hitters[!sample, ]  
  
# perform best subset selection  
best\_subset <- regsubsets(Salary ~ ., train, nvmax = 19)  
  
# compute test validation errors  
test\_m <- model.matrix(Salary ~ ., data = test)  
validation\_errors <- vector("double", length = 19)  
  
for(i in 1:19) {  
 coef\_x <- coef(best\_subset, id = i) # extract coefficients for model size i  
 pred\_x <- test\_m[ , names(coef\_x)] %\*% coef\_x # predict salary using matrix algebra  
 validation\_errors[i] <- mean((test$Salary - pred\_x)^2) # compute test error btwn actual & predicted salary  
}  
  
# plot validation errors  
plot(validation\_errors, type = "b")



A more robust approach is to perform cross validation. But before we do, let窶冱 turn our our approach above for computing test errors into a function. Our function pretty much mimics what we did above. The only complex part is how we extracted the formula used in the call to regsubsets. I suggest you work through this line-by-line to understand what each step is doing.

predict.regsubsets <- function(object, newdata, id ,...) {  
 form <- as.formula(object$call[[2]])   
 mat <- model.matrix(form, newdata)  
 coefi <- coef(object, id = id)  
 xvars <- names(coefi)  
 mat[, xvars] %\*% coefi  
 }

We now try to choose among the models of different sizes using k-fold cross-validation. This approach is somewhat involved, as we must perform best subset selection within each of the k training sets. First, we create a vector that allocates each observation to one of k = 10 folds, and we create a matrix in which we will store the results.

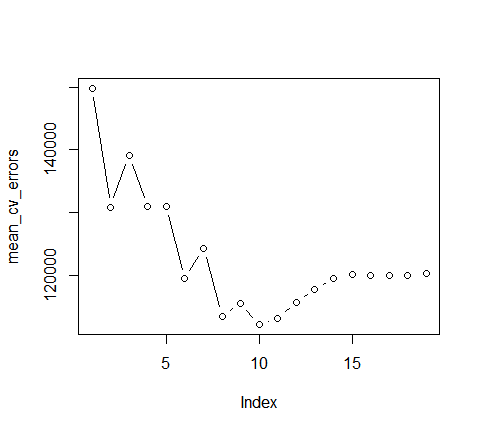
k <- 10  
set.seed(1)  
folds <- sample(1:k, nrow(hitters), replace = TRUE)  
cv\_errors <- matrix(NA, k, 19, dimnames = list(NULL, paste(1:19)))  
cv\_errors  
## 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19  
## [1,] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA  
## [2,] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA  
## [3,] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA  
## [4,] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA  
## [5,] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA  
## [6,] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA  
## [7,] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA  
## [8,] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA  
## [9,] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA  
## [10,] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA

Now we write a for loop that performs cross-validation. In the jth fold, the elements of folds that equal j are in the test set, and the remainder are in the training set. We make our predictions for each model size, compute the test errors on the appropriate subset, and store them in the appropriate slot in the matrix cv\_errors.

for(j in 1:k) {  
   
 # perform best subset on rows not equal to j  
 best\_subset <- regsubsets(Salary ~ ., hitters[folds != j, ], nvmax = 19)  
   
 # perform cross-validation  
 for( i in 1:19) {  
 pred\_x <- predict.regsubsets(best\_subset, hitters[folds == j, ], id = i)  
 cv\_errors[j, i] <- mean((hitters$Salary[folds == j] - pred\_x)^2)  
 }  
 }

This has given us a 10ﾃ?19 matrix, of which the (i,j)th element corresponds to the test MSE for the ith cross-validation fold for the best j-variable model. We use the colMeans function to average over the columns of this matrix in order to obtain a vector for which the jth element is the cross-validation error for the j-variable model.

mean\_cv\_errors <- colMeans(cv\_errors)  
plot(mean\_cv\_errors, type = "b")



We see that our more robust cross-validation approach selects an 11-variable model. We can now perform best subset selection on the full data set in order to obtain the 11-variable model.

final\_best <- regsubsets(Salary ~ ., data = hitters , nvmax = 19)  
coef(final\_best, 11)  
## (Intercept) AtBat Hits Walks CAtBat   
## 135.7512195 -2.1277482 6.9236994 5.6202755 -0.1389914   
## CRuns CRBI CWalks LeagueN DivisionW   
## 1.4553310 0.7852528 -0.8228559 43.1116152 -111.1460252   
## PutOuts Assists   
## 0.2894087 0.2688277

#### Additional resources

This will get you started with approaches for performing linear model selection; however, understand that there are other approaches for more sophisticated model selection procedures. The following resources will help you learn more:

* [An Introduction to Statistical Learning](http://www-bcf.usc.edu/~gareth/ISLR/)
* [Applied Predictive Modeling](http://appliedpredictivemodeling.com/)
* [Elements of Statistical Learning](https://statweb.stanford.edu/~tibs/ElemStatLearn/)

### Naive Bayes

The **Naive Bayes classifier** is a simple probabilistic classifier which is based on Bayes theorem but with strong assumptions regarding independence. Historically, this technique became popular with application in emal filtering, spam detection and document categorization.

Although it is often outperformed by other techniques, and desipite the naive design and oversimplified assumptions, this classifier can perform well in many complex real-world problems. And since it is a resource efficient algorithm that is fast and scales well, it is definitely a machine learning algorithms to have in your toolkit.

#### tl;dr

This tutorial serves as an introduction to naive Bayes classifier and covers:

1. [Replication requirment](#Naive_Bayes_RR): what you will need to reproduce the analysis in this tutorial.
2. A naive overview: A closer look behind the naive Bayes classifier and its pros and cons
3. caret: Implementing with the caret package.
4. H2O: Implementing with the h2o package.
5. Learning more Where to go from here.

#### Replication requirements

This tutorial leverages the following packages.

library(rsample) # data splitting  
library(dplyr) # data transformation  
library(ggplot2) # data visualization  
library(caret) # implementing with caret  
library(h2o) # implemeting with h2o

To illustrate the naive Bayes classifier, we will use the attrition data that has been included in the rsample package. The goal is to predict employee attrition.

# convert some numeric variables to factors  
  
attrition <- attrition %>%   
 mutate(  
 JobLevel = factor(JobLevel),  
 StockOptionLevel = factor(StockOptionLevel),  
 TrainingTimesLastYear = factor(TrainingTimesLastYear)  
 )  
  
# Create training (70%) and test (30%) sets for the attrition data.  
# Use set.seed for reproducibility  
set.seed(123)  
split <- initial\_split(attrition, prop = .7, strata = "Attrition")  
train <- training(split)  
test <- testing(split)  
  
# distribution of Attrition rates across train & test set  
table(train$Attrition) %>% prop.table()  
##   
## No Yes   
## 0.838835 0.161165  
##   
## No Yes   
## 0.838835 0.161165  
table(test$Attrition) %>% prop.table()  
##   
## No Yes   
## 0.8386364 0.1613636  
##   
## No Yes   
## 0.8386364 0.1613636

#### A Naive overview

##### The idea

The naive Bayes classifier is founded on Bayesian probability, which originated from [Reverend Thomas Bayes](https://en.wikipedia.org/wiki/Thomas_Bayes). Bayesian probability incorporates the concept of conditional probability, the probabilty of event A given that event B has occurred [denoted as . In the context of our attrition data, we are seeking the probability of an employee belonging to attrition class (where and ) given that its predictor values are . This can be written as .

The Bayesian formula for claculating the probability is

where:

* is the *prior* probability of the outcome. Essentially, based on the historical data, what is the probability of an employee attriting or not. As we saw in the above section preparing our training and test sets, our prior probability of an employee attriting was about 16% and the probability of not attriting was about 84%.
* is the probability of the predictor variables (same as .Essentially, based on the historical data, what is the probability of each observed combination of predictor variables. When new data comes in, this becomes our *evidence*.
* is the *conditional probability or likelihood*. Essentially, for each class of the response variable (i.e. attrit or not attrit), what is the probability of observing the predictor values.
* is called our *posterior probability*. By combining our observed information, we are updating our a priori information on probabilities to compute a posterior probability that an observation has class .

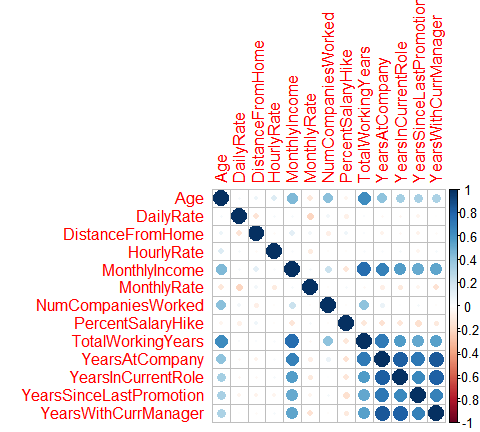
We can re-write Eq.(1) in plaini english as:

Although Eq. (1) has simplistic beauty on its surface, it becomes complex and interactable as the number of predictor variables grow. In fact, to compute the posterior probability for a response variable with *m* classes and a data set with *p* predictors, Eq. (1) would require probabilities computed. Sor for our attrition data, we have 2 classes (attrition vs. non-attrition) and 31 variables, requiring 2,147,483,648 probabilities computed.

##### The simplified classifier

Consequently, the naive Bayes classifier makes a simplifying assumption (hence the name) to allow the computation to scale. With naive Bayes, we assume that the predictor variables are *conditionally independent* of one another given the response value. This is an extremely strong assumption. We can see quickly that our attrition data violates this as we have several moderately to strongly correlated variables.

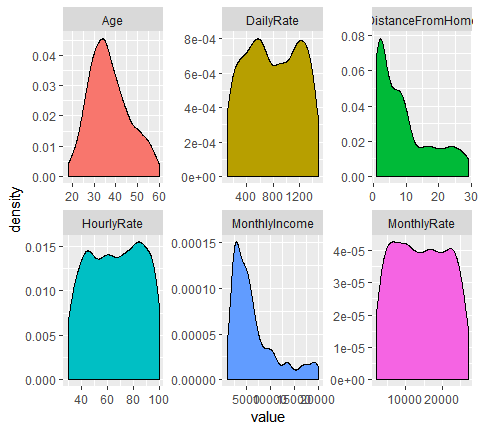
train %>%  
 filter(Attrition == "Yes") %>%   
 select\_if(is.numeric) %>%   
 cor() %>%   
 corrplot::corrplot()



However, by making this assumption, we can simplify our calculation such that the posterior probability is simply the product of the probability distribution for each individual variable conditioned on the response category (Eq. 2). Now we are only required to compute probabilities (this equates to 62 probabilities for our data set), a far more managable task.

For categorical variables, this computation is quite simple as you just use the frequencies from the data. However, when including continuous predictor variables often an assumption of normality is made so that we can use the probability from the variable窶冱 probability density function. If we pick a handful of our numeric features we quickly see assumption of normality is not always fair.

train %>%   
 dplyr::select(Age,DailyRate,DistanceFromHome, HourlyRate, MonthlyIncome, MonthlyRate) %>%   
 gather(metric, value) %>%   
 ggplot(aes(value, fill=metric))+  
 geom\_density(show.legend = FALSE)+  
 facet\_wrap(~metric, scales="free")



Granted, some numeric features may be normalized with a Box-Cox transformation; however, as you will see in this tutorial we can also use non-parametric kernel density estimators to try get a more accurate representation of continuous variable probabilities. Ultimately, transforming the distributions and selecting an estimator is part of the modeling development and tuning process.

##### Laplace Smoother

One additional issue to be aware of - since naﾃｯve Bayes uses the product of feature probabilities conditioned on each class, we run into a serious problem when new data includes a feature value that never occurs for one or more levels of a response class. What results is for this individual feature adn this zero will ripple through the entire multiplication of all features and this zero will ripple through the entire multiplication of all features and will always force the posterior probability to be zero for that class.

A solution to this problem involves using the **Laplace smooother**. The Laplace smooother adds a small number to each of the counts in the frequencies for each feature, which ensures that each feature has a nonzero probability of occuring for each class. Typically, a value of one to two for the Laplace smoother is sufficient, but this is a tuning parameter to incorporate and optimize with cross validation.

##### Advantage and Shortcomins

The naﾃｯve Bayes classifier is simple (both intuitively and computationally), fast, performs well with small amounts of training data, and scales well to large data sets. The greatest weakness of the naﾃｯve Bayes classifier is that it relies on an often-faulty assumption of equally important and independent features which results in biased posterior probabilities. Although this assumption is rarely met, in practice, this algorithm works surprisingly well. This is primarily because what is usually needed is not a propensity (exact posterior probability) for each record that is accurate in absolute terms but just a reasonably accurate rank ordering of propensities.

For example, we may not care about the exact posterior probability of attrition, we just want to know for a given observation, is the posterior probability of attriting larger than not attriting. Even when the assumption is violated, the rank ordering of the records窶? propensities is typically preserved. Consequentely, naﾃｯve Bayes is often a surprisingly accurate algorithm; however, on average it rarely can compete with the accuracy of advanced tree-based methods (random forests & gradient boosting machines) but is definitely worth having in your toolkit.

#### Implementation

There are several packages to apply naive Bayes (i.e., e1071, klaR, naivebayes, bnclassify). This tutorial demonstrates using the caret and h2o packages. caret allows us to use different naive Bayes packages above but in a common framework, and also allows for easy cross validation and tuning. h2o allows us to perform naive Bayes in a powerful and scalable architecture.

##### caret

First, we apply a naive Bayes model with 10-fold cross validation, which gets 83% accuracy. Consider about 83% of our obsevations in our training set do not attrit, our overall accuracy is not better than if we just predicted “No” attrition for every observation.

library(caret)  
  
# create response and feature data  
features <- setdiff(names(train), "Attrition")  
features  
## [1] "Age" "BusinessTravel"   
## [3] "DailyRate" "Department"   
## [5] "DistanceFromHome" "Education"   
## [7] "EducationField" "EnvironmentSatisfaction"   
## [9] "Gender" "HourlyRate"   
## [11] "JobInvolvement" "JobLevel"   
## [13] "JobRole" "JobSatisfaction"   
## [15] "MaritalStatus" "MonthlyIncome"   
## [17] "MonthlyRate" "NumCompaniesWorked"   
## [19] "OverTime" "PercentSalaryHike"   
## [21] "PerformanceRating" "RelationshipSatisfaction"  
## [23] "StockOptionLevel" "TotalWorkingYears"   
## [25] "TrainingTimesLastYear" "WorkLifeBalance"   
## [27] "YearsAtCompany" "YearsInCurrentRole"   
## [29] "YearsSinceLastPromotion" "YearsWithCurrManager"  
  
x <- train[, features]  
y <- train$Attrition  
  
# set up 10-fold cross validation procedure  
train\_control <- trainControl(  
 method = "cv",  
 number = 10  
)  
  
# train model  
nb.m1 <- caret::train(  
 x = x,  
 y = y,  
 method = "nb",  
 trControl = train\_control  
)  
# results   
confusionMatrix(nb.m1)  
## Cross-Validated (10 fold) Confusion Matrix   
##   
## (entries are percentual average cell counts across resamples)  
##   
## Reference  
## Prediction No Yes  
## No 76.6 8.2  
## Yes 7.3 8.0  
##   
## Accuracy (average) : 0.8456

We can turn the few hyperparameters that a naive Bayes model has.

* usekernel: parameter allows us to use a kernel density estimate for continuous variables versus a guassian density estimate,
* adjust: allows us to adjust the bandwidth of the kernel density (larger numbers mean more flexible density estimate),
* fL: allows us to incorporate the Laplace smoother.

If we just tuned our modelwith the above parameters, we are able to lift our accuracy to 85%; however, by incorporating some preprocessing of our features (normalize with Box Cox, standardize with center-scaling, and reducing with PCA), we actually get about another 2% lift in our accuracy.

# set up tuning grid  
search\_grid <- expand.grid(  
 usekernel = c(TRUE, FALSE),  
 fL = 0:5,  
 adjust = seq(0, 5, by = 1)  
)  
  
# train model  
nb.m2 <- train(  
 x = x,  
 y = y,  
 method = "nb",  
 trControl = train\_control,  
 tuneGrid = search\_grid,  
 preProc = c("BoxCox", "center", "scale", "pca")  
 )  
  
# top 5 model  
nb.m2$results %>%   
 top\_n(5, wt=Accuracy) %>%   
 arrange(Desc(Accuracy))  
## usekernel fL adjust Accuracy Kappa AccuracySD KappaSD  
## 1 TRUE 1 3 0.8737864 0.4435322 0.02858175 0.1262286  
## 2 TRUE 0 2 0.8689320 0.4386202 0.02903618 0.1155707  
## 3 TRUE 2 3 0.8689320 0.4750282 0.02830559 0.0970368  
## 4 TRUE 2 4 0.8689320 0.4008608 0.02432572 0.1234943  
## 5 TRUE 4 5 0.8689320 0.4439767 0.02867321 0.1354681  
  
  
# plot search grid results  
plot(nb.m2)

# results for best model  
confusionMatrix(nb.m2)  
  
## Cross-Validated (10 fold) Confusion Matrix   
##   
## (entries are percentual average cell counts across resamples)  
##   
## Reference  
## Prediction No Yes  
## No 80.8 9.5  
## Yes 3.1 6.6  
##   
## Accuracy (average) : 0.8738

We can assess the accuracy on our final holdout test set. Its obvious that our model is not capturing a large percentage of our actual attritions (illustrated by our low specificity).

pred <- predict(nb.m2, newdata = test)  
confusionMatrix(pred, test$Attrition)  
## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction No Yes  
## No 349 41  
## Yes 20 30  
##   
## Accuracy : 0.8614   
## 95% CI : (0.8255, 0.8923)  
## No Information Rate : 0.8386   
## P-Value [Acc > NIR] : 0.10756   
##   
## Kappa : 0.4183   
## Mcnemar's Test P-Value : 0.01045   
##   
## Sensitivity : 0.9458   
## Specificity : 0.4225   
## Pos Pred Value : 0.8949   
## Neg Pred Value : 0.6000   
## Prevalence : 0.8386   
## Detection Rate : 0.7932   
## Detection Prevalence : 0.8864   
## Balanced Accuracy : 0.6842   
##   
## 'Positive' Class : No   
##

#### h2o

Lets go ahead and start up h2o:

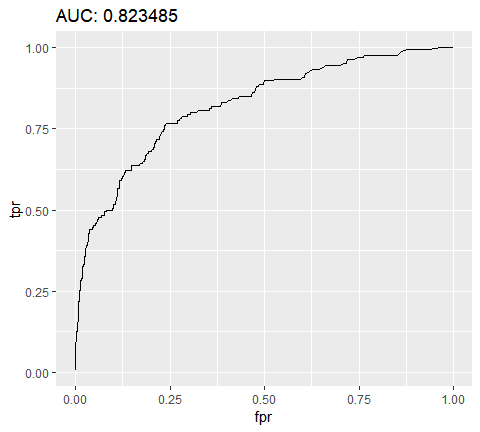
# start up h2o  
library(h2o)  
h2o.no\_progress()  
h2o.init()  
##   
## H2O is not running yet, starting it now...  
##   
## Note: In case of errors look at the following log files:  
## C:\Users\KOJIKM~1.MIZ\AppData\Local\Temp\RtmpOepbM3/h2o\_KojiKM\_Mizumura\_started\_from\_r.out  
## C:\Users\KOJIKM~1.MIZ\AppData\Local\Temp\RtmpOepbM3/h2o\_KojiKM\_Mizumura\_started\_from\_r.err  
##   
##   
## Starting H2O JVM and connecting: . Connection successful!  
##   
## R is connected to the H2O cluster:   
## H2O cluster uptime: 3 seconds 996 milliseconds   
## H2O cluster timezone: Asia/Tokyo   
## H2O data parsing timezone: UTC   
## H2O cluster version: 3.22.1.1   
## H2O cluster version age: 5 months !!!   
## H2O cluster name: H2O\_started\_from\_R\_KojiKM.Mizumura\_sjc551   
## H2O cluster total nodes: 1   
## H2O cluster total memory: 1.96 GB   
## H2O cluster total cores: 4   
## H2O cluster allowed cores: 4   
## H2O cluster healthy: TRUE   
## H2O Connection ip: localhost   
## H2O Connection port: 54321   
## H2O Connection proxy: NA   
## H2O Internal Security: FALSE   
## H2O API Extensions: Algos, AutoML, Core V3, Core V4   
## R Version: R version 3.6.0 (2019-04-26)

We can compute a naﾃｯve Bayes model in h2o with h2o.naiveBayes. Here we use the default model with no Laplace smoother. We experience similar results on our training cross validation as we did using caret.

# create feature names  
y <- "Attrition"  
x <- setdiff(names(train), y)  
  
library(magrittr)  
# h2o cannot ingest ordered factors  
train.h2o <- train %>%  
 mutate\_if(is.factor, factor, ordered=FALSE) %>%   
 as.h2o()  
  
# train model  
nb.h2o <- h2o.naiveBayes(  
 x = x,  
 y = y,  
 training\_frame = train.h2o,  
 nfolds = 10,  
 laplace = 0  
)  
  
# assess results  
h2o.confusionMatrix(nb.h2o)  
## Confusion Matrix (vertical: actual; across: predicted) for max f1 @ threshold = 0.803034027013974:  
## No Yes Error Rate  
## No 815 49 0.056713 =49/864  
## Yes 74 92 0.445783 =74/166  
## Totals 889 141 0.119417 =123/1030  
## Confusion Matrix (vertical: actual; across: predicted) for max f1 @ threshold = 0.751627773135859:  
## No Yes Error Rate  
## No 812 52 0.060185 =52/864  
## Yes 77 89 0.463855 =77/166  
## Totals 889 141 0.125243 =129/1030

We can also do some feature preprocessing as we did with caret and tune the Laplace smoother using h2o.grid. We don窶冲 see much improvement.

# do a little preprocessing  
preprocess <- preProcess(train, method = c("BoxCox", "center", "scale", "pca"))  
train\_pp <- predict(preprocess, train)  
test\_pp <- predict(preprocess, test)  
  
# convert to h2o objects  
train\_pp.h2o <- train\_pp %>%   
 mutate\_if(is.factor, factor, ordered=FALSE) %>%   
 as.h2o()  
  
test\_pp.h2o <- test\_pp %>%   
 mutate\_if(is.factor, factor, ordered=FALSE) %>%   
 as.h2o()  
  
# get new feature names -> PCA preprocessing reduced and changed some fe  
x <- setdiff(names(train\_pp), "Attrition")  
y <- "Attrition"  
  
# create tuning grid  
hyper\_params <- list(  
 laplace = seq(0,5, by = 0.5)  
)  
  
# build grid search  
grid <- h2o.grid(  
 algorithm = "naivebayes",  
 grid\_id = "nb\_grid",  
 x = x,  
 y = y,  
 training\_frame = train\_pp.h2o,  
 nfolds = 10,  
 hyper\_params = hyper\_params  
)  
  
# sort the grid models by mse  
sorted\_grid <- h2o.getGrid("nb\_grid", sort\_by = "accuracy",  
 decreasing = TRUE)  
sorted\_grid  
## H2O Grid Details  
## ================  
##   
## Grid ID: nb\_grid   
## Used hyper parameters:   
## - laplace   
## Number of models: 11   
## Number of failed models: 0   
##   
## Hyper-Parameter Search Summary: ordered by decreasing accuracy  
## laplace model\_ids accuracy  
## 1 0.5 nb\_grid\_model\_2 0.8786407766990292  
## 2 4.0 nb\_grid\_model\_9 0.8776699029126214  
## 3 3.5 nb\_grid\_model\_8 0.8766990291262136  
## 4 2.5 nb\_grid\_model\_6 0.874757281553398  
## 5 5.0 nb\_grid\_model\_11 0.870873786407767  
## 6 3.0 nb\_grid\_model\_7 0.8669902912621359  
## 7 1.5 nb\_grid\_model\_4 0.8631067961165049  
## 8 4.5 nb\_grid\_model\_10 0.8631067961165049  
## 9 2.0 nb\_grid\_model\_5 0.8592233009708738  
## 10 1.0 nb\_grid\_model\_3 0.8475728155339806  
## 11 0.0 nb\_grid\_model\_1 0.8203883495145631  
## H2O Grid Details  
## ================  
##   
## Grid ID: nb\_grid   
## Used hyper parameters:   
## - laplace   
## Number of models: 11   
## Number of failed models: 0   
##   
## Hyper-Parameter Search Summary: ordered by decreasing accuracy  
## laplace model\_ids accuracy  
## 1 2.5 nb\_grid\_model\_5 0.8572815533980582  
## 2 5.0 nb\_grid\_model\_10 0.8572815533980582  
## 3 3.0 nb\_grid\_model\_6 0.8533980582524272  
## 4 2.0 nb\_grid\_model\_4 0.8504854368932039  
## 5 1.5 nb\_grid\_model\_3 0.8466019417475728  
## 6 4.5 nb\_grid\_model\_9 0.8407766990291262  
## 7 0.5 nb\_grid\_model\_1 0.8398058252427184  
## 8 4.0 nb\_grid\_model\_8 0.8339805825242719  
## 9 3.5 nb\_grid\_model\_7 0.8281553398058252  
## 10 1.0 nb\_grid\_model\_2 0.8194174757281554  
## 11 0.0 nb\_grid\_model\_0 0.8009708737864077  
  
# grab top model id  
best\_h2o\_model <- sorted\_grid@model\_ids[[1]]  
best\_model <- h2o.getModel(best\_h2o\_model)  
  
# confusion matrix of best model  
h2o.confusionMatrix(best\_model)  
## Confusion Matrix (vertical: actual; across: predicted) for max f1 @ threshold = 0.384614509559317:  
## No Yes Error Rate  
## No 793 71 0.082176 =71/864  
## Yes 58 108 0.349398 =58/166  
## Totals 851 179 0.125243 =129/1030  
  
# ROC curve  
auc <- h2o.auc(best\_model, xval = TRUE)  
fpr <- h2o.performance(best\_model, xval = TRUE) %>% h2o.fpr() %>% .[['fpr']]  
tpr <- h2o.performance(best\_model, xval = TRUE) %>% h2o.tpr() %>% .[['tpr']]  
  
data.frame(fpr = fpr,  
 tpr = tpr) %>%   
 ggplot(aes(fpr, tpr))+  
 geom\_line()+  
 ggtitle(sprintf('AUC: %f', auc))



Once we窶况e identified the optimal model we can assess on our test set.

# evaluate on test set  
h2o.performance(best\_model, newdata = test\_pp.h2o)  
## H2OBinomialMetrics: naivebayes  
##   
## MSE: 0.1036054  
## RMSE: 0.321878  
## LogLoss: 0.3961723  
## Mean Per-Class Error: 0.2148364  
## AUC: 0.8100118  
## pr\_auc: 0.5257453  
## Gini: 0.6200237  
##   
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:  
## No Yes Error Rate  
## No 304 65 0.176152 =65/369  
## Yes 18 53 0.253521 =18/71  
## Totals 322 118 0.188636 =83/440  
##   
## Maximum Metrics: Maximum metrics at their respective thresholds  
## metric threshold value idx  
## 1 max f1 0.209868 0.560847 117  
## 2 max f2 0.169826 0.669856 133  
## 3 max f0point5 0.512767 0.618182 50  
## 4 max accuracy 0.512767 0.877273 50  
## 5 max precision 0.965328 1.000000 0  
## 6 max recall 0.000016 1.000000 399  
## 7 max specificity 0.965328 1.000000 0  
## 8 max absolute\_mcc 0.512767 0.497362 50  
## 9 max min\_per\_class\_accuracy 0.169826 0.788618 133  
## 10 max mean\_per\_class\_accuracy 0.169826 0.788675 133  
##   
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>, xval=<T/F>)`  
## H2OBinomialMetrics: naivebayes

# predict new data  
h2o.predict(nb.h2o, newdata = test\_pp.h2o)  
## predict No Yes  
## 1 No 0.7518712 0.248128833  
## 2 No 0.9545763 0.045423728  
## 3 No 0.9972380 0.002761993  
## 4 No 0.9892572 0.010742778  
## 5 No 0.7700278 0.229972157  
## 6 No 0.2774100 0.722590033  
##   
## [440 rows x 3 columns]  
## predict No Yes  
## 1 No 0.9851309 0.014869060  
## 2 No 0.9109746 0.089025413  
## 3 No 0.9947524 0.005247592  
## 4 No 0.9694504 0.030549563  
## 5 No 0.9683716 0.031628419  
## 6 No 0.9836910 0.016308996  
##   
## [440 rows x 3 columns]  
  
# shut down h2o  
h2o.shutdown(prompt = FALSE)  
## [1] TRUE  
## [1] TRUE

#### Learning more

Although we did not see much improvement over the baseline response class proportions in this example, the naﾃｯve Bayes classifier is often hard to beat in terms of CPU and memory consumption as shown by Huang, J. (2003), and in certain cases its performance can be very close to more complicated and slower techniques. Consquently, its a solid technique to have in your toolkit. If you want to dig deeper into this classifier, I would start with:

* [Andrew Moore窶冱 tutorials](http://www.cs.cmu.edu/~./awm/tutorials/naive.html)
* [Naive Bayes classifiers by Kevin Murphy](https://datajobsboard.com/wp-content/uploads/2017/01/Naive-Bayes-Kevin-Murphy.pdf)
* [Data Mining and Predictive Analytics, Ch. 14](https://www.amazon.com/Mining-Predictive-Analytics-Daniel-Chantal/dp/8126559136/ref=sr_1_1?ie=UTF8&qid=1524231609&sr=8-1&keywords=data+mining+and+predictive+analytics+2nd+edition+%2C+by+larose)

### Logistic regression

Logistic regression (aka logit regressionor logit model) was developed by statistician David Cox in 1958 and is a regression model where the response variable Y is categorical. Logistic regression allows us to estimate the probability of a categorical response based on one or more predictor variables (). It allows one to say that the presence of a predictor increases (or decreases) the probability of a given outcome by a specific percentage. This tutorial covers the case when Y is binary 窶? that is, where it can take only two values, 窶?0窶? and 窶?1窶?, which represent outcomes such as pass/fail, win/lose, alive/dead or healthy/sick. Cases where the dependent variable has more than two outcome categories may be analysed with multinomial logistic regression, or, if the multiple categories are ordered, in ordinal logistic regression. However, *discriminant analysis* has become a popular method for multi-class classification so our next tutorial will focus on that technique for those instances.

#### tl;dr

This turorial serves as an introudction to logistic regression and covers:

1. [Replication requirements](#log-pre-requisite): What you窶冤l need to reproduce the analysis in this tutorial
2. [Why logistic regression](#why-logit): Why use logistic regression?
3. [Preparing our data](#logit-data): Prepare our data for modeling
4. Simple Logistic regression: Predicting the probability of response Y with a single predictor variable X
5. Multiple Logistic regression: Predicting the probability of response Y with multiple predictor variables
6. Model evaluation & diagnostics: How well does the model fit the data? Which predictors are most important? Are the predictions accurate?

#### Replication requirements

This tutorial primarily leverages the Default data provided by the ISLR package. This is simulated data set containing information on ten thousand customers such as whether the customer defaulted, is a student, the average balance carried by the customer and the income of the customer. We窶冤l also use a few packages that provide data manipulation, visualization, pipeline modeling functions, and model output tidying functions.

library(tidyverse)  
library(modelr)  
library(broom)  
  
# load data  
(default <- as\_tibble(ISLR::Default))  
## # A tibble: 10,000 x 4  
## default student balance income  
## <fct> <fct> <dbl> <dbl>  
## 1 No No 730. 44362.  
## 2 No Yes 817. 12106.  
## 3 No No 1074. 31767.  
## 4 No No 529. 35704.  
## 5 No No 786. 38463.  
## 6 No Yes 920. 7492.  
## 7 No No 826. 24905.  
## 8 No Yes 809. 17600.  
## 9 No No 1161. 37469.  
## 10 No No 0 29275.  
## # ... with 9,990 more rows

#### Why Logistic regression

Linear regression is not approproate in the case of qualitative response. Why not? Suppose that we are trying to predict the medical condition of a patient in the emergency room on the basis of her symptoms. In this simplified example, there are three possible diagnoses: *stroke*, *drug overdose*, and *epileptic seizure*. We could consider encoding these values as a quantitative response variable, Y , as follows:

$$
Y = 1, if stroke; \\
2, if drug overdose; \\
3, if epileptic seizure
$$

Using this coding, least squares could be used to fit a linear regression model to predict Y on the basis of a set of predictors . . Unfortunately, this coding implies an ordering on the outcomes, putting drug overdose in between stroke and epileptic seizure, and insisting that the difference between stroke and drug overdose is the same as the difference between drug overdose and epileptic seizure. In practice there is no particular reason that this needs to be the case. For instance, one could choose an equally reasonable coding,

which would imply a totally different relationship among the three conditions. Each of these codings would produce fundamentally different linear models that would ultimately lead to different sets of predictions on test observations.

More relevant to our data, if we are trying to classify a customer as a high- vs. low-risk defaulter based on their balance we could use linear regression; however, the left figure below illustrates how linear regression would predict the probability of defaulting. Unfortunately, for balances close to zero we predict a negative probability of defaulting; if we were to predict for very large balances, we would get values bigger than 1. These predictions are not sensible, since of course the true probability of defaulting, regardless of credit card balance, must fall between 0 and 1.

default %>% colnames()  
## [1] "default" "student" "balance" "income"  
  
lm <- lm(default ~ balance, data=default)  
glm <- glm(default ~ balance, data=default, family = "binomial")

To avoid this problem, we must model p(X) using a function that gives outputs between 0 and 1 for all values of X. Many functions meet this description. In logistic regression, we use the logistic function, which is defined in Eq. 1 and illustrated in the right figure above.

$$
p(x) = \frac{e^{\beta\_o + \beta\_1X}}{1+e^{\beta\_o + \beta\_1X}
$$

#### Preparing our data

As in the regression tutorial, we will split our data into a training (60%) and testing (40%) data sets so we can assess how well our model performs on an out-of-sample data set.

set.seed(123)  
sample <- sample(c(TRUE, FALSE), nrow(default), replace = T, prob = c(0.6,0.4))  
train <- default[sample, ]  
test <- default[!sample, ]

#### Simple logistic regression

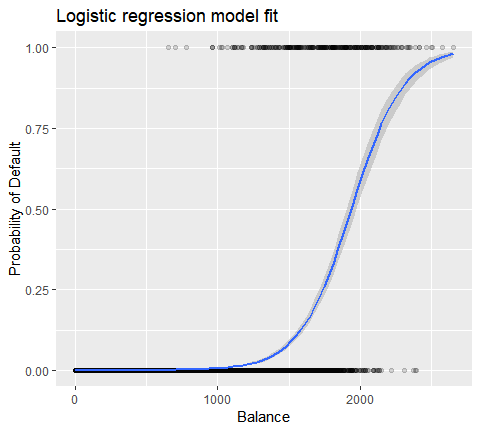
We will fit a logistic regression model in order to predict the probability of a customer defaulting based on the average balance carried by the customer. The glm function fits generalized linear models, a class of models that includes logistic regression. The syntax of the glm function is similar to that of lm, except that we must pass the argument family = binomial in order to tell R to run a logistic regression rather than some other type of generalized linear model.

model1 <- glm(default ~ balance, family = "binomial",  
 data = train)

In the background the glm, uses *maximum likelihood* to fit the model. The basic intuition behind using maximum likelihood to fit a logistic regression model is as follows: we seek estimates for and such that the predicted probability of default for each individual, using Eq.1, corresponds as closely as possible to the individual’s observed default status. In other words, we try to find and 1 such that plugging these estimates into the model for , given in Eq. 1, yields a number close to one for all individuals who defaulted, and a number close to zero for all individuals who did not. This intuition can be formalized using a mathematical equation called a likelihood function:

The estimates are chosen to *maximize* this likelihood function. Maximum likelihood is very general approach that is used to fit many of the non-linear models that we will examine in future totorial. What resuls is an S-shaped probability curve illustrated below (note that to plot the logistic regression fit line, we need to convert our response variable to a binary coded variable)

default %>%  
 mutate(prob = ifelse(default == "Yes", 1, 0)) %>%  
 ggplot(aes(balance, prob)) +  
 geom\_point(alpha = .15) +  
 geom\_smooth(method = "glm", method.args = list(family = "binomial")) +  
 ggtitle("Logistic regression model fit") +  
 xlab("Balance") +  
 ylab("Probability of Default")



Similar to linear regression we can assess the model using summary or glance. Note that the coefficient output format is similar to what we saw in linear regression; however, the goodness-of-fit details at the bottom of summary differ. We窶冤l get into this more later but just note that you see the word deviance. Deviance is analogous to the sum of squares calculations in linear regression and is a measure of the lack of fit to the data in a logistic regression model. The null deviance represents the difference between a model with only the intercept (which means 窶從o predictors窶?) and a saturated model (a model with a theoretically perfect fit). The goal is for the model deviance (noted as *Residual deviance*) to be lower; smaller values indicate better fit. In this respect, the null model provides a baseline upon which to compare predictor models.

summary(model1)  
##   
## Call:  
## glm(formula = default ~ balance, family = "binomial", data = train)  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -2.2905 -0.1395 -0.0528 -0.0189 3.3346   
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) -1.101e+01 4.887e-01 -22.52 <2e-16 \*\*\*  
## balance 5.669e-03 2.949e-04 19.22 <2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 1723.03 on 6046 degrees of freedom  
## Residual deviance: 908.69 on 6045 degrees of freedom  
## AIC: 912.69  
##   
## Number of Fisher Scoring iterations: 8  
##   
## Call:  
## glm(formula = default ~ balance, family = "binomial", data = train)  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -2.2905 -0.1395 -0.0528 -0.0189 3.3346   
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) -1.101e+01 4.887e-01 -22.52 <2e-16 \*\*\*  
## balance 5.669e-03 2.949e-04 19.22 <2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 1723.03 on 6046 degrees of freedom  
## Residual deviance: 908.69 on 6045 degrees of freedom  
## AIC: 912.69  
##   
## Number of Fisher Scoring iterations: 8

##### Asessing coefficients

The below table shows the coefficient estimates and related information that result from fitting a logistic regression model in order to predict the probability of default = Yes using balance. Bear in mind that the coefficient estimates from logistic regression characterize the relationship between the predictor and response variable on a *log-odds* scale (see Ch. 3 of ISLR1 for more details). Thus, we see that $\hatﾎｲ\_1=0.0057$ ; this indicates that an increase in balance is associated with an increase in the probability of default. To be precise, a one-unit increase in balance is associated with an increase in the log odds of default by 0.0057 units.

broom::tidy(model1)  
## # A tibble: 2 x 5  
## term estimate std.error statistic p.value  
## <chr> <dbl> <dbl> <dbl> <dbl>  
## 1 (Intercept) -11.0 0.489 -22.5 2.66e-112  
## 2 balance 0.00567 0.000295 19.2 2.53e- 82  
## term estimate std.error statistic p.value  
## 1 (Intercept) -11.006277528 0.488739437 -22.51972 2.660162e-112  
## 2 balance 0.005668817 0.000294946 19.21985 2.525157e-82

We can further interpret the *balance* coefficient as - for every one dollar increase in monthly balance carried, the odds of the customer defaulting increases by a factor of 1.0057.

exp(coef(model1))  
## (Intercept) balance   
## 1.659718e-05 1.005685e+00

Many aspects of the coefficient output are similar to those discussed in the linear regression output. For example, we measure the confidence intervals and accuracy of the coefficient estimates by computing their standard errors. For instance, $\hatﾎｲ\_1$ has a p-value < and the probability of defaulting. We can also use the standard errors to get confidence intervals as we did in the linear regression tutorial:

confint(model1)  
## 2.5 % 97.5 %  
## (Intercept) -12.007610373 -10.089360652  
## balance 0.005111835 0.006269411

##### Making predictions

Once the coefficients have benn estimated, it is a simple matter to comupte the probability of default for any given credit card balance. Mathematically, using the coefficient estimates from our model we predict that the default probability for an individual with a balance of $1,000 is less than 0.5%

$$ p(x) = = 0.004785

, where \_o = -11.0063, 0.0057 $$

We can predict the probability of defaulting in R using the predict function (be sure to include type = "response"). Here we compare the probability of defaulting based on balances of $1000 and $2000. As you can see as the balance moves from $1000 to $2000 the probability of defaulting increases signficantly, from 0.5% to 58%!

predict(model1,   
 data.frame(balance = c(1000, 2000)),   
 type = "response")  
## 1 2   
## 0.004785057 0.582089269  
## 1 2   
## 0.004785057 0.582089269

One can also use qualitative predictors with the logistic regression model. As an example, we can fit a model that uses the student variable.

model2 <- glm(default ~ student, family = "binomial", data = train)  
  
tidy(model2)  
## # A tibble: 2 x 5  
## term estimate std.error statistic p.value  
## <chr> <dbl> <dbl> <dbl> <dbl>  
## 1 (Intercept) -3.55 0.0934 -38.1 0   
## 2 studentYes 0.441 0.149 2.96 0.00311  
## term estimate std.error statistic p.value  
## 1 (Intercept) -3.5534091 0.09336545 -38.05914 0.000000000  
## 2 studentYes 0.4413379 0.14927208 2.95660 0.003110511

The coefficient associated with student = Yes is positive, and the associated p-value is statistically significant. This indicates that students tend to have higher default probabilities than non-students. In fact, this model suggests that a student has nearly twice the odds of defaulting than non-students. However, in the next section we窶冤l see why.

$$ p(default = Yes|student = Yes) = =0.0426 \

p(default = Yes|student = No) = =0.0426

$$

predict(model2, data.frame(student = factor(c("Yes", "No"))), type = "response")  
## 1 2   
## 0.04261206 0.02783019  
## 1 2   
## 0.04261206 0.02783019

#### Multiple logistic regression

We can also extend our model as seen in Eq. 1 so that we can predict a binary response using multiple predictors where ) are p predictors:

Let窶冱 go ahead and fit a model that predicts the probability of *default* based on the balance, income (in thousands of dollars), and student status variables. There is a surprising result here. The p-values associated with balance and student=Yes status are very small, indicating that each of these variables is associated with the probability of defaulting. However, the coefficient for the student variable is negative, indicating that students are less likely to default than non-students. In contrast, the coefficient for the student variable in model 2, where we predicted the probability of default based only on student status, indicated that students have a greater probability of defaulting. What gives?

model3 <- glm(default ~ balance + income + student, family = "binomial", data = train)  
tidy(model3)  
## # A tibble: 4 x 5  
## term estimate std.error statistic p.value  
## <chr> <dbl> <dbl> <dbl> <dbl>  
## 1 (Intercept) -10.9 0.648 -16.8 1.47e-63  
## 2 balance 0.00591 0.000310 19.0 7.90e-81  
## 3 income -0.00000501 0.0000108 -0.465 6.42e- 1  
## 4 studentYes -0.809 0.313 -2.58 9.78e- 3

The right-hand panel of the figure below provides an explanation for this discrepancy. The variables student and balance are correlated. Students tend to hold higher levels of debt, which is in turn associated with higher probability of default. In other words, students are more likely to have large credit card balances, which, as we know from the left-hand panel of the below figure, tend to be associated with high default rates. Thus, even though an individual student with a given credit card balance will tend to have a lower probability of default than a non-student with the same credit card balance, the fact that students on the whole tend to have higher credit card balances means that overall, students tend to default at a higher rate than non-students. This is an important distinction for a credit card company that is trying to determine to whom they should offer credit. A student is riskier than a non-student if no information about the student窶冱 credit card balance is available. However, that student is less risky than a non-student with the same credit card balance!

This simple example illustrates the dangers and subtleties associated with performing regressions involving only a single predictor when other predictors may also be relevant. The results obtained using one predictor may be quite different from those obtained using multiple predictors, especially when there is correlation among the predictors. This phenomenon is known as *confounding*.

In the case of multiple predictor variables sometimes we want to understand which variable is the most influential in predicting the response (Y) variable. We can do this with varImp from the caret package. Here, we see that *balance* is the most important by a large margin whereas student status is less important followed by income (which was found to be insignificanbt anyways (p = .64)).

caret::varImp(model3)  
## Overall  
## balance 19.0403764  
## income 0.4647343  
## studentYes 2.5835947

As before, we can easily make predictions with this model. For example, a student with a credit card balance of and an income of has an estimated probability of default of

$$ p(X) = p(X) = = 0.054

$$

A non-student with the same balance and income has an estimated probability of default of

new.df <- tibble(balance = 1500, income = 40, student = c("Yes", "No"))  
predict(model3, new.df, type = "response")  
## 1 2   
## 0.05437124 0.11440288  
## 1 2   
## 0.05437124 0.11440288

Thus, we see that for the given balance and income (although income is insignificant) a student has about half the probability of defaulting than a non-student.

#### Mode evaluation & diagnostics

So far three logistic regression models have been built and the coefficients have been examined. However, some critical questions remain. Are the models any good? How well does the model fit the data? And how accurate are the predictions on an out-of-sample data set?

##### Goodness of fit

In the linear regression tutorial we saw how the F-statistic, and adjusted and residual diagnostics inform us of how good the model fits the data. Here, we will look at a few ways to assess the goodness of fit ofr our logit models.

##### Likelihood ratio test

First, we can use a *Likelihood Ratio Test* to assess if our models are improving the fit. Adding predictor variables to a model will almost always improve the model fit (i.e. increase the log likelihood and reduce the model deviance compared to the null deviance), but it is necessary to test whether the observed difference in model fit is statistically significant. We can use *anova* to perform this test. The results indicate that, compared to model1, model3 reduces the residual deviance by over 13 (remember, a goal of logistic regression is to find a model that minimizes deviance residuals). More imporantly, this improvement is statisticallly significant at p = 0.001. This suggests that model3 does provide an improved model fit.

anova(model1, model3, test="Chisq")  
## Analysis of Deviance Table  
##   
## Model 1: default ~ balance  
## Model 2: default ~ balance + income + student  
## Resid. Df Resid. Dev Df Deviance Pr(>Chi)   
## 1 6045 908.69   
## 2 6043 895.02 2 13.668 0.001076 \*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

##### pseudo

Unlike linear regression with ordinary least squares estimation, there is no statistic which explains the proportion of variance in the dependent variable that is explained by the predictors. However, there are a number of pseudo metrics that could be of value. Most notable is [McFadden’s](http://stats.stackexchange.com/questions/82105/mcfaddens-pseudo-r2-interpretation) , which is defined as

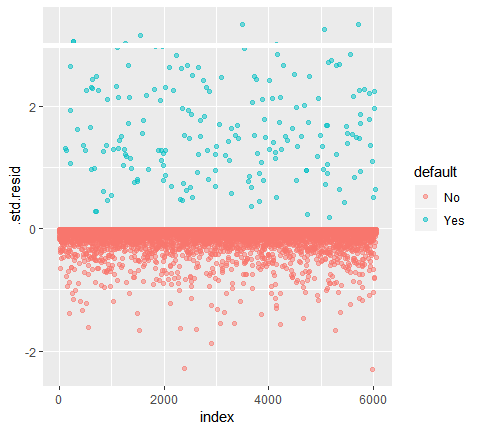
where is the log likelihood value for the fitted model and ln(LM\_0) is the log likelohood for the null model with only an intercept as a predictor. The measure ranges from 0 to just under 1, with values closer to zero indicating that the model has no predictive power. However, unlike in linear regression, models rarely achieve a high McFadden . In fact, in McFadden窶冱 own words models with a McFadden pseudo represents a very good fit. We can assess McFadden’s pseudo values for our models with:

list(  
 model1 = pscl::pR2(model1)["McFadden"],   
 model2 = pscl::pR2(model2)["McFadden"],   
 model3 = pscl::pR2(model3)["McFadden"])  
## $model1  
## McFadden   
## 0.4726215   
##   
## $model2  
## McFadden   
## 0.004898314   
##   
## $model3  
## McFadden   
## 0.4805543

##### Residual assessment

Keep in mind that logistic regression does not assume the residuals are normally distributed nor that the variance is constant. However, the deviance residual is useful for determining if individual points are not well fit by the model. Here we can fit the standardized deviance residuals to see how many exceed 3 standard deviations. First we extract several useful bits of model results with augment and then proceed to plot.

model1\_data <- augment(model1) %>%   
 mutate(index = 1:n())  
  
ggplot(model1\_data, aes(index, .std.resid, col=default))+  
 geom\_point(alpha = .5)+  
 geom\_ref\_line(h = 3)

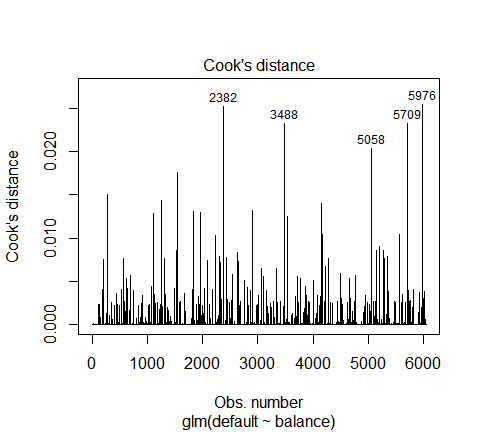


Those standardized residuals that exceed 3 represent possible outliers and may deserve closer attention. We can filter for these residuals to get a closer look. We see that all these observations represent customers who defaulted with budgets that are much lower than the normal defaulters.

model1\_data %>%   
 filter(abs(.std.resid) > 3)  
## # A tibble: 8 x 10  
## default balance .fitted .se.fit .resid .hat .sigma .cooksd .std.resid  
## <fct> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 Yes 1119. -4.66 0.175 3.06 2.84e-4 0.386 0.0151 3.06  
## 2 Yes 1119. -4.66 0.175 3.06 2.84e-4 0.386 0.0151 3.06  
## 3 Yes 1135. -4.57 0.171 3.03 2.97e-4 0.386 0.0144 3.03  
## 4 Yes 1067. -4.96 0.189 3.15 2.46e-4 0.386 0.0175 3.15  
## 5 Yes 961. -5.56 0.216 3.33 1.80e-4 0.385 0.0232 3.33  
## 6 Yes 1144. -4.52 0.169 3.01 3.03e-4 0.386 0.0140 3.01  
## 7 Yes 1013. -5.26 0.203 3.25 2.11e-4 0.385 0.0203 3.25  
## 8 Yes 962. -5.55 0.216 3.33 1.80e-4 0.385 0.0232 3.33  
## # ... with 1 more variable: index <int>

Similar to linear regression we can also identify influential observations with Cook窶冱 distance values. Here we identify the top 5 largest values.

plot(model1, which = 4, id.n = 5)



And we can investigate these further as well. Here we see that the top five influential points include:

* hose customers who defaulted with very low balances and
* two customers who did not default, yet had balances over $2,000

This means if we were to remove these observations (not recommended), the shape, location, and confidence interval of our logistic regression S-curve would likely shift.

model1\_data %>%   
 top\_n(5, .cooksd)  
## # A tibble: 5 x 10  
## default balance .fitted .se.fit .resid .hat .sigma .cooksd .std.resid  
## <fct> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 No 2388. 2.53 0.241 -2.28 3.98e-3 0.387 0.0252 -2.29  
## 2 Yes 961. -5.56 0.216 3.33 1.80e-4 0.385 0.0232 3.33  
## 3 Yes 1013. -5.26 0.203 3.25 2.11e-4 0.385 0.0203 3.25  
## 4 Yes 962. -5.55 0.216 3.33 1.80e-4 0.385 0.0232 3.33  
## 5 No 2391. 2.55 0.242 -2.29 3.95e-3 0.387 0.0254 -2.30  
## # ... with 1 more variable: index <int>

#### Validation of predicted values

##### Classification rate

When developing models for prediction, the most critical metric is regarding how well the model does in predicting the target variable on out-of-sample observations. First, we need to use the estimated models to predict values on our training data set(train). When using predict be sure to include type=response so that the prediction returns the probability of default.

test.predicted.m1 <- predict(model1, newdata = test, type = "response")  
test.predicted.m2 <- predict(model2, newdata = test, type = "response")  
test.predicted.m3 <- predict(model3, newdata = test, type = "response")

Now we can compare the predicted target variable versus the observed values for each model and see which performs the best. We can start by using the confusion matrix, which is a table that describes the classification performance for each model on the test data. Each quadrant of the table has an important meaning. In this case the 窶廸o窶? and 窶弸es窶? in the rows represent whether customers defaulted or not. The 窶廡ALSE窶? and 窶弋RUE窶? in the columns represent whether we predicted customers to default or not.

* **true positives** (Bottom-right quadrant): these are cases in which we predicted the customer would default and they did.
* **true negatives** (Top-left quadrant): We predicted no default, and the customer did not default.
* **false positives** (Top-right quadrant): We predicted yes, but they didn窶冲 actually default. (Also known as a 窶弋ype I error.窶?)
* **false negatives** (Bottom-left): We predicted no, but they did default. (Also known as a 窶弋ype II error.窶?)

The results show that model1 and model3 are very similar. 96% of the predicted observations are true negatives and about 1% are true positives. Both models have a type II error of less than 3% in which the model predicts the customer will not default but they actually did. And both models have a type I error of less than 1% in which the models predicts the customer will default but they never did. model2 results are notably different; this model accurately predicts the non-defaulters (a result of 97% of the data being non-defaulters) but never actually predicts those customers that default!

list(  
 model1 = table(test$default, test.predicted.m1 > 0.5) %>% prop.table() %>% round(3),  
 model2 = table(test$default, test.predicted.m2 > 0.5) %>% prop.table() %>% round(3),  
 model3 = table(test$default, test.predicted.m3 > 0.5) %>% prop.table() %>% round(3))  
## $model1  
##   
## FALSE TRUE  
## No 0.962 0.003  
## Yes 0.025 0.010  
##   
## $model2  
##   
## FALSE  
## No 0.965  
## Yes 0.035  
##   
## $model3  
##   
## FALSE TRUE  
## No 0.963 0.003  
## Yes 0.026 0.009

We also want to understand the misclassification (aka *error*) rates (or we could flip this for the accuracy rates). . We don窶冲 see much improvement between models 1 and 3 and although model 2 has a low error rate don窶冲 forget that it never accurately predicts customers that actually default.

test %>%  
 mutate(m1.pred = ifelse(test.predicted.m1 > 0.5, "Yes", "No"),  
 m2.pred = ifelse(test.predicted.m2 > 0.5, "Yes", "No"),  
 m3.pred = ifelse(test.predicted.m3 > 0.5, "Yes", "No")) %>%  
 summarise(m1.error = mean(default != m1.pred),  
 m2.error = mean(default != m2.pred),  
 m3.error = mean(default != m3.pred))  
## # A tibble: 1 x 3  
## m1.error m2.error m3.error  
## <dbl> <dbl> <dbl>  
## 1 0.0278 0.0349 0.0281  
## # A tibble: 1 ﾃ? 3  
## m1.error m2.error m3.error  
## <dbl> <dbl> <dbl>  
## 1 0.02782697 0.03491019 0.02807994

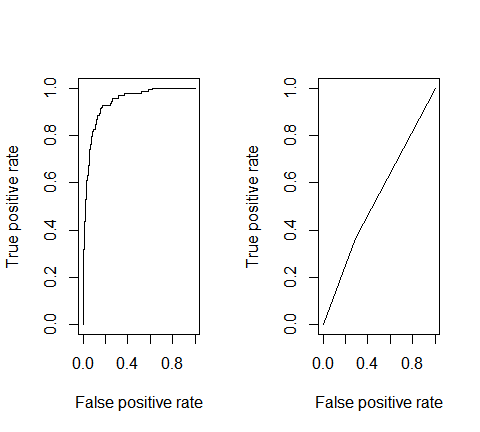
We can gain some additional insights by looking at the raw values (not percentages) in our confusion matrix. Lets look at model 1 to illustrate. We see that there are a total of customers that defaulted. Of the total defaults, were not predicted. Alternatively, we could say that only of default occurrences were predicted - this is known as the the precision (also known as sensitivity) of our model. So while the overall error rate is low, the precision rate is also low, which is not good!

table(test$default, test.predicted.m1>0.5)  
##   
## FALSE TRUE  
## No 3803 12  
## Yes 98 40

With classification models you will also here the terms *sensitivity* and *specificity* when characterizing the performance of the model. As mentioned above sensitivity is synonymous to precision. However, the specificity is the percentage of non-defaulters that are correctly identified, here (the accuracy here is largely driven by the fact that 97% of the observations in our data are non-defaulters). The importance between *sensitivityy* and *specificity* is dependent on context. In this case, a credit card company is likely to be more concerned with sensititivy since they want to reduce their risk. Therefore, they may be more concerned with tuning a model so that their *sensititivy*/*precision* is improved.

The receiving operating characteristic (ROC) is a visual measure of classifier performance. Using the proportion of positive data points that are correctly considered as positive and the proportion of negative data points that are mistakenly considered as positive, we generate a graphic that shows the trade off between the rate at which you can correctly predict something with the rate of incorrectly predicting something. Ultimately, we窶决e concerned about the area under the ROC curve, or AUC. That metric ranges from 0.50 to 1.00, and values above 0.80 indicate that the model does a good job in discriminating between the two categories which comprise our target variable. We can compare the ROC and AUC for model窶冱 1 and 2, which show a strong difference in performance. We definitely want our ROC plots to look more like model 1窶冱 (left) rather than model 2窶冱 (right)!

library(ROCR)  
  
par(mfrow=c(1, 2))  
  
prediction(test.predicted.m1, test$default) %>%  
 performance(measure = "tpr", x.measure = "fpr") %>%  
 plot()  
  
prediction(test.predicted.m2, test$default) %>%  
 performance(measure = "tpr", x.measure = "fpr") %>%  
 plot()



And to compute the AUC numerically we can use the following. Remember, AUC will range from .50 - 1.00. Thus, model 2 is a very poor classifying model while model 1 is a very good classying model.

# model1 AUC  
prediction(test.predicted.m1, test$default) %>%   
 performance(measure = "auc") %>%   
 .@y.values  
## [[1]]  
## [1] 0.939932  
## [[1]]  
## [1] 0.939932  
  
# model 2 AUC  
prediction(test.predicted.m2, test$default) %>%  
 performance(measure = "auc") %>%  
 .@y.values  
## [[1]]  
## [1] 0.5386955  
## [[1]]  
## [1] 0.5386955

We can continue to 窶徼une窶? our models to improve these classification rates. If you can improve your AUC and ROC curves (which means you are improving the classification accuracy rates) you are creating \_窶徑ift窶拈, meaning you are lifting the classification accuracy.

#### Additional resources

This will get you up and running with logistic regression. Keep in mind that there is a lot more you can dig into so the following resources will help you learn more:

An Introduction to Statistical Learning Applied Predictive Modeling Elements of Statistical Learning