ML Basic

JEFFREY CHAN

12/23/2020

Linear Regression Essentials in R

load / install the requirement packages

```r
library(tidyverse)
```

```
## -- Attaching packages ----------

## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.3     v dplyr   1.0.2
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.0

## -- Conflicts --------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
library(caret)
```

```
## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift
```

```r
theme_set(theme_bw())
```

Preparing the data

sample_n(data, sample data), will randomly pick # of sample data

```r
data("marketing", package = "datarium")
sample_n(marketing,3)
```

```
##   youtube facebook newspaper sales
## 1  268.80     2.88     18.72 13.92
## 2   15.72     0.48     30.72  6.36
## 3  238.92    36.72     46.44 21.96
```

```r
# split the data into training and test set
set.seed(123)

# We'll randomly split the data into training set (80% for building a
# predictive model) and test set (20% for evaluating the model).
```

```
training.samples <- marketing$sales %>%
  createDataPartition(p=0.8, list = F)
train.data <- marketing[training.samples, ]
test.data <- marketing[-training.samples, ]

# build model
model <- lm(sales ~., data = train.data)

# summarize the model
summary(model)
```

```
##
## Call:
## lm(formula = sales ~ ., data = train.data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -10.7142  -0.9939   0.3684   1.4494   3.3619
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 3.594142   0.420815   8.541 1.05e-14 ***
## youtube     0.044636   0.001552  28.758  < 2e-16 ***
## facebook    0.188823   0.009529  19.816  < 2e-16 ***
## newspaper   0.002840   0.006442   0.441     0.66
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.043 on 158 degrees of freedom
## Multiple R-squared:  0.8955, Adjusted R-squared:  0.8935
## F-statistic: 451.2 on 3 and 158 DF,  p-value: < 2.2e-16
```

```
summary(model)$coefficient
```

```
##                Estimate  Std. Error    t value     Pr(>|t|)
## (Intercept) 3.594141778 0.420814685  8.5409134 1.054115e-14
## youtube     0.044635905 0.001552128 28.7578721 1.125356e-64
## facebook    0.188823227 0.009528828 19.8159966 1.090250e-44
## newspaper   0.002839757 0.006441963  0.4408217 6.599447e-01
```

```
# make prediction
predictions <- model %>% predict(test.data)

# model performance
# (a) Prediction error, RMSE
RMSE(predictions, test.data$sales)
```

```
## [1] 1.965508
```

```
# (b) R-square
caret::R2(predictions, test.data$sales)
```

```
## [1] 0.9049049
```

Simple Linear Regression

```r
model_you <- lm(sales ~ youtube, data = train.data)
summary(model_you)$coef
```

```
##               Estimate  Std. Error  t value     Pr(>|t|)
## (Intercept) 8.58914961 0.616044182 13.94242 1.987874e-29
## youtube     0.04671639 0.003003398 15.55451 8.019035e-34
```

Make prediction

```r
newdata <- data.frame(youtube = c(0,1000))
model_you %>% predict(newdata)
```

```
##        1        2
##  8.58915 55.30554
```

Multiple Linear Regression

```r
model3 <- lm(sales ~ youtube + facebook + newspaper, data = train.data)
summary(model3)$coef
```

```
##                Estimate  Std. Error    t value     Pr(>|t|)
## (Intercept) 3.594141778 0.420814685  8.5409134 1.054115e-14
## youtube     0.044635905 0.001552128 28.7578721 1.125356e-64
## facebook    0.188823227 0.009528828 19.8159966 1.090250e-44
## newspaper   0.002839757 0.006441963  0.4408217 6.599447e-01
```

Quick note, i have a lot of predictor, we can simply use ~. to include all the predictor

```r
model3_1 <- lm(sales ~., data = train.data)
summary(model3_1)$coef
```

```
##                Estimate  Std. Error    t value     Pr(>|t|)
## (Intercept) 3.594141778 0.420814685  8.5409134 1.054115e-14
## youtube     0.044635905 0.001552128 28.7578721 1.125356e-64
## facebook    0.188823227 0.009528828 19.8159966 1.090250e-44
## newspaper   0.002839757 0.006441963  0.4408217 6.599447e-01
```

Col1: b0 / y intercept is 3.73546064 Col2: std.error = 0.44062. this represent the accuracy of the coefficients. We always want small value for std.error Col3: T value is the t statistics estimate / std error = t value Col4: P value for the T statistics. The smaller the p value the more significant the estimate is.

Let's make prediction for the values

```r
# New advertising budget
newdata <- data.frame( youtube = 2000, facebook = 1000, newspaper = 1000)

# predict sales values
model3_1 %>% predict(newdata)
```

```
##        1
## 284.5289
```

Interpretation.

Before using the model for predictions, i need to access the statsitical significance of the model. We need to apply summary(model_name)

```r
summary(model)
```

```
##
## Call:
```

```
## lm(formula = sales ~ ., data = train.data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -10.7142  -0.9939   0.3684   1.4494   3.3619
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 3.594142   0.420815   8.541 1.05e-14 ***
## youtube     0.044636   0.001552  28.758  < 2e-16 ***
## facebook    0.188823   0.009529  19.816  < 2e-16 ***
## newspaper   0.002840   0.006442   0.441     0.66
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.043 on 158 degrees of freedom
## Multiple R-squared:  0.8955, Adjusted R-squared:  0.8935
## F-statistic: 451.2 on 3 and 158 DF,  p-value: < 2.2e-16
```

The residuals, should be normally distributed. Looks like our data is kind of normally distributed. by theory, mean should be zero. Q1 and Q3, min and max should be around the same with + or - sign.

coefficients, shows the parameter values and their significance. If they are significant, it will be shown with starts

RSE / R^2 / F statistics are used to check how well the model fits to our data

First step, always check the F statistics and the associated p value and the bottom of the model summary

Coefficients significance

```r
summary(model)$coef
```

```
##                 Estimate  Std. Error    t value     Pr(>|t|)
## (Intercept) 3.594141778 0.420814685  8.5409134 1.054115e-14
## youtube     0.044635905 0.001552128 28.7578721 1.125356e-64
## facebook    0.188823227 0.009528828 19.8159966 1.090250e-44
## newspaper   0.002839757 0.006441963  0.4408217 6.599447e-01
```

Important, from the summary table , we can see that youtube, facebook advertising budget are significantly changing the sales while by newspaper, there is not much of a change.

Also, to interpret the intercept, we can say every 1000 dollar i invest in facebook advertising i will have a return of 1000* 0.19398450 = 193 sale unit. So do youtube, 1000* 0.04516611 = 45.16 sale units

Since newspaper does not affect the outcome much, lets remove it from the model

```r
model4 <- lm(sales ~ youtube + facebook, data = train.data)
summary(model4)
```

```
##
## Call:
## lm(formula = sales ~ youtube + facebook, data = train.data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -10.8127  -1.0073   0.3236   1.4643   3.3454
##
## Coefficients:
```

```
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 3.658580   0.393609   9.295   <2e-16 ***
## youtube     0.044650   0.001548  28.846   <2e-16 ***
## facebook    0.190165   0.009006  21.114   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 2.038 on 159 degrees of freedom
## Multiple R-squared:  0.8954, Adjusted R-squared:  0.894 
## F-statistic: 680.2 on 2 and 159 DF,  p-value: < 2.2e-16
```

now our equation can be written as sales = 3.577663 + 0.045287(youtube) + 0.190299(facebook)

Model accuracy

Well, after knowing it is significant, we would like to know how well the model fits our data. This process will be referred to as the goodness of fit The overall quality of the linear regression fit can be shown by model summary RSE, R^2, adjusted R^2, F statistics

RSE / model sigma -> prediction error. it is the observed out come - predicted value (Y_i - Y bar). small rse is the best the model fits to the data. dividing the RSE by the average value of the outcome variable will give me the prediction error rate. Which should be as small as possible.

in our example, we have 1.853 for RSE.

```
(1.853/(mean(train.data$sales)))*100
```

```
## [1] 10.99196
```

It is 10.90% which is low.

R-squared and adjusted R squared R^2 is ranges between 0 and 1. Higher the R square the better the model. High R square = observed data is very close to the prediction data. So the quality of the regression line is pretty good.R^2 = pearson ^2. Here is the trick tho, if i have a lot of parameters, R^2 will increase along with it.

However, the adjusted R^2 is the correction of the for the number of parameters in the predictive model.Therefore, I should always read adjusted R^2 because it will make the correction according the to incorrect R^2

When adjusted R^2 = 0 thats mean the model did not explain much about the variability in the outcome. In our outcome, adjusted R^2 is 0.9112 so it is pretty good.

Lastly, F statistics gives the overall significance of the model. it tells us whether at least one predictor variables has non zero coefficient. In a simple linear regression (one parameter), it wont be interesting because it is just a duplicated info given by the t test from the coef table.

The F statistic becomes more important once we start using multiple predictors as in multiple linear regression.

So according to what we have, our F statistics equal 825.4 with 2 parameters and 159 df, with a p-value of 2.2e^-16. which is highly significant. In order to read the p value, $p < 0.05$ will be significant.

Making predictions

Procedure to make predictions 1) predict the sales values based on new advertising budgets in the test data 2) Assess the model performance by computing: the prediction error RMSE (Root Mean Squared Error), representing the average difference between the observed known outcome values in the test data and the predicted outcome values by the model. The lower the RMSE, the better the model. The R^2, representing the correlation between the observed outcome values and the predicted outcome values. the higher the r^2, the better the model.

```
# make predictions
predictions <- model %>% predict(test.data)

# model performance
# (a) compute the prediction error, RMSE
RMSE(predictions, test.data$sales)
```

```
## [1] 1.965508
```

```
caret::R2(predictions, test.data$sales)
```

```
## [1] 0.9049049
```

```
(RMSE(predictions, test.data$sales)/ mean(test.data$sales))*100
```

```
## [1] 11.77248
```

The % error is 16.39% is alright

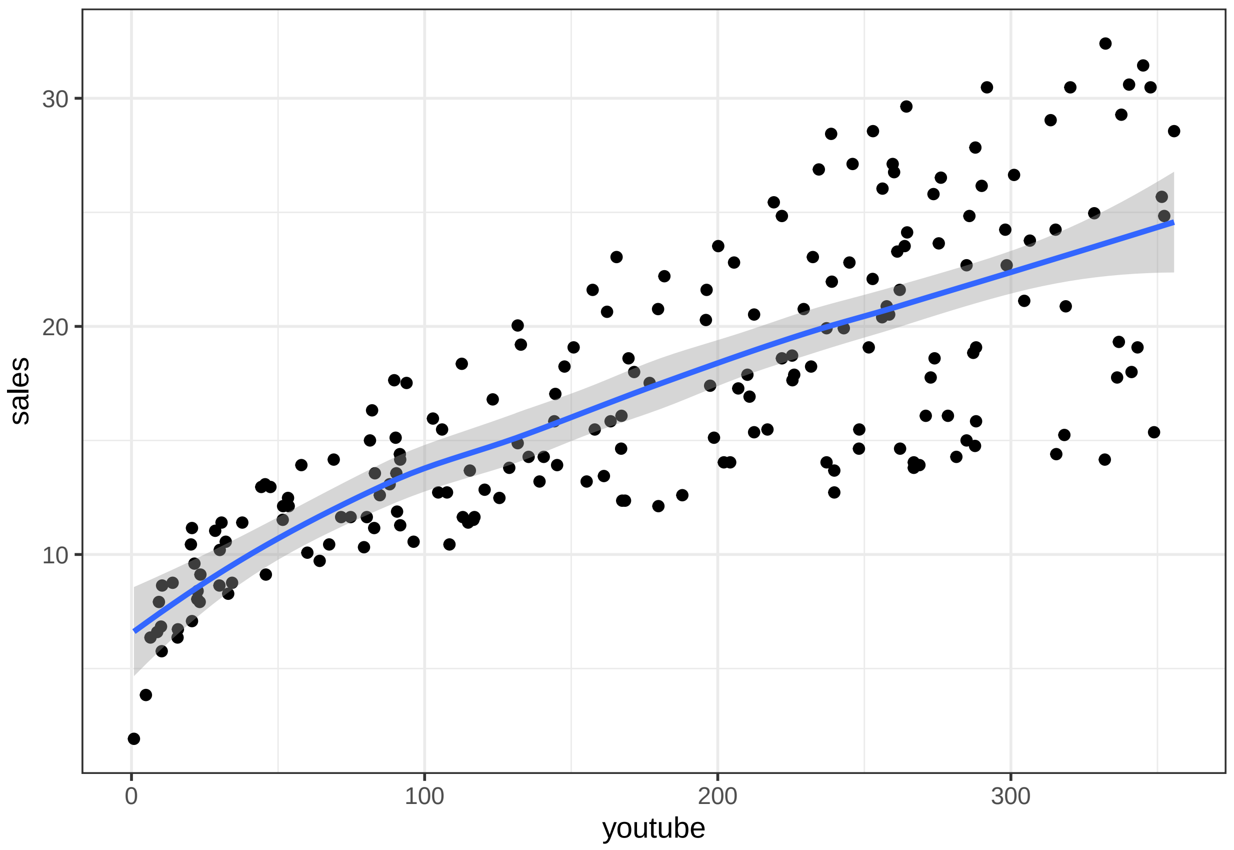
Discussion

This section discuss basic linear regression and provides practical examples in R for computing simple and multiple linear regression model. Also, we have learnt how the accuracy of the model.

The idea of linear regression is to see the predictor relationship with the response. It can be easily be visualized by a basic plot without working a lot on lm() summary and more.

```
ggplot(marketing, aes(youtube, sales)) +
  geom_point() +
  stat_smooth()
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



As we can expected, since we know that the Beta_1 is positive from what we have done before, we can expect that the slope is positive and the scatter dots are following the positive slope.

Interaction Effect in Multiple Regression: Essentials

Considering our example, the additive model assumes that, the effect on sales of youtube advertising is independent of the effect of facebook advertising.

This assumption might not be true. For example, spending money on facebook advertising may increase the effectiveness of youtube advertising on sales. In marketing, this is known as a synergy effect, and in statistics it is referred to as an interaction effect (James et al. 2014).

Interaction effects

```
# build the model
# use this:
model12 <- lm(sales ~ youtube*facebook, data = train.data)
summary(model12)
```

```
##
## Call:
## lm(formula = sales ~ youtube * facebook, data = train.data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.6325 -0.5051  0.2666  0.7425  1.8109
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)      8.179e+00  3.306e-01  24.741  < 2e-16 ***
## youtube          1.859e-02  1.660e-03  11.196  < 2e-16 ***
## facebook         2.781e-02  1.016e-02   2.739  0.00687 **
## youtube:facebook 9.145e-04  4.952e-05  18.467  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.15 on 158 degrees of freedom
## Multiple R-squared:  0.9669, Adjusted R-squared:  0.9662
## F-statistic:  1537 on 3 and 158 DF,  p-value: < 2.2e-16
```

Make predictions

```
predictions12 <- model12 %>% predict(test.data)
# model performance
# (a) prediction error, RMSE
RMSE(predictions12, test.data$sales)
```

```
## [1] 1.055644
```

```
# (b) R-square
caret::R2(predictions12, test.data$sales)
```

```
## [1] 0.9716356
```

```
# % error
(RMSE(predictions12, test.data$sales) / mean(test.data$sales))*100
```

```
## [1] 6.322813
```

Those values are pretty good. high R^2 and 10.67% error

```
summary(model12)$coef
```

```
##                      Estimate   Std. Error   t value     Pr(>|t|)
## (Intercept)      8.1786320203 3.305698e-01 24.741018 3.213476e-56
## youtube          0.0185856131 1.659981e-03 11.196278 8.614530e-22
## facebook         0.0278145861 1.015569e-02  2.738818 6.874607e-03
## youtube:facebook 0.0009145105 4.952241e-05 18.466598 2.668477e-41
```

As you can see that all of them are significant (Pr(>|t|)). so it means there is an interaction relationship between the two predictor variables (youtube and facebook advertising)

our model will be like sales = 7.89 + 0.0189949052(youtube) + 0.0330506939(facebook) + 0.0008751096 (youtube*facebook)

comparing the additive and the interaction models.

The prediction error RMESE of the interaction model is 1.721177 compared with the prediction error of the addictive model 2.642146, interaction model is lower.

R^2 of the interaction model is 0.9373706, and for the addictive model has 0.8322611. Interaction model has a better R^2

Lastly, these result suggest that the model with the interaction term is better than the model that contains only main effects.

So for this specific data, we should go for the model with the interaction model.

Discussion, after finding addictive model which is significant, we should also check if the interaction model is also significant. If they do, we should adapt the interaction model.

Regression with Categorical Variables: Dummy Coding Essentials in R

```
library(car)
```

```
## Loading required package: carData

##
## Attaching package: 'car'

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

## The following object is masked from 'package:purrr':
##
##     some
```

```
# load data
data("Salaries", package = "car")
```

```
## Warning in data("Salaries", package = "car"): data set 'Salaries' not found
```

```
# Inspect the data
sample_n(Salaries, 3)
```

```
##       rank discipline yrs.since.phd yrs.service    sex salary
## 1     Prof          A            22          15   Male 166800
## 2     Prof          A            39          36 Female 137000
## 3 AsstProf          A            11           4   Male  78785
```

Categorical variables with two levels From our data set, we would like to investigate differences in salaries between males and females. Based on the gender, we can say m = 1, female = 0 b0 + b1 if person is male bo if person is female. B0 is average salary among females B0 + B1 = average salary among males B1 is average difference in salary between males and females

```
mwage <- lm(salary ~ sex, data = Salaries)
summary(mwage)$coef
```

```
##              Estimate Std. Error   t value     Pr(>|t|)
## (Intercept) 101002.41   4809.386 21.001103 2.683482e-66
## sexMale      14088.01   5064.579  2.781674 5.667107e-03
```

From the above result, average salary for female is 101002.41. Male = B0 + B1 = 101002.41 + 14088.01 = 115090.40

the P value for both sex is very significant, which is suggesting that there is a statistical evidence of a difference in average salary between the genders.

```
contrasts(Salaries$sex)
```

```
##        Male
## Female    0
## Male      1
```

I can use the function relevel() to set the baseline category to males as follow

```
Salaries <- Salaries %>%
  mutate(sex = relevel(sex, ref = "Male"))
```

```
mwage1 <- lm(salary ~ sex , data = Salaries)
summary(mwage1)$coef
```

```
##              Estimate Std. Error   t value      Pr(>|t|)
## (Intercept) 115090.42   1587.378 72.503463 2.459122e-230
## sexFemale   -14088.01   5064.579 -2.781674  5.667107e-03
```

Categorical variables with more than two levels

Generally, categorical variable with n levels will be transformed in to n-1 variables each with two levels. These n-1 new variables contain the same info than the single variable. This recording creates a table called contrast matrix.

In salaries, data has three levels, asstprof, assocprof, and prof. These variables could be dummy coded in to two variables, one called assocprof and one prof.

if rank = assocprof, then the column assocprof would be coded with 1 and prof with 0 if rank = prof, then the col assocprof would be coded with a 0 and prof would be coded with a 1 if rank = asstprof then both cols assocprof and prof would be coded with a 0

```r
res <- model.matrix(~rank, data=Salaries)
head(res[,-1])
```

```
##   rankAssocProf rankProf
## 1             0        1
## 2             0        1
## 3             0        0
## 4             0        1
## 5             0        1
## 6             1        0
```

```r
head(res)
```

```
##   (Intercept) rankAssocProf rankProf
## 1           1             0        1
## 2           1             0        1
## 3           1             0        0
## 4           1             0        1
## 5           1             0        1
## 6           1             1        0
```

R will always use the first level as a reference and interpret the remaining levels relative to the first level

The following code will give you the level

```r
levels(Salaries$rank)
```

```
## [1] "AsstProf"  "AssocProf" "Prof"
```

Indeed, AsstProf will be the reference level.

Also ANOVA(analysis of variance) is just a special case of linear model where the predictors are categorical variables. R understands the fact that ANOVA and regression are both examples of linear models, it lets you extract the classic ANOVA table from the regression model using the R base anova() function or the Anova() function. We generally use Anova() function because it automatically takes care of unbalanced designs.

Lets predict the salary from using a multiple regression procedure

```r
mwage2 <- lm(salary ~ yrs.service + rank + discipline + sex, data = Salaries)
Anova(mwage2)
```

```
## Anova Table (Type II tests)
##
## Response: salary
##                 Sum Sq  Df  F value    Pr(>F)
## yrs.service 3.2448e+08   1   0.6324    0.4270
## rank        1.0288e+11   2 100.2572 < 2.2e-16 ***
## discipline  1.7373e+10   1  33.8582 1.235e-08 ***
## sex         7.7669e+08   1   1.5137    0.2193
## Residuals   2.0062e+11 391
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
Anova(mwage1)
```

```
## Anova Table (Type II tests)
##
## Response: salary
##               Sum Sq  Df F value   Pr(>F)
## sex       6.9800e+09   1  7.7377 0.005667 **
## Residuals 3.5632e+11 395
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

When we take rank, discipline, yrs of service in to consideration, the variable sex is no longer significant combined with the variation in salary between individuals.

```r
summary(mwage2)
```

```
##
## Call:
## lm(formula = salary ~ yrs.service + rank + discipline + sex,
##     data = Salaries)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -64202 -14255  -1533  10571  99163
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)   73122.92    3245.27  22.532  < 2e-16 ***
## yrs.service     -88.78     111.64  -0.795 0.426958
## rankAssocProf 14560.40    4098.32   3.553 0.000428 ***
## rankProf      49159.64    3834.49  12.820  < 2e-16 ***
## disciplineB   13473.38    2315.50   5.819 1.24e-08 ***
## sexFemale     -4771.25    3878.00  -1.230 0.219311
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 22650 on 391 degrees of freedom
## Multiple R-squared:  0.4478, Adjusted R-squared:  0.4407
## F-statistic: 63.41 on 5 and 391 DF,  p-value: < 2.2e-16
```

```r
levels(Salaries$discipline)
```

```
## [1] "A" "B"
```

apply ??salaries and check discipline a and b is corresponding to what field from the documentation, dis A is theoretical, dis B is applied department.

For example, from discipline B (applied departments) is significantly associated with an average increase of 13473.38 in salary compared (there is a difference) to discipline theoretical departments. So that is the reason why discipline B is significant

Nonlinear Regression Essentials in R: Polynomial and Spline Regression Models

http://www.sthda.com/english/articles/40-regression-analysis/162-nonlinear-regression-essentials-in-r-polynomial-and-spline-regression-models/

In this section, you'll learn how to compute non-linear regression models and how to compare the different models in order to choose the one that fits the best your data.

Will also be using RMSE and R2 metric to compare the different models' accuracy Recall, RMSE -> model prediction error. Thats the average difference of the observed outcome values and predicted outcome values.

R2 represents the squared correlation. How accurate is the data, if it is exactly on the regression line, that the R^2 will be 1

The best model is the model with the lowest RMSE and the highest R2

```r
library(tidyverse)
library(caret)
theme_set(theme_classic())
```

Prepare the data

we will use the boston data from MASS package

```r
# Load the data, if the package is not installed, instal it now and load the library
if(!require("MASS")){
  install.packages("MASS")
  library(MASS)
}
```

```
## Loading required package: MASS
```

```
##
## Attaching package: 'MASS'
```

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

```r
data("Boston", package = "MASS")
str(Boston)
```

```
## 'data.frame':    506 obs. of  14 variables:
##  $ crim   : num  0.00632 0.02731 0.02729 0.03237 0.06905 ...
##  $ zn     : num  18 0 0 0 0 0 12.5 12.5 12.5 12.5 ...
##  $ indus  : num  2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7.87 7.87 ...
##  $ chas   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ nox    : num  0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524 0.524 0.524 ...
##  $ rm     : num  6.58 6.42 7.18 7 7.15 ...
##  $ age    : num  65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9 ...
##  $ dis    : num  4.09 4.97 4.97 6.06 6.06 ...
##  $ rad    : int  1 2 2 3 3 3 5 5 5 5 ...
##  $ tax    : num  296 242 242 222 222 222 311 311 311 311 ...
##  $ ptratio: num  15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 15.2 ...
##  $ black  : num  397 397 393 395 397 ...
##  $ lstat  : num  4.98 9.14 4.03 2.94 5.33 ...
##  $ medv   : num  24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 ...
```

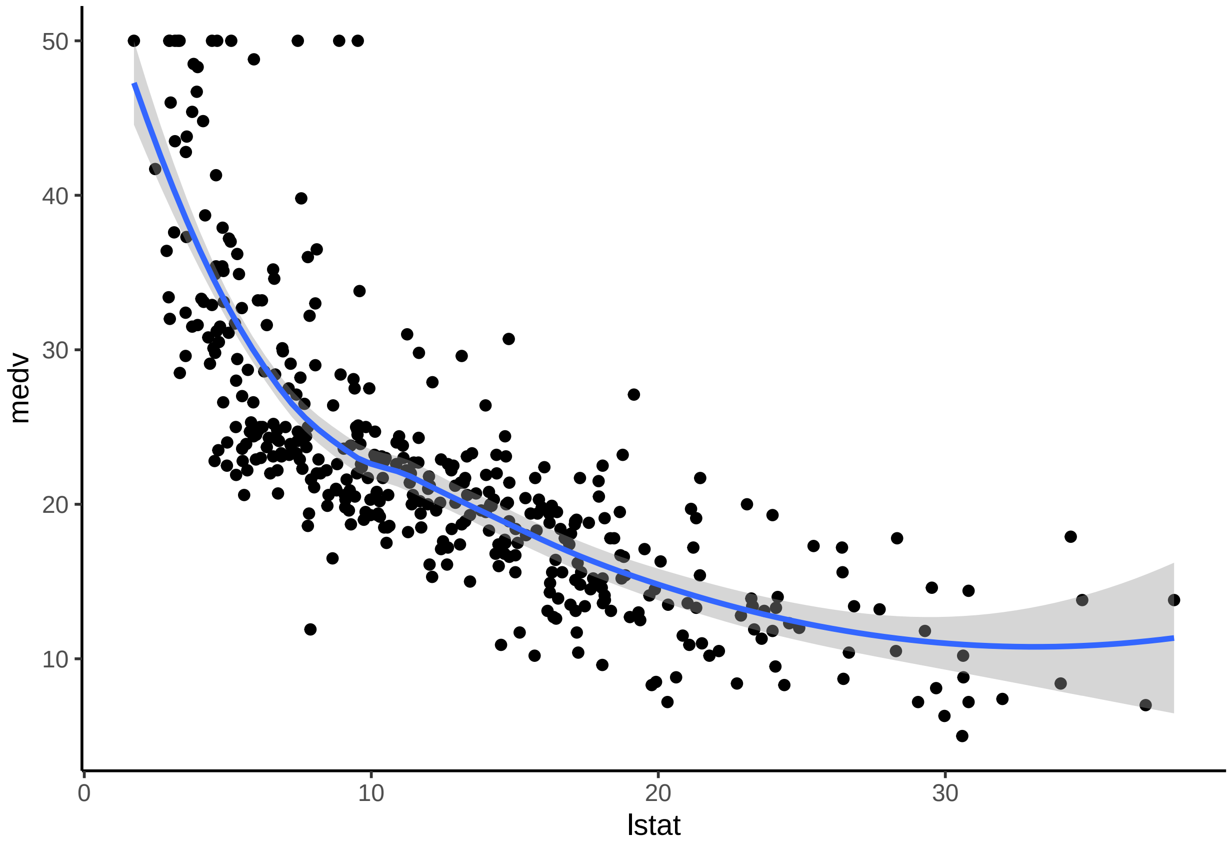
```r
# Split the data in to training and test set
set.seed(123)
ts <- Boston$medv %>%
  createDataPartition(p =0.8, list = F)
trd <- Boston[ts, ]
ted <- Boston[-ts, ]
```

Visualize the scatter plot of the medv vs lstat varaibles, both medv and lstat is from the Boston dataset

use stat_smooth when i want to display the results with non standard geom

```
ggplot(trd, aes(lstat,medv)) +
  geom_point() +
  stat_smooth()
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



It is obvious to see that the blue line is a curve

Linear Regression

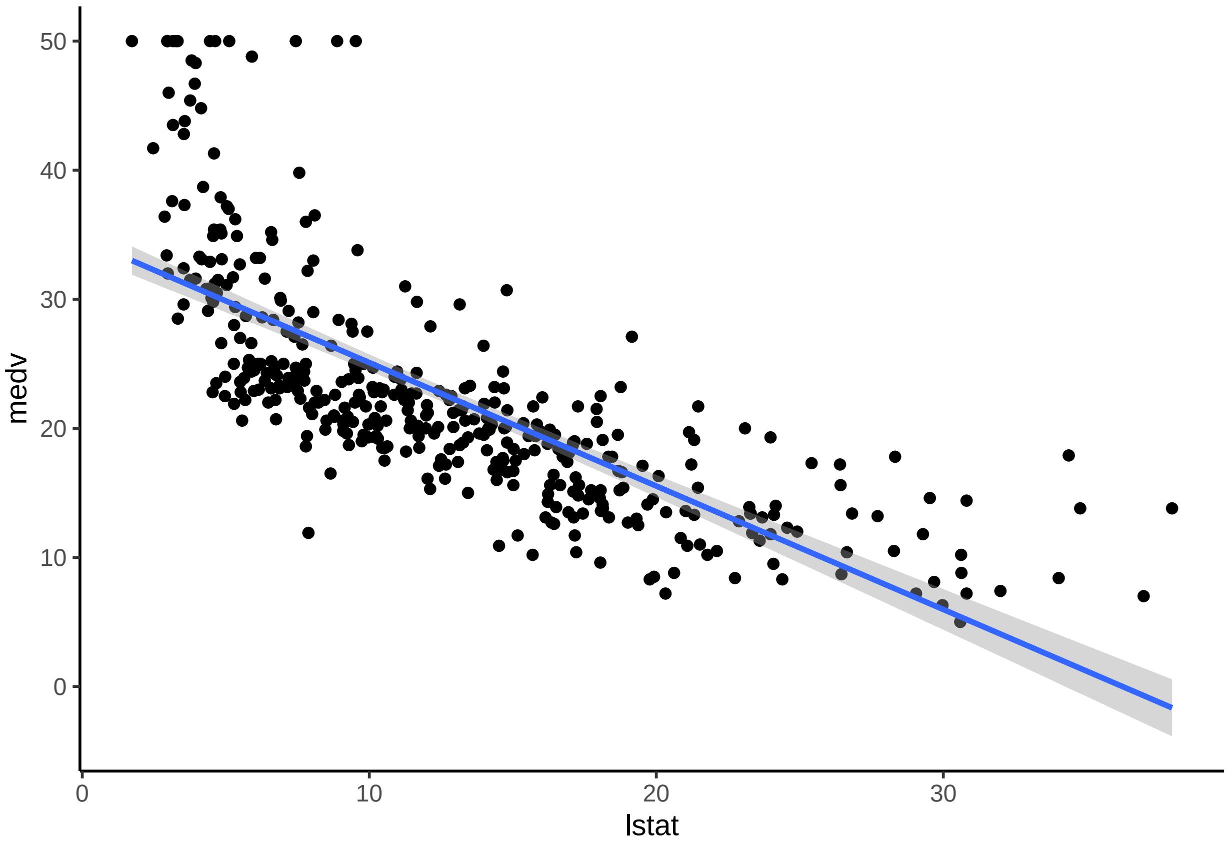
```
# Build the model
modelc <- lm(medv ~ lstat, data = trd)
# make prediction
pmodelc <- modelc %>% predict(ted)
data.frame(
  rmseL <- RMSE(pmodelc, ted$medv),
  r2L <- caret::R2(pmodelc, ted$medv),
  perrorL <- rmseL / mean(ted$medv)
)
```

```
##   rmseL....RMSE.pmodelc..ted.medv. r2L....caret..R2.pmodelc..ted.medv.
## 1                         6.503817                              0.513163
##   perrorL....rmseL.mean.ted.medv.
## 1                       0.2874712
```

I have a pretty high percent error 26.82%

visualize the data

```
ggplot(trd, aes(lstat, medv)) +
  geom_point() +
  stat_smooth(method = lm, formula = y ~ x)
```



Polynomial Regression

Lets make it polynomial regression medv is the response.

we should have this following formula medv = b0 + b1 * lstat + b2*lstat^2 in r to make that ^2 we need to apply I(x^2)

```
modelc1 <- lm(medv ~ lstat + I(lstat^2), data = trd)
summary(modelc1)
```

```
##
## Call:
## lm(formula = medv ~ lstat + I(lstat^2), data = trd)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -15.3654  -3.8250  -0.6439   2.2733  25.2922
##
```

```
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 42.573638   0.964887   44.12   <2e-16 ***
## lstat       -2.267309   0.135846  -16.69   <2e-16 ***
## I(lstat^2)   0.041198   0.004095   10.06   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.501 on 404 degrees of freedom
## Multiple R-squared:  0.6418, Adjusted R-squared:   0.64
## F-statistic: 361.9 on 2 and 404 DF,  p-value: < 2.2e-16
```

Alternative way to create 2 degree regression model

```
modelc2<- lm(medv ~ poly(lstat, 2, raw =T), data = trd)
summary(modelc2)
```

```
##
## Call:
## lm(formula = medv ~ poly(lstat, 2, raw = T), data = trd)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -15.3654  -3.8250  -0.6439   2.2733  25.2922
##
## Coefficients:
##                           Estimate Std. Error t value Pr(>|t|)
## (Intercept)              42.573638   0.964887   44.12   <2e-16 ***
## poly(lstat, 2, raw = T)1 -2.267309   0.135846  -16.69   <2e-16 ***
## poly(lstat, 2, raw = T)2  0.041198   0.004095   10.06   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.501 on 404 degrees of freedom
## Multiple R-squared:  0.6418, Adjusted R-squared:   0.64
## F-statistic: 361.9 on 2 and 404 DF,  p-value: < 2.2e-16
```

```
modelc2p <- modelc2 %>% predict(ted)
p2 <- c(RMSE(modelc2p,ted$medv), caret::R2(modelc2p, ted$medv),
       RMSE(modelc2p, ted$medv) / mean(ted$medv))
```

As you can see they are the same. intercepts and the coefficient of beta1 and beta2^2 are all significant. as well as the F statistic P value is also small. Indeed, we have an Adjusted R^2 0.6329 which is ok.

The following is the 6th order

```
modelc6 <- lm(medv ~ poly(lstat, 6, raw = T), data = trd)
summary(modelc6)
```

```
##
## Call:
## lm(formula = medv ~ poly(lstat, 6, raw = T), data = trd)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -13.1962  -3.1527  -0.7655   2.0404  26.7661
##
```

```
## Coefficients:
##                            Estimate Std. Error t value Pr(>|t|)
## (Intercept)               7.788e+01  6.844e+00  11.379  < 2e-16 ***
## poly(lstat, 6, raw = T)1 -1.767e+01  3.569e+00  -4.952 1.08e-06 ***
## poly(lstat, 6, raw = T)2  2.417e+00  6.779e-01   3.566 0.000407 ***
## poly(lstat, 6, raw = T)3 -1.761e-01  6.105e-02  -2.885 0.004121 **
## poly(lstat, 6, raw = T)4  6.845e-03  2.799e-03   2.446 0.014883 *
## poly(lstat, 6, raw = T)5 -1.343e-04  6.290e-05  -2.136 0.033323 *
## poly(lstat, 6, raw = T)6  1.047e-06  5.481e-07   1.910 0.056910 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.188 on 400 degrees of freedom
## Multiple R-squared:  0.6845, Adjusted R-squared:  0.6798
## F-statistic: 144.6 on 6 and 400 DF,  p-value: < 2.2e-16
```

```
modelc6p <- modelc6 %>% predict(ted)
p6 <- c(RMSE(modelc6p,ted$medv), caret::R2(modelc6p, ted$medv),
        RMSE(modelc6p, ted$medv) / mean(ted$medv))
```

From this point we can see that after degree 3 it is no longer significant. Then we will just simply use degree 3 for our model

```
modelc5 <- lm(medv ~ poly(lstat, 5, raw = T), data = trd)
summary(modelc5)
```

```
##
## Call:
## lm(formula = medv ~ poly(lstat, 5, raw = T), data = trd)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -13.1519  -3.1235  -0.5927   2.0962  27.1286
##
## Coefficients:
##                            Estimate Std. Error t value Pr(>|t|)
## (Intercept)               6.765e+01  4.273e+00  15.831  < 2e-16 ***
## poly(lstat, 5, raw = T)1 -1.177e+01  1.786e+00  -6.588 1.40e-10 ***
## poly(lstat, 5, raw = T)2  1.220e+00  2.580e-01   4.727 3.16e-06 ***
## poly(lstat, 5, raw = T)3 -6.385e-02  1.644e-02  -3.884 0.000120 ***
## poly(lstat, 5, raw = T)4  1.577e-03  4.714e-04   3.345 0.000901 ***
## poly(lstat, 5, raw = T)5 -1.459e-05  4.954e-06  -2.945 0.003421 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.205 on 401 degrees of freedom
## Multiple R-squared:  0.6816, Adjusted R-squared:  0.6777
## F-statistic: 171.7 on 5 and 401 DF,  p-value: < 2.2e-16
```

```
modelc5p <- modelc5 %>% predict(ted)
p5 <- c(RMSE(modelc5p,ted$medv), caret::R2(modelc5p, ted$medv), RMSE(modelc5p, ted$medv) / mean(ted$med
modelc3 <- lm(medv ~ poly(lstat, 3, raw = T), data = trd)
modelc3p <- modelc3 %>% predict(ted)
p3 <- c(RMSE(modelc3p,ted$medv), caret::R2(modelc3p, ted$medv), RMSE(modelc3p, ted$medv) / mean(ted$med
modelc4 <- lm(medv ~ poly(lstat, 4, raw = T), data = trd)
modelc4p <- modelc4 %>% predict(ted)
```

```
p4 <- c(RMSE(modelc4p,ted$medv), caret::R2(modelc4p, ted$medv), RMSE(modelc4p, ted$medv) / mean(ted$med
perrorp4 <- RMSE(modelc4p, ted$medv) / mean(ted$medv)
modelc7 <- lm(medv ~ poly(lstat, 7, raw = T), data = trd)
modelc7p <- modelc7 %>% predict(ted)
p7 <- c(RMSE(modelc7p,ted$medv), caret::R2(modelc7p, ted$medv), RMSE(modelc7p, ted$medv) / mean(ted$med
```

Lets see the RMSE, R2 and percent error p2, p3,p4, p5, p6

```
temp <- cbind(p2,p3,p4,p5,p6,p7)
row.names(temp) <-c("RMSE", "R2", "% error")
temp
```

```
##                p2        p3        p4        p5        p6        p7
## RMSE    5.6307274 5.5007142 5.3929523 5.2703735 5.3495124 5.3569154
## R2      0.6351934 0.6521415 0.6664867 0.6829474 0.6759031 0.6753441
## % error 0.2488803 0.2431336 0.2383705 0.2329525 0.2364505 0.2367777
```

There are couple patterns are going on. First im looking at the table above, we can see that p5 is the dip of the % error. after that it bounces back and has a small changes at p7. Also, if we look at the anova(p1 : p7) the significance level stops at 4.

```
anova(modelc2,modelc3,modelc4,modelc5, modelc6, modelc7)
```

```
## Analysis of Variance Table
##
## Model 1: medv ~ poly(lstat, 2, raw = T)
## Model 2: medv ~ poly(lstat, 3, raw = T)
## Model 3: medv ~ poly(lstat, 4, raw = T)
## Model 4: medv ~ poly(lstat, 5, raw = T)
## Model 5: medv ~ poly(lstat, 6, raw = T)
## Model 6: medv ~ poly(lstat, 7, raw = T)
##   Res.Df   RSS Df Sum of Sq       F    Pr(>F)
## 1    404 12224
## 2    403 11624  1    599.27 22.2105 3.381e-06 ***
## 3    402 11100  1    524.61 19.4432 1.334e-05 ***
## 4    401 10865  1    234.93  8.7072  0.003356 **
## 5    400 10767  1     98.14  3.6375  0.057210 .
## 6    399 10766  1      1.00  0.0372  0.847094
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

So i would say degree 4 is the best option.

```
modelc4 <- lm(medv ~ poly(lstat, 4, raw = T), data = trd)
modelc4p <- modelc4 %>% predict(ted)
p4 <- c(RMSE(modelc4p,ted$medv), caret::R2(modelc4p, ted$medv), RMSE(modelc4p, ted$medv) / mean(ted$med
p4
```

```
## [1] 5.3929523 0.6664867 0.2383705
```

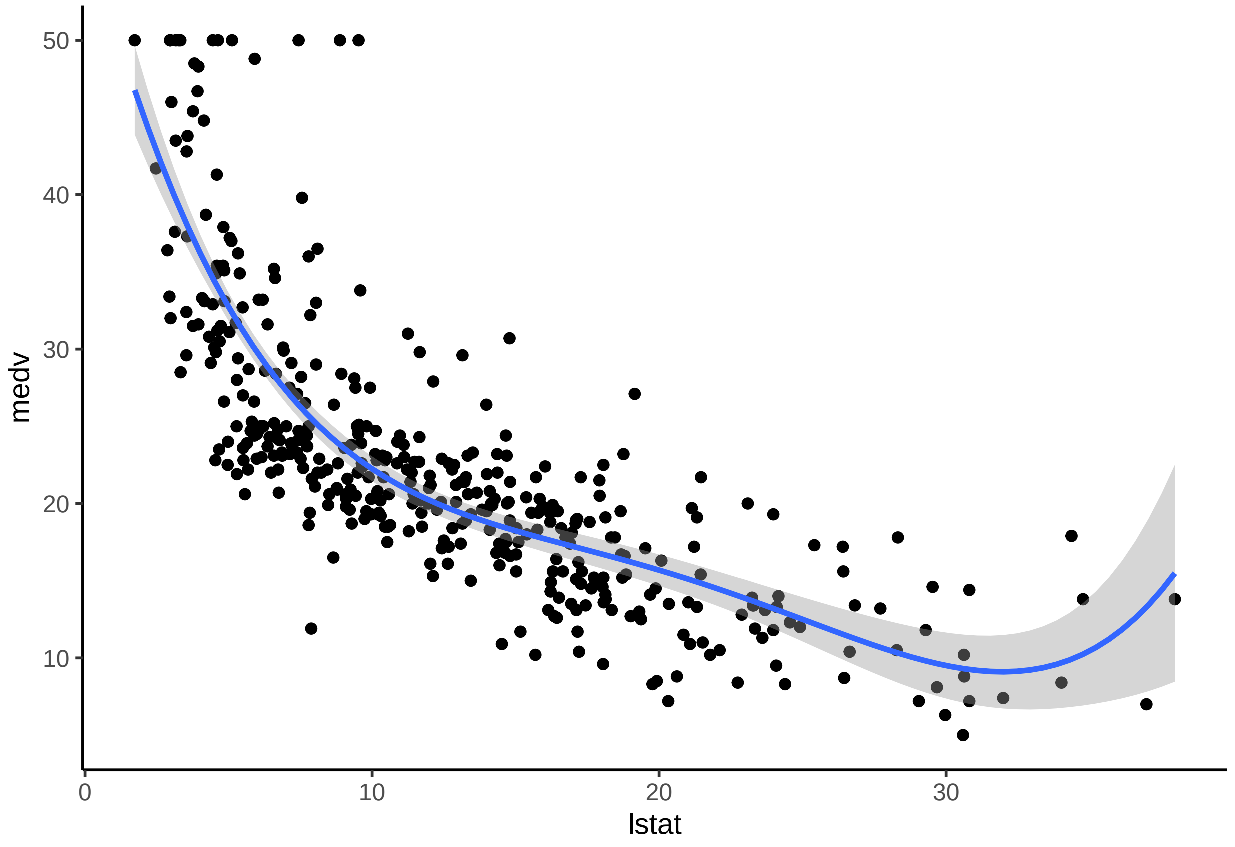
```
summary(modelc4)
```

```
##
## Call:
## lm(formula = medv ~ poly(lstat, 4, raw = T), data = trd)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -13.6093  -3.1719  -0.6806   2.2075  27.1453
##
## Coefficients:
##                             Estimate Std. Error t value Pr(>|t|)
## (Intercept)                5.764e+01  2.615e+00  22.047  < 2e-16 ***
## poly(lstat, 4, raw = T)1 -7.098e+00  8.303e-01  -8.549 2.62e-16 ***
## poly(lstat, 4, raw = T)2  5.007e-01  8.419e-02   5.947 5.94e-09 ***
## poly(lstat, 4, raw = T)3 -1.643e-02  3.336e-03  -4.925 1.24e-06 ***
## poly(lstat, 4, raw = T)4  1.948e-04  4.468e-05   4.359 1.66e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.255 on 402 degrees of freedom
## Multiple R-squared:  0.6747, Adjusted R-squared:  0.6715
## F-statistic: 208.5 on 4 and 402 DF,  p-value: < 2.2e-16
```

The residuals distribution is alright. By theory, the median should be 0 and 1Q, 3Q, and min max should be balanced. so this is not the best model but thats all we have.

Let's visualize the data

```
ggplot(trd, aes(lstat, medv)) +
  geom_point() +
  stat_smooth(method = lm, formula = y ~ poly(x, 4, raw = T))
```



Log Transformation

When i have a non linear relationship, i can also try a log transformation of the predictor variables.

```r
# build the model
modelcLog <- lm(medv ~ log(lstat), data = trd)

# make predictions
pmodelcLog <- modelcLog %>% predict(ted)

# model performance
(rmseLog <- RMSE(pmodelcLog, ted$medv))
```

```
## [1] 5.467124
```

```r
(r2Log <-  caret::R2(pmodelcLog, ted$medv))
```

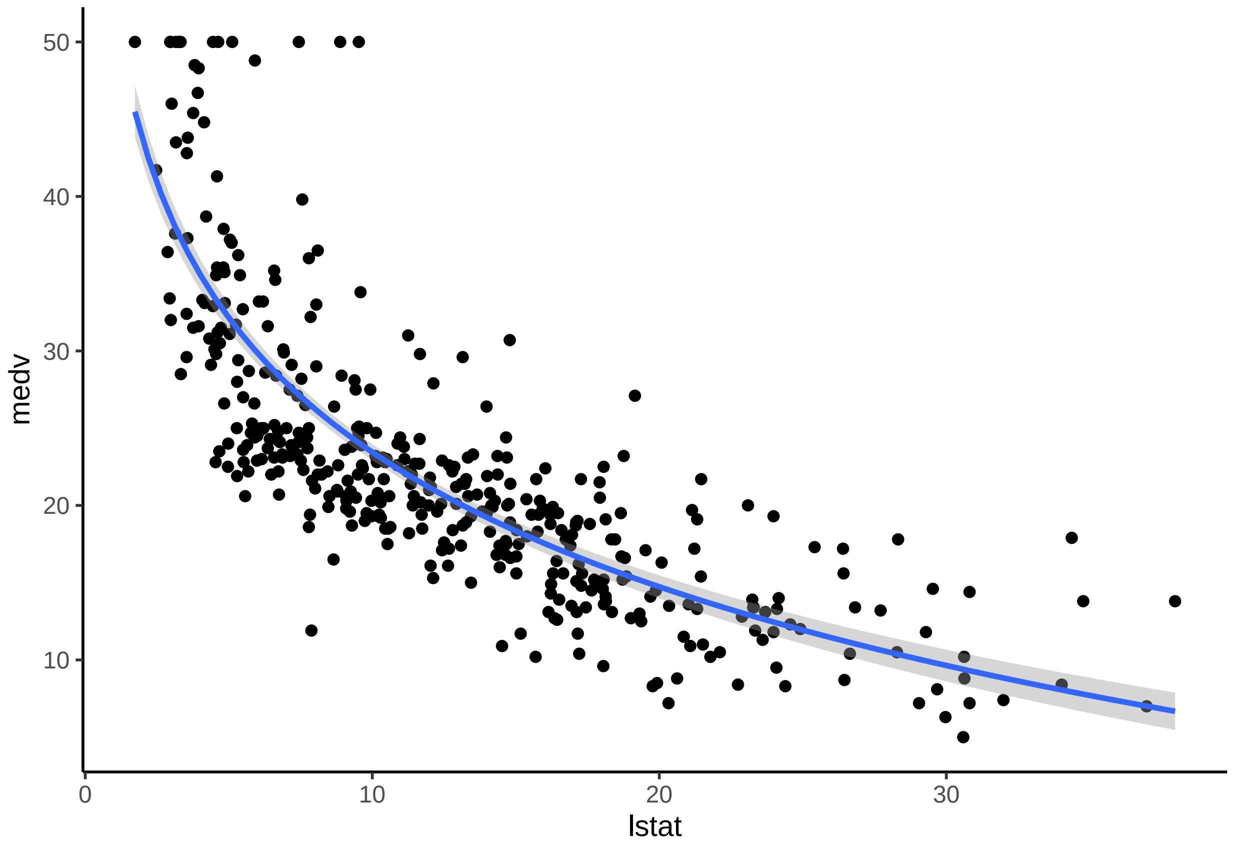
```
## [1] 0.6570091
```

```r
(perrorLog <- rmseLog/ mean(ted$medv))
```

```
## [1] 0.2416489
```

visualize the data

```r
ggplot(trd, aes(lstat, medv)) +
  geom_point() +
  stat_smooth(method = lm, formula = y ~ log(x))
```



Spline regression

```r
library(splines)
# splines becomes a base package so you shouldnt be install
# it if you are using R version 4.0.2
knots <- quantile(trd$lstat, p = c(.25,.5,.75))
# build model
modelcSp <- lm(medv ~ bs(lstat, knots = knots), data = trd)
# make prediction
pmodelcSp <- modelcSp %>% predict(ted)

# model performance
(rmseSp <- RMSE(pmodelcSp, ted$medv))
```

```
## [1] 5.317372
```

```r
(r2Sp <-  caret::R2(pmodelcSp, ted$medv))
```

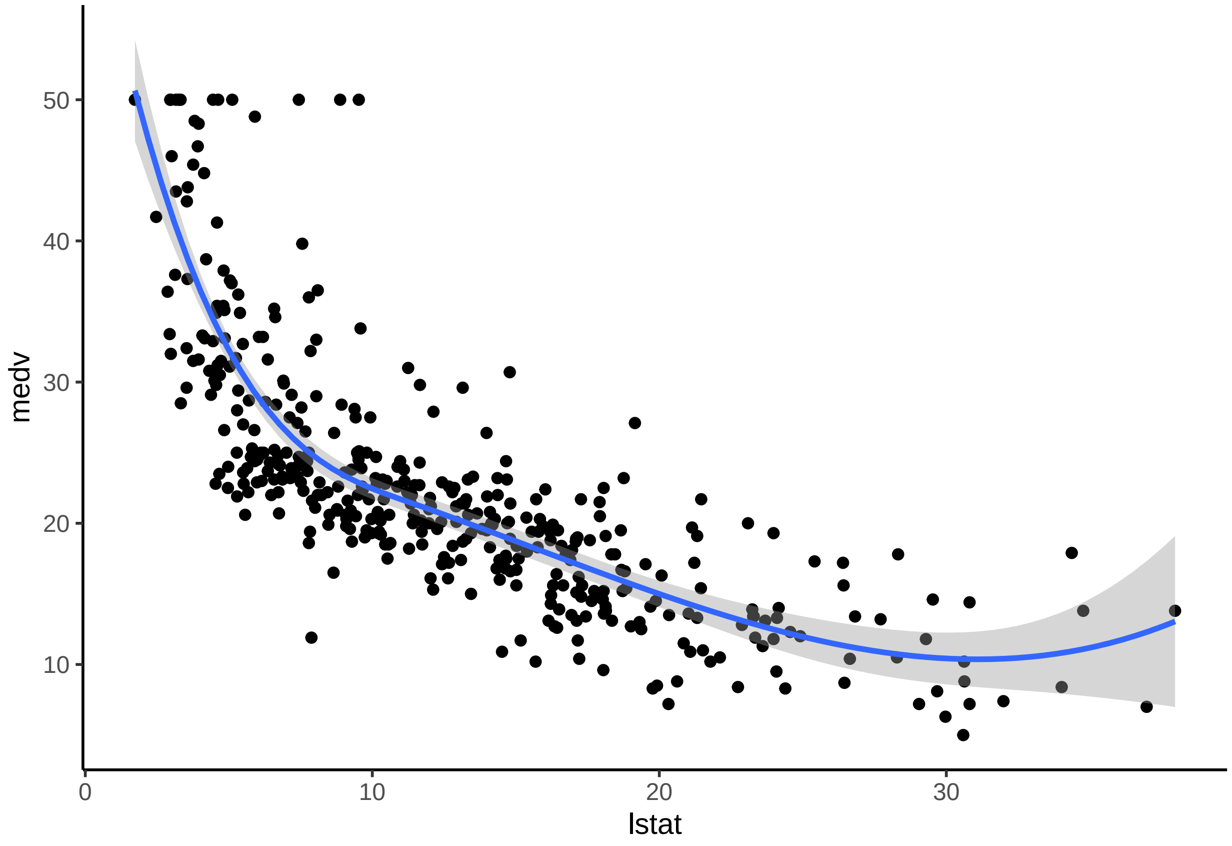
```
## [1] 0.6786367
```

```r
(perrorSp <- rmseSp/ mean(ted$medv))
```

```
## [1] 0.2350299
```

Lets visualize the data

```r
ggplot(trd, aes(lstat, medv)) +
  geom_point() +
  stat_smooth(method = lm, formula = y ~ splines::bs(x, df =4))
```



Generalized additive models

Here is another problem. Once I know that the model is non linear relationship in my data, the polynomial terms might not be flexible enough to capture the relationship, and the spline terms require specifying the knots. Therefore, we have Generalized additive models, GAM, are a technique to automatically fit a spline regression. We need mgcv package

```r
library(mgcv)
```

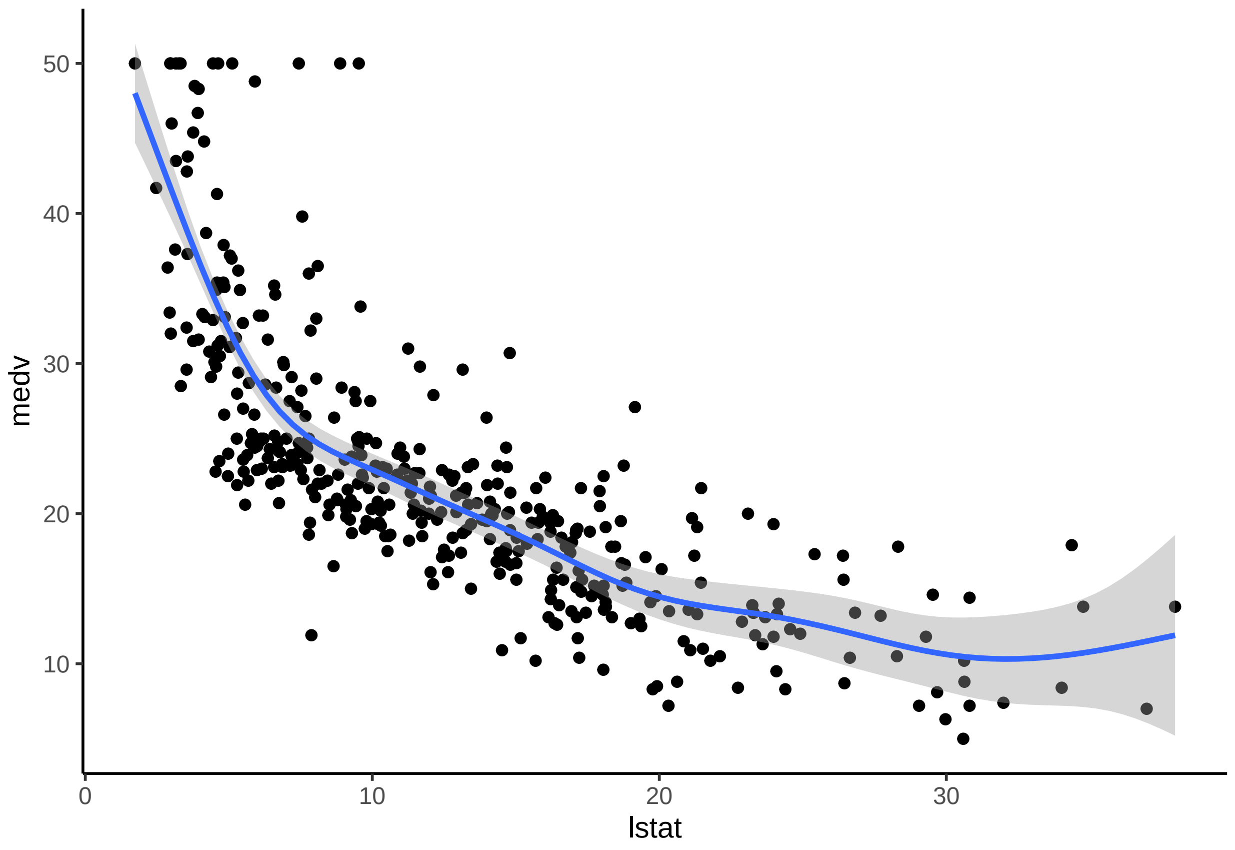
```
## Loading required package: nlme

##
## Attaching package: 'nlme'

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

## This is mgcv 1.8-31. For overview type 'help("mgcv-package")'.
```

```r
# build the model
modelcGam <- gam(medv ~s(lstat), data = trd)
# make predictions
pmodelcGam <- modelcGam %>% predict(ted)
# Model performance
data.frame(
  rmsegam <- RMSE(pmodelcGam, ted$medv),
  r2gam <- caret::R2(pmodelcGam, ted$medv),
  perrorGam <- rmsegam / mean(ted$medv))
```

```
##   rmsegam....RMSE.pmodelcGam..ted.medv.
## 1                              5.318856
##   r2gam....caret..R2.pmodelcGam..ted.medv. perrorGam....rmsegam.mean.ted.medv.
## 1                                0.6760512                           0.2350954
```

Visualize the plot

```r
ggplot(trd, aes(lstat, medv)) +
  geom_point() +
  stat_smooth(method = gam, formula = y ~ s(x))
```



Lets compare all the models linear, p^4, log, splines, and GAM

```
(fourmodelsPercentError <- rbind(perrorL, perrorp4, perrorLog, perrorSp, perrorGam))
```

```
##                [,1]
## perrorL   0.2874712
## perrorp4  0.2383705
## perrorLog 0.2416489
## perrorSp  0.2350299
## perrorGam 0.2350954
```

Among all 5 models, the best model we have is splines. why? think of how it works. it's like integral, it breaks in to small little part and find the slope thats why the stat_smooth fits the best of the data. Therefore Sp is the best model

Multiple Linear Regression in R

A little bit duplicated with the first section, however, in this section we are dealing with more than one variables.

With three predictor variables (x), the prediction of y is expressed by the following equation:

y = b0 + b1*$x1 + b2$*x2 + b3*x3

Outcome: 1) build and interpret a multiple linear regression model in R 2) Check the overall quality of the model

```
data("marketing", package = "datarium")
modelLS <- lm(sales ~ youtube + facebook + newspaper, data = marketing)
summary(modelLS)
```

```
##
```

```
## Call:
## lm(formula = sales ~ youtube + facebook + newspaper, data = marketing)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -10.5932  -1.0690   0.2902   1.4272   3.3951
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.526667   0.374290   9.422   <2e-16 ***
## youtube      0.045765   0.001395  32.809   <2e-16 ***
## facebook     0.188530   0.008611  21.893   <2e-16 ***
## newspaper   -0.001037   0.005871  -0.177     0.86
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.023 on 196 degrees of freedom
## Multiple R-squared:  0.8972, Adjusted R-squared:  0.8956
## F-statistic: 570.3 on 3 and 196 DF,  p-value: < 2.2e-16
```

Let's interpret the data

First we might want to look at the F statistics' p value We have a pretty high significant value. This mean that, at least one of the variable is significantly related to the outcome. You can think of it is true that the parameter (youtube / facebook / newspaper) has the relationship with the response variable

You should have this question, so which one has the relationship with the response variable?

let's take a look with the following code

```r
summary(modelLS)$coef
```

```
##                 Estimate  Std. Error    t value     Pr(>|t|)
## (Intercept)  3.526667243 0.374289884  9.4222884 1.267295e-17
## youtube      0.045764645 0.001394897 32.8086244 1.509960e-81
## facebook     0.188530017 0.008611234 21.8934961 1.505339e-54
## newspaper   -0.001037493 0.005871010 -0.1767146 8.599151e-01
```

Look at the T statistics is checking if the b0, b1, b2, b3 is 0 or not. Look at the newspaper, we have -0.001037493 as the beta3 right? and look at the t value, -0.1767146.

The T value for the newspaper is pretty close to 0 and look at B3, we have -0.001037493, it simply no use at all so we can simply create another model without the news paper.

But the rest, youtube, facebook have a high impact for the sales. There is a relationship with the sales. So more money to put in to youtube and facebook, we have more feedback from the sales.

Lets create another model

```r
modelLSyf <- lm(sales ~ youtube + facebook, data = marketing)
summary(modelLSyf)
```

```
##
## Call:
## lm(formula = sales ~ youtube + facebook, data = marketing)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -10.5572  -1.0502   0.2906   1.4049   3.3994
```

```
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.50532    0.35339   9.919   <2e-16 ***
## youtube      0.04575    0.00139  32.909   <2e-16 ***
## facebook     0.18799    0.00804  23.382   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.018 on 197 degrees of freedom
## Multiple R-squared:  0.8972, Adjusted R-squared:  0.8962
## F-statistic: 859.6 on 2 and 197 DF,  p-value: < 2.2e-16
```

95% confident interval for the coefficient

```r
confint(modelLSyf)
```

```
##                  2.5 %     97.5 %
## (Intercept) 2.80841159 4.20222820
## youtube     0.04301292 0.04849671
## facebook    0.17213877 0.20384969
```

Model accuracy assessment

As we know, simple linear regression, the overall quality is based on R^2 and Residual standard error (RSE)

Remember one thing, the more parameters we have the higher the R^2. Eventhough the parameter does not have much effect on the Y response, it will still in play with the R square. Thats the reason why we have adjusted R^2

Residual standard error (RSE), or sigma

The RSE estimate gives a measure of error of prediction. we want small RSE

```r
sigma(modelLSyf)/mean(marketing$sales)
```

```
## [1] 0.1199045
```

This number is not to bad. so we have approximately 12% error rate.

Some cool trick, we dont have to type all the parameters. we can simply do

model <- lm(sales ~. data = marketing) that ~ means all

now if we would like to drop the newspaper from the parameter list model <- lm(sales ~. - newspaper, data = marketing)

Alternative model <- update(model, ~. - newspaper)

Predict in R: Model Predictions and Confidence Intervals

Outcome: predict outcome for new observations data, display confidence intervals and the prediction intervals.

```r
# load the data
data("cars", package = "datasets")
# build the model
modelcar <- lm(dist ~ speed, data = cars)
modelcar
```

```
##
## Call:
## lm(formula = dist ~ speed, data = cars)
```

```
##
## Coefficients:
## (Intercept)        speed
##     -17.579        3.932
```

Prediction for new data set

```r
# create data
cardata <- data.frame(speed <- c(12,19,24))
# predict data
predict(modelcar, newdata = cardata)
```

```
##        1        2        3
## 29.60981 57.13667 76.79872
```

##Confidence interval

by default the confidence interval is 95%

```r
predict(modelcar, newdata = cardata, interval = "confidence")
```

```
##        fit      lwr      upr
## 1 29.60981 24.39514 34.82448
## 2 57.13667 51.82913 62.44421
## 3 76.79872 68.38765 85.20978
```

From our newly created data. Lets take 19 miles per hr as our example. Distance is in ft we can say that 19 miles per hr has an average stopping dist between 51.82913 to 62.44421. the regression line has predict the value for 19mph with a stopping dist @ 57.13667

Prediction interval prediction interval gives an uncertainty around a single value.

```r
predict(modelcar, newdata = cardata, interval = "prediction")
```

```
##        fit       lwr       upr
## 1 29.60981 -1.749529  60.96915
## 2 57.13667 25.761756  88.51159
## 3 76.79872 44.752478 108.84495
```

From the table above, we can say that 95% prediction intervals tell us that with a speed of 19 mph with the stopping distance is 25.76 to 88.51. This means that, based on our model 95% of the cars with a speed of 19 mph have a stopping distance between 25.76 and 88.51

Now you should have at least one question in your mind at this point. WTH? what is the difference between prediction interval and confidence interval?

The key word for both interpreting is average.

prediction interval is predicting a single future value at that point.

confidence interval is predicting the mean.

https://stats.stackexchange.com/questions/16493/difference-between-confidence-intervals-and-prediction-intervals

More on prediction interval or confidence interval

A prediction interval reflects the uncertainty around a single value, while a confidence interval reflects the uncertainty around the mean prediction values. Thus, a prediction interval will be generally much wider than a confidence interval for the same value.

Generally, we are interested in specific individual predictions. So a prediction interval would be more appropriate. Using a confidence interval when you should be using a prediction interval will be underestimate the uncertainty in a given predicted value

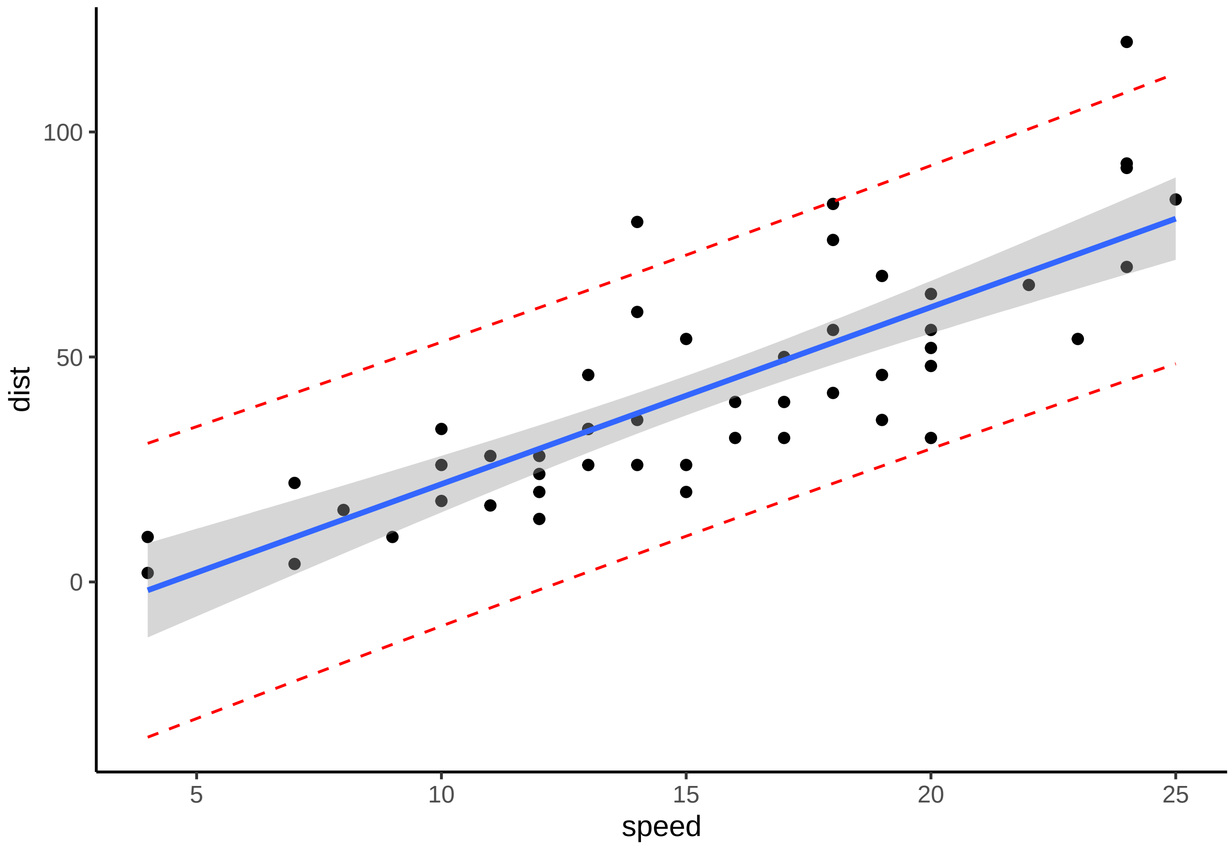
lets build the prediction band and confidence inteval

```r
# build the model
data("cars", package = "datasets")
modelcar1 <- lm(dist~speed, data =cars)
# add the predictions
p.int <- predict(modelcar1, interval = "prediction")
```

```
## Warning in predict.lm(modelcar1, interval = "prediction"): predictions on current data refer to _futu
```

```r
carD <- cbind(cars, p.int)
# Regression line + confidence intervals
p <- ggplot(carD, aes(speed, dist)) +
  geom_point() +
  stat_smooth(method = lm) +
  geom_line(aes(y = lwr), color = "red", linetype = "dashed") + #prediction interval
  geom_line(aes(y = upr), color = "red", linetype = "dashed") # prediction interval
p
```

```
## `geom_smooth()` using formula 'y ~ x'
```



Section 2 - Regression Model Diagnostics - 3 Sub Sections

Intro This section is focusing in making diagnostics to detect potential problems in the data.

Sub-Section 1 Linear Regression Assumptions and Diagnostics in R: Essentials

By the title, you can tell we are going to check if the model works well for the data.

1) inspect the significance of the regression beta coefficients
2) R^2

Those are what we have learnt from the previous sections

This section, we will learn the additional steps to evaluate how well the model fits the data.

For example, linear regression model makes the assumption that the relationship between x and y are linear. but that might not be true. The relationship could be polynomial or logarithmic

the data might contain outlines which will affect the result.

Thats why we need to diagnose the regression model and detect potential problems and check the assumptions is met by the linear regression model

in order to do so, we will check the distribution of residuals errors, which will tell us more about my data.

Main idea for this section. - explaining residuals errors and fitted values. - present linear regression assumptions, as well as potential problems you can will tackles while performing regression analysis - We will talk about some built in diagnostic plots in R to test the assumptions about our linear regression model.

Lets get started

```
if(!require("broom")){
  install.packages("broom")
  library(tidyverse)
library(broom)
}
```

```
## Loading required package: broom
```

```
theme_set(theme_classic())
```

```
# load the data
data("marketing", package = "datarium")
# inspect the data
sample_n(marketing, 3)
```

```
##   youtube facebook newspaper sales
## 1   10.44    58.68     90.00  8.64
## 2  345.12    51.60     86.16 31.44
## 3  195.96    37.92     63.48 20.28
```

Build the regression model

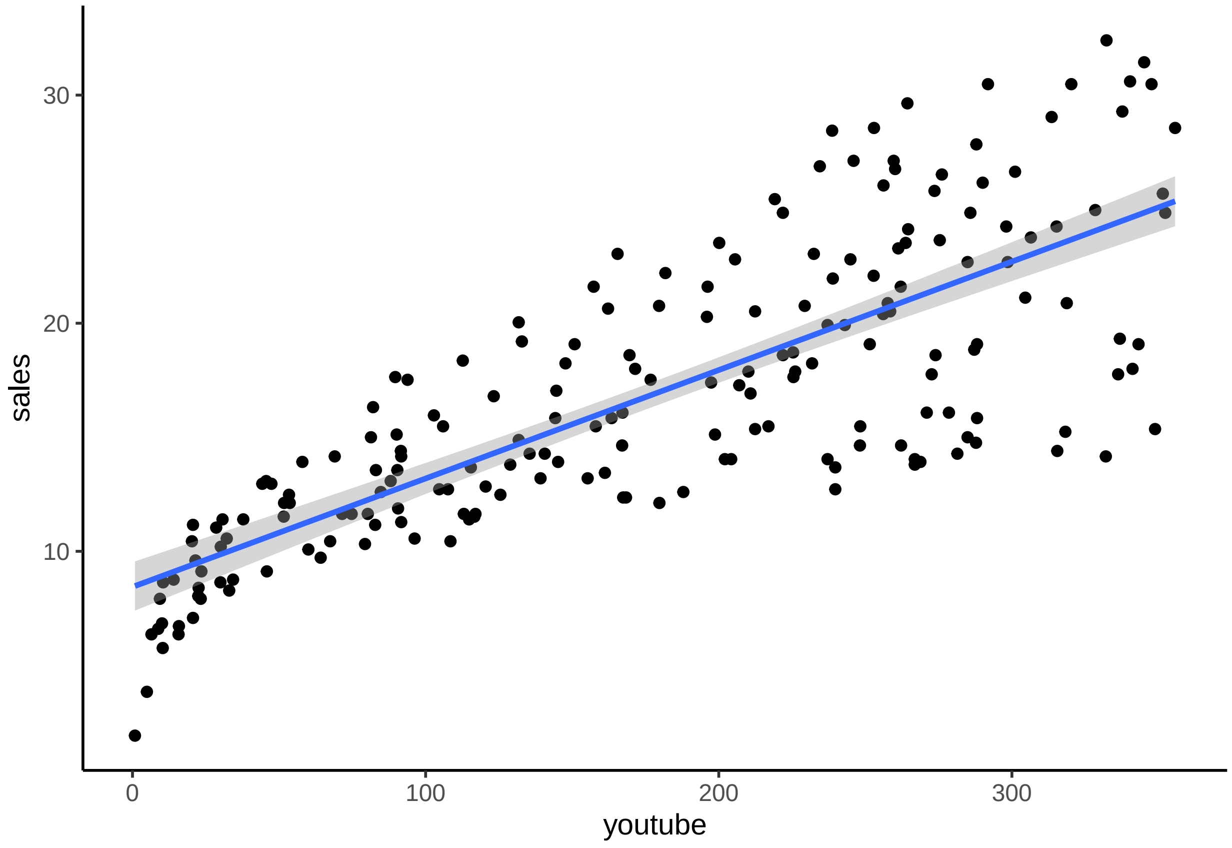
```
modelm <- lm(sales ~ youtube, data = marketing)
modelm
```

```
##
## Call:
## lm(formula = sales ~ youtube, data = marketing)
##
## Coefficients:
## (Intercept)      youtube
##     8.43911      0.04754
```

Fitted value (predicted value) is the value that is predicted from the regression line.

```
ggplot(marketing, aes(youtube, sales)) +
  geom_point() +
  stat_smooth(method = lm)
```

```
## ‘geom_smooth()‘ using formula ’y ~ x’
```



augment is from broom package

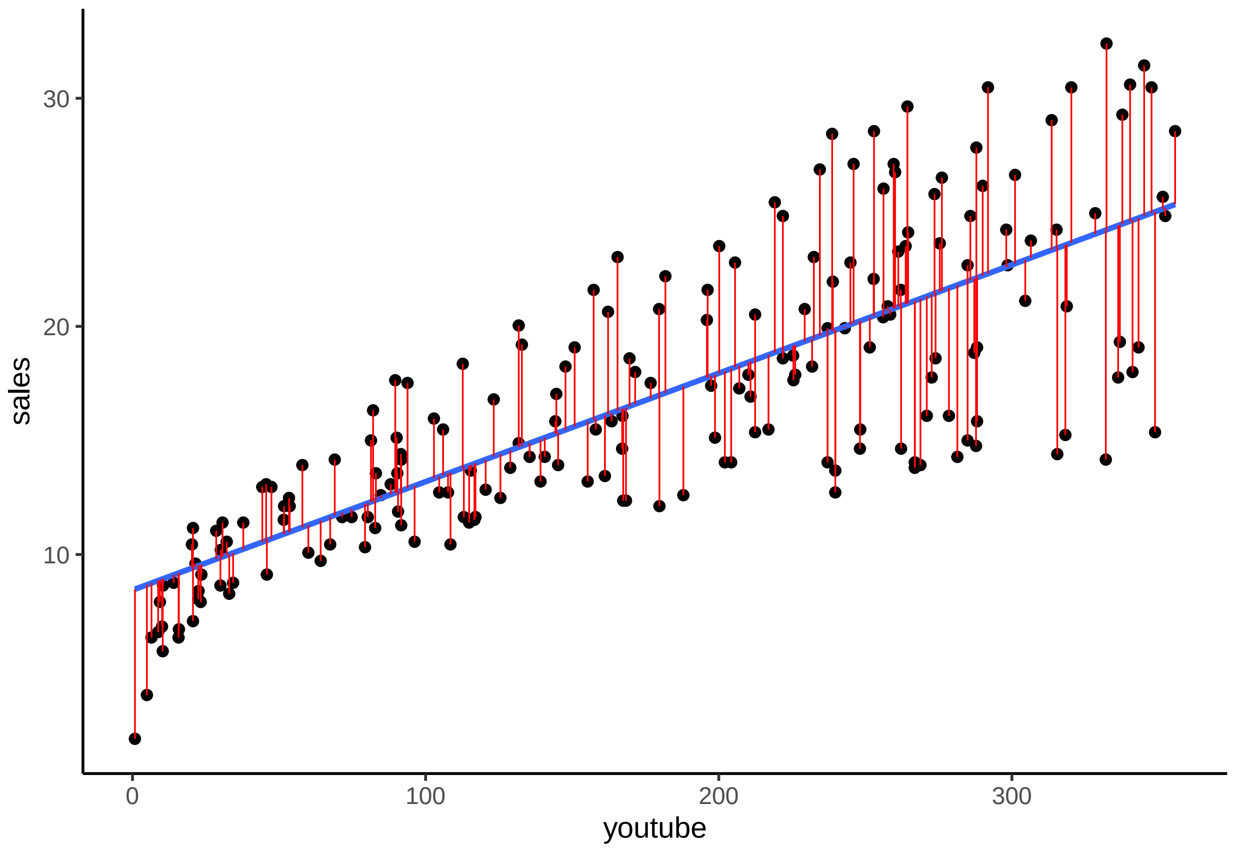
```
model.diag.metrics <- augment(modelm)
head(model.diag.metrics)
```

```
## # A tibble: 6 x 8
##   sales youtube .fitted .resid .std.resid    .hat .sigma   .cooksd
##   <dbl>   <dbl>   <dbl>  <dbl>      <dbl>   <dbl>  <dbl>     <dbl>
## 1 26.5    276.    21.6   4.96      1.27   0.00970   3.90 0.00794
## 2 12.5     53.4   11.0   1.50      0.387  0.0122    3.92 0.000920
## 3 11.2     20.6    9.42  1.74      0.449  0.0165    3.92 0.00169
## 4 22.2    182.    17.1   5.12      1.31   0.00501   3.90 0.00434
## 5 15.5    217.    18.8  -3.27     -0.839  0.00578   3.91 0.00205
## 6  8.64    10.4    8.94 -0.295    -0.0762 0.0180    3.92 0.0000534
```

Lets plot the residuals error in red

```
ggplot(model.diag.metrics, aes(youtube, sales)) +
  geom_point() +
  stat_smooth(method = lm, se = F) +
  geom_segment(aes(xend = youtube, yend = .fitted), color = "red", size =0.3)
```

```
## ‘geom_smooth()‘ using formula ’y ~ x’
```



In order to check the regression assumption, we need to see the distribution of the residuals

Linear regression makes several assumptions about the data, such as :

Linearity of the data. The relationship between the predictor (x) and the outcome (y) is assumed to be linear.

Normality of residuals. The residual errors are assumed to be normally distributed.

Homogeneity of residuals variance. The residuals are assumed to have a constant variance (homoscedasticity)

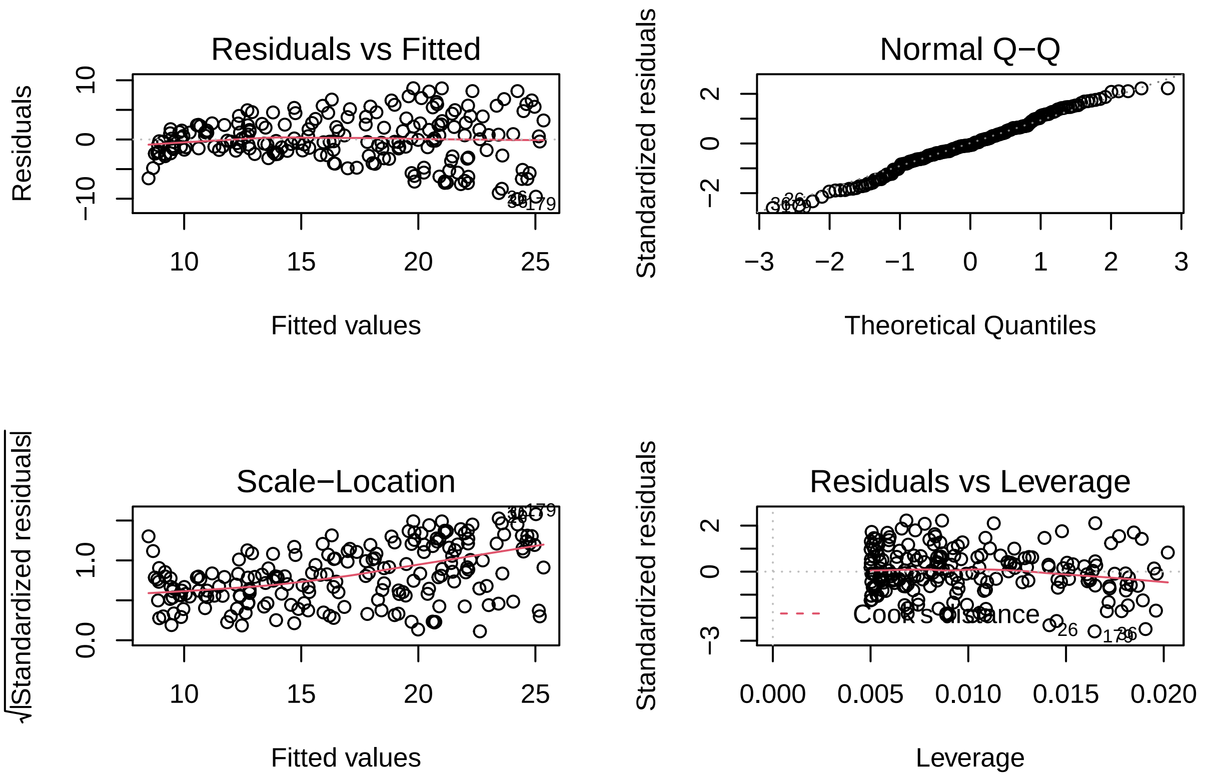
Independence of residuals error terms.

You should check whether or not these assumptions hold true. Potential problems include:

Non-linearity of the outcome - predictor relationships Heteroscedasticity: Non-constant variance of error terms. Presence of influential values in the data that can be: Outliers: extreme values in the outcome (y) variable High-leverage points: extreme values in the predictors (x) variable

Those problems can be solved by diagnostic plot

```r
par(mfrow = c(2,2))
plot(modelm)
```



How to read it?

https://data.library.virginia.edu/diagnostic-plots/

Residuals vs Fitted. Used to check the linear relationship assumptions. A horizontal line, without distinct patterns is an indication for a linear relationship, what is good.

Normal Q-Q. Used to examine whether the residuals are normally distributed. It's good if residuals points follow the straight dashed line.

Scale-Location (or Spread-Location). Used to check the homogeneity of variance of the residuals (homoscedasticity). Horizontal line with equally spread points is a good indication of homoscedasticity. This is not the case in our example, where we have a heteroscedasticity problem.

Residuals vs Leverage. Used to identify influential cases, that is extreme values that might influence the regression results when included or excluded from the analysis. This plot will be described further in the next sections.

Lets create a better table for model.diag.metrics

```
names(model.diag.metrics)
```

```
## [1] "sales"      "youtube"    ".fitted"    ".resid"     ".std.resid"
## [6] ".hat"       ".sigma"     ".cooksd"
```

Before and after

```
index <- c(1:nrow(model.diag.metrics))
model.diag.metrics1 <- data.frame(index,model.diag.metrics[ , c(-7)])
names(model.diag.metrics1)
```

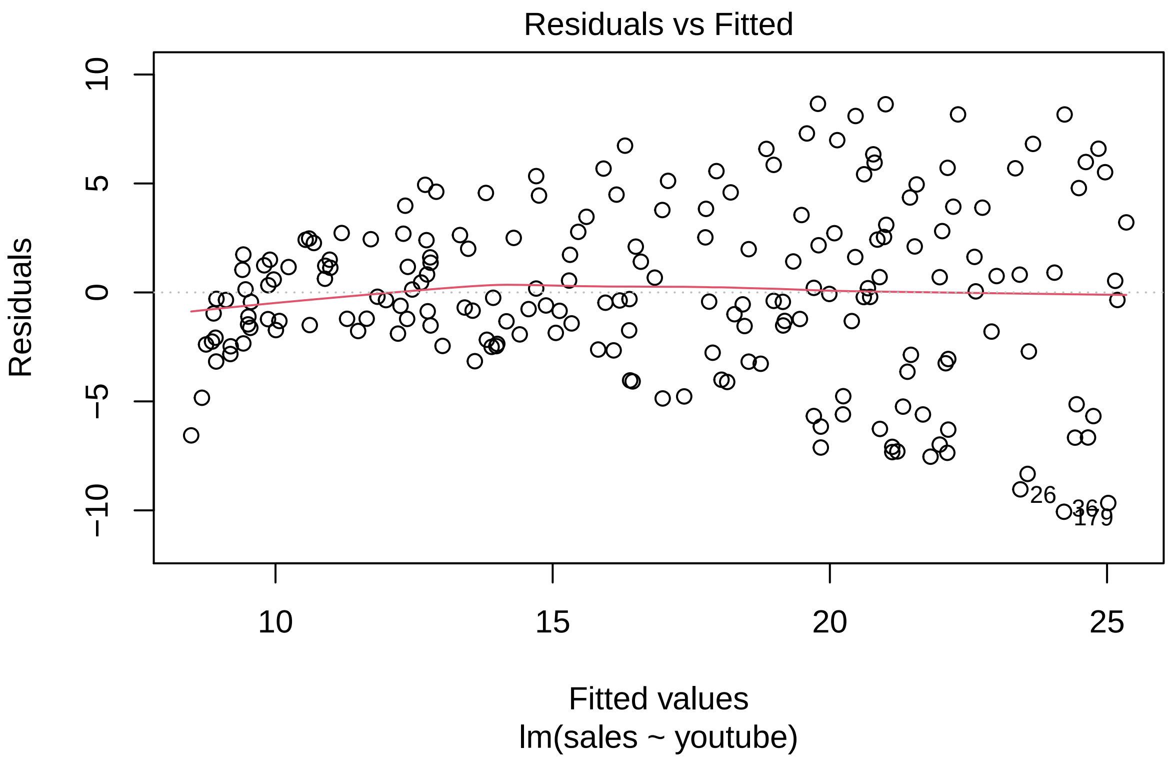
```
## [1] "index"      "sales"      "youtube"    ".fitted"    ".resid"
## [6] ".std.resid" ".hat"       ".cooksd"
```

fitted values = predicted value from the regression line .resid = residual errors .hat = outliers .std.resid =

detect outliers, extreme values -> standardized residuals = residuals / standard errors .cooksd = Cook's distance -> detect influential values outliers or high leverage point

Let's look at the plot one by one

```r
plot(modelm,1)
```



Good plot would be show no fitted pattern, which means the red line should be a horizontal line at 0. If the red line is not at 0 or around 0 there might be a problem with our linear model, which means we need to use non linear model. In our example, plot 1 residuals vs fitted, there is no pattern in the residual plot, so we can assume it is a linear relationship sales ~ youtube.

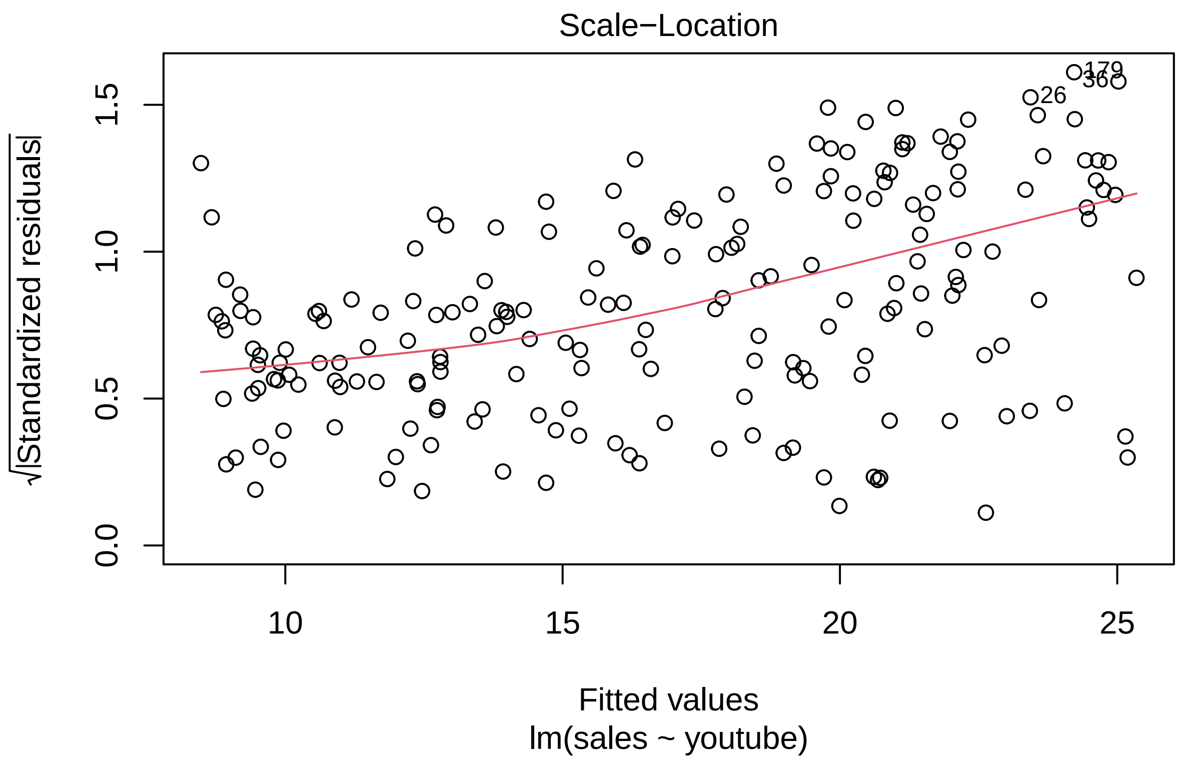
if the residual plot is non linear relationship, you need to transform your data ith log(x), sqrt(x), x^2 in the regression and so on.

You might ask how do you know if it is non linear? well it will be obvious that red line is something else like quadratic or something. reference link: non linear https://i0.wp.com/blogs.sas.com/content/iml/files/2019/06/residsmooth1.png?ssl=1

Homogeneity of variance

This assumption can be checked by examining the scale - location plot, also known as the spread- location plot

```r
plot(modelm,3)
```



Good plot for scale - location is the data set is close together and close the the line. since this plot is spreading wider when x increases, so this is not a good plot.

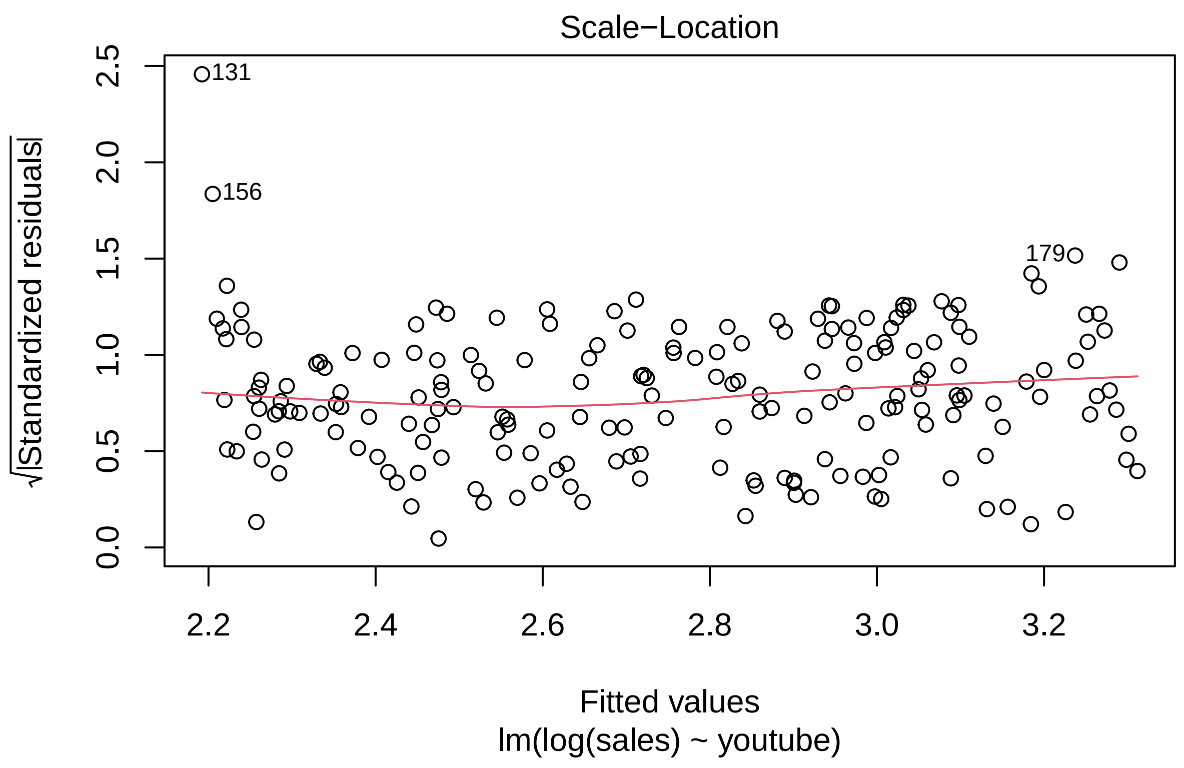
So i can say not a constant variances in the residuals error (heterosedasticity)

Remember, by theory epsilon has mean zero and the variance is constant.

a possible solution to reduce the variance is to use log or square root transformation for variable y

lets try again by transforming the plot

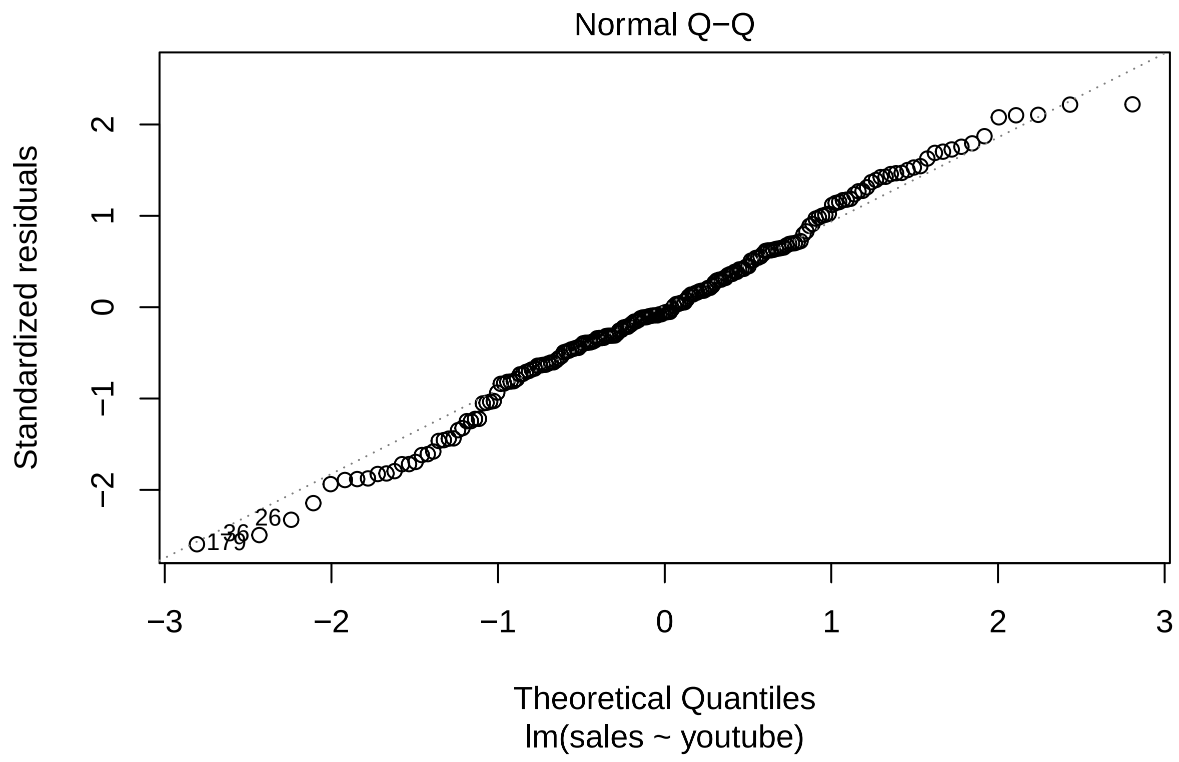
```r
modelmLog <- lm(log(sales) ~ youtube, data = marketing)
plot(modelmLog,3)
```



This plot looks a lot better

Normality of Residuals this is also called QQ plot. This plot can check the normality assumption. The normal probability plot of residuals should approximately follow a straight line like below.

```r
plot(modelm, 2)
```



Outliers and high Leverage points

The outliers will directly affect the RSE because it is so far away from the regression line. Outliers can be identified by examining the standardized residual (or studentized residual), which is the residual divided by the estimated standard error.

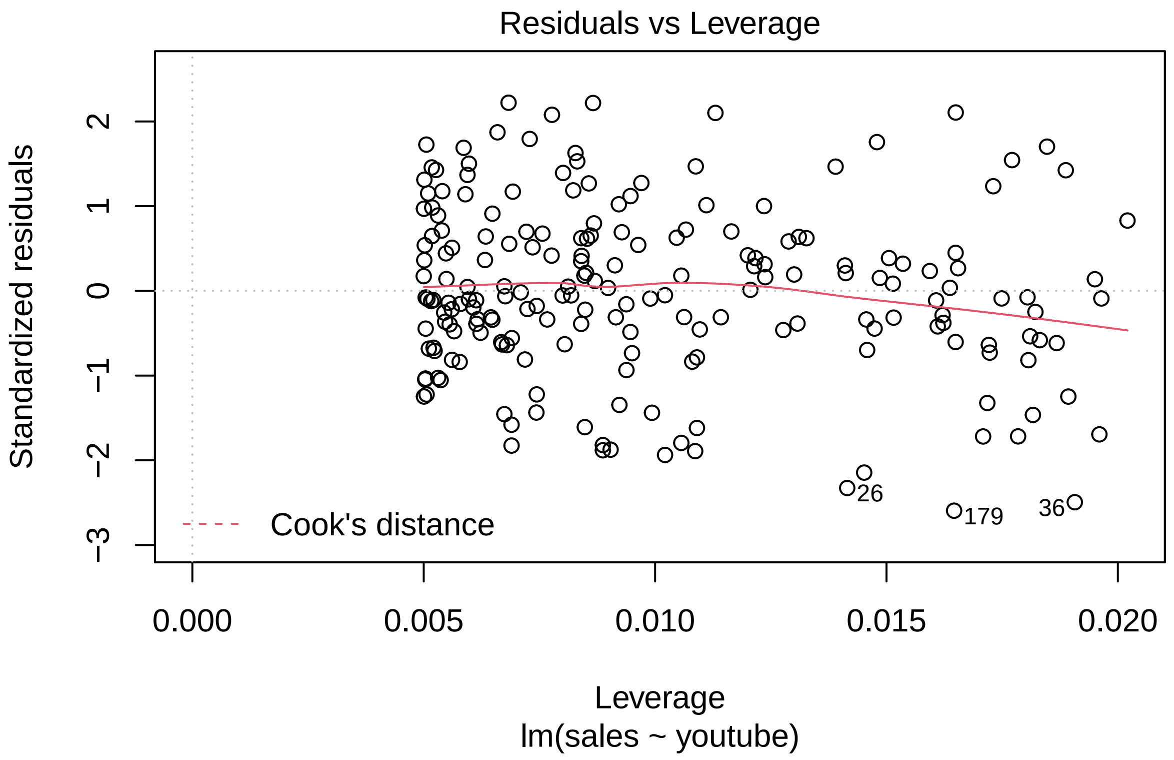
Observations whose standardized residuals are greater than 3 in absolute value are possible outliers (James et al. 2014).

High leverage points (hat value)

A value of this statistic above 2(p + 1)/n indicates an observation with high leverage (P. Bruce and Bruce 2017); where, p is the number of predictors and n is the number of observations.

Residuals vs Leverage plot

```
plot(modelm, 5)
```



The plot will highlight the most extreme points. 26, 179, 36. as you can see no outliers that have exceed 3 standard deviations, which is a good plot.

Influential values

Not all outliers (extreme data points) are influential in linear regression analysis

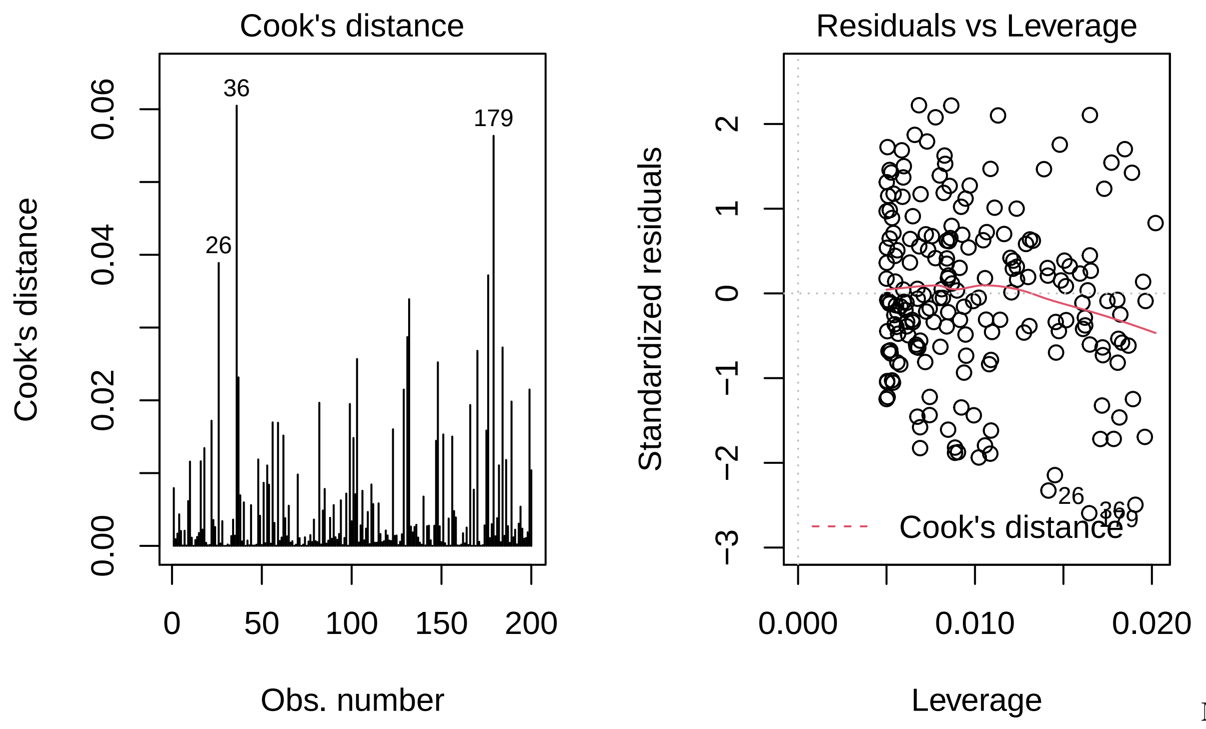
We use Cook's distance to determine the influence of a value. This metric defines influence as a combination of leverage and residual size.

High influence if Cook's distance exceeds 4/ (n - p - 1) n = # of observations p = number of predictor variables.

again residuals vs leverage plot can help us to find influential observations. outlying values are generally located at the upper right corner or lower right corner.

Those corners will have direct influential against a regression line.

```r
par(mfrow=c(1,2))
# Cook's distance
plot(modelm, 4)
# Residuals vs Leverage
plot(modelm, 5)
```



Now we have been talking about Cook's distance for the last 10 mins, so how do I know if the outliers are influencing the regression line?

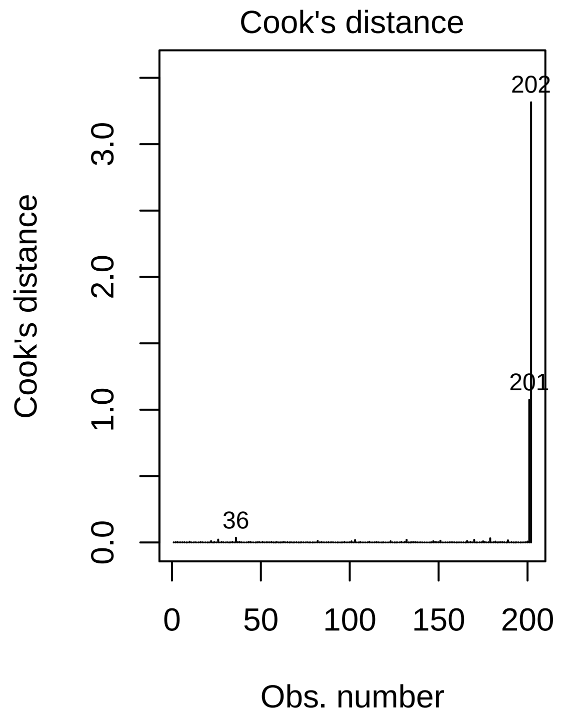
First from the Residuals vs Leverage plot, i am not seeing any red dashed line, so that means my outliers is not affecting the regression line a lot. If we see the dashed line that's mean we are close to the Cook's distance line which means it is some how affecting the regression line seriously. but usually if the outliers are within the Cook's distance we are all good.

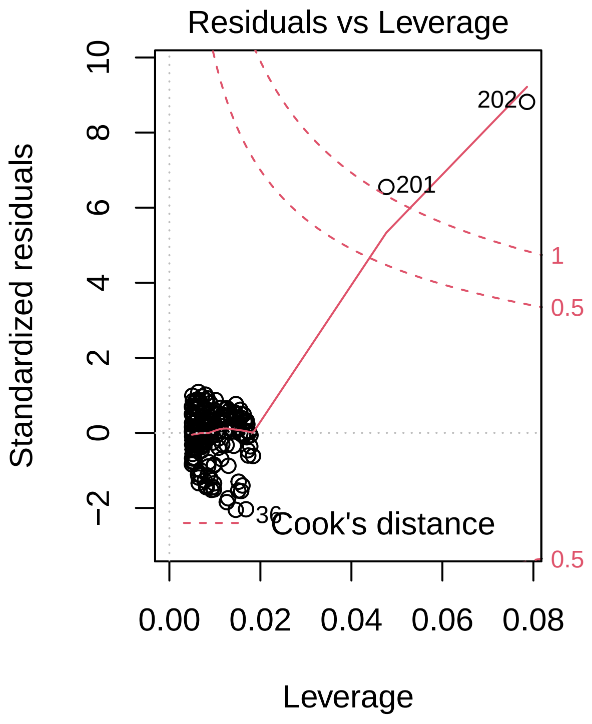
By default, only top 3 most extreme values are labeled on Cook's distance plot, if I want to add more i can use the following code plot(modelm, 4, id.n = 5)

if i would like to access those distance later, i can use the following code model.diag.metrics %?% top_n(3, wt = .cooksd)

Lets create something that is outside the Cook's distance

```r
dfcook <- data.frame(
  x <- c(marketing$youtube, 500,600),
  y <- c(marketing$sales, 80,100)
)
modelm2 <- lm(y~x, dfcook)
par(mfrow = c(1,2))
# Cook's distance
plot(modelm2, 4)
# Residuals vs Leverage
plot(modelm2, 5)
```





As you can see from the Residuals vs Leverage, you can see there are 2 outliers are outside of the dashed line (Cook's distance) those are the outliers that are affecting the regression line directly. In the other words, those data are outside of the cook's distance would simply mean they have a high cook's value.

The plot identified the influential observation as #201 and #202. If you exclude these points from the analysis, the slope coefficient changes from 0.06 to 0.04 and R2 from 0.5 to 0.6. Pretty big impact!

There are a lot of concepts that must be understood, so take your time to go thru it again.

Discussion.

This section describes linear regression assumptions and how to diagnose the potential problems in the model You must visualizing the residuals and the patterns in residuals is not a stop signal. your current regression model might not be the best way to understand your data.

Here are the potential problems: non - linear relationships between the outcome and the predictor variables. When facing to this problem, one solution is to include a quadratic term, such as polynomial terms or log transformation. See Chapter

Existence of important variables that you left out from your model. Other variables you didn't include (e.g., age or gender) may play an important role in your model and data.

Presence of outliers. If you believe that an outliers have occurred due to an error in data collection and entry, then one solution is to simply remove the concerned observation.

Sub-Section 2 - Multi-collinearity Essentials and VIF in R

In multiple regression, 2 or more parameters might be correlated with each other is called collinearity.

There is an extreme situation, called multicollinearity, which collinearity exists between three or more variables, even if no pair of variables have a particularly high correlation. To simplify that, there is redundancy between parameter variables.

If we have multicollinearity in our model, it will become unstable.

VIF (variance inflation factor) measures how much the variance of a regression coefficient is inflated due to multicollinearity.

"The smallest possible value of VIF is one (absence of multicollinearity). As a rule of thumb, a VIF value that exceeds 5 or 10 indicates a problematic amount of collinearity (James et al. 2014)."

So in this section, we will be detecting the multicollinearity in a regression model using R

Required R packages tidyverse, caret

```
library("tidyverse")
library("caret")
```

Data Preparation

```
# load the data
data("boston", package = "MASS")
```

```
## Warning in data("boston", package = "MASS"): data set 'boston' not found
```

```
# split the data into training and test set
set.seed(123)
btr.samples <- Boston$medv %>%
  createDataPartition(p=0.8, list = F)
btrd <- Boston[btr.samples, ]
bted <- Boston[-btr.samples,]
```

Build a regression model

```
# build the model
modelbos <- lm(medv ~., data = btrd)

# make predictions
bpred <- modelbos %>% predict(bted)

# model performance
brmse <- RMSE(bpred, bted$medv)
br2 <- caret::R2(bpred, bted$medv)
brmse
```

```
## [1] 4.588948
```

```
br2
```

```
## [1] 0.761126
```

Detecting multicollinearity need to install car package for vif()

```
car::vif(modelbos)
```

```
##     crim       zn    indus     chas      nox       rm      age      dis
## 1.844585 2.323334 3.956419 1.066866 4.447852 1.911905 3.213786 4.065420
##      rad      tax  ptratio    black    lstat
## 8.152643 9.660128 1.851645 1.364539 3.131835
```

From there we can see that tax has a 9.660128 score, which is a problem.

Dealing with multicollinearity

```
# build a model excluding the tax variable
modelbos.cleaned <- lm(medv ~. -tax, data = btrd)
```

```r
# make predictions
modelbos.cleaned.pred <- modelbos.cleaned %>% predict(bted)

#  performance
brmse.cleaned <- RMSE(modelbos.cleaned.pred, bted$medv)
br2.cleaned <- caret::R2(modelbos.cleaned.pred, bted$medv)
brmse.cleaned
```

```
## [1] 4.644396
```

```r
br2.cleaned
```

```
## [1] 0.7558647
```

Now we have a lower R2 and a higher RMSE

This section is simply detecting and how to deal with multicollinearity in regression models. anything that is above 5 or 10, those parameters must be removed.

Note that, in a large data set presenting multiple correlated predictor variables, you can perform principal component regression and partial least square regression strategies.

Sub-Section 3 - Confounding Variable Essentials

library required gapminder

```r
library(gapminder)
lm(lifeExp ~ gdpPercap, data = gapminder)
```

```
##
## Call:
## lm(formula = lifeExp ~ gdpPercap, data = gapminder)
##
## Coefficients:
## (Intercept)    gdpPercap
##   5.396e+01    7.649e-04
```

So obviously the continent is an important variable like countries in Europe are estimated to have a higher life expectancy compared to countries in Africa. So by adding continent as a confounding variable will increase the accuracy of our model.

```r
lm(lifeExp ~ gdpPercap + continent, data = gapminder)
```

```
##
## Call:
## lm(formula = lifeExp ~ gdpPercap + continent, data = gapminder)
##
## Coefficients:
##       (Intercept)          gdpPercap  continentAmericas      continentAsia
##         4.789e+01          4.453e-04          1.359e+01          8.658e+00
##   continentEurope   continentOceania
##         1.757e+01          1.815e+01
```

Regression Model Validation - 3 sections

Info

In this big section, we will talk about goodness of the model, how well the model fits the training data used to build the model and how accurate is the model in predicting the outcome for new unseen test observations.

In this part, you will learn techniques for assessing regression model accuracy and for validating the performance of the model.

Sub section 1 - Regression Model Accuracy Metrics: R-square, AIC, BIC, Cp and more

This section, we will be discussing different statistical regression metrics for measuring the performance of the regression model

Model Performance metrics

R-Squared: interval -> +/ - 1 +- 1 is the max or min. Closer to 1 the better it is. It measures how close the actual data to the regression line. If we have 1 that means all the actual data are on exactly lies on the regression line.

Root mean squared error RMSE: We want a small RMSE. RMSE is simply the square root of MSE. MSE is exactly the epsilon term from the regression model. Formula = MSE = actual - theoretical value) ˆ2

Residual Standard Error (RSE): it is the same as model sigma. We want small RSE. In real world practice, the difference between RMSE and RSE is very small, specifically when we are working on large scale multivariate data.

Mean Absolute error (MAE): it is similar with RMSE. It's measures the prediction error. formula = MAE = mean(abs(actual - theoretical))

Recall that, the more variable we add, Rˆ2 will increase as well. So, we need another method to measure the accuracy of the model

Concerning R2, we have adjusted R- Squared, which adjusts the R2 for having too many variables. Additionally, we have AIC, AICc, BIC, and Mallows Cp, that are commonly used for model evaluation and selection. Those are unbiased estimate of the model prediction error MSE. We want small # from those result.

AIC (Akaike's information Criteria): this is developed by the Japanese Statistician. The idea of AIC is to penalize the inclusion of additional variables to a model. it adds a penalty that increase the error when including additional terms. The lower the AIC, the better the model.

AICc: is a version of AIC corrected for a small sample sizes.

BIC (Bayesian information criteria): is derived from AIC with a stronger penalty for including additional variables to the model.

Mallows Cp: is derived from AIC developed by Colin Mallows.

Generally speaking, we use Adjusted R2, AIC, BIC, and Cp to measure the regression model quality and for comparing models.

Lets get in to the code

```r
if(!require("modelr")){
  install.packages("modelr")
  library(tidyverse)
  library(modelr)
  library(broom)
}
```

```
## Loading required package: modelr
```

```
##
## Attaching package: 'modelr'

## The following object is masked from 'package:broom':
##
##     bootstrap
```

Example of data

```
# load data
data("swiss")
# inspect the data
sample_n(swiss,3)
```

```
##            Fertility Agriculture Examination Education Catholic
## Porrentruy      76.1        35.3           9         7    90.57
## Entremont       69.3        84.9           7         6    99.68
## Aigle           64.1        62.0          21        12     8.52
##            Infant.Mortality
## Porrentruy             26.6
## Entremont              19.8
## Aigle                  16.5
```

Build the regression models model1 all parameters model2 except examination

```
models1 <- lm(Fertility ~., data = swiss)
models2 <- lm(Fertility ~., -Examination, data = swiss)
```

Now, lets check the model quality, the following code will look like duplicate because im showing how to find those result in different ways.

summary() will return Rsquared and adjusted R squared and the RSE AIC() and BIC() computes the AIC and BIC respectively.

```
summary(models1)
```

```
##
## Call:
## lm(formula = Fertility ~ ., data = swiss)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -15.2743  -5.2617   0.5032   4.1198  15.3213
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)      66.91518   10.70604   6.250 1.91e-07 ***
## Agriculture      -0.17211    0.07030  -2.448  0.01873 *
## Examination      -0.25801    0.25388  -1.016  0.31546
## Education        -0.87094    0.18303  -4.758 2.43e-05 ***
## Catholic          0.10412    0.03526   2.953  0.00519 **
## Infant.Mortality  1.07705    0.38172   2.822  0.00734 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.165 on 41 degrees of freedom
## Multiple R-squared:  0.7067, Adjusted R-squared:  0.671
## F-statistic: 19.76 on 5 and 41 DF,  p-value: 5.594e-10
```

```r
AIC(models1)
```

```
## [1] 326.0716
```

```r
BIC(models1)
```

```
## [1] 339.0226
```

rsquare(), rmse() and mae() [modelr package], computes, respectively, the R2, RMSE and the MAE.

```r
models1.r2 <- rsquare(models1, data =swiss)
models1.rmse <- rmse(models1, data = swiss)
models1.mae <- mae(models1, data = swiss)
```

R2(), RMSE() and MAE() [caret package], computes, respectively, the R2, RMSE and the MAE.

```r
library(caret)
models1.pred <- models1 %>% predict(swiss)
models1.R2 <- caret::R2(models1.pred, swiss$Fertility)
models1.RMSE <- RMSE(models1.pred, swiss$Fertility)
models1.MAE <- MAE(models1.pred, swiss$Fertility)
```

glance() [broom package], computes the R2, adjusted R2, sigma (RSE), AIC, BIC.

```r
library(broom)
glance(models1)
```

```
## # A tibble: 1 x 12
##   r.squared adj.r.squared sigma statistic  p.value    df logLik   AIC   BIC
##       <dbl>         <dbl> <dbl>     <dbl>    <dbl> <dbl>  <dbl> <dbl> <dbl>
## 1     0.707         0.671  7.17      19.8 5.59e-10     5  -156.  326.  339.
## # ... with 3 more variables: deviance <dbl>, df.residual <int>, nobs <int>
```

Now lets compare the performance

```r
name <- c("adj.r.squared", "sigma", "AIC" , "BIC", "p.value")
models1.temp <- c(glance(models1))
models1.performance <- models1.temp[c(2,3,8,9,5)]
models2.temp <- c(glance(models2))
models2.performance <- models2.temp[c(2,3,8,9,5)]
models.performance <- rbind(models1.performance, models2.performance)
models.performance
```

```
##                     adj.r.squared sigma    AIC      BIC      p.value
## models1.performance 0.670971      7.165369 326.0716 339.0226 5.593799e-10
## models2.performance 0.7630908     6.888644 174.5797 183.1118 2.493363e-06
```

From the above table we can see couple things

We have a better R^2 from model2.performance, which means models2 explain better for the predicted value, the model from models2 has a better accuracy.

RSE: models1 = 7.165369, models2 = 6.888644 RSE = sigma. So from models2 we have a smaller sigma which is a good thing. so models2 wins in this case.

Now, AIC and BIC. it is obvious that models2 has a better numbers than models1. remember we want small number from AIC and BIC. so models2 wins again

F statistics p value.

```r
5.593799e-10 > 2.493363e-06
```

```
## [1] FALSE
```

if you can compare it by looking at it, just calculate it with R. so models1 has a smaller F statistics. However, imo, i would take models2.performance as my model because of all those numbers compared with models1, models2 has a better accuracy.

```r
sigma(models1)/mean(swiss$Fertility)
```

```
## [1] 0.1021544
```

```r
sigma(models2)/mean(swiss$Fertility)
```

```
## [1] 0.0982092
```

Lastly, as you can see, models2 has a better prediction error.

Discussion

We have gone thru the overall performance of a regression model

The most important metrics are the Adjusted R^2, RMSE, AIC, and BIC. These metrics are also used as the basis of model comparison and optimal model selection.

Note that regression metrics are all internal measures. They tell you how well the model fits to the data in hand, training data set.

In general, we don't care how well the method works on the training data. Rater, we are interested in the accuracy of the predictions that we obtain when we apply our method to previously unseen test data. However, the test data is not always available making the test error very difficult to estimate. Because of this situation, we have cross-validation and bootstrap that are applied for estimating the test error (the prediction error rate) using training data.

sub section 2 - Cross-Validation Essentials in R

intro
Cross validation refers to a set of methods for measuring the performance of a given predictive model on new test data set

Idea of Cross validation:
1) Training set to build the model
2) Testing set to see the prediction error

Cross validation is also known as re-sampling method

In this section you will learn:
1) most commonly used statistical metrics that measure the performance of a regression model in predicting the outcome of new test data

2) validation set approach (data split)
   Leave one out cross validation
   K - fold Cross validation
   Repeated K fold cross validation

Those 4 methods have their advantages and drawbacks. Use it that fit the best of your problem. Mostly, K fold cross validation is recommended

required R packages

```r
library(tidyverse)
library(caret)
```

Preper the data

```r
# load the data
data("swiss")
# inspect the data
sample_n(swiss,3)
```

```
##              Fertility Agriculture Examination Education Catholic
## La Vallee         54.3        15.2          31        20     2.15
## Paysd'enhaut      72.0        63.5           6         3     2.56
## Entremont         69.3        84.9           7         6    99.68
##              Infant.Mortality
## La Vallee                10.8
## Paysd'enhaut             18.0
## Entremont                19.8
```

To see the accuracy of the model on predicting the outcome. simply we want to estimate the prediction error.

1) R^2
2) RMSE: square root the MSE
3) compute the prediction errors

R^2, RMSE, MAE are used to measure the regression model performance during the cross-validation.

Cross validation method 1) reserve a small sample of data set 2) build (or train) the model using the remaining part of the data set (unseen data) 3) test the effectiveness of the model on the reserved sample data set. If the model works well on the test data set, then it is good. (MSE)

The Validation set Approach

We split the data in to two part 80% for the training and 20% for the testing

```r
# split the data in to two part
set.seed(1234)
training.samples.swiss <- swiss$Fertility %>%
  createDataPartition(p =0.8, list =F)
train.data.swiss <- swiss[training.samples.swiss, ]
test.data.swiss <- swiss[-training.samples.swiss, ]
# build the model
model.swiss <- lm(Fertility ~., data = train.data.swiss)
# make predictions and compute the R2, RMSE, MAE
swiss.pred <- model.swiss %>%
  predict(test.data.swiss)
swiss.performance <- c(caret::R2(swiss.pred, test.data.swiss$Fertility),
                       RMSE(swiss.pred, test.data.swiss$Fertility),
                       MAE(swiss.pred, test.data.swiss$Fertility))
swiss.prediction.percent.error <- RMSE(swiss.pred, test.data.swiss$Fertility) / mean(test.data.swiss$Fe
swiss.performance
```

```
## [1] 0.7845619 7.2493973 5.8573987
```

```r
(swiss.prediction.percent.error*100)
```

```
## [1] 10.44394
```

When building the model, we always want to use the lowest test sample RMSE. RMSE and MAE are measured in the same scale as the outcome variable.

Note that, the validation set method is only useful if you have a lot of data. Disadvantage is that, we build a model on a fraction of the data set only. We might have leaving out some interesting information about data, leading to higher bias. So the test error rate can be highly variable, depending on which observations are included in the training set and which observations are included in the validation set.

LOOCV (leave one out cross validation)

Process 1) leave out one data point and build the model on the rest of the data 2) Test the model against the data point that is left out at step 1 and record the test error associated with the prediction 3) repeat the process for all data points 4) get the overall prediction error by taking the average of all these test error estimates recorded at step 2

code:

```r
# define training control
train.control <- trainControl(method = "LOOCV")

# Train the model
loocv.model <- train(Fertility ~., data = swiss, method = "lm", trControl = train.control)

# Summarize the result
print(loocv.model)
```

```
## Linear Regression
##
## 47 samples
##  5 predictor
##
## No pre-processing
## Resampling: Leave-One-Out Cross-Validation
## Summary of sample sizes: 46, 46, 46, 46, 46, 46, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   7.738618  0.6128307  6.116021
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

Good thing about LOOCV is that we reduce the potential bias. However, the process is repeated a lot of time, so it will take a long time if we have a large data set. Because the model will be tested each data point at each iteration, which might yield a higher variation in the prediction error, if some data points are outliers. So we need a good ratio of testing data points, thats why we have k fold cross validation method.

K Fold Cross Validation

1) Randomly split the data set into k-subsets (or k-fold)(for example 5 subsets)
2) Reserve one subset and train the model on all other subsets
3) Test the model on the reserved subset and record the prediction error
4) Repeat this process until each of the k subsets has served as the test set.
5) Compute the average of the k recorded errors. This is called the cross-validation error serving as the performance metric for the model.

KFCV is a powerful method to estimate the accuracy of a model.

"The most obvious advantage of k-fold CV compared to LOOCV is computational. A less obvious but potentially more important advantage of k-fold CV is that it often gives more accurate estimates of the test error rate than does LOOCV (James et al. 2014)."

the question will become, how to choose the right value of K

small k -> more biased and undesirable. high K = less biased but suffer from large variability.

high value will lead us to LOOCV approach. small K will take use to validation set approach

Most often, we use K= 5 / k = 10. These values have been shown empirically to yield test error rate estimates that suffer neither from excessively high bias nor form very high variance.

The following we use k = 10

```r
# define training control
set.seed(321)
train.control.k <- trainControl(method = "cv", number = 10)

# train the model
k.model <- train(Fertility ~., data = swiss, method = "lm", trControl = train.control.k)

# summarize the results
print(k.model)
```

```
## Linear Regression
##
## 47 samples
##  5 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 42, 41, 43, 42, 43, 42, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   7.462458  0.6749535  6.07299
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

If you check both model, k.model has a better R squared MAE and RMSE.

```r
print(loocv.model)
```

```
## Linear Regression
##
## 47 samples
##  5 predictor
##
## No pre-processing
## Resampling: Leave-One-Out Cross-Validation
## Summary of sample sizes: 46, 46, 46, 46, 46, 46, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   7.738618  0.6128307  6.116021
##
```

```
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

Lets go for repeated K fold cross validation

```r
set.seed(432)
# Define training control
train.control.k.repeat <- trainControl(method = "repeatedcv", number = 10,
                                       repeats = 3)
# train the model
model.k.repeat <- train(Fertility ~., data = swiss, method = "lm",
                        trControl = train.control)

# summarize the results
print(model.k.repeat)
```

```
## Linear Regression
##
## 47 samples
##  5 predictor
##
## No pre-processing
## Resampling: Leave-One-Out Cross-Validation
## Summary of sample sizes: 46, 46, 46, 46, 46, 46, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   7.738618  0.6128307  6.116021
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

Non repeat K fold RMSE Squared MAE
7.462458 0.6749535 6.07299 if you compared non repeat K fold, the non repeat k fold is a better model because it has a higher R squared and lower RMSE MAE,

In this section, we have go thru 4 different methods for assessing the performance of a model on unseen test data. 1) validation set approach, 2) leave-one-out cross-validation, 3) k-fold cross-validation 4) repeated k fold cross-validation

We generally recommend the repeated K fold cross-validation to estimate the prediction error rate. It can be used in regression and classification settings.

another method is using bootstrap re-sampling methods, which consists of repeatedly and randomly selecting a sample of n observations from the original data set, and to evaluate the model performance on each copy.

Sub-Section 3 - Bootstrap Re-sampling Essentials in R

bootstrap re-sampling method can be used to measure the accuracy of a predictive model. Also, it can measure the uncertainty associated with any statistical estimator.

Bootstrap re-sampling consist of repeatedly selecting a sample of n observations from the original data set and evaluate the model on each copy. An average standard error is then calculated with the results provide an indication of the overall variance of the model performance.

```r
library(tidyverse)
library(caret)
```

Load the data from swiss

```
# load the data
data("swiss")
# inspect the data
sample_n(swiss, 3)
```

```
##            Fertility Agriculture Examination Education Catholic
## Courtelary      80.2        17.0          15        12     9.96
## Grandson        71.7        34.0          17         8     3.30
## Boudry          70.4        38.4          26        12     5.62
##            Infant.Mortality
## Courtelary             22.2
## Grandson               20.0
## Boudry                 20.3
```

Bootstrap procedure

The bootstrap method is used to quantify the uncertainty associated with a given statistical estimator or with a predictive model.

It consists of randomly selecting a sample of n observations from the original data set. This subset, called bootstrap data set is then used to evaluate the model.

This procedure is repeated a large number of times and the standard error of the bootstrap estimate is then calculated. The results provide an indication of the variance of the models performance.

Note that, the sampling is performed with replacement, which means that the same observation can occur more than once in the bootstrap data set.

Evaluating a predictive model performance

sample size 100

```
# define training control
train.control.bs <- trainControl(method = "boot", number = 100)

# train the model
model.bs <- train(Fertility ~., data = swiss, method = "lm", trControl = train.control.bs)

# summarize the results
print(model.bs)
```

```
## Linear Regression
##
## 47 samples
##  5 predictor
##
## No pre-processing
## Resampling: Bootstrapped (100 reps)
## Summary of sample sizes: 47, 47, 47, 47, 47, 47, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   8.127003  0.6189821  6.531412
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

the model is alright. not too bad RMSE and MAE, Rsquare is .609 is okay

Lets talk about quantifying an estimator uncertainty and confidence intervals

Simply saying, we want to estimate the accuracy of the linear regression beta coefficient using bootstrap method. 1) Create a simple function, model_coef(), that takes the swiss data set as well as the indices for the observations, and returns the regression coefficients. 2) apply the function boot_fun() to the full data set of 47 observations in order to compute the coefficients

```
model_coef <- function(data, index){
  coef(lm(Fertility ~., data = data, subset = index))
}
model_coef(swiss, 1:47)
```

```
##      (Intercept)      Agriculture      Examination        Education 
##       66.9151817       -0.1721140       -0.2580082       -0.8709401 
##         Catholic Infant.Mortality 
##        0.1041153        1.0770481
```

Let use boot() function to compute the standard error of 500 bootstrap estimates for the coefficients

```
library(boot)
```

```
## 
## Attaching package: 'boot'

## The following object is masked from 'package:car':
## 
##     logit

## The following object is masked from 'package:lattice':
## 
##     melanoma
```

```
boot(swiss, model_coef, 500)
```

```
## 
## ORDINARY NONPARAMETRIC BOOTSTRAP
## 
## 
## Call:
## boot(data = swiss, statistic = model_coef, R = 500)
## 
## 
## Bootstrap Statistics :
##       original        bias    std. error
## t1* 66.9151817 -0.2277779630 10.93143313
## t2* -0.1721140 -0.0029933680  0.06556137
## t3* -0.2580082 -0.0232585784  0.26616110
## t4* -0.8709401  0.0202862351  0.23240613
## t5*  0.1041153 -0.0004434766  0.03281364
## t6*  1.0770481  0.0332737829  0.44912907
```

The original column corresponds to the regression coef, with the associated standard errors t1 corresponds to the intercept, t2 corresponds to agriculture and so on

```
names(swiss)
```

```
## [1] "Fertility"        "Agriculture"      "Examination"      "Education"       
## [5] "Catholic"         "Infant.Mortality"
```

the standard error of the regression coefficient associated with Agriculture is 0.07. The standard errors measure the variability / accuracy of the beta coef. it can be used to compute the confidence intervals of the coefficients.

```
summary(lm(Fertility ~., data = swiss))$coef
```

```
##                    Estimate  Std. Error   t value     Pr(>|t|)
## (Intercept)      66.9151817 10.70603759  6.250229 1.906051e-07
## Agriculture      -0.1721140  0.07030392 -2.448142 1.872715e-02
## Examination      -0.2580082  0.25387820 -1.016268 3.154617e-01
## Education        -0.8709401  0.18302860 -4.758492 2.430605e-05
## Catholic          0.1041153  0.03525785  2.952969 5.190079e-03
## Infant.Mortality  1.0770481  0.38171965  2.821568 7.335715e-03
```

The bootstrap approach does not rely on any of these assumptions made by the linear model, and so it is likely giving a more accurate estimate of the coefficients and standard errors than the summary function.

#Section 4 - Model Selection Essentials in R

Best subsets regression means test all possible combination of the predictors, and then select the best model.

Stepwise regression means adding and deleting predictors in order to find the best performing model with a reduced set of variables.

Penalized Regression (ridge and lasso regression) and Principal components based regression method.

This this section we will talk about 1) Best subsets selection 2) Stepwise Selection 3) penalized Regression 4) Dimension Reduction Methods

load libraries and data Quick guide, lapply will simply load all the libraries from the vector

```
lib <- c("tidyverse", "caret", "leaps")
lapply(lib, require, character.only = T)
```

```
## Loading required package: leaps
```

```
## [[1]]
## [1] TRUE
##
## [[2]]
## [1] TRUE
##
## [[3]]
## [1] TRUE
```

```
data("swiss")
sample_n(swiss, 3)
```

```
##              Fertility Agriculture Examination Education Catholic
## Vevey             58.3        26.8          25        19    18.46
## Orbe              57.4        54.1          20         6     4.20
## V. De Geneve      35.0         1.2          37        53    42.34
##              Infant.Mortality
## Vevey                    20.9
## Orbe                     15.3
## V. De Geneve             18.0
```

Computing the best subset regression

nvmax = 5 is because we only have 5 variables

```r
modelsub <- regsubsets(Fertility ~., data = swiss, nvmax =5)
summary(modelsub)
```

```
## Subset selection object
## Call: regsubsets.formula(Fertility ~ ., data = swiss, nvmax = 5)
## 5 Variables  (and intercept)
##                  Forced in Forced out
## Agriculture          FALSE      FALSE
## Examination          FALSE      FALSE
## Education            FALSE      FALSE
## Catholic             FALSE      FALSE
## Infant.Mortality     FALSE      FALSE
## 1 subsets of each size up to 5
## Selection Algorithm: exhaustive
##          Agriculture Examination Education Catholic Infant.Mortality
## 1  ( 1 ) " "         " "         "*"       " "      " "
## 2  ( 1 ) " "         " "         "*"       "*"      " "
## 3  ( 1 ) " "         " "         "*"       "*"      "*"
## 4  ( 1 ) "*"         " "         "*"       "*"      "*"
## 5  ( 1 ) "*"         "*"         "*"       "*"      "*"
```

Now, at the bottom of the table, row 1 to 5. It simply means the best 1, 2, . . . , 5 variables model is what.

The best alone model is simply include education. Best 2 variables model is include education and catholic and so on for the next 3 models.

Now the problem is, which one to choose. Now, we need some statistical metrics or strategies to compare all the performance, and pick the best one, which means low BIC, cp, and high Adj R2.

```r
perf.modelsub <- summary(modelsub)
df.performance.modelsub <- data.frame(
  Adj.R2 = which.max(perf.modelsub$adjr2),
  CP = which.min(perf.modelsub$cp),
  BIC = which.min(perf.modelsub$bic)
)
df.performance.modelsub
```

```
##   Adj.R2 CP BIC
## 1      5  4   4
```

As we can see from the table above, Adj.R2 is the best with model 5, however; if we are looking at the CP and BIC, we should shift to model 4 instead.

here is a important question, are those metrics from a test data or training data? You might need to sleep on it if you are having trouble to identify what is training data and testing data.

Another approach is to select a models based on prediction error computed on a new test data using k fold cross validation techniques.

K fold cross-validation

Remember, we either set k to 5 or 10.

Lets make a function

```r
#       id: model id
#   object: regsubsets object
#     data: data used to fit regsubsets
#  outcome: outcome varaible
```

```r
get_model_formula <- function(id, object, outcome){
  # get model data
  getModelData <- summary(object)$which[id,-1]
  # get outcome varaible
  form <- as.formula(object$call[[2]])
  outcome <- all.vars(form)[1]
  # get model predictors
  predictors.a <- names(which(getModelData == T))
  predictors.a <- paste(predictors.a, collapse = "+")
  # build model formula
  as.formula(paste0(outcome, "~", predictors.a))
}
```

lets get the best 3 variable model formula,

```r
get_model_formula(3, modelsub, "Fertility")
```

```
## Fertility ~ Education + Catholic + Infant.Mortality
## <environment: 0x7fde4735d158>
```

let's build the get_cv_error() function for cross-validation error for a given model

```r
get_cv_error <- function(model.formula, data){
  set.seed(3)
  train.controlsub <- trainControl(method = "cv", number = 5)
  cv <- train(model.formula, data = data,
              method = "lm", trControl = train.control)
  cv$results$RMSE
}
```

lets get the error

```r
# compute cross-validation error
model.ids <- 1:5
cv.errors <- map(model.ids, get_model_formula, modelsub, "Fertility") %>%
  map(get_cv_error, data = swiss) %>%
  unlist()
cv.errors
```

```
## [1] 9.591367 8.617911 7.857175 7.614933 7.738618
```

Now, get the smallest RMSE

```r
which.min(cv.errors)
```

```
## [1] 4
```

```r
coef(modelsub, 4)
```

```
##      (Intercept)      Agriculture        Education         Catholic 
##       62.1013116       -0.1546175       -0.9802638        0.1246664 
## Infant.Mortality 
##        1.0784422
```

This entire section has gone thru how to find the best model out of multiple variables.

Sub-Section 2 - Stepwise Regression Essentials in R

Stepwise Regression is constantly adding and removing predictor variables and see who has the best performance, which implies lowers prediction error.

There are three methods.

Forward selection, starts with no predictors and then stop when there is no longer statistically significant.

Backward selection (or backward elimination), starts from all possible predictors, and removes the least contribute predictors and stop when all predictors are statistically significant.

Stepwise selection (sequential replacement), which is a combination of forward and backward selections. start with no predictors, and sequentially adding the most contributing predictors (like forward selection). After adding each new variable, remove any variables that are no longer provide an improvement in the model fit (like backward selection).

Restriction Forward selection and stepwise selection can be applied in the high dimensional configuration, where the number of sample n is inferior to the number of predictors p, such as in genomic fields.

Backward selection requires that the number of samples n is larger than the number of variables p, so that the full model can be fit.

library tidyverse, caret, leaps, MASS

```r
libsteps <- c("tidyverse", "caret", "leaps", "MASS")
lapply(libsteps, require, character.only = T)
```

```
## [[1]]
## [1] TRUE
##
## [[2]]
## [1] TRUE
##
## [[3]]
## [1] TRUE
##
## [[4]]
## [1] TRUE
```

```r
# fit the full model
full.model <- lm(Fertility ~., data = swiss)
# stepwise regression model
step.model <- stepAIC(full.model, direction = "both", trace = F)
```

regsubsets() from leaps package, which has the tuning parameter nvmax. please see the previous section where i have initially introduced you this parameter. It returns multiple models with different size up to nvmax. you need to compare the performance of those models for picking the best one. regsubsets() has the option method for you to specify which selection you would like to make. backward, forward, seqrep

```r
modelselection <- regsubsets(Fertility ~., data = swiss, nvmax =5, method = "seqrep")
summary(modelselection)
```

```
## Subset selection object
## Call: regsubsets.formula(Fertility ~ ., data = swiss, nvmax = 5, method = "seqrep")
## 5 Variables  (and intercept)
##                  Forced in Forced out
## Agriculture          FALSE      FALSE
## Examination          FALSE      FALSE
## Education            FALSE      FALSE
## Catholic              FALSE      FALSE
## Infant.Mortality      FALSE      FALSE
## 1 subsets of each size up to 5
## Selection Algorithm: 'sequential replacement'
##          Agriculture Examination Education Catholic Infant.Mortality
## 1  ( 1 ) " "         " "         "*"       " "      " "
## 2  ( 1 ) " "         " "         "*"       "*"      " "
## 3  ( 1 ) " "         " "         "*"       "*"      "*"
## 4  ( 1 ) "*"         "*"         "*"       "*"      " "
## 5  ( 1 ) "*"         "*"         "*"       "*"      "*"
```

train() from caret package also has a easier work flow to perform stepwise selection using the leaps and mass package. It has an option named method, with the following values "leapBackward", "leapForward", "leapSeq"

Following code will also apply 10 k fold cv to estimate the average prediction error.

```
set.seed(999)
train.control.selection <- trainControl(method = "cv", number =10)
step.model.selection <- train(Fertility ~., data = swiss,
                              method = "leapBackward",
                              tuneGrid = data.frame(nvmax = 1:5),
                              trControl = train.control)
step.model.selection
```

```
## Linear Regression with Backwards Selection
##
## 47 samples
##  5 predictor
##
## No pre-processing
## Resampling: Leave-One-Out Cross-Validation
## Summary of sample sizes: 46, 46, 46, 46, 46, 46, ...
## Resampling results across tuning parameters:
##
##   nvmax  RMSE      Rsquared   MAE
##   1      9.591367  0.3991715  8.107915
##   2      9.930877  0.3740050  8.283843
##   3      8.059676  0.5771867  6.506707
##   4      7.614933  0.6236409  6.071138
##   5      7.738618  0.6128307  6.116021
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was nvmax = 4.
```

Again, we want small # for RMSE and MAE, also we want as close to 1 as possible for r^2

```
step.model.selection$bestTune
```

```
##   nvmax
## 4     4
```

the bestTune wil give us the best model to select

```
summary(step.model.selection$finalModel)
```

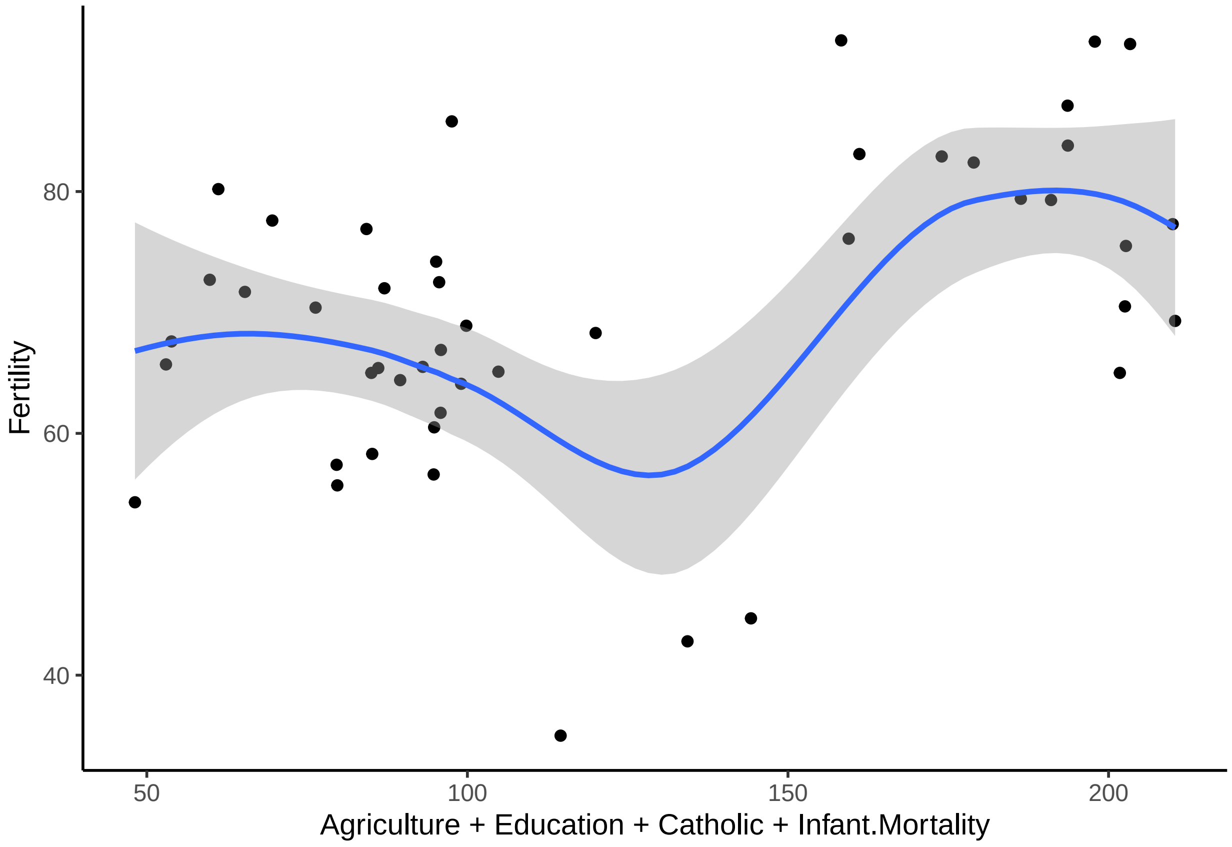
```
## Subset selection object
## 5 Variables  (and intercept)
##                    Forced in Forced out
## Agriculture            FALSE      FALSE
## Examination            FALSE      FALSE
## Education              FALSE      FALSE
## Catholic               FALSE      FALSE
## Infant.Mortality       FALSE      FALSE
## 1 subsets of each size up to 4
## Selection Algorithm: backward
##          Agriculture Examination Education Catholic Infant.Mortality
## 1  ( 1 ) " "         " "         "*"       " "      " "
## 2  ( 1 ) " "         " "         "*"       "*"      " "
## 3  ( 1 ) " "         " "         "*"       "*"      "*"
## 4  ( 1 ) "*"         " "         "*"       "*"      "*"
```

```
coef(step.model.selection$finalModel, 4)
```

```
##      (Intercept)      Agriculture        Education         Catholic
##       62.1013116       -0.1546175       -0.9802638        0.1246664
## Infant.Mortality
##        1.0784422
```

```
step.model.selection.lm <- lm(Fertility ~ Agriculture + Education + Catholic + Infant.Mortality, data =
ggplot(step.model.selection.lm, aes(x = Agriculture + Education + Catholic + Infant.Mortality, Fertilit
  geom_point() +
  stat_smooth()
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



In MASS library, it has a function called stepAIC. you can use the direction parameter and set the value to both, forward, and backward for the selection.

In addition, caret package has method to compute stepwise regression using the MASS package, with method = "lmStepAIC"

please read the documentation on how to apply selection method.

Conclusion,

Stepwise regression is very useful and important for high - dimensional data that has multiple predictor variables. Other alternatives are the penalized regression (ridge and lasso regression) and the principal components based regression method (PCR and PLS)

Sub-Section 3 - Penalized Regression Essentials: Ridge, Lasso & Elastic Net

The standard linear model doesn't work well when we have a lot of parameters. So we have alternative method which is penalized regression. This will add a constraint in the equation, which is known as shrinkage or regularization methods.

what the penalty does is to reduce the coefficient values towards zero. This will eliminate those less contribution variables to the regression model, down to zero or at least close to zero.

In the mean time, the reduction requires the tuning parameter (lambda) to determines the amount of shrinkage.

In this section we will discuss the most commonly used penalized regression methods, including ridge regression, lasso regression and elastic net regression.

Library tidyverse, caret, glmnet (computing penalized regression)

```
libpen <- c("tidyverse", "caret", "glmnet")
lapply(libpen, require, character.only = T)
```

```
## Loading required package: glmnet

## Loading required package: Matrix

##
## Attaching package: 'Matrix'

## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack

## Loaded glmnet 4.0-2

## [[1]]
## [1] TRUE
##
## [[2]]
## [1] TRUE
##
## [[3]]
## [1] TRUE
```

```
data("Boston", package = "MASS")
# By now i hope you are already familiar wit the next 5 lines of code
set.seed(323)
training.samples.pen <- Boston$medv %>%
  createDataPartition(p = 0.8, list = F)
train.data.pen <- Boston[training.samples.pen, ]
test.data.pen <- Boston[-training.samples.pen, ]
# Predictor variables
x.pen <- model.matrix(medv~., train.data.pen)[, -1]
y.pen <- train.data.pen$medv
```

glmnet() is for computing penalized linear regression models. glmnet(x.pen, y.pen, alpha =1, lambda = NULL)

x: matrix of predictor variables y: the response or outcome variable, which is a binary variable. alpha: the elasticnet mixing parameter. Allowed values include: "1": for lasso regression "0": for ridge regression a value between 0 and 1 (say 0.3) for elastic net regression. lambda: a numeric value defining the amount of shrinkage. Should be specify by analyst.

In penalized regression, you need to specify a constant lambda to adjust the amount of the coefficient shrinkage. The best lambda for your data, can be defined as the lambda that minimize the cross-validation prediction error rate. This can be determined automatically using the function cv.glmnet().

Lets dig into the real code

Computing Ridge Regression

```
# find the best lambda using cross- validation
set.seed(32423)
cv.pen <- cv.glmnet(x.pen, y.pen, alpha = 0)
cv.pen$lambda.min
```

```
## [1] 0.6619773
```

Let's use this lambda

```
model.pen <- glmnet(x.pen, y.pen, alpha = 0, lambda = cv.pen$lambda.min)
# Display regression coefficients
coef(model.pen)
```

```
## 14 x 1 sparse Matrix of class "dgCMatrix"
##                        s0
## (Intercept)  28.232030472
## crim         -0.089534050
## zn            0.036242574
## indus        -0.031221170
## chas          2.134380075
## nox         -12.039681211
## rm            3.920996982
## age          -0.006864208
## dis          -1.083784241
## rad           0.163168176
## tax          -0.005723189
## ptratio      -0.845413054
## black         0.008149447
## lstat        -0.444799608
```

Make predictions on the test data

```
x.pen.test <- model.matrix(medv ~., test.data.pen)[,-1]
pred.pen <- model.pen %>% predict(x.pen.test) %>% as.vector()
```

```r
# model performance metrics
pen.performance <- data.frame(
  RMSE = RMSE(pred.pen, test.data.pen$medv),
  R2 = caret::R2(pred.pen, test.data.pen$medv),
  perError = RMSE(pred.pen, test.data.pen$medv) / mean(test.data.pen$medv)
)
pen.performance
```

```
##       RMSE        R2 perError
## 1 5.223382 0.7698707 0.226606
```

Computing Lasso Regression

```r
# find the best lambda using cross- validation
set.seed(222)
cv.lasso <- cv.glmnet(x.pen,y.pen, alpha =1)
# display the best lambda value
cv.lasso$lambda.min
```

```
## [1] 0.03001987
```

build the model with training data

```r
model.lasso <- glmnet(x.pen, y.pen, alpha = 1, lambda = cv.lasso$lambda.min)
# display regression coefficients
coef(model.lasso)
```

```
## 14 x 1 sparse Matrix of class "dgCMatrix"
##                        s0
## (Intercept)  34.945127094
## crim         -0.102476332
## zn            0.046148759
## indus         .
## chas          1.869352924
## nox         -16.237862317
## rm            3.733797896
## age          -0.001125746
## dis          -1.357155920
## rad           0.267631419
## tax          -0.009948424
## ptratio      -0.932842063
## black         0.008107730
## lstat        -0.492879263
```

Make prediction

```r
x.test.lasso <- model.matrix(medv ~., test.data.pen)[, -1]
pred.lasso <- model.lasso %>% predict(x.test.lasso) %>% as.vector()
# model performance metrics
lasso.perf <- data.frame(
  RMSE = RMSE(pred.lasso, test.data.pen$medv),
  R2 = caret::R2(pred.lasso, test.data.pen$medv),
  perError = RMSE(pred.lasso, test.data.pen$medv) / mean(test.data.pen$medv)
)
lasso.perf
```

```
##       RMSE        R2  perError
## 1 5.123271 0.7727101 0.2222628
```

Last one, elastic net regression

Build the model, it is more simple then the previous twos. We can use caret work flow to implement everything. caret will automatically select the best tuning parameters alpha and lambda. Why? because it will tests a range of possible alpha and lambda values, and select the best #.

Lets dive in to the code

```
set.seed(32132)
model.elastic <- train(
  medv ~., data = train.data.pen, method = "glmnet",
  trControl = trainControl("cv", number = 10),
  tuneLength = 10
)
# Best tuning parameter
model.elastic$bestTune
```

```
##    alpha     lambda
## 15   0.2 0.08710736
```

Coefficient of the final model. You need to specify the best lambda

```
coef(model.elastic$finalModel, model.elastic$bestTune$lambda)
```

```
## 14 x 1 sparse Matrix of class "dgCMatrix"
##                         1
## (Intercept)  34.470824042
## crim         -0.102830127
## zn            0.045773890
## indus         .
## chas          1.916667364
## nox         -15.961162078
## rm            3.760890396
## age          -0.002868719
## dis          -1.348319420
## rad           0.263050842
## tax          -0.009777302
## ptratio      -0.928301292
## black         0.008226864
## lstat        -0.486022274
```

Make Prediction

```
x.test.elastic <-model.matrix(medv ~., test.data.pen)[, -1]
pred.elastic <- model.elastic %>% predict(test.data.pen)
# model performance metrics
elastic.perf <- data.frame(
  RMSE = RMSE(pred.elastic, test.data.pen$medv),
  R2 = caret::R2(pred.elastic, test.data.pen$medv),
  perError = RMSE(pred.elastic,test.data.pen$medv) / mean(test.data.pen$medv)
)
elastic.perf
```

```
##       RMSE       R2  perError
## 1 5.127825 0.772687 0.2224604
```

```r
penalized.performance <- rbind.data.frame(pen.performance , lasso.perf, elastic.perf)
Pen.Test.Name <- c("Ridge", "Lasso", "Elastic")
penalized.performance <- cbind.data.frame(Pen.Test.Name, penalized.performance)
penalized.performance
```

```
##   Pen.Test.Name     RMSE        R2  perError
## 1         Ridge 5.223382 0.7698707 0.2266060
## 2         Lasso 5.123271 0.7727101 0.2222628
## 3       Elastic 5.127825 0.7726870 0.2224604
```

As you can see the table above, we can either use Lasso model or Elastic. Imo, i would use Lasso, because of smaller RMSE, slightly higher R2, but more importantly, i have smaller percent error. If i have huge data set, 0.1% does make a different.

The following code is repeating the process above, but when we use caret package, it will be simpler.

Setup a grid range of lambda values

```r
lambda <-  10^seq(-3,3, length = 100)
# Ridge Regression
# Build the model
set.seed(123)
model.ridge <- train(
  medv ~., data = train.data.pen, method = "glmnet",
  trControl = trainControl("cv", number = 10),
  tuneGrid = expand.grid(alpha = 0, lambda = lambda)
  )
# Model coefficients
coef(model.ridge$finalModel, model.ridge$bestTune$lambda)
```

```
## 14 x 1 sparse Matrix of class "dgCMatrix"
##                         1
## (Intercept)  28.191064926
## crim         -0.089284990
## zn            0.036328235
## indus        -0.030992288
## chas          2.130837097
## nox         -11.960838032
## rm            3.917523323
## age          -0.006704166
## dis          -1.081966364
## rad           0.162842416
## tax          -0.005741076
## ptratio      -0.844484238
## black         0.008152874
## lstat        -0.445670277
```

```r
# Make predictions
ridge.pred <- model.ridge %>% predict(test.data.pen)
# Model prediction performance
ridge.perf <- data.frame(
  RMSE = RMSE(ridge.pred, test.data.pen$medv),
  R2 = caret::R2(ridge.pred, test.data.pen$medv),
  perError = RMSE(ridge.pred, test.data.pen$medv) / mean(test.data.pen$medv)
)
```

Lasso Regression

```
# Build the model
set.seed(123)
lasso.model <- train(
  medv ~., data = train.data.pen, method = "glmnet",
  trControl = trainControl("cv", number = 10),
  tuneGrid = expand.grid(alpha = 1, lambda = lambda)
  )
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo, :
## There were missing values in resampled performance measures.
```

```
# Model coefficients
coef(lasso.model$finalModel, lasso.model$bestTune$lambda)
```

```
## 14 x 1 sparse Matrix of class "dgCMatrix"
##                        1
## (Intercept)  34.702436000
## crim         -0.101281965
## zn            0.045572769
## indus         .
## chas          1.866717118
## nox         -16.116721558
## rm            3.741896185
## age          -0.000923771
## dis          -1.343836057
## rad           0.262070223
## tax          -0.009718290
## ptratio      -0.930921265
## black         0.008082672
## lstat        -0.493240898
```

```
# Make predictions
lasso.pred <- lasso.model %>% predict(test.data.pen)
# Model prediction performance
lasso.perf <- data.frame(  RMSE = RMSE(lasso.pred, test.data.pen$medv),
                             R2 = caret::R2(lasso.pred, test.data.pen$medv),
                  perError = RMSE(lasso.pred, test.data.pen$medv) / mean(test.data.pen$medv)
)
```

Elastic net regression

```
# Build the model
set.seed(123)
elastic.model <- train(
  medv ~., data = train.data.pen, method = "glmnet",
  trControl = trainControl("cv", number = 10),
  tuneLength = 10
  )
# Model coefficients
coef(elastic.model$finalModel, elastic.model$bestTune$lambda)
```

```
## 14 x 1 sparse Matrix of class "dgCMatrix"
##                        1
## (Intercept)  3.454809e+01
## crim        -1.006986e-01
## zn           4.528772e-02
## indus        .
## chas         1.866481e+00
## nox         -1.604215e+01
## rm           3.746872e+00
## age         -8.610915e-04
## dis         -1.336128e+00
## rad          2.587846e-01
## tax         -9.580351e-03
## ptratio     -9.294301e-01
## black        8.069791e-03
## lstat       -4.931172e-01
```

```
# Make predictions
predictions <- elastic.model %>% predict(test.data.pen)
# Model prediction performance
elastic.perf <- data.frame(
  RMSE = RMSE(predictions, test.data.pen$medv),
  R2 = caret::R2(predictions, test.data.pen$medv),
  perError = RMSE(predictions, test.data.pen$medv) / mean(test.data.pen$medv)
)
```

The performance of the different models - ridge, lasso and elastic net - can be easily compared using caret. The best model is defined as the one that minimizes the prediction error.

```
modelsss <- list(ridge = model.ridge, lasso = lasso.model, elastic = elastic.model)
resamples(modelsss) %>% summary(metric = "RMSE")
```

```
##
## Call:
## summary.resamples(object = ., metric = "RMSE")
##
## Models: ridge, lasso, elastic
## Number of resamples: 10
##
## RMSE
##             Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## ridge   2.985268 3.890679 4.468207 4.653439 5.288179 6.996436    0
## lasso   3.010413 3.868703 4.438512 4.621414 5.251428 6.920674    0
## elastic 3.008035 3.865748 4.438152 4.621372 5.251111 6.922004    0
```

Amount all 3 models, elastic has the lowest median RMSE.

However, if we conclude everything together, lasso would be my first option

```
penalized.performance
```

```
##   Pen.Test.Name     RMSE        R2  perError
## 1         Ridge 5.223382 0.7698707 0.2266060
## 2         Lasso 5.123271 0.7727101 0.2222628
## 3       Elastic 5.127825 0.7726870 0.2224604
```

Sub- section 4 - rincipal Component and Partial Least Squares Regression Essentials

Intro

This section will be talking about dimension reduction. It is very useful if we have a large data set with multiple correlated predictor variables.

The way how reduction methods work is: First summarizing the original predictors into few new variables called principal components (PCs), which are used as predictors to fit the linear regression model. It will avoid multicollinearity between predictors, which is a big issue in regression setting.

When using dimension reduction methods, it's generally recommended to standardize each predictor to make them comparable. Standardization consists of dividing the predictor by its standard deviation.

Two well known regression methods based on dimension reduction: Principal Component Regression (PCR) Partial Least Squares (PLS)

Principal Component Regression

PCR means summarize the original predictor variables into few new variables also known as principal components (PC), which are a linear combination of the original data.

Those PC are used to build the linear regression model. Those PC are chosen by cross-validation (CV).

Note that, PCR is suitable when the data set contains highly correlated predictors.

Partial Least Squares Regression

Downside of PCR is that, we cant guarantee that the selected principal components are associated with the outcome. The selection of the principal components that are included in the model is not supervised by the outcome variable.

The alternative to PCR is Partial Least Squares (PLS) Regression. PLS identifies new principal components that not only summarizes the original predictors, but also that are related to the outcome. Then those components are used to fit the regression model. Compared to PCR, PLS uses a dimension reduction strategy that is supervised by the outcome.

Like PCR, PLS is convenient for data with highly- correlated predictors. The # of PCs used in PLS is generally chosen by the Cross-Validation. To make variables comparable, predictors and the outcome variables should be generally standardized. Prepare the data

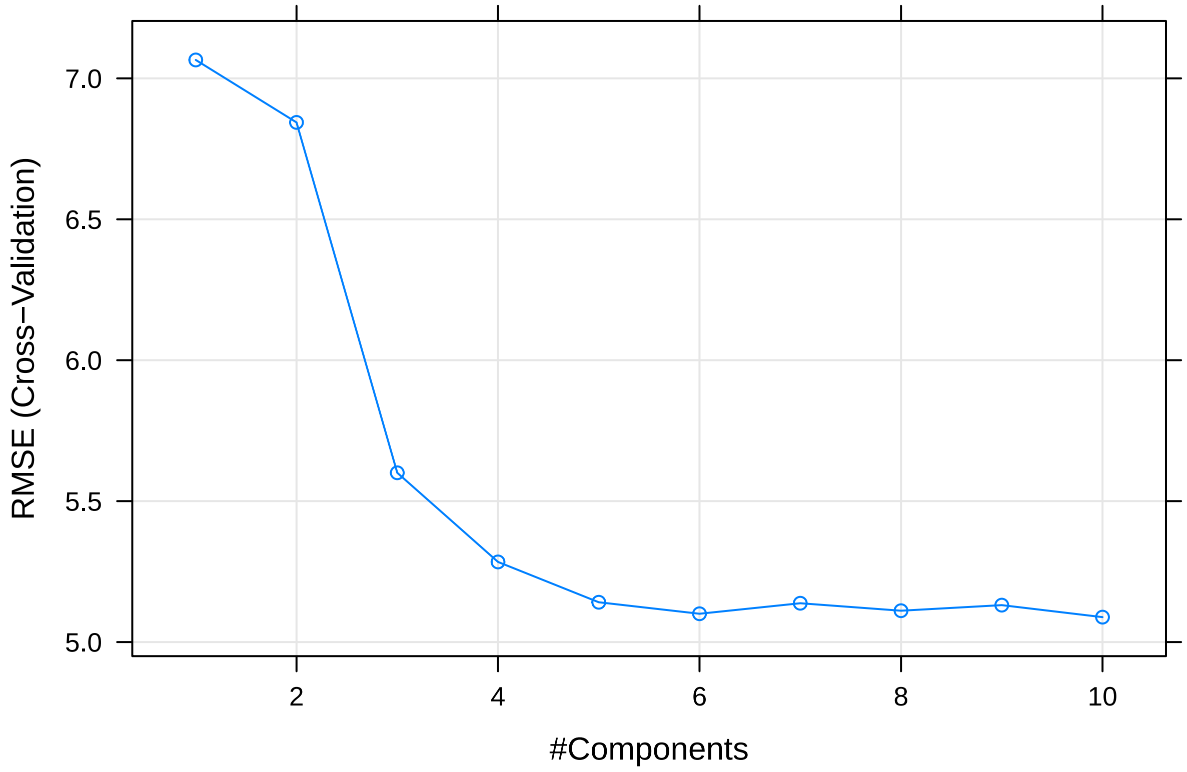
```r
library(tidyverse)
library(caret)
library(pls)
```

```
##
## Attaching package: 'pls'

## The following object is masked from 'package:caret':
##
##     R2

## The following object is masked from 'package:stats':
##
##     loadings
```

```r
# Load the data
data("Boston", package = "MASS")
# Split the data into training and test set
set.seed(123)
training.samples <- Boston$medv %>%
  createDataPartition(p = 0.8, list = FALSE)
train.data  <- Boston[training.samples, ]
test.data <- Boston[-training.samples, ]
```

Computing Principal Component Regression

```
# Build the model on training set
set.seed(123)
model <- train(
  medv~., data = train.data, method = "pcr",
  scale = TRUE,
  trControl = trainControl("cv", number = 10),
  tuneLength = 10
  )
# Plot model RMSE vs different values of components
plot(model)
```



From the plot, we can see that 6 or 10 is our best principal components number. which also gives the smallest prediction error RMSE.

Without guessing we can use the following code

```
# Print the best tuning parameter ncomp that
# minimize the cross-validation error, RMSE
model$bestTune
```

```
##    ncomp
## 10    10
```

It said, the best model is 10

```r
# Summarize the final model
summary(model$finalModel)
```

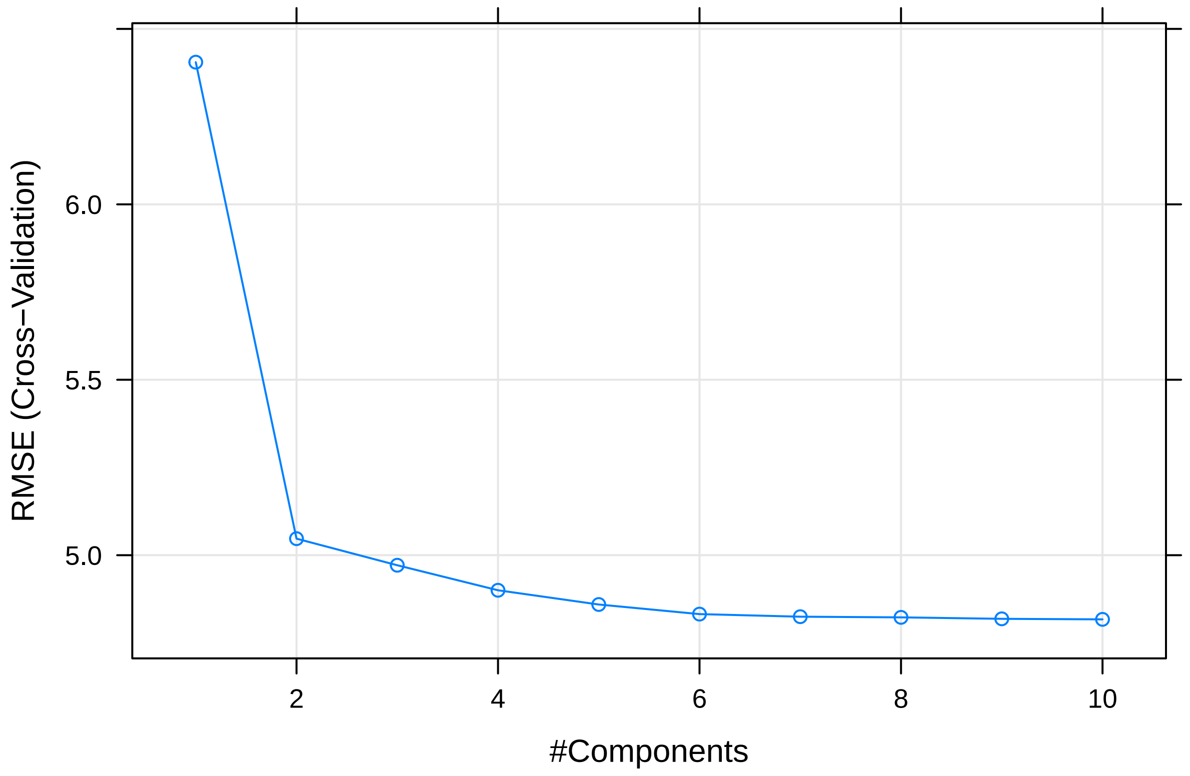
```
## Data:    X dimension: 407 13
##  Y dimension: 407 1
## Fit method: svdpc
## Number of components considered: 10
## TRAINING: % variance explained
##           1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## X           48.02    58.82    67.97    74.78    81.04    86.16    90.21
## .outcome    37.80    43.76    62.57    67.24    68.66    69.30    69.31
##           8 comps  9 comps  10 comps
## X           93.21    95.31     96.92
## .outcome    69.87    69.88     70.64
```

From the last two rows, we want big number. For 10 comps, 96.92% of the variation (or information) contained in the predictors are captured by 6 principal components. Also, setting ncomp = 10 will captures 70.64% of the information in the outcome variable, which is pretty good.

```r
# Make predictions
predictions <- model %>% predict(test.data)
# Model performance metrics
pcr.performance <-data.frame(
  RMSE = caret::RMSE(predictions, test.data$medv),
  Rsquare = caret::R2(predictions, test.data$medv),
  perError = caret::RMSE(predictions, test.data$medv) / mean(test.data$medv)
)
```

2

```r
# Build the model on training set
set.seed(123)
model <- train(
  medv~., data = train.data, method = "pls",
  scale = TRUE,
  trControl = trainControl("cv", number = 10),
  tuneLength = 10
  )
# Plot model RMSE vs different values of components
plot(model)
```



```r
# Print the best tuning parameter ncomp that
# minimize the cross-validation error, RMSE
model$bestTune
```

```
##    ncomp
## 10    10
```

```r
# Summarize the final model
summary(model$finalModel)
```

```
## Data:    X dimension: 407 13
##  Y dimension: 407 1
## Fit method: oscorespls
## Number of components considered: 10
## TRAINING: % variance explained
##           1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## X           46.91    57.21    64.47    69.89    75.76    79.59    83.17
## .outcome    49.21    70.10    71.68    72.72    73.13    73.28    73.38
##           8 comps  9 comps  10 comps
## X           85.82    90.83     92.65
## .outcome    73.45    73.46     73.46
```

```r
# Make predictions
predictions <- model %>% predict(test.data)
# Model performance metrics
pls.performance <- data.frame(
  RMSE = caret::RMSE(predictions, test.data$medv),
  Rsquare = caret::R2(predictions, test.data$medv),
  perError = caret::RMSE(predictions, test.data$medv) / mean(test.data$medv)
)
rbind.data.frame(PCR = pcr.performance, PLS = pls.performance)
```

```
##         RMSE   Rsquare  perError
## PCR 4.935417 0.7270573 0.2181473
## PLS 4.590163 0.7610013 0.2028869
```

Unfortunately, we get exactly the same data, so we can either use both model.

Comment:

I have been suffering from the following error "Error in UseMethod("R2") : no applicable method for 'R2' applied to an object of class"c('double', 'numeric')""

The way I fixed this problem is to add caret::R2()

Section 4 - Classification methods essential - total sub sections 9

Classification Methods Essentials

From the previous 3 sections, we have talked about how to predict a quantitative or continuous outcome variables based on one or multiple predictor variables.

In this section, classification, the out come variables are qualitative (or categorical). Classification means a set of machine learning methods for prediction the class (or category) of individuals on the basis of one of multiple predictor variables.

Topics we be talked about in this section:

- Logistic regression, for binary classification tasks
- Stepwise and penalized logistic regression for variable selections
- Logistic regression assumptions and diagnostics
- Multi-nomial logistic regression, an extension of the logistic regression for multi-class classification tasks
- Discriminant analysis, for binary and multi-class classification problems
- Naive Bayes classifier
- Support vector machines
- Classification model evaluation

Classification simply refer to what type of thing is it. For example, input is apple, when you pass it in to those methods above, it will try to find out what is your input, calculated in probability, and tell you what is the input.

You would need a cutoff point for probability so that you can make things work.

Sub-section 1 - Logistic Regression Essential in R

Let get started with Logistic Regression Essentials in R

In Logistic Regression, it will tell you yes or no. Is it an apple? Yes / No thats it.

The equation usually comes in individuals based on one or multiple predictor variables (x). It returns Y / N like Bernoulli or Binomial distribution, success of failure.

Logistic Regression comes from a family called Generalized Linear Model (GLM). It is a extension of linear regression model. Other synonyms are binary logistic regression, binomial logistic regression, and logit model.

Logistic Regression does not return directly the class of observation, it tells you the probability if it belongs to that specific class or not. So thats mean you need to decide the threshold probability at which the category flips from one to the other. By default, it is set to $p = 0.5$. However, in reality it should be settled based on the analysis purpose.

This section you will learn how to:

- Define the logistic regression equation and key terms such as log-odds and logit
- Perform Logistic Regression in R and interpret the results.
- Make predictions on new test data and evaluate the model accuracy

You will need to know what is logistic regression.

Here is the formula - Log[p/(1-p)] = b0 + b1 * x + b2 * X2 … + bn * Xn

Probability can be calculated from the odds as p = Odds/(1 + Odds)

```r
library(tidyverse)
library(caret)
theme_set(theme_bw())
```

Preparing the data

Logistic Regression works for continuous and / or categorical predictor variables.

Performing the following steps might improve the accuracy of your model - remove potential outliers

- make sure that the predictor variables are normally distributed. if not, you use log, root, box-cox transformation.
- Remove highly correlated predictors to minimize over fitting. The presence of highly correlated predictors might lead to an unstable model solution.

We will be using PimaIndiansDiabetes2 from mlbench package.

Let get into the code. (dont for get to install the packages if you havent)

```r
set.seed(3213)
data("PimaIndiansDiabetes2", package = "mlbench")
PimaIndiansDiabetes2 <- na.omit(PimaIndiansDiabetes2)
# inspect the data
sample_n(PimaIndiansDiabetes2, 3)
```

```
##     pregnant glucose pressure triceps insulin mass pedigree age diabetes
## 225        1     100       66      15      56 23.6    0.666  26      neg
## 390        3     100       68      23      81 31.6    0.949  28      neg
## 764       10     101       76      48     180 32.9    0.171  63      neg
```

```r
# split the data
training.samples.in <- PimaIndiansDiabetes2$diabetes %>%
  createDataPartition(p = 0.8, list = F)
train.data.in <- PimaIndiansDiabetes2[training.samples.in, ]
test.data.in <- PimaIndiansDiabetes2[-training.samples.in, ]
```

Computing Logistic Regression glm() -> generalized linear model, can be used to compute logistic regression. you will need to specify the option family = binomial, which tells R that we want to fit logistic regression.

```r
# fit the model
model.in <- glm(diabetes ~., data = train.data.in, family = binomial)
# summarize the model
summary(model)
```

```
## Data:    X dimension: 407 13
##  Y dimension: 407 1
## Fit method: oscorespls
## Number of components considered: 10
## TRAINING: % variance explained
##            1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## X           46.91    57.21    64.47    69.89    75.76    79.59    83.17
## .outcome    49.21    70.10    71.68    72.72    73.13    73.28    73.38
##           8 comps  9 comps  10 comps
## X           85.82    90.83     92.65
## .outcome    73.45    73.46     73.46
```

```
# make predictions
probabilities <- model.in %>% predict(test.data.in, type = "response")
pred.in <- ifelse(probabilities > 0.5, "pos", "neg")
# model accuracy
mean(pred.in == test.data$diabetes)
```

```
## [1] NaN
```

Simple Logistic Regression

when you see simple, most likely it refers to one predictor variable.

The following R code builds a model to predict the probability of being diabetes positive based on the plasma glucose concentration.

```
model.slr <- glm(diabetes ~ glucose, data = train.data.in, family = binomial)
summary(model.slr)$coef
```

```
##                Estimate  Std. Error   z value     Pr(>|z|)
## (Intercept) -6.06547775 0.699285600 -8.673820 4.17859e-18
## glucose      0.04258861 0.005335583  7.981998 1.43983e-15
```

Both beta0 and beta1 are at significance level.

Now the logistic equation can be written as p = exp(-6.32 + 0.043 * glucose) / [1 + exp(-6.32 + 0.043 * glucose)]

With this formula, each new glucose plasma concentration value, you can predict the probability of the individuals in being diabetes positive.

We can use predict() and use type = “response” to directly obtain the probabilities

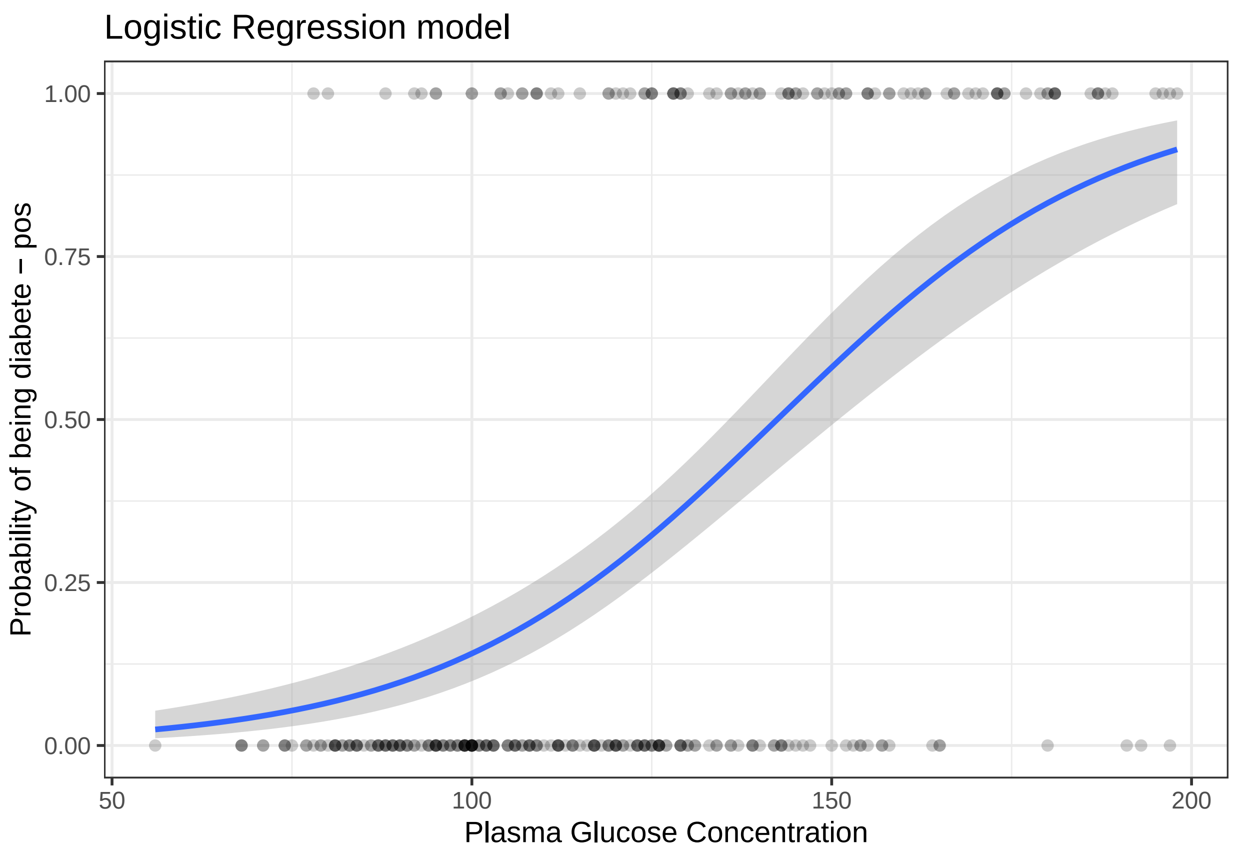
```
newdata.in <- data.frame(glucose = c(20,180))
prob.in <- model.slr %>% predict(newdata.in, type = "response")
pred.class <- ifelse(probabilities > 0.5, "pos", "neg")
pred.class
```

```
##    17    32    33    44    54    64    86    96    99   100   129   143   154
## "neg" "pos" "neg" "pos" "pos" "neg" "neg" "pos" "neg" "neg" "neg" "neg" "pos"
##   159   172   182   245   260   266   278   290   293   294   296   317   321
## "neg" "neg" "neg" "neg" "pos" "neg" "neg" "neg" "pos" "neg" "pos" "neg" "neg"
##   329   360   361   374   375   380   397   403   410   426   429   442   443
## "neg" "pos" "pos" "neg" "neg" "neg" "neg" "neg" "pos" "pos" "neg" "neg" "neg"
##   470   479   486   487   488   500   528   539   540   541   552   554   555
## "pos" "neg" "neg" "neg" "pos" "pos" "neg" "neg" "pos" "neg" "neg" "neg" "neg"
##   569   573   589   595   600   607   610   613   634   638   639   641   645
## "pos" "neg" "pos" "neg" "neg" "pos" "neg" "pos" "neg" "neg" "neg" "neg" "neg"
##   648   655   671   674   696   701   708   716   727   739   746   752   764
## "pos" "neg" "pos" "pos" "neg" "neg" "neg" "pos" "neg" "neg" "neg" "neg" "neg"
```

We can plot the model

```
train.data.in %>%
  mutate(prob = ifelse(diabetes == "pos", 1, 0)) %>%
  ggplot(aes(glucose,prob)) +
  geom_point(alpha = 0.2) +
  geom_smooth(method = "glm", method.args = list(family = "binomial")) +
  labs(
    title = "Logistic Regression model",
    x = "Plasma Glucose Concentration",
    y = "Probability of being diabete - pos")
```

```
## `geom_smooth()` using formula 'y ~ x'
```



Multiple logistic regression

```
model.mlr <- glm(diabetes ~., data = train.data.in, family = binomial)
summary(model.mlr)$coef
```

```
##                  Estimate  Std. Error    z value     Pr(>|z|)
## (Intercept) -10.089618441 1.388240569 -7.2679179 3.650706e-13
## pregnant      0.077794364 0.061220899  1.2707158 2.038298e-01
## glucose       0.041184804 0.006673737  6.1711763 6.778380e-10
## pressure     -0.007240783 0.013297101 -0.5445384 5.860710e-01
## triceps       0.004093379 0.019665341  0.2081519 8.351103e-01
## insulin      -0.001537553 0.001458665 -1.0540820 2.918454e-01
## mass          0.076636938 0.030830689  2.4857355 1.292840e-02
## pedigree      1.349689676 0.482622420  2.7965748 5.164746e-03
## age           0.040761698 0.020543442  1.9841708 4.723680e-02
```

```r
summary(model.mlr)
```

```
##
## Call:
## glm(formula = diabetes ~ ., family = binomial, data = train.data.in)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.8567  -0.6368  -0.3698   0.6538   2.5606
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -10.089618   1.388241  -7.268 3.65e-13 ***
## pregnant      0.077794   0.061221   1.271  0.20383
## glucose       0.041185   0.006674   6.171 6.78e-10 ***
## pressure     -0.007241   0.013297  -0.545  0.58607
## triceps       0.004093   0.019665   0.208  0.83511
## insulin      -0.001538   0.001459  -1.054  0.29185
## mass          0.076637   0.030831   2.486  0.01293 *
## pedigree      1.349690   0.482622   2.797  0.00516 **
## age           0.040762   0.020543   1.984  0.04724 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 398.80  on 313  degrees of freedom
## Residual deviance: 270.99  on 305  degrees of freedom
## AIC: 288.99
##
## Number of Fisher Scoring iterations: 5
```

From the table above, all those betas are at their significance levels, are they?

remember we use 0.05. so for those has less than 02, they are actually not significance. why? is 0.3 smaller than 0.05? no so 0.3 is not signifigant at all… Dont mess up this part.

Estimate is the coefficient for that specific beta.

Std.Error is the standard error of the coefficient estimates. This is also representing the accuracy of the coefficients. We want small numbers from this col.

z value is the z statistic, which is the coefficient estimate (col 2) divided by the standard error of the estimate (col3)

Pr(>|z|) is the p value corresponding to the z statistic. We want small value from this.

So let clean up the formula

```r
model.mlr2 <- glm(diabetes ~ pregnant + glucose + mass,
                  data = train.data.in,
                  family = binomial)
summary(model.mlr2)
```

```
##
## Call:
## glm(formula = diabetes ~ pregnant + glucose + mass, family = binomial,
##     data = train.data.in)
```

```
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.2088  -0.6653  -0.3985   0.6924   2.3954
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -8.553573   1.058420  -8.081 6.40e-16 ***
## pregnant     0.148458   0.043908   3.381 0.000722 ***
## glucose      0.038563   0.005364   7.189 6.51e-13 ***
## mass         0.073867   0.022322   3.309 0.000936 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 398.80  on 313  degrees of freedom
## Residual deviance: 286.16  on 310  degrees of freedom
## AIC: 294.16
##
## Number of Fisher Scoring iterations: 5
```

Interpretation

Coding wise, everything above is the most important thing you will need to know. Also, there is another concept.

for example, in our data, we know the the coefficient for glucose is 0.036900. It means if we have 1 unit of glucose glucose -> e^0.036900 =

```
exp(0.036900)
```

```
## [1] 1.037589
```

So it will increase 1.037589. so if we have large coefficient for beta, it is very bad. just like pregnant.

```
exp(0.176226)
```

```
## [1] 1.192708
```

We will make predictions using the test data in order to evaluate the performance of our logistic regression model.

```
prob.test <- model.mlr2 %>% predict(test.data.in, type = "response")
head(prob.test)
```

```
##         17         32         33         44         54         64
## 0.34977533 0.57906696 0.05301269 0.93879576 0.87107401 0.28028253
```

let convert those number to either pos or neg

```
pred.class.test <- ifelse(probabilities > 0.5, "pos", "neg")
head(pred.class.test)
```

```
##    17    32    33    44    54    64
## "neg" "pos" "neg" "pos" "pos" "neg"
```

Assessing model accuracy

```
mean(pred.class.test == test.data.in$diabetes)
```

```
## [1] 0.7435897
```

From here, we can see that we have 25.64103% error rate. So our model is alright, not too bad.

In this section, we have talked about how logistic regression works. We have gone thru how to make predictions and assess the model accuracy. Logistic model output is very easy to interpret compared to other classification methods. The reason why it is easy to interpret compared to other classification methods. Also, because of it's simplicity, we will less likely to have over fitting problem than flexible methods, like decision trees.

A lot of concepts from linear regression will remain it's ground for logistic regression modeling. Look at what we did towards the end of the section. We still perform some diagnostics to make sure that the assumptions made by the model are met for our data. In addition, we need to see how good the model is in predicting the outcome for the new test data set. We also have gone thru the classification accuracy, but not that other important performance metric exist.

If we have a lot of parameters, we might want to use stepwise regression, lasso regression techniques to see which parameters are significant. We can also add interaction terms in the model or include the spline terms. We can also fit generalized additive models when linearity of the predictor cannot be assumed. this can be done using mgcv package.

```
library("mgcv")
# fit the model
gam.model <- gam(diabetes ~ s(glucose) + mass + pregnant, data = train.data.in, family = "binomial")
# summarize model
summary(gam.model)
```

```
##
## Family: binomial
## Link function: logit
##
## Formula:
## diabetes ~ s(glucose) + mass + pregnant
##
## Parametric coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -3.87420    0.80288  -4.825  1.4e-06 ***
## mass         0.07387    0.02232   3.309 0.000936 ***
## pregnant     0.14846    0.04391   3.381 0.000722 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##            edf Ref.df Chi.sq  p-value
## s(glucose)   1      1  51.69 6.52e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.327   Deviance explained = 28.2%
## UBRE = -0.063181  Scale est. = 1         n = 314
```

```
prob.gam <- gam.model %>% predict(test.data.in, type = "response")
pred.gam <- ifelse(probabilities > 0.5, "pos", "neg")
# model accuracy
mean(pred.gam == test.data.in$diabetes)
```

```
## [1] 0.7435897
```

As we know, Logistic regression is limited to only two class classification. If we want more outcome, we might need multi-nomial logistic regression for multi-class classification problem.

Sub-section 2 - Stepwise Logistic Regression Essentials in R

In this section, we will be talking about stepwise logistic regression that consists of automatically selecting a reduced number of predictor variables for building the best performing logistic regression model.

load up and prepare the data

```r
library(tidyverse)
library(caret)
data("PimaIndiansDiabetes2", package = "mlbench")
PimaIndiansDiabetes2 <- na.omit(PimaIndiansDiabetes2)
set.seed(123)
training.samples <- PimaIndiansDiabetes2$diabetes %>%
  createDataPartition(p = 0.8, list = FALSE)
train.data  <- PimaIndiansDiabetes2[training.samples, ]
test.data <- PimaIndiansDiabetes2[-training.samples, ]
```

Compute the stepwise logistic regression

The stepwise logistic regression can be easily computed using the R function stepAIC() available in the MASS package. It performs model selection by AIC. It has an option called direction, which can have the following values: “both”, “forward”, “backward”

Full model

```r
full.model <- glm(diabetes ~., data = train.data, family = binomial)
coef(full.model)
```

```
##   (Intercept)      pregnant       glucose      pressure       triceps 
## -1.053400e+01  1.005031e-01  3.709621e-02 -3.875933e-04  1.417771e-02 
##       insulin          mass      pedigree           age 
##  5.939876e-04  7.997447e-02  1.329149e+00  2.718224e-02
```

Perform Stepwise variable selection

```r
step.model <- full.model %>% stepAIC(trace = FALSE)
coef(step.model)
```

```
##  (Intercept)     pregnant      glucose         mass     pedigree          age 
## -10.77109289   0.09903127   0.03813593   0.09512159   1.36220306   0.02956352
```

Compare the full and the stepwise model

```r
probabilities <- full.model %>% predict(test.data, type = "response")
predicted.classes <- ifelse(probabilities > 0.5, "pos", "neg")
observed.classes <- test.data$diabetes
mean(predicted.classes == observed.classes)
```

```
## [1] 0.7564103
```

```r
probabilities <- predict(step.model,test.data, type = "response")
predicted.classes <- ifelse(probabilities > 0.5, "pos", "neg")
oberseved.classes <- test.data$diabetes
mean(predicted.classes == observed.classes)
```

```
## [1] 0.7820513
```

As you can see, we have a better model after applying the stepwise function and selection the most impactful parameters to the model.

Just to remember, the more simpler model the better it is.

Sub-section 3 - Penalized Logistic Regression Essentials in R: Ridge, Lasso and Elastic Net

Penalized logistic regression imposes a penalty to the logistic model for having to many variables (parameters). This will reduce / lower the coefficients those parameters who have less contribution to the model towards to zero. This entire process is also called regularization.

Most common penalized regression include:

Ridge regression: Will set less meaningful parameters close to zero

Lasso regression: will set less meaningful parameters to exactly zero

Elastic net regression: combination of ridge and lasso regression.

In this section, we will penalized logistic regression, by lasso regression for automatically selecting the optimal and most contributing parameters to the model.

Let get in to the code

```r
library(tidyverse)
library(caret)
library(glmnet)
data("PimaIndiansDiabetes2", package = "mlbench")
PimaIndiansDiabetes2 <- na.omit(PimaIndiansDiabetes2)
set.seed(1523)
training.samples <- PimaIndiansDiabetes2$diabetes %>%
  createDataPartition(p = 0.8, list = FALSE)
train.data  <- PimaIndiansDiabetes2[training.samples, ]
test.data <- PimaIndiansDiabetes2[-training.samples, ]
# Dumy code categorical predictor variables
x <- model.matrix(diabetes~., train.data)[,-1]
# Convert the outcome (class) to a numerical variable
y <- ifelse(train.data$diabetes == "pos", 1, 0)
```

Lasso

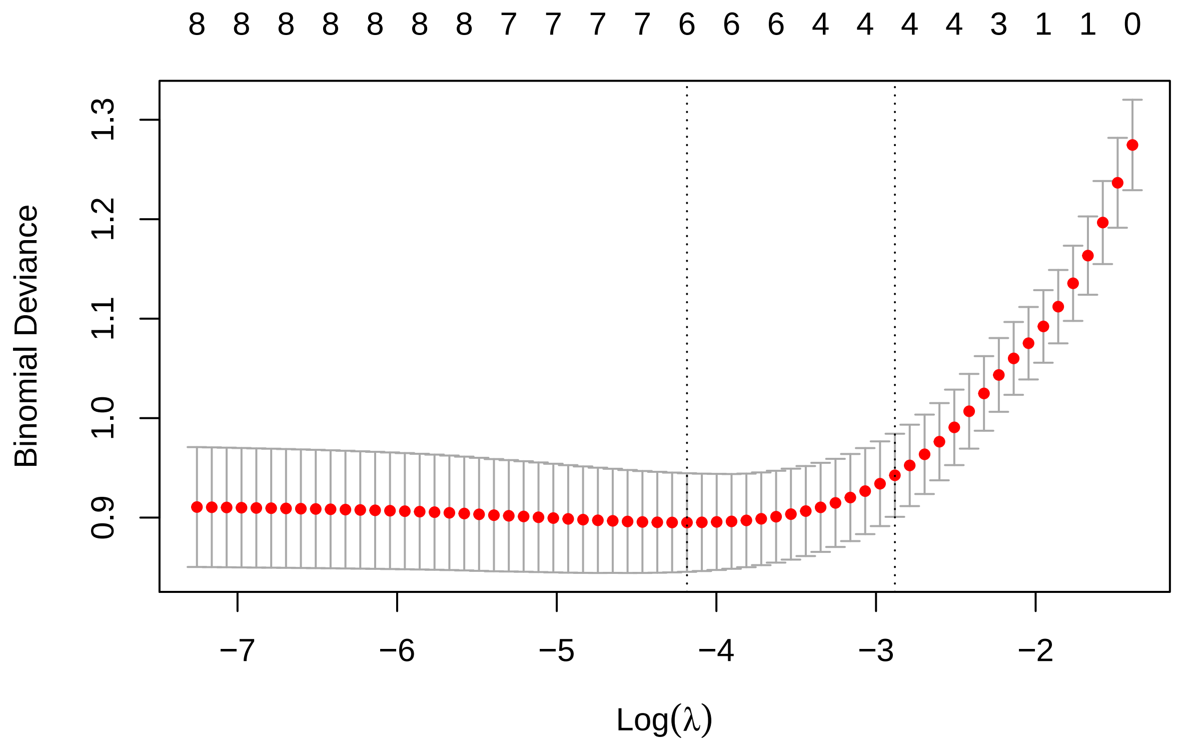
```r
cv.lasso <- cv.glmnet(x, y, alpha = 1, family = "binomial")
model <-glmnet(x,y, alpha = 1, family = "binomial",
               lambda = cv.lasso$lambda.min)
coef(model)
```

```
## 9 x 1 sparse Matrix of class "dgCMatrix"
##                       s0
## (Intercept) -8.881100796
## pregnant     0.024907745
## glucose      0.035390024
## pressure     .
## triceps      0.003541696
## insulin      .
## mass         0.049682735
## pedigree     1.392519720
## age          0.033904233
```

```
x.test <- model.matrix(diabetes ~., test.data)[,-1]
prob <- model %>% predict(newx = x.test)
pred <- ifelse(probabilities > 0.5, "pos", "neg")
ob.classes <- test.data$diabetes
mean(pred == ob.classes)
```

```
## [1] 0.6538462
```

```
plot(cv.lasso)
```



The plot displays the cross-validation error according to the log of lambda. the left dashed vertical line indicates that the log of the optimal value of lambda is approximately -4 which is the one that minimizes the prediction error, where will give you the most accurate model.

```
cv.lasso$lambda.min
```

```
## [1] 0.01523502
```

Generally speaking, the purpose of regularization is to balance accuracy and simplicity, which implies that the smallest number of predictors that also gives the highest accuracy model. the function cv.glment() finds also the value of lambda that gives the simplest model, but also lies within one standard error of the optimal value of lambda. This value is called lambda.1se

```
cv.lasso$lambda.1se
```

```
## [1] 0.05604018
```

use lambda.min as the best lambda

```
coef(cv.lasso, cv.lasso$lambda.min)
```

```
## 9 x 1 sparse Matrix of class "dgCMatrix"
##                        1
## (Intercept) -8.880824643
## pregnant     0.024915092
## glucose      0.035389385
## pressure     .
## triceps      0.003552386
## insulin      .
## mass         0.049670689
## pedigree     1.392494214
## age          0.033900357
```

using lambda.1se as the best lambda

```r
coef(cv.lasso, cv.lasso$lambda.1se)
```

```
## 9 x 1 sparse Matrix of class "dgCMatrix"
##                       1
## (Intercept) -5.77203688
## pregnant     .
## glucose      0.02742876
## pressure     .
## triceps      .
## insulin      .
## mass         0.01766794
## pedigree     0.66392511
## age          0.02109841
```

The next code, we will compute the final model using lambda.min and then access the model accuracy against the test data.

```r
# Final model with lambda.min
lasso.model <- glmnet(x, y, alpha = 1, family = "binomial",
                      lambda = cv.lasso$lambda.min)
# Make prediction on test data
x.test <- model.matrix(diabetes ~., test.data)[,-1]
probabilities <- lasso.model %>% predict(newx = x.test)
predicted.classes <- ifelse(probabilities > 0.5, "pos", "neg")
# Model accuracy
observed.classes <- test.data$diabetes
mean(predicted.classes == observed.classes)
```

```
## [1] 0.7692308
```

lambda.1se

```r
# Final model with lambda.1se
lasso.model <- glmnet(x, y, alpha = 1, family = "binomial",
                      lambda = cv.lasso$lambda.1se)
# Make prediction on test data
x.test <- model.matrix(diabetes ~., test.data)[,-1]
probabilities <- lasso.model %>% predict(newx = x.test)
predicted.classes <- ifelse(probabilities > 0.5, "pos", "neg")
# Model accuracy rate
observed.classes <- test.data$diabetes
mean(predicted.classes == observed.classes)
```

```
## [1] 0.7179487
```

Compute the full logistic model

```r
# Fit the model
full.model <- glm(diabetes ~., data = train.data, family = binomial)
# Make predictions
probabilities <- full.model %>% predict(test.data, type = "response")
predicted.classes <- ifelse(probabilities > 0.5, "pos", "neg")
# Model accuracy
observed.classes <- test.data$diabetes
mean(predicted.classes == observed.classes)
```

```
## [1] 0.7948718
```

According to the author, even tho we have a 0.7948718 accuracy for the full logistic model, we would pick lasso.min as our lambda. Because the model is far less complicated. According to the bias variance trade off, all things equal, simpler model should be always preferred because it is less likely to overfit the training data.

For variable selection, an alternative to the penalized logistic regression is the stepwise logistic regression.

Sub-section 4 - Logistic Regression Assumptions and Diagnostics in R

This section will go thru the major assumptions and provides practical guide to check whether these assumptions hold true for your data, which means it this section will tell us if we are building a good model.

Key concept:

Outcome for Logistic regression is either yes or no 1 or 0 or positive or negative

There is a linear relationship between the logit of the outcome and each predictor variables. logit(p) = log(p/(1-p)), where p is the probabilites of the outcome.

There is no influential values (extremevalues or outliers)

there is no high intercorrelations (multi collinearity) among the predictors

To have a better accuracy of your model, we need to make sure those assumption = T

```r
library(tidyverse)
library(broom)
theme_set(theme_classic())
# Load the data
data("PimaIndiansDiabetes2", package = "mlbench")
PimaIndiansDiabetes2 <- na.omit(PimaIndiansDiabetes2)
# Fit the logistic regression model
model <- glm(diabetes ~., data = PimaIndiansDiabetes2,
               family = binomial)
# Predict the probability (p) of diabete positivity
probabilities <- predict(model, type = "response")
predicted.classes <- ifelse(probabilities > 0.5, "pos", "neg")
head(predicted.classes)
```

```
##     4     5     7     9    14    15
## "neg" "pos" "neg" "pos" "pos" "pos"
```

Logistic regression diagnostics

Linearity assumption

Simply means we plot the data and check if they are linear or not.

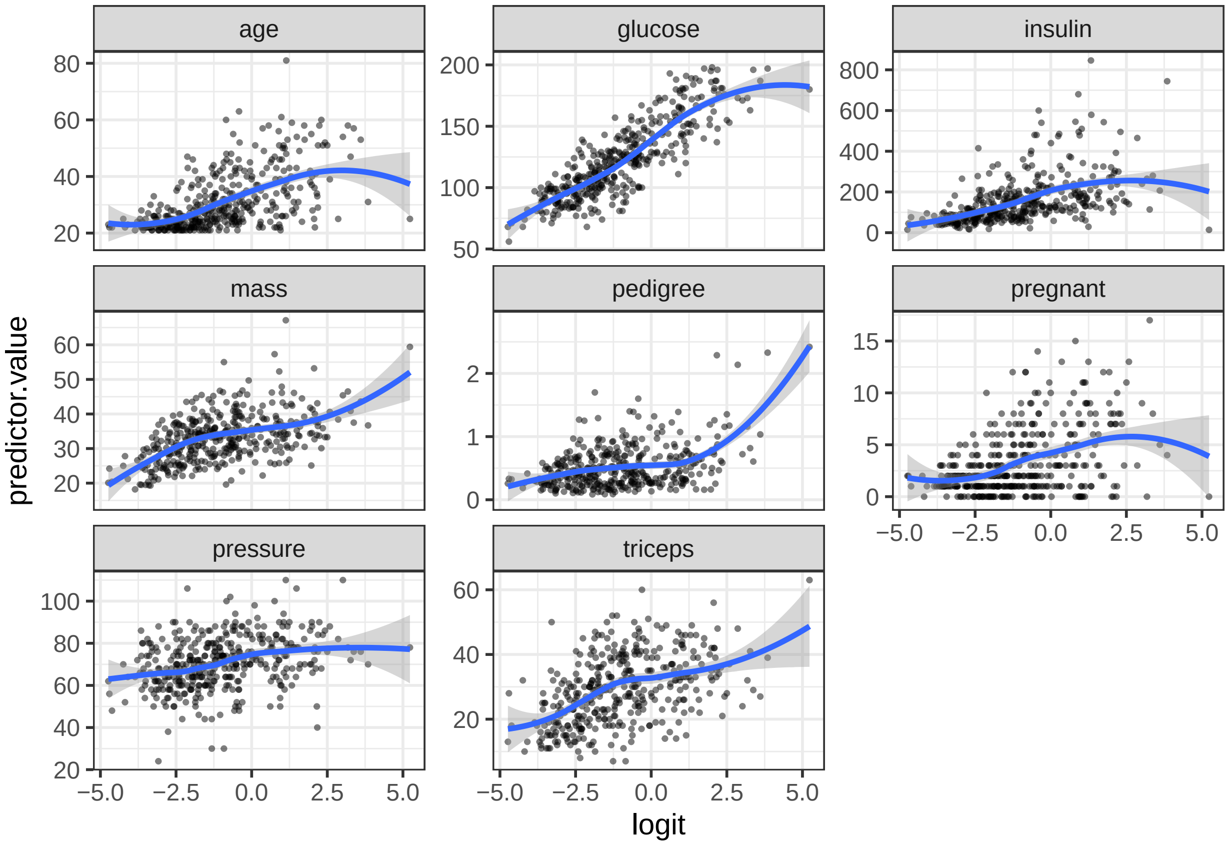
We need to remove qualitative variables from the original data frame and bind the logit values to the data

```
# select only numeric predictors
mydata <- PimaIndiansDiabetes2 %>% dplyr::select_if(is.numeric)
predictors <- colnames(mydata)
# bind the logit and tiding the data for plot
mydata <- mydata %>% mutate(logit = log(probabilities/(1-probabilities))) %>%
  gather(key = "predictors", value = "predictor.value", -logit)
```

Create a scatter plots

```
ggplot(mydata, aes(logit, predictor.value)) +
  geom_point(size = 0.5, alpha =0.5) +
  geom_smooth(method = "loess") +
  facet_wrap(~predictors, scales = "free_y") +
  theme_bw()
```

```
## `geom_smooth()` using formula 'y ~ x'
```

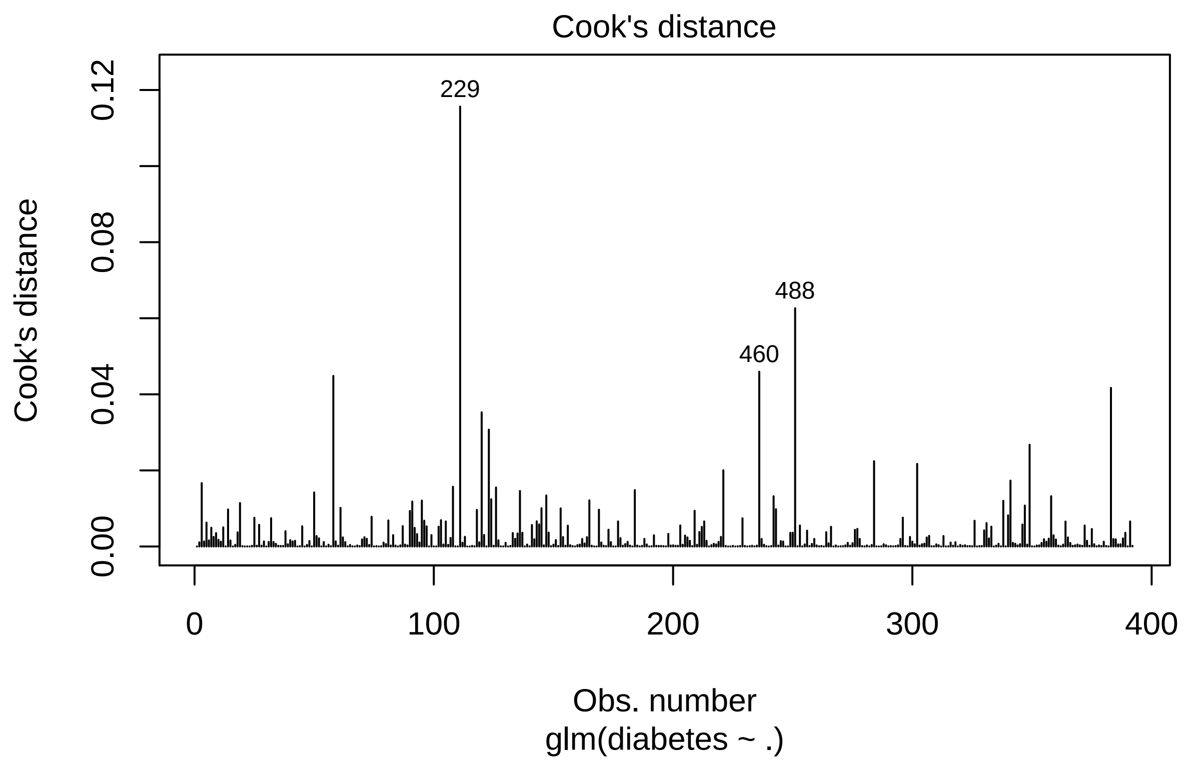


As we can see, glucose, mass, pregnant, pressure and triceps are all quite linearly associated with the diabetes outcome in logit scale. Which means age and pedigree is not linear so that they might need transformations. What should we do? poly, fractional poly, spline would work

Influential values

We can view those extreme values in the data by visualizing the Cook's distance values with n = 3

```r
plot(model, which = 4, id.n=3)
```

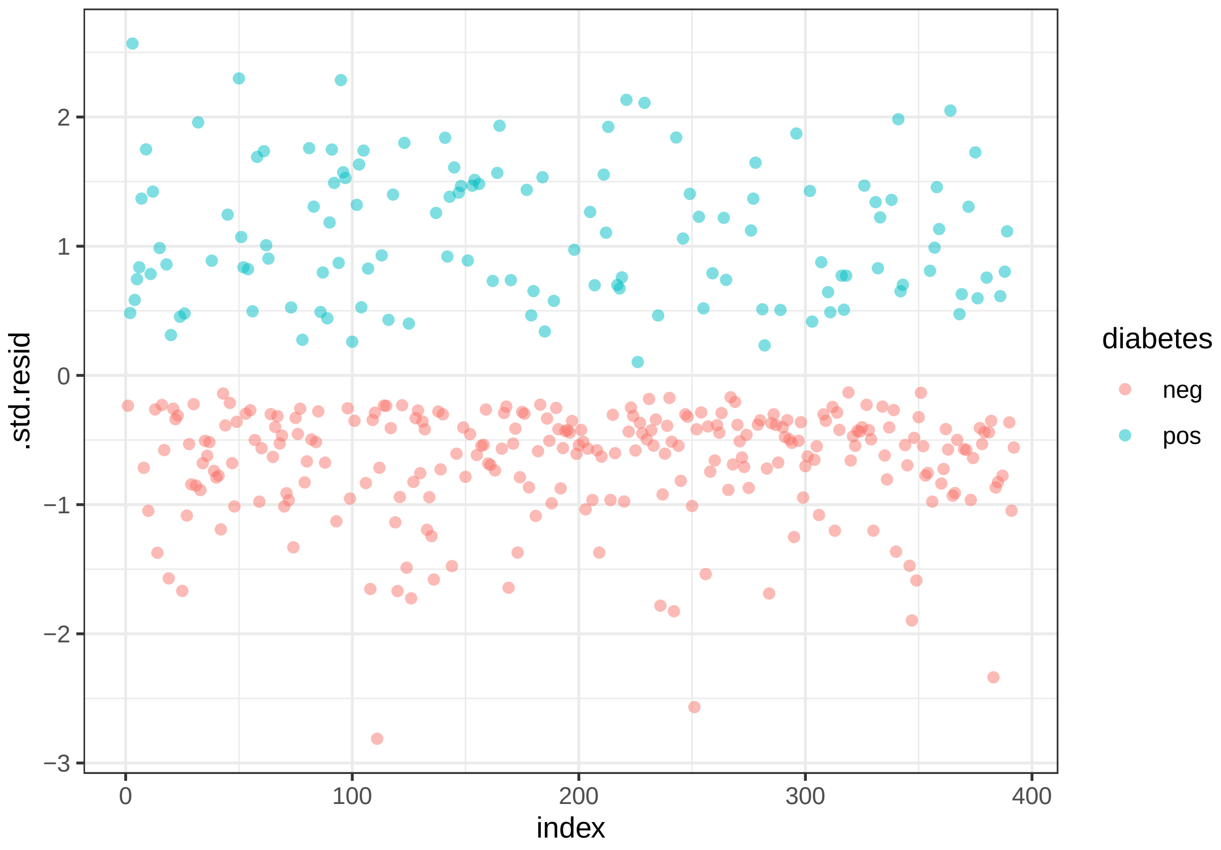


Not all outliers are influential observations. We can check standardized residual error for potential influential observations.

```r
# extract model results
model.data <- augment(model) %>% mutate(index = 1:n())
model.data %>% top_n(3, .cooksd)
```

```
## # A tibble: 3 x 17
##   .rownames diabetes pregnant glucose pressure triceps insulin  mass pedigree
##   <chr>     <fct>       <dbl>   <dbl>    <dbl>   <dbl>   <dbl> <dbl>    <dbl>
## 1 229       neg             4     197       70      39     744  36.7     2.33
## 2 460       neg             9     134       74      33      60  25.9     0.46
## 3 488       neg             0     173       78      32     265  46.5     1.16
## # ... with 8 more variables: age <dbl>, .fitted <dbl>, .resid <dbl>,
## #   .std.resid <dbl>, .hat <dbl>, .sigma <dbl>, .cooksd <dbl>, index <int>
```

```r
ggplot(model.data, aes(index, .std.resid)) +
  geom_point(aes(color = diabetes), alpha = .5) +
  theme_bw()
```



Filter potential influential data points with abs(.std.res) > 3

```
model.data %>% filter(abs(.std.resid) >3)
```

```
## # A tibble: 0 x 17
## # ... with 17 variables: .rownames <chr>, diabetes <fct>, pregnant <dbl>,
## #   glucose <dbl>, pressure <dbl>, triceps <dbl>, insulin <dbl>, mass <dbl>,
## #   pedigree <dbl>, age <dbl>, .fitted <dbl>, .resid <dbl>, .std.resid <dbl>,
## #   .hat <dbl>, .sigma <dbl>, .cooksd <dbl>, index <int>
```

empty which means we do not have any influential observations in our data.

If i have outliers in a continuous predictor, potential solutions include:

1) Removing the concerned records
2) Transform the data into log scale
3) use non parametric methods

Multi-collinearity

Multi-collinearity means one or more parameters are super highly correlated, which is an important task in regression analysis and should be fixed by removing those specific variables. we can use vif() to cehck

we dont want it to exceeds 5 or 10. Otherwise those are the problem.

```
car::vif(model)
```

```
## pregnant  glucose pressure  triceps  insulin     mass pedigree      age
## 1.892387 1.378937 1.191287 1.638865 1.384038 1.832416 1.031715 1.974053
```

Discussion

In this section, we have describes the main assumptions of logistic regression model, and provides examples of R code to diagnose potential problems in the data including non linearity between the predictor variables and the logit of the outcome. The present of influential observations in the data and multicollinearity among predictors.Fixing those potential problems might improve considerably the goodness of the model.

Sub-section 5 - Multi nomial Logistic Regression Essentials in R

Multinomial logistic regression is an extension of the logistic regression for multi class classification tasks. It is used when the outcome involves more than two classes.

nnet -> computing multinomial logistic regression

```r
library(tidyverse)
library(caret)
library(nnet)
```

```
##
## Attaching package: 'nnet'

## The following object is masked from 'package:mgcv':
##
##     multinom
```

```r
# load up the data
data("iris")
# split the data
set.seed(3333)
training.samples <- iris$Species %>% createDataPartition(p = 0.8, list = F)
train.data <- iris[training.samples, ]
test.data <- iris[-training.samples, ]
```

Computing multinomial logistic regression

```r
# feed the model
model <- nnet::multinom(Species ~., data = train.data)
```

```
## # weights:  18 (10 variable)
## initial  value 131.833475
## iter  10 value 14.477999
## iter  20 value 3.410071
## iter  30 value 2.671949
## iter  40 value 2.257685
## iter  50 value 2.154485
## iter  60 value 2.107296
## iter  70 value 2.059608
## iter  80 value 1.996874
## iter  90 value 1.739233
## iter 100 value 1.718397
## final  value 1.718397
## stopped after 100 iterations
```

```r
summary(model)
```

```
## Call:
## nnet::multinom(formula = Species ~ ., data = train.data)
```

```
##
## Coefficients:
##            (Intercept) Sepal.Length Sepal.Width Petal.Length Petal.Width
## versicolor    53.12350    -8.517859   -19.68636     25.73491   -9.242463
## virginica    -61.61498   -14.472089   -40.66622     47.86102   50.014648
##
## Std. Errors:
##            (Intercept) Sepal.Length Sepal.Width Petal.Length Petal.Width
## versicolor    202.7656     165.8230    257.4329     204.3598    58.89326
## virginica     227.0715     166.3074    259.7037     209.2708    50.43342
##
## Residual Deviance: 3.436793
## AIC: 23.43679
```

```
pred.classes <- model %>% predict(test.data)
head(pred.classes)
```

```
## [1] setosa setosa setosa setosa setosa setosa
## Levels: setosa versicolor virginica
```

```
# model accuracy
mean(pred.classes == test.data$Species)
```

```
## [1] 0.9
```

Thats a pretty high accuracy.

Discussion

So we have gone thru multinomial logistic regression in R. This method is used for multi classes problems. However, we dont use it that often. However, Discriminant analysis is more popular for multiple class classification.

Sub-section 6 - Discriminant Analysis Essentials in R

Discriminant analysis is used to predict the probability of belonging to a given class based on one or multiple predictor variables. It works with continuous / categorical predictor variables.

In discriminant analysis is more suitable for predicting the category of an observation in the situation where the outcome variable contains more than two classes. in addition, it's more stable than the logistic regression for multi-class classification problems.

Remember, both logistic and discriminant analysis can be used for binary classification.

Here are the discriminant analysis methods will be discussed

- Linear discriminant analysis (LDA) it uses linear combinations of predictors to predict the class of a given observation. Assumes that the predictor variables (p) are normally distributed and the classes have identical variances (for univariate analysis, $p = 1$) or identical covariance matrices (for multivariate analysis, $p > 1$)
- Quadratic discriminant analysis (QDA) more flexible than LDA. There is no assumption that the covariance matrix of classes is the same
- Mixture discriminant analysis (MDA) each class is assumed to be a Gaussian mixture of sub classes
- Flexible Discriminant Analysis (FDA) non linear combinations of predictors are used such as splines
- Regularized Discriminant Analysis (RDA) Regularization (shrinkage) improves the estimate of the covariance matrices in situations where the number of predictors are larger than the number of samples

in the training data. This leads to an improvement of the discriminant analysis.

```r
library(tidyverse)
library(caret)
theme_set(theme_classic())
data("iris")
set.seed(321312)
training.samples <- iris$Species %>% createDataPartition(p =0.8, list =F)
train.data <- iris[training.samples, ]
test.data <- iris[-training.samples, ]
# Normalize the data. Categorical variables are automatically ignored.
# Estimate pre processing parameters
preproc.param <- train.data %>% preProcess(method = c("center", "scale"))
# transform the data using the estimated parameters
train.transf <- preproc.param %>% predict(train.data)
test.transf <- preproc.param %>% predict(test.data)
```

Linear Discriminant Analysis LDA

LDA starts by finding directions that maximize the separation between classes. Then use those directions to predict the class of individuals. These directions, called linear discriminant, are a linear combinations of predictor variables.

LDA assumes that predictors are normally distributed and that the different classes have class- specific means and equal variance / covariance.

before performing LDA, consider:

- Inspecting the univariate distributions of each variable and make sure that they are normally distributed. If not, you can transform them using log and root for exponential distributions and box-cox for skewed distributions.
- Removing outliers from your data and standardize the variables to make their scale comparable.

The linear discriminant analysis can be easily computed using the function lda() from the mass package

```r
library(MASS)
# feed the model
model <- lda(Species ~., data = train.transf)
pred <- model %>% predict(test.transf)
# model accuracy
mean(pred$class ==test.transf$Species)
```

```
## [1] 0.9666667
```

```r
model
```

```
## Call:
## lda(Species ~ ., data = train.transf)
##
## Prior probabilities of groups:
##     setosa versicolor  virginica
##  0.3333333  0.3333333  0.3333333
##
## Group means:
##            Sepal.Length Sepal.Width Petal.Length Petal.Width
## setosa      -0.97995883   0.8949649   -1.2890497  -1.2485759
## versicolor   0.07021834  -0.6925766    0.2643596   0.1556638
## virginica    0.90974049  -0.2023883    1.0246901   1.0929121
```

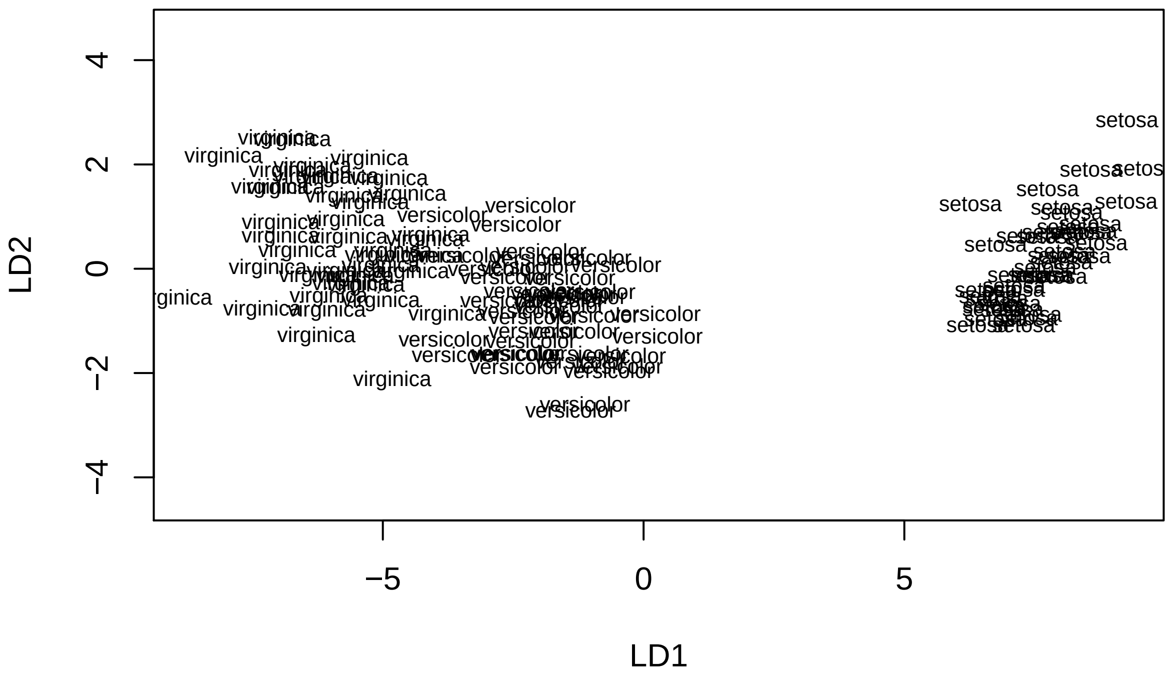
```
##
## Coefficients of linear discriminants:
##                     LD1        LD2
## Sepal.Length  0.8055912  0.1269374
## Sepal.Width   0.7225173  0.9781999
## Petal.Length -3.6651301 -1.6646138
## Petal.Width  -2.4644317  2.1330713
##
## Proportion of trace:
##    LD1    LD2
## 0.9902 0.0098
```

LDA determines group means and computes for each individual, the probability of belonging to the different groups. The individual is then affected to the group with the highest probability score.

lda() outputs contain the following elements:

- Prior probabilities of groups: the proportion of training observations in each group. for example, there are 31% of the training observations in the samosa group
- Group means: group center of gravity. Shows the mean of each variable in each group
- coefficients of linear discriminants: shows the linear combination of predictor variables that are used to form the LDA decision rule. For example: LD1 = 0.8055912 * Sepal.Length + (Sepal.Width * 0.7225173) - (Petal.Length * 3.6651301) - (Petal.Width * 2.4644317) LD2 = (Sepal.Length * 0.1269374) + (Sepal.Width * 0.9781999) - (Petal.Length * 1.6646138) + (Sepal.Width * 2.1330713)

```r
model <- lda(Species~., data = train.transf)
plot(model)
```



```r
pred <- model %>% predict(test.transf)
names(pred)
```

```
## [1] "class"     "posterior" "x"
```

The predict() function returns the following elements.

Class: predicted classes of observations Posterior: is a matrix whose columns are the groups, rows are the individuals and values are the posterior probability that the corresponding observation belongs to the groups x: contains the linear discriminant, described above

```r
# predicted classes
head(pred$class,6)
```

```
## [1] setosa setosa setosa setosa setosa setosa
## Levels: setosa versicolor virginica
```

```r
# predicted probabilities of class membership
head(pred$posterior, 6)
```

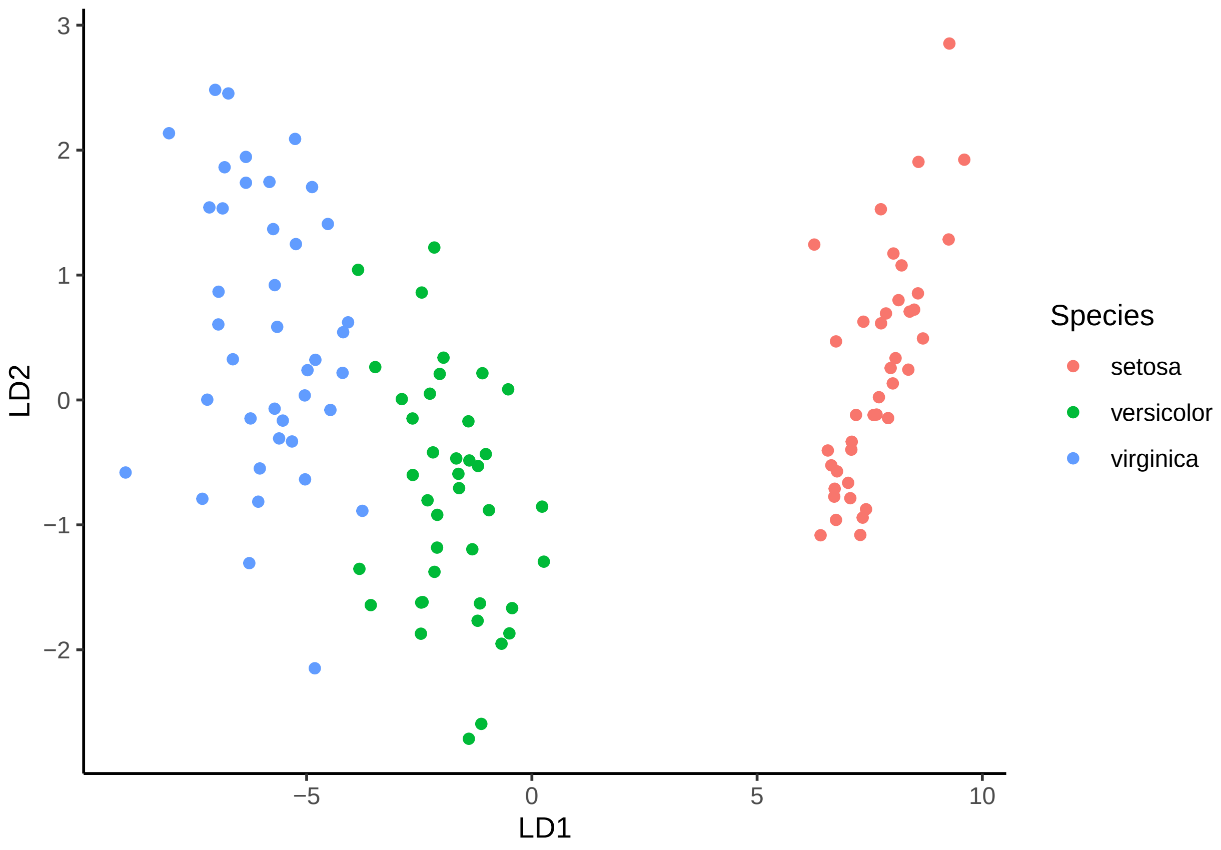
```
##    setosa   versicolor    virginica
## 3       1 4.491605e-19 5.291106e-39
## 5       1 1.879460e-22 3.312918e-43
## 7       1 4.511179e-18 4.134197e-37
## 8       1 4.854289e-20 3.675412e-40
## 15      1 1.304265e-30 5.720773e-54
## 22      1 2.504062e-20 1.226805e-39
```

```r
# linear discriminants
head(pred$x, 3)
```

```
##        LD1        LD2
## 3 7.401022 -0.2872264
## 5 8.135698  0.5369781
## 7 7.092506  0.3167861
```

create the LDA plot using ggplot2

```r
lda.data <- cbind(train.transf, predict(model)$x)
ggplot(lda.data, aes(LD1,LD2)) +
  geom_point(aes(color = Species))
```



Model accuracy

```
mean(pred$class == test.transf$Species)
```

```
## [1] 0.9666667
```

we have a super high model accuracy

Note that, by default, the probability cutoff used to decide group-membership is 0.5 (random guessing). For example the number of observations in the setosa group can be recalculated using:

```
sum(pred$posterior[,1] >= .5)
```

```
## [1] 10
```

In some situations, you might want to increase the precision of the model. In this case you can fine-tune the model by adjusting the posterior probability cutoff. For example, you can increase or lower the cutoff.

Variable selection: note that, if the predictor variables are standardized before computing LDA, the discriminator weights can be used as measures of variable importance for feature selection.

Quadratic Discriminant analysis - QDA

QDA is little bit more flexible than LDA, in the sense that it does not assumes the equality of variance / covariance. in other words, for QDA the co variance matrix can be different for each class.

LDA tends to be a better than QDA when you have a small training set.

QDA is recommended if the training set is very large, so that the variance of the classifier is not a major issue, or if the assumption of a common covariance matrix for the k classes is clearly untenable

QDA can be computed using the R function qda() from the mass package

```r
library(MASS)
# fit the model
model <- qda(Species ~., data = train.transf)
model
```

```
## Call:
## qda(Species ~ ., data = train.transf)
##
## Prior probabilities of groups:
##     setosa versicolor  virginica
##  0.3333333  0.3333333  0.3333333
##
## Group means:
##            Sepal.Length Sepal.Width Petal.Length Petal.Width
## setosa      -0.97995883   0.8949649   -1.2890497  -1.2485759
## versicolor   0.07021834  -0.6925766    0.2643596   0.1556638
## virginica    0.90974049  -0.2023883    1.0246901   1.0929121
```

```r
# make predictions
pred <- model %>% predict(test.transf)
# model acccracy
mean(pred$class == test.transf$Species)
```

```
## [1] 0.9666667
```

Mixture Discriminant analysis - MDA

The LDA classifier assume that each class comes from a single normal (or Gaussian) distribution. this is too restrictive. For MDA, there are classes, and each class is assumed to be a Gaussian mixture of sub classes, where each data point has a probability of belonging to each class. Equality of co variance matrix, among classes, is still assumed.

```r
library(mda)
```

```
## Loading required package: class
```

```
## Loaded mda 0.5-2
```

```r
model <- mda(Species ~., data= train.transf)
model
```

```
## Call:
## mda(formula = Species ~ ., data = train.transf)
##
## Dimension: 4
##
## Percent Between-Group Variance Explained:
##     v1     v2     v3     v4
##  95.13  99.00  99.77 100.00
##
## Degrees of Freedom (per dimension): 5
##
## Training Misclassification Error: 0.025 ( N = 120 )
##
## Deviance: 7.029
```

make predictions

```
pred.classes <- model %>% predict(test.transf)
# model accuracy
mean(pred.classes == test.transf$Species)
```

```
## [1] 0.9666667
```

Fliexible discriminant analysis - FDA

FDA is a more flexible extension of LDA that uses non linear combinations of predictors such as splines. FDA is useful to model muli variate non-normality or non linear relationships among variables within each group for allowing a more accurate classification.

```
library(mda)
model <- fda(Species ~., data = train.transf)
predict.classes <- model %>% predict(test.transf)
mean(predict.classes == test.transf$Species)
```

```
## [1] 0.9666667
```

Regularized discriminant analysis

RDA builds a classification rule by regularizing the group covariance matrices. Allowing a more robust model against multicollinearity in the data. It could be useful for large multivariate data set containing highly correlated predictors.

Regularized discriminant analysis is a kind of a trade - off between LDA and QDA. Recall that, in LDA we assume equality of covariance matrix for all of the classes. QDA assuems different covariance matrices for all the classes. Regularized discriminant analysis is an intermediate between LDA and QDA.

RDA shrinks the separate covariances of QDA toward a common co variance as in LDA. this improves the estimate of the covariance matrices in situations where the number of predictors is larger than the number of samples in the training data, potentially leading to an improvement of the model accuracy.

```
library(klaR)
model <- rda(Species ~., data = train.transf)
pred <- model %>% predict(test.transf)
mean(pred$class == test.transf$Species)
```

```
## [1] 0.9666667
```

Discussion

We have descirbed linear discriminant analysis (LDA) and extensions for predicting the class of an observations based on multiple predictor variables. Discriminant analysis is more suitable to multi class classification problems compared to the logistic regression.

LDA assumes that the different classes have the same variacne or covariacne matrix. we have described many extension of LDA in this section. the most popular extension of LDA is the quadratic discriminant analysis (QDA) which is more flexible than LDA in the sense that it does not assume the quality of the group covariance matrices.

LDA tends to be better than QDA for small data set. QDA is recommended for large training data set.

Sub-section 7 - Naive Bayes Classifier Essentials

The Naive Bayes classifier is a simple and powerful method that can be used for binary and multiclass classification problems.

Naive Bayes classifier predicts the class membership probability of observations using Bayes theorem, which is based on conditional probability

Observations are assigned to the class with the largest probability score.

We will be using klaR and caret package in this sections

```
library(tidyverse)
library(caret)
library(klaR)
data("PimaIndianDiabetes2", package ="mlbench")
```

```
## Warning in data("PimaIndianDiabetes2", package = "mlbench"): data set
## 'PimaIndianDiabetes2' not found
```

```
PimaIndiansDiabetes2 <- na.omit(PimaIndiansDiabetes2)
set.seed(3323)
training.samples <- PimaIndiansDiabetes2$diabetes %>%
  createDataPartition( p =0.8, list=F)
train.data <- PimaIndiansDiabetes2[training.samples, ]
test.data <- PimaIndiansDiabetes2[-training.samples, ]
```

Computing Naive Bayes

```
library("klaR")
model <- NaiveBayes(diabetes ~., data = train.data)
pred <- model %>% predict(test.data)
mean(pred$class == test.data$diabetes)
```

```
## [1] 0.7948718
```

Using caret R package

```
library(klaR)
set.seed(3233)
model <- train(diabetes ~., data = train.data, method ="nb",
               trControl = trainControl("cv", number =10))
predicted.classes <- model %>% predict(test.data)
mean(predicted.classes == test.data$diabetes)
```

```
## [1] 0.7820513
```

Discussion

This chapter introduces the basics of Naive Bayes classification and provides practical examples in R using the klaR and caret package.

Sub-section 8 - SVM Model: Support Vector Machine Essentials

Support Vector Machine (SVM) is a machine learning technique used for classification tasks. SVM will create a boundaries. let n = group, boundaries = n-1. 3 groups i will have 2 lines that separates the data points. The line that im referring to can be a linear / non linear. it can be used for two class or multi - class classification problems.

In practice, those are usually non linear. Technically, svm algorithm perform a non linear classification using kernel trick. Most common kernel transformations are polynomial / radial kernel.

Also, there is also an extension of the SVM for regression which is called support vector regression.

In this chapter, we will be building SVM classifier using the caret R package

```r
library(tidyverse)
library(caret)
data("PimaIndiansDiabetes2", package = "mlbench")
pima.data <- na.omit(PimaIndiansDiabetes2)
set.seed(123)
training.samples <- pima.data$diabetes %>%
  createDataPartition( p = 0.8, list = F)
train.data <- pima.data[training.samples, ]
test.data <- pima.data[-training.samples, ]
```

SVM linear Classifier

```r
set.seed(123)
model <- train(diabetes ~., data = train.data, method = "svmLinear",
              trControl = trainControl("cv", number = 10),
              preProcess = c("center", "scale"))
pred.classes <- model %>% predict(test.data)
head(pred.classes)
```

```
## [1] neg neg pos pos neg neg
## Levels: neg pos
```

Compute model accuracy rate

```r
mean(pred.classes == test.data$diabetes)
```

```
## [1] 0.7692308
```

There is a parameter C, also known as cost. the C determines the possible of misclassifications. It end up imposes a penalty to the model for making an error. Big C = less likely SVM algorithm will mis classify a point. So we want Big C

by default caret builds the SVM linear classifier using C = 1. you can check this by typing model in R console. It is possibly to check all C and choose the optimal one to maximize the model cross validation accuracy.

```r
set.seed(123)
model <- train(
  diabetes ~., data = train.data, method = "svmLinear",
  trControl = trainControl("cv", number = 10),
  tuneGrid = expand.grid(C = seq(0,2,length = 20)),
  preProcess = c("center", "scale"))
```

```
## Warning: model fit failed for Fold01: C=0.0000 Error in .local(x, ...) :
##   No Support Vectors found. You may want to change your parameters

## Warning: model fit failed for Fold02: C=0.0000 Error in .local(x, ...) :
##   No Support Vectors found. You may want to change your parameters

## Warning: model fit failed for Fold03: C=0.0000 Error in .local(x, ...) :
##   No Support Vectors found. You may want to change your parameters

## Warning: model fit failed for Fold04: C=0.0000 Error in .local(x, ...) :
##   No Support Vectors found. You may want to change your parameters
## Warning: model fit failed for Fold05: C=0.0000 Error in .local(x, ...) :
##   No Support Vectors found. You may want to change your parameters
```

```
## Warning: model fit failed for Fold06: C=0.0000 Error in .local(x, ...) :
##   No Support Vectors found. You may want to change your parameters
```

```
## Warning: model fit failed for Fold07: C=0.0000 Error in .local(x, ...) :
##   No Support Vectors found. You may want to change your parameters
```

```
## Warning: model fit failed for Fold08: C=0.0000 Error in .local(x, ...) :
##   No Support Vectors found. You may want to change your parameters
```

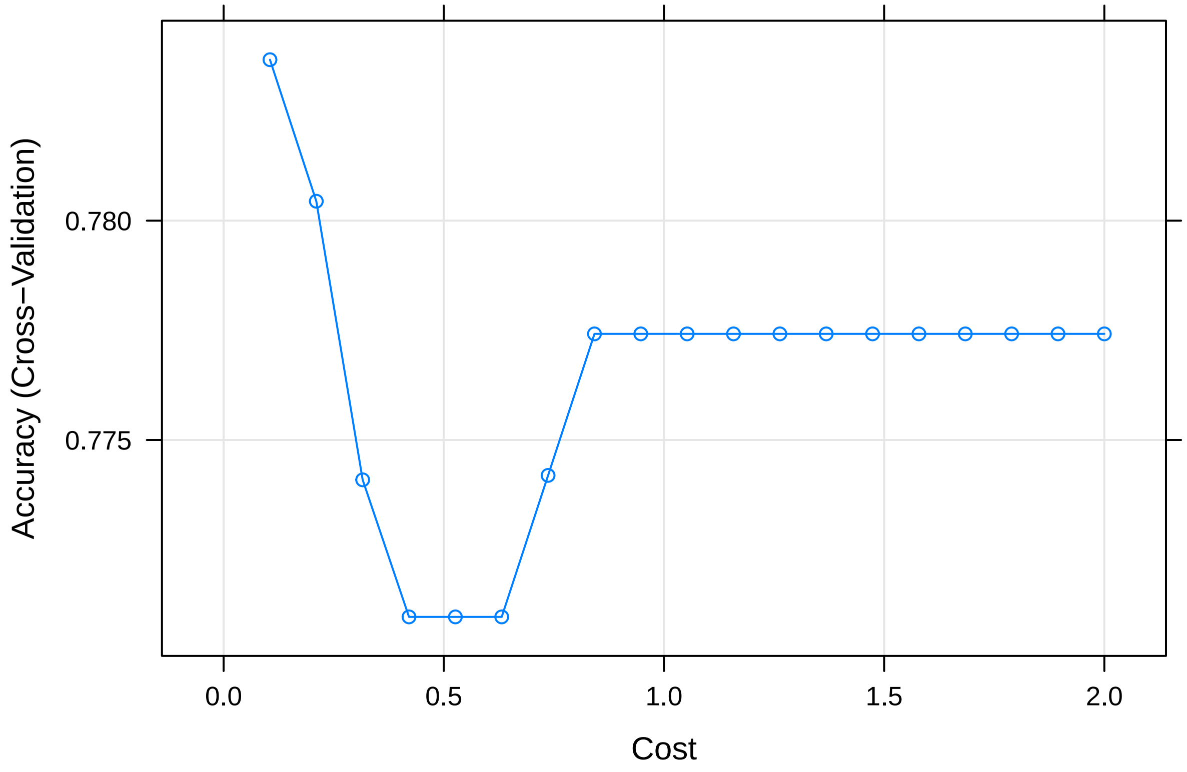
```
## Warning: model fit failed for Fold09: C=0.0000 Error in .local(x, ...) :
##   No Support Vectors found. You may want to change your parameters
```

```
## Warning: model fit failed for Fold10: C=0.0000 Error in .local(x, ...) :
##   No Support Vectors found. You may want to change your parameters
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo, :
## There were missing values in resampled performance measures.
```

```
## Warning in train.default(x, y, weights = w, ...): missing values found in
## aggregated results
```

```r
plot(model)
```



Get the best C

```r
model$bestTune
```

```
##           C
## 2 0.1052632
```

```r
# pred with best tunned data
predicted.classes <- model %>% predict(test.data)
```

```
mean(predicted.classes == test.data$diabetes)
```

```
## [1] 0.7564103
```

SVM classifier using Non-Linear Kernel

To build a non linear SVM classifier, we need to use either polynomial kernel or radial kernel function. the caret package can be used to easily computes the polynomial and the radial SVM non linear models. The package automatically take the optimal values for the model tuning parameters, where optimal is defined as values the maximize the model accuracy.

Computing SVM using radial basis kernel

```
set.seed(123)
model <- train(diabetes ~., data = train.data, method = "svmRadial",
               trControl = trainControl("cv", number = 10),
               preProcess = c("center", "scale"),
               tuneLength = 10)
model$bestTune
```

```
##       sigma    C
## 1 0.1134145 0.25
```

make prediction

```
pred.classes <- model %>% predict(test.data)
mean(pred.classes == test.data$diabetes)
```

```
## [1] 0.7307692
```

Compute SVM using polynomial basis kernel

```
set.seed(123)
model <- train(diabetes~., data = train.data, method = "svmPoly",
               trControl = trainControl("cv", number = 10),
               preProcess = c("center", "scale"),
               tuneLength = 4)
model$bestTune
```

```
##    degree scale   C
## 38      3  0.01 0.5
```

```
# make prediction
pred.classes <- model %>% predict(test.data)
mean(pred.classes == test.data$diabetes)
```

```
## [1] 0.7307692
```

Discussion

looks like our SVM linear model has the highest accuracy. therefore we will use that as our final model.

Sub - section 9 - Evaluation of Classification Model Accuracy: Essentials

After building a classification model, i will need to evaluate the performance of the model.

So i will need to estimate the model prediction accuracy and prediction errors using a new test data set. because we know the actual outcome of observations in the test data, the performance of the predictive model can be accessed by comparing the predicted outcome values against the known outcome values.

We will be covering the commonly used metrics and methods for assessing the performance of predictive classification model:

Average classification accuracy, representing the proportion of correctly classified observations.

Confusion matrix, which is 2x2 table showing four parameters, including the number of true positives, true negatives, false negatives and false positive

Precision, Recall and Specificity which are three major performance metrics describing a ppredictive classification model

ROC curve, which is a graphical summary of the overall performance of the model, showing the proportion of true positives and false positives at all possible values of probability cutoff. the area under the curve (AUC) summarizes the overall performance of the classifier.

build a classification model

```
library(tidyverse)
library(caret)
data("PimaIndiansDiabetes2", package = "mlbench")
pima.data <- na.omit(PimaIndiansDiabetes2)
set.seed(123)
training.samples <- pima.data$diabetes %>%
  createDataPartition(p = 0.8, list =F)
train.data <- pima.data[training.samples, ]
test.data <- pima.data[-training.samples,]
```

use lda model

```
library(MASS)
fit <- lda(diabetes~., data = train.data)
pred <- predict(fit, test.data)
pred.prob <- pred$posterior[, 2]
pred.classes <- pred$class
observed.classes <- test.data$diabetes
```

Overall Classification accuracy

The overall classification accuracy rate corresponds to the proportion of observations that have been correctly classified. Determining the raw classification accuracy is the first step in assessing the performance of a model.

Inversely, the classification error rate is defined as the proportion of observations that have been misclassified. Error rate = 1 - accuracy

The raw classification accuracy and error can be easily computed by comparing the observed classes in the test data against the predicted classes by the model:

```
accuracy <- mean(observed.classes == pred.classes)
accuracy
```

```
## [1] 0.7692308
```

```
error <- mean(observed.classes != pred.classes)
error
```

```
## [1] 0.2307692
```

From the output, we can see the the lienar discriminant analysis correctly predicted the individual outcome in 77% of the cases. This is by far better than random guessing. The misclassification error rate can be calculated as 100 - 77 = 23

In our example, a binary classifier can make two types of erros

diabetes positive to diabetes negative diabetes negative to diabetes positive

we can use confusion matrix to see the predicted outcome values against the known outcome values and see the error

Confusion matrix

We can use table() to produce a confusion matrix in order to find the number of observations were correctly or incorrectly classified. It compares the observed and the predicted outcome values and shows the number of correct and incorrect predictions categorized by type of outcome

```
# Confusion matrix, number of cases
table(observed.classes, pred.classes)
```

```
##                 pred.classes
## observed.classes neg pos
##              neg  44   8
##              pos  10  16
```

```
table(observed.classes, pred.classes) %>% prop.table() %>% round(digits = 4)
```

```
##                 pred.classes
## observed.classes    neg    pos
##              neg 0.5641 0.1026
##              pos 0.1282 0.2051
```

The diagonal elements of the confusion matrix indicate correct predictions, while the off diagonals represent incorrect predictions. Correct prediction -> 0.5651 and 0.2051 So the correct classification rate is the sum of the number of case on the diagonal divided by the sample size in the test data.

```
(44+16)/(nrow(test.data))
```

```
## [1] 0.7692308
```

Please look at the table from the below links https://bit.ly/2JDOw69

each cell has it's own meaning

Cell a: we predicted diabetes - negative and the individuals were diabetes - negative

Cell b: We predictd diabetes - positive, but the individuals didn't actually have diabetes. (type 1 error)

Cell c: We predicted diabetes - negative, but they did have diabetes (Type 2 error)

Cell d: These are cases in which we predicted the individuals would be diabetes positive and they were.

So diagonal is the correct prediction. So the non diagonal numbers are the wrong prediction.

Precision, Recall and Specificity

In addition to the raw classification accuracy, there are many other metrics that are widely used to examine the performance of a classification model, including:

Precision, which is the proportion of true positives among all the individuals that have been predicted to be diabetes-positive by the model. This represents the accuracy of a predicted positive outcome. Precision = TruePositives/(TruePositives + FalsePositives).

Sensitivity (or Recall), which is the True Positive Rate (TPR) or the proportion of identified positives among the diabetes-positive population (class = 1). Sensitivity = TruePositives/(TruePositives + FalseNegatives).

Specificity, which measures the True Negative Rate (TNR), that is the proportion of identified negatives among the diabetes-negative population (class = 0). Specificity = TrueNegatives/(TrueNegatives + FalseNegatives).

False Positive Rate (FPR), which represents the proportion of identified positives among the healthy individuals (i.e. diabetes-negative). This can be seen as a false alarm. The FPR can be also calculated as 1-specificity. When positives are rare, the FPR can be high, leading to the situation where a predicted positive is most likely a negative.

Sensitivy and Specificity are commonly used to measure the performance of a predictive model.

These above mentioned metrics can be easily computed using the function confusionMatrix() [caret package].

In two-class setting, you might need to specify the optional argument positive, which is a character string for the factor level that corresponds to a "positive" result (if that makes sense for your data). If there are only two factor levels, the default is to use the first level as the "positive" result.

```
confusionMatrix(pred.classes, observed.classes, positive = "pos")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction neg pos
##        neg  44  10
##        pos   8  16
##
##                Accuracy : 0.7692
##                  95% CI : (0.66, 0.8571)
##     No Information Rate : 0.6667
##     P-Value [Acc > NIR] : 0.03295
##
##                   Kappa : 0.4706
##
##  Mcnemar's Test P-Value : 0.81366
##
##             Sensitivity : 0.6154
##             Specificity : 0.8462
##          Pos Pred Value : 0.6667
##          Neg Pred Value : 0.8148
##              Prevalence : 0.3333
##          Detection Rate : 0.2051
##    Detection Prevalence : 0.3077
##       Balanced Accuracy : 0.7308
##
##        'Positive' Class : pos
##
```

The model accuracy is around 76.92%.

In our example, sensitivity is 61.54%, which implies that diabetes-positive individuals that were correctly identified by the model as diabetes-positive

The specificity of the model is 84.62% which impies that the proportion of diabetes-negative individuals that were correctly identified by the model as diabetes-negative.

The model precision or the proportion of positive predicted value is 66.67%

In the medical science, sensitivity and specificity are two important metrics that characterize the performance of classifier or screening test. In medical diagnostic, we are concerned about the minimal wrong positive diagnosis. So we want high specificity, which we have 84.62%. To improve the sensitivity / precision, we need to test different probability cutoff for test positive / negative.

ROC curve

ROC curve (receiver operating characteristics curve) is a popular graphical measure for the performance or the accuracy of a classifier, which corresponds to the total proportion of correctly classified observations.

For example, the accuracy of a medical diagnostic test can be assessed by considering the two possible types of errors: false positives, and false negatives. In classification point of view, the test will be declared positive when the corresponding predicted probability, returned by the classifier algorithm, is above a fixed threshold. This threshold is generally set to 0.5 (i.e., 50%), which corresponds to the random guessing probability.

So, in reference to our diabetes data example, for a given fixed probability cutoff:

The true positive rate (or fraction) is the proportion of identified positives among the diabetes-positive population. Recall that, this is also known as the sensitivity of the predictive classifier model. And the false positive rate is the proportion of identified positives among the healthy (i.e. diabetes-negative) individuals. This is also defined as 1-specificity, where specificity measures the true negative rate, that is the proportion of identified negatives among the diabetes-negative population.

Since we don't usually know the probability cutoff in advance, the ROC curve is typically used to plot the true positive rate (or sensitivity on y-axis) against the false positive rate (or "1-specificity" on x-axis) at all possible probability cutoffs. This shows the trade off between the rate at which you can correctly predict something with the rate of incorrectly predicting something. Another visual representation of the ROC plot is to simply display the sensitive against the specificity.

The Area Under the Curve (AUC) summarizes the overall performance of the classifier, over all possible probability cutoffs. It represents the ability of a classification algorithm to distinguish 1s from 0s (i.e, events from non-events or positives from negatives).

For a good model, the ROC curve should rise steeply, indicating that the true positive rate (y-axis) increases faster than the false positive rate (x-axis) as the probability threshold decreases.

So, the "ideal point" is the top left corner of the graph, that is a false positive rate of zero, and a true positive rate of one. This is not very realistic, but it does mean that the larger the AUC the better the classifier.

The AUC metric varies between 0.50 (random classifier) and 1.00. Values above 0.80 is an indication of a good classifier.

In this section, we'll show you how to compute and plot ROC curve in R for two-class and multiclass classification tasks. We'll use the linear discriminant analysis to classify individuals into groups.

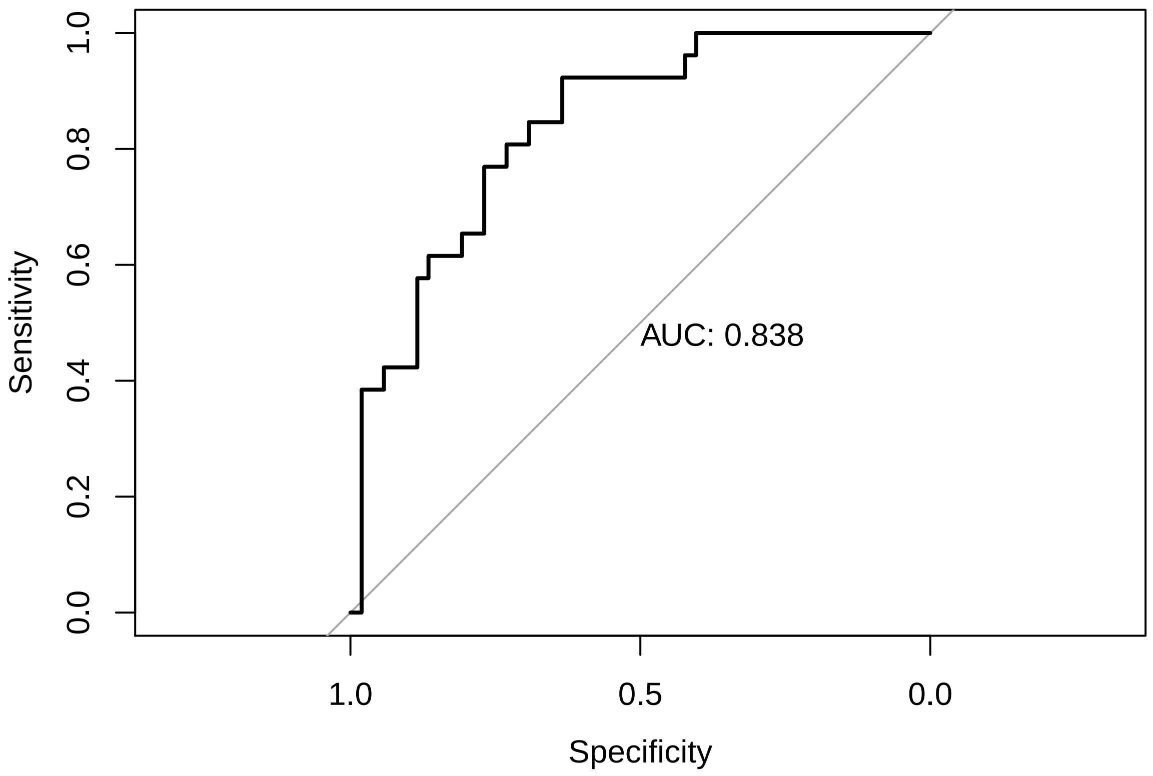
```r
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```r
# compute roc
res.roc <- roc(observed.classes, pred.prob)
```

```
## Setting levels: control = neg, case = pos

## Setting direction: controls < cases
```

```r
plot(res.roc, print.auc = T)
```



The gray diagonal line represents a classifier no better than random chance.

A highly performing classifier will have an ROC that rises steeply to the top-left corner, that is it will correctly identify lots of positives without mis classifying lots of negatives as positives.

In our example, the auc is 0.85, which is close to the max (which is 1). so our classifier can be considered as very good. A classifier that performs no better than chance is expected to have an AUC 0.5 when evaluated on an independent test set not used to train the model.

If we want a classifier model with a specificity of at lest 60%, then the sensitivity is about 0.88%. The corresponding probability threshold can be extract as follow

```r
# extract some interesting results
roc.data <-data_frame(
  thresholds = res.roc$thresholds,
  sensitivity = res.roc$sensitivities,
  specificity = res.roc$specificities
)
```

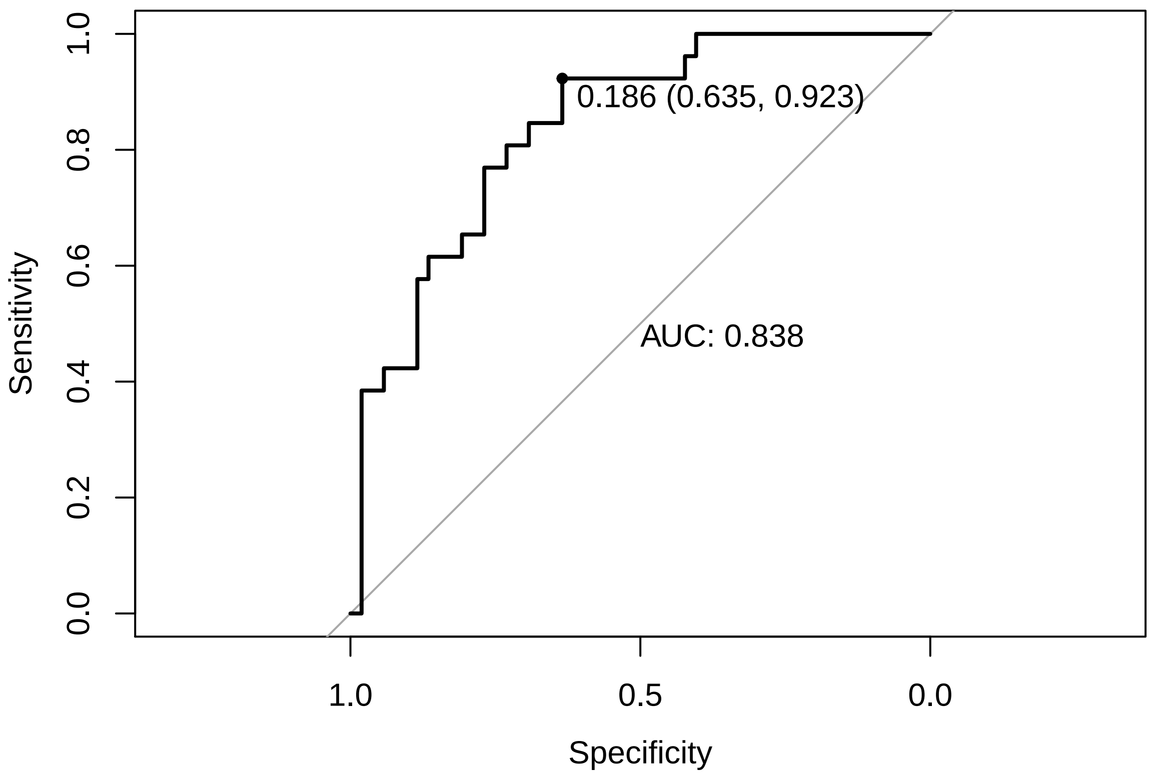
```
## Warning: `data_frame()` is deprecated as of tibble 1.1.0.
## Please use `tibble()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```r
# get the probability threshold for specificity = 0.6
roc.data %>% filter(specificity >= 0.6)
```

```
## # A tibble: 45 x 3
##    thresholds sensitivity specificity
##         <dbl>       <dbl>       <dbl>
##  1      0.180       0.923       0.615
##  2      0.186       0.923       0.635
##  3      0.193       0.885       0.635
##  4      0.208       0.846       0.635
##  5      0.227       0.846       0.654
##  6      0.237       0.846       0.673
##  7      0.249       0.846       0.692
##  8      0.267       0.808       0.692
##  9      0.278       0.808       0.712
## 10      0.284       0.808       0.731
## # ... with 35 more rows
```

The best threshold with the highest sum sensitivity + specificity can be printed as follow. There might be more than one threshold.

```r
plot.roc(res.roc, print.auc = T, print.thres = "best")
```



Section - 6 Articles - Statistical Machine Learning Essentials

Statistical Machine learning refers to a set of powerful automated algorithms that are used to predict an outcome variable based on multiple predictor variables. The algorithms automatically improve their performance through learning from the data, which means they are data driven and do not seek to impose linear or other overall structure on the data. the entire concept is referring to they are non - parametric

The different machine learning methods can be used for both

Classification, where the outcome variable is a categorical variable True or False, positive or negative. Regression, where the outcome variable is a continuous variable

This section we will be covering the following methods:

K-Nearest Neighbors, which predict the outcome of a new observation x as the average outcome of the k most similar observations to x

Decision trees, which build a set of decision rules describing the relationship between predictors and the outcome. These rules are used to predict the outcome of a new observations

Ensemble learning, including bagging, random forest and boosting. These machine learning algorithm are based on decision trees. They produce many tree models from the training data sets, and use the average as the predictive model. These results to the top-performing predictive modeling techniques.

#KNN algorithm

##KNN algorithm for classification

To classify a given new observation, the k nearest neighbors method starts by identifying the k most similar training observations to our new observations, and then assigns new observation to the class containing the majority of its neighbors

KNN algorithm for regression

Similarly to predict a continuous outcome value for given new observation, the KNN algorithm computes the average outcome value of the k training observations that are the most similar to new observations, and returns this value as new observation predicted outcome value.

Similarity measures

Note that, the dis similarity between observations is generally determind using Euclidean distance measure, which is very sensitive to the scale one which predictor variable measurements are made. So, it's generally recommended to standardize the predictor variables for making their scales comparable.

The following sections hows how to build a k nearest neighbor predictive model for classification and regression settings.

load the data and libraries

```r
library(tidyverse)
library(caret)
data("PimaIndiansDiabetes2", package = "mlbench")
PimaIndiansDiabetes2 <- na.omit(PimaIndiansDiabetes2)
set.seed(123)
training.samples <- PimaIndiansDiabetes2$diabetes %>%
  createDataPartition(p = 0.8, list = F)
train.data <- PimaIndiansDiabetes2[training.samples, ]
test.data <- PimaIndiansDiabetes2[-training.samples, ]
```

Compute KNN classifier

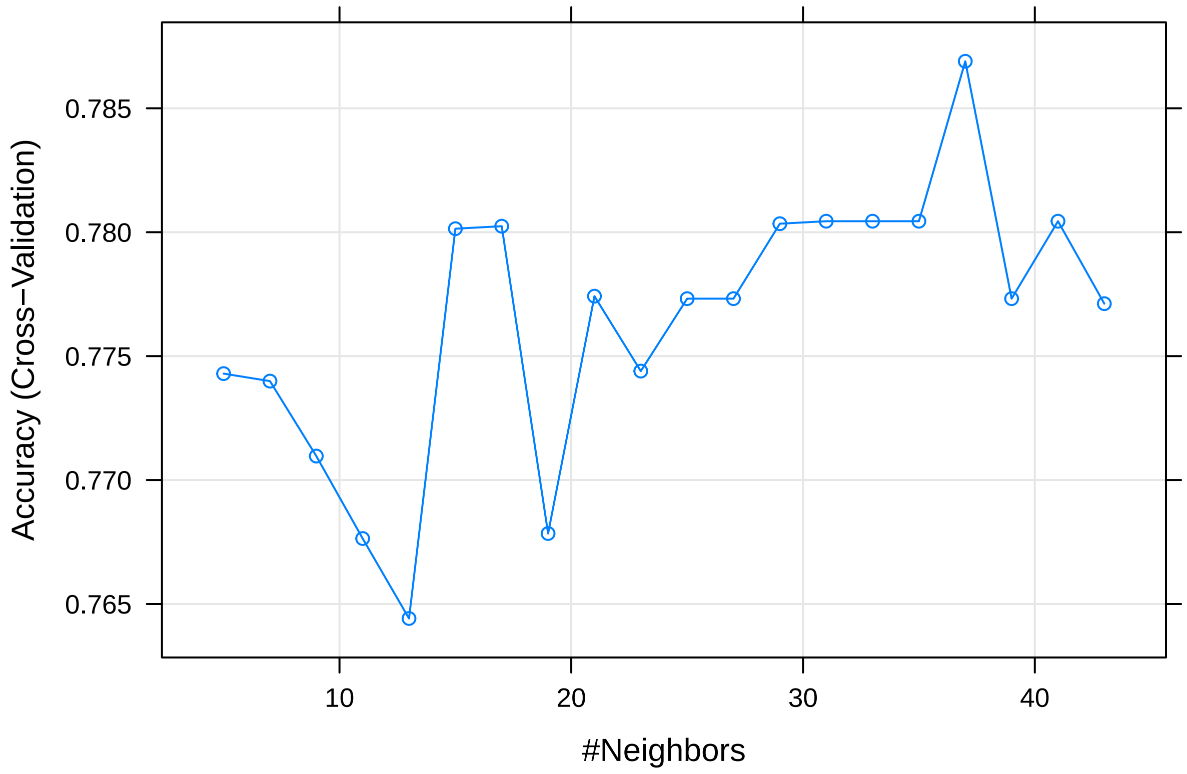
We'll use the caret package, which automatically tests different possible values of k, then chooses the optimal k that minimizes the cross-validation ("cv") error, and fits the final best KNN model that explains the best our data.

Additionally caret can automatically preprocess the data in order to normalize the predictor variables.

We'll use the following arguments in the function train():

trControl, to set up 10-fold cross validation preProcess, to normalize the data tuneLength, to specify the number of possible k values to evaluate

```r
set.seed(123)
model <- train(
  diabetes ~., data = train.data, method = "knn",
  trControl = trainControl("cv", number = 10),
  preProcess = c("center", "scale"),
  tuneLength = 20
)
# plot model accuracy vs different values of k
plot(model)
```



```r
model$bestTune
```

```
##     k
## 17 37
```

```r
# make prediction on the test data
pred.classes <- model %>% predict(test.data)
#compute model accuracy rate
mean(pred.classes == test.data$diabetes)
```

```
## [1] 0.7692308
```

Now, we have an overall prediction accuracy of 76.92% which is pretty good.

KNN for regression

In this section, we will predict a continuous variable using KNN

we will use Boston data from MASS package for predicting the median house value (mdev), in Boston Suburbs, using different predictor variables.

prep the data

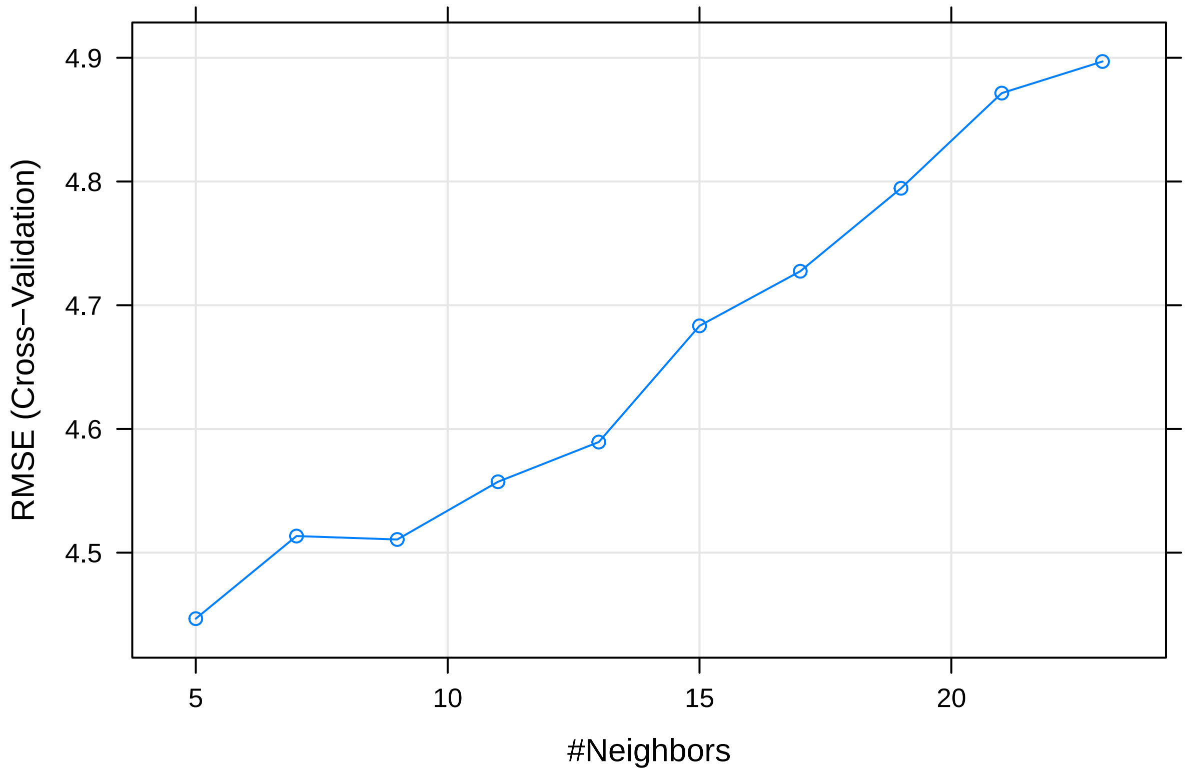
```
data("Boston", package = "MASS")
set.seed(123)
training.samples <-Boston$medv %>% createDataPartition(p=0.8,list = F)
train.data <- Boston[training.samples, ]
test.data <- Boston[-training.samples, ]
```

Compute KNN using caret

The best K is the one that minimize the prediction error RMSE (root mean squard error)

The rmse corresponds to the square root of the average difference between the observed known outcome values and the predicted values, RMSE = mean((observed - predicted)^2) %>% sqrt(). The lower the rmse, the better the model

```
set.seed(123)
model <- train(
  medv~., data = train.data, method = "knn",
  trControl = trainControl("cv", number = 10),
  preProcess = c("center", "scale"),
  tuneLength = 10
)
plot(model)
```



```
model$bestTune
```

```
##   k
## 1 5
```

```r
pred <- model %>% predict(test.data)
head(pred)
```

```
## [1] 32.60 27.28 19.18 20.88 18.86 18.06
```

```r
RMSE(predictions, test.data$medv)
```

```
## [1] 4.590163
```

This chapter describes the basics of KNN (k-nearest neighbors) modeling, which is conceptually, one of the simpler machine learning method.

It's recommended to standardize the data when performing the KNN analysis. We provided R codes to easily compute KNN predictive model and to assess the model performance on test data.

When fitting the KNN algorithm, the analyst needs to specify the number of neighbors (k) to be considered in the KNN algorithm for predicting the outcome of an observation. The choice of k considerably impacts the output of KNN. k = 1 corresponds to a highly flexible method resulting to a training error rate of 0 (overfitting), but the test error rate may be quite high.

You need to test multiple k-values to decide an optimal value for your data. This can be done automatically using the caret package, which chooses a value of k that minimize the cross-validation error

CART Model: Decision Tree Essentials

Decision tree is very important and is a powerful / popular predictive machine learning technique that is used for classification and regression. So it is also known as classification and regression trees (CART).

To implement cart algorithm, we need to install RPART.

```r
library(tidyverse)
library(caret)
library(rpart)
```

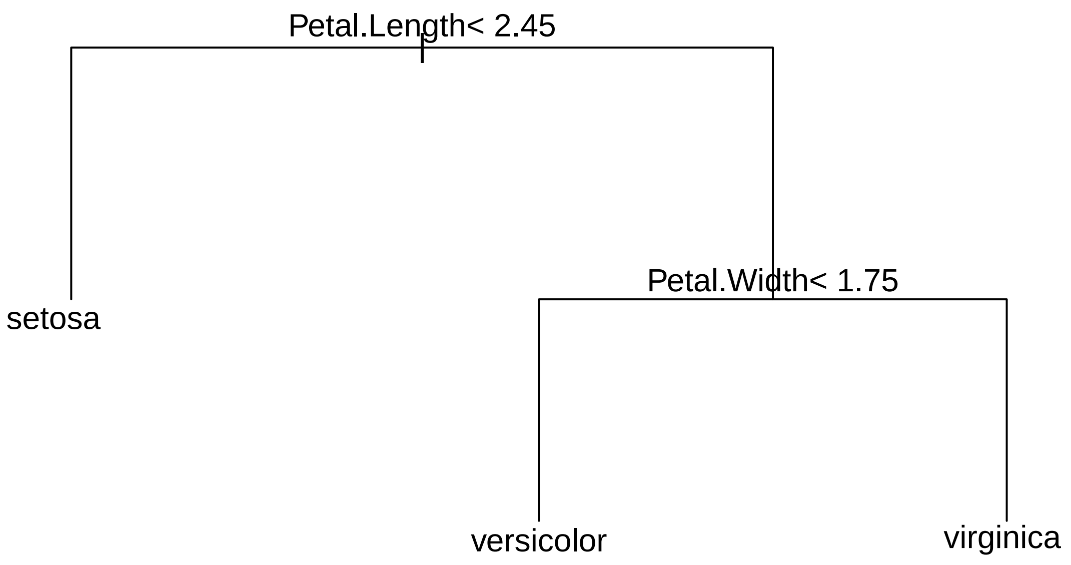
Decision Tree Algorithm

The algorithm of decision tree models works by repeatedly partitioning the data in to multiple sub spaces. So that the outcomes in each final sub space is as homogeneous as possible. This is technically called recursive partitioning.

2 possible results

continuous variable for regression trees categorical variable for classification trees

```r
library(rpart)
model <- rpart(Species ~.,data = iris)
par(xpd = NA) #otherwise on some devices the test is clipped
plot(model)
text(model, digits =4)
```



The plot shows the different possible splitting rules that can be used effectively predict the type of outcome(here, iris species). for example, the top split assigns observations having Petal.length <2.45 to the left branch, where the predicted species are setosa.

the different rules in tree can be printed as fllow:

```r
print(model, digits = 2)
```

```
## n= 150
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 150 100 setosa (0.333 0.333 0.333)
##   2) Petal.Length< 2.5 50   0 setosa (1.000 0.000 0.000) *
##   3) Petal.Length>=2.5 100  50 versicolor (0.000 0.500 0.500)
##     6) Petal.Width< 1.8 54   5 versicolor (0.000 0.907 0.093) *
##     7) Petal.Width>=1.8 46   1 virginica (0.000 0.022 0.978) *
```

If you have some sort of knoweldge in CS, it is actually almost the same thing with the tree algorithm you have seen in CS.

These rules are produced by repeatedly splitting the predictor variables, starting with the variable that has the highest association with the response variable. The process continues until some predetermined stopping criteria are met.

The resulting tree is composed of decision nodes, branches and leaf nodes. The tree is placed from upside to down, so the root is at the top and leaves indicating the outcome is put at the bottom.

Each decision node corresponds to a single input predictor variable and a split cutoff on that variable. The leaf nodes of the tree are the outcome variable which is used to make predictions.

The tree grows from the top (root), at each node the algorithm decides the best split cutoff that results to the greatest purity (or homogeneity) in each subpartition.

The tree will stop growing by the following three criteria (Zhang 2016):

all leaf nodes are pure with a single class; a pre-specified minimum number of training observations that cannot be assigned to each leaf nodes with any splitting methods; The number of observations in the leaf node reaches the pre-specified minimum one. A fully grown tree will overfit the training data and the resulting model might not be performant for predicting the outcome of new test data. Techniques, such as pruning, are used to control this problem.

Choosing the trees split points

Technically, for regression modeling, the split cutoff is defined so that the residual sum of squared error (RSS) is minimized across the training samples that fall within the subpartition.

Recall that, the RSS is the sum of the squared difference between the observed outcome values and the predicted ones, RSS = sum((Observeds - Predicteds)^2). See Chapter @ref(linear-regression)

In classification settings, the split point is defined so that the population in subpartitions are pure as much as possible. Two measures of purity are generally used, including the Gini index and the entropy (or information gain).

For a given subpartition, Gini = sum(p(1-p)) and entropy = -1* sum(p * log(p)), where p is the proportion of misclassified observations within the subpartition.

The sum is computed across the different categories or classes in the outcome variable. The Gini index and the entropy varie from 0 (greatest purity) to 1 (maximum degree of impurity)

Making predictions

```
newdata <- data.frame(Sepal.Length = 6.5, Sepal.Width = 3.0,
                      Petal.Length = 5.2, Petal.Width = 2.0)
model %>% predict(newdata, "class")
```

```
##         1
## virginica
## Levels: setosa versicolor virginica
```

Classification trees

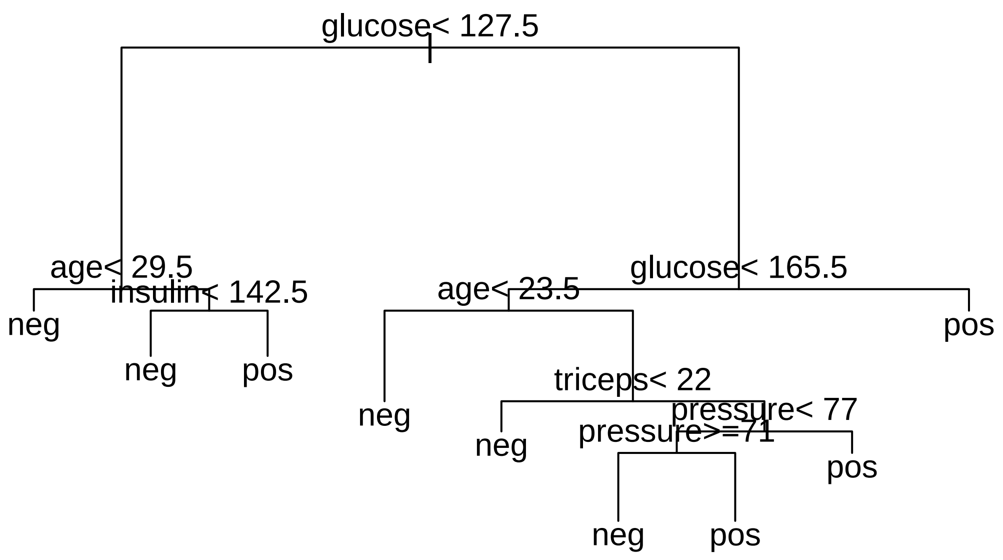
```
data("PimaIndiansDiabetes2", package = "mlbench")
PimaIndiansDiabetes2 <- na.omit(PimaIndiansDiabetes2)
sample_n(PimaIndiansDiabetes2, 3)
```

```
##     pregnant glucose pressure triceps insulin mass pedigree age diabetes
## 648        0     179       50      36     159 37.8    0.455  22      pos
## 554        1      88       62      24      44 29.9    0.422  23      neg
## 575        1     143       86      30     330 30.1    0.892  23      neg
```

```
set.seed(123)
training.samples <- PimaIndiansDiabetes2$diabetes %>%
  createDataPartition(p = 0.8, list = FALSE)
train.data  <- PimaIndiansDiabetes2[training.samples, ]
test.data <- PimaIndiansDiabetes2[-training.samples, ]
```

Fully Grown Trees

```
set.seed(123)
model1 <-rpart(diabetes~., data = train.data, method = "class")
par(xpd = NA)
plot(model1)
text(model1, digits = 3)
```



make predictions on the test data

```r
predicted.classes <-model1 %>% predict(test.data, type = "class")
head(predicted.classes)
```

```
##  19  21  32  55  64  71
## neg neg neg pos pos neg
## Levels: neg pos
```

```r
mean(predicted.classes == test.data$diabetes)
```

```
## [1] 0.7564103
```

However, this full tree including all predictor appears to be very complex and can be difficult to interpret in the situation where you have a large data sets with multiple predictors.

Additionally, it is easy to see that, a fully grown tree will overfit the training data and might lead to poor test set performance.

A strategy to limit this overfitting is to prune back the tree resulting to a simpler tree with fewer splits and better interpretation at the cost of a little bias (James et al. 2014, P. Bruce and Bruce (2017)).

Pruning the tree

Briefly, our goal here is to see if a smaller subtree can give us comparable results to the fully grown tree. If yes, we should go for the simpler tree because it reduces the likelihood of overfitting.

One possible robust strategy of pruning the tree (or stopping the tree to grow) consists of avoiding splitting a partition if the split does not significantly improves the overall quality of the model.

In rpart package, this is controlled by the complexity parameter (cp), which imposes a penalty to the tree for having two many splits. The default value is 0.01. The higher the cp, the smaller the tree.

A too small value of cp leads to overfitting and a too large cp value will result to a too small tree. Both cases decrease the predictive performance of the model.

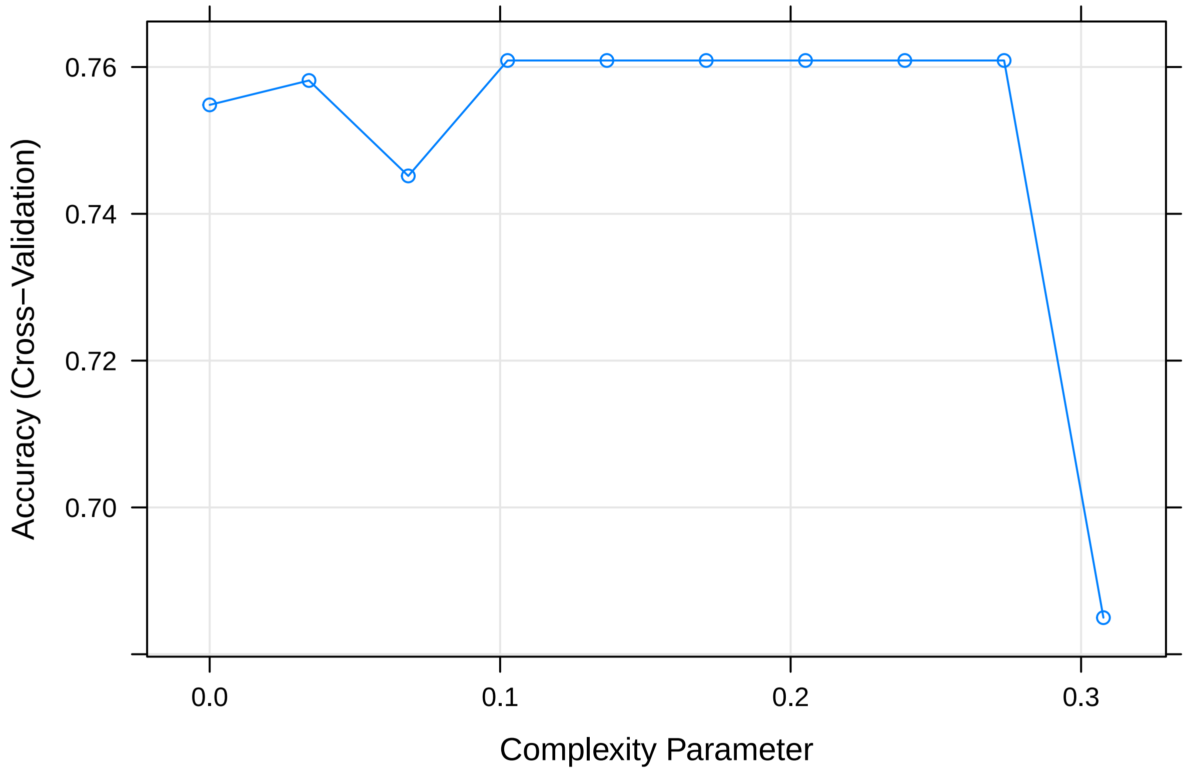
An optimal cp value can be estimated by testing different cp values and using cross-validation approaches to determine the corresponding prediction accuracy of the model. The best cp is then defined as the one that maximize the cross-validation accuracy (Chapter @ref(cross-validation)).

Pruning can be easily performed in the caret package workflow, which invokes the rpart method for automatically testing different possible values of cp, then choose the optimal cp that maximize the cross-validation ("cv") accuracy, and fit the final best CART model that explains the best our data.

You can use the following arguments in the function train() [from caret package]:

trControl, to set up 10-fold cross validation tuneLength, to specify the number of possible cp values to evaluate. Default value is 3, here we'll use 10.

```
set.seed(123)
model2 <- train(
  diabetes ~., data = train.data, method = "rpart",
  trControl = trainControl("cv", number = 10),
  tuneLength = 10
)
# plot model accuracy vs different values of cp (complexity parameter)
plot(model2)
```

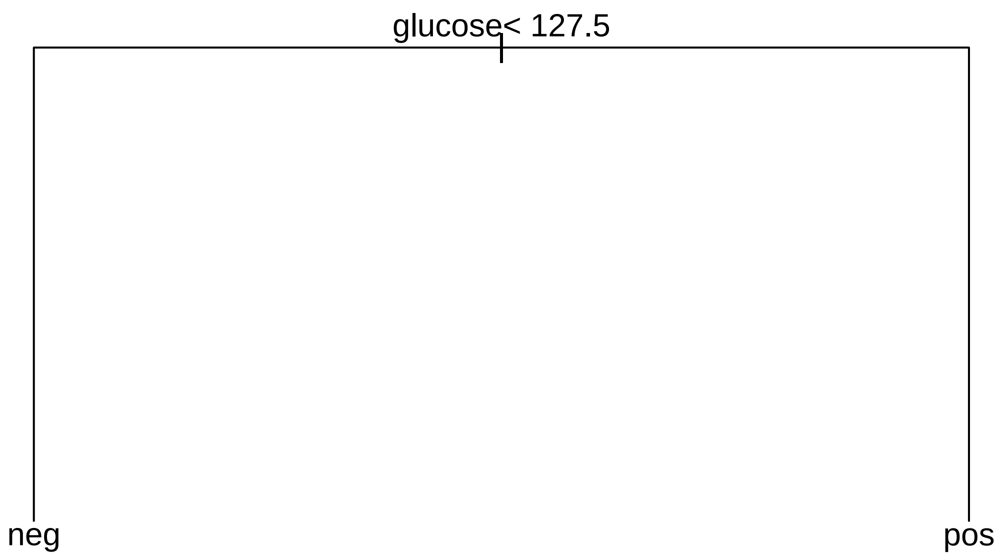


Get the best tune parameter cp that maximizes the model accuracy

```
model2$bestTune
```

```
##          cp
## 9 0.2735043
```

```
par(xpd = NA)
plot(model2$finalModel)
text(model2$finalModel, digits =3)
```



```
model2$finalModel
```

```
## n= 314
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 314 104 neg (0.6687898 0.3312102)
##   2) glucose< 127.5 198  30 neg (0.8484848 0.1515152) *
##   3) glucose>=127.5 116  42 pos (0.3620690 0.6379310) *
```

```
predicted.classes <- model2 %>% predict(test.data)
mean(predicted.classes == test.data$diabetes)
```

```
## [1] 0.7307692
```

From the output above, it can be seen that the best value for the complexity parameter (cp) is 0.2735043, allowing a simpler tree, easy to interpret, with an overall accuracy of 0.7307692, which is comparable to the accuracy (0.7564103) that we have obtained with the full tree. The prediction accuracy of the pruned tree is even better compared to the full tree.

Taken together, we should go for this simpler model.

##Regression trees

Previously, we described how to build a classification tree for predicting the group (i.e. class) of observations. In this section, we'll describe how to build a tree for predicting a continuous variable, a method called regression analysis

The R code is identical to what we have seen in previous sections. Pruning should be also applied here to limit overfiting.

Similarly to classification trees, the following R code uses the caret package to build regression trees and to predict the output of a new test data set.

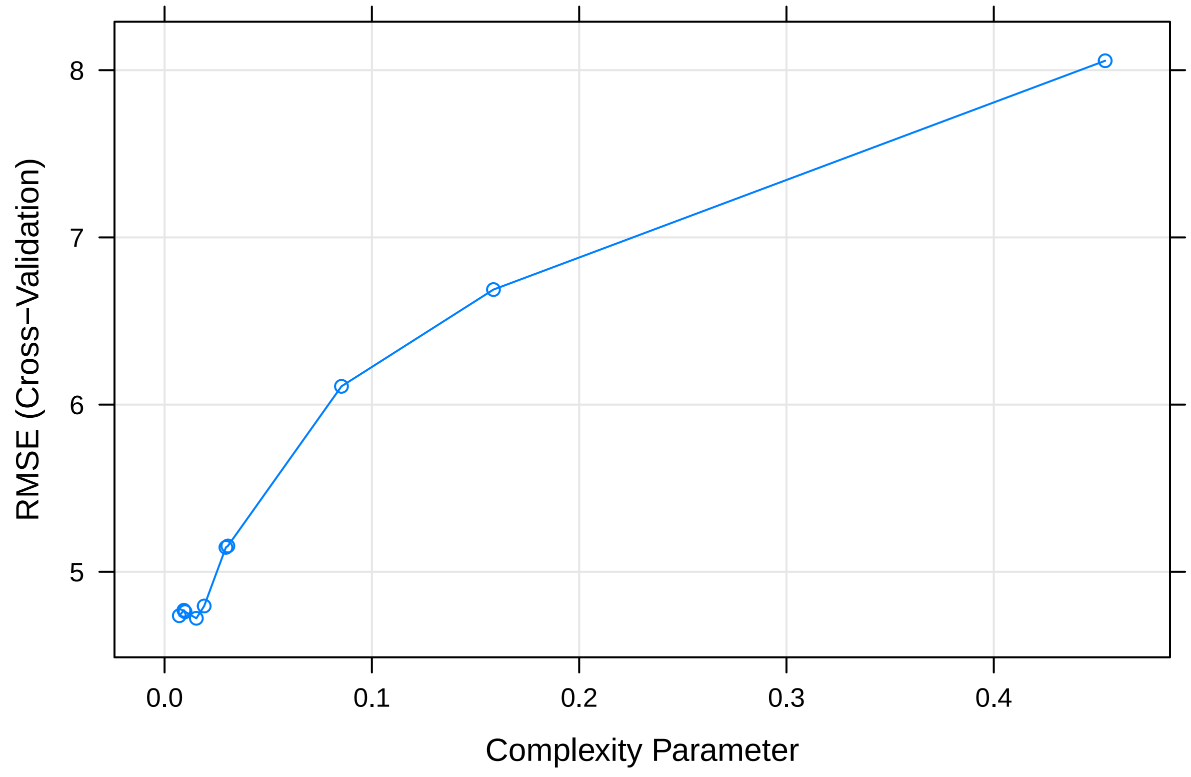
```
# Load the data
data("Boston", package = "MASS")
# Inspect the data
sample_n(Boston, 3)
```

```
##      crim zn indus chas   nox    rm  age    dis rad tax ptratio black lstat
## 1 4.55587  0 18.10    0 0.718 3.561 87.9 1.6132  24 666    20.2 354.7  7.12
## 2 0.06076  0 11.93    0 0.573 6.976 91.0 2.1675   1 273    21.0 396.9  5.64
## 3 0.14476  0 10.01    0 0.547 5.731 65.2 2.7592   6 432    17.8 391.5 13.61
##   medv
## 1 27.5
## 2 23.9
## 3 19.3
```

```
# Split the data into training and test set
set.seed(123)
training.samples <- Boston$medv %>%
  createDataPartition(p = 0.8, list = FALSE)
train.data  <- Boston[training.samples, ]
test.data <- Boston[-training.samples, ]
# Fit the model on the training set
set.seed(123)
model <- train(
  medv ~., data = train.data, method = "rpart",
  trControl = trainControl("cv", number = 10),
  tuneLength = 10
  )
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo, :
## There were missing values in resampled performance measures.
```

```
# Plot model error vs different values of
# cp (complexity parameter)
plot(model)
```



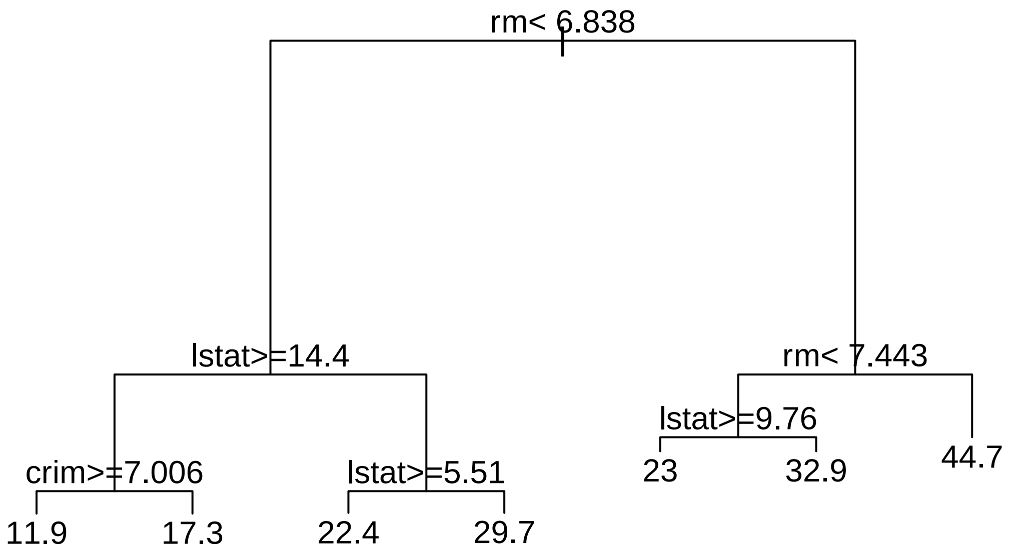
```
# Print the best tuning parameter cp that
# minimize the model RMSE
```

```
model$bestTune
```

```
##          cp
## 4 0.015327
```

Plot the final model

```
# Plot the final tree model
par(xpd = NA) # Avoid clipping the text in some device
plot(model$finalModel)
text(model$finalModel, digits = 3)
```



```
# Decision rules in the model
model$finalModel
```

```
## n= 407
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 407 34125.5800 22.51057
##    2) rm< 6.8375 334 12681.2200 19.62695
##      4) lstat>=14.4 146  2607.9150 15.05822
##        8) crim>=7.006285 60   659.0840 11.86000 *
##        9) crim< 7.006285 86   906.9406 17.28953 *
##      5) lstat< 14.4 188  4659.1330 23.17500
##       10) lstat>=5.51 167  2966.0360 22.35329 *
##       11) lstat< 5.51 21   683.6381 29.70952 *
##    3) rm>=6.8375 73  5959.9890 35.70411
##      6) rm< 7.443 49  2037.2070 31.28367
##       12) lstat>=9.76 8   440.5750 23.02500 *
##       13) lstat< 9.76 41   944.5190 32.89512 *
##      7) rm>=7.443 24  1010.4700 44.72917 *
```

```
# Make predictions on the test data
predictions <- model %>% predict(test.data)
head(predictions)
```

```
##         3         6         9        11        14        15
## 32.89512 29.70952 17.28953 17.28953 22.35329 22.35329
```

```r
# Compute the prediction error RMSE
RMSE(predictions, test.data$medv)
```

```
## [1] 4.484213
```

Conditional inference tree

The conditional inference tree (ctree) uses significance test methods to select and split recursively the most related predictor variables to the outcome. This can limit overfitting compared to the classical rpart algorithm.

At each splitting step, the algorithm stops if there is no dependence between predictor variables and the outcome variable. Otherwise the variable that is the most associated to the outcome is selected for splitting.

```r
# Load the data
data("PimaIndiansDiabetes2", package = "mlbench")
pima.data <- na.omit(PimaIndiansDiabetes2)
# Split the data into training and test set
set.seed(123)
training.samples <- pima.data$diabetes %>%
  createDataPartition(p = 0.8, list = FALSE)
train.data  <- pima.data[training.samples, ]
test.data <- pima.data[-training.samples, ]
```

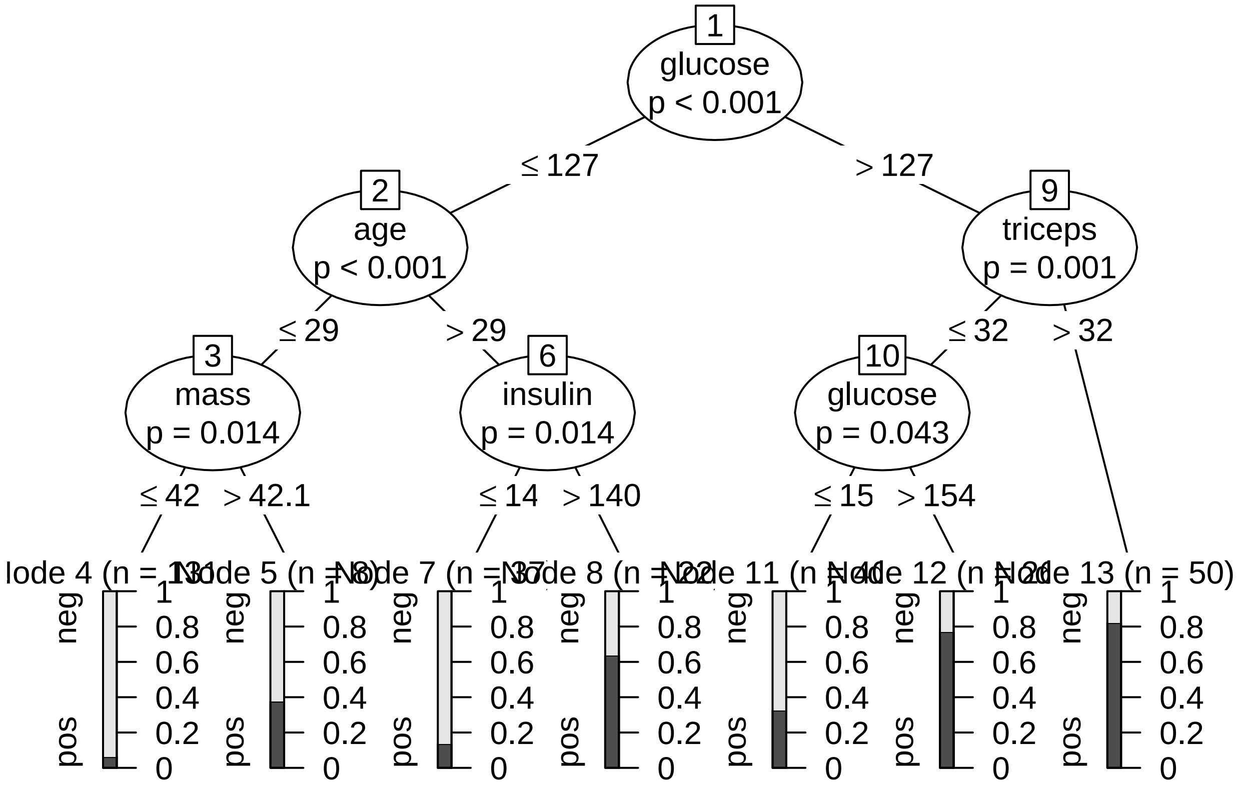
Build conditional trees using the tuning parameters maxdepth and mincriterion for controlling the tree size. caret package selects automatically the optimal tuning values for your data, but here we'll specify maxdepth and mincriterion. The following example create a classification tree:

```r
library(party)
```

```
## Loading required package: grid

## Loading required package: mvtnorm

## Loading required package: modeltools

## Loading required package: stats4

##
## Attaching package: 'modeltools'

## The following object is masked from 'package:car':
##
##     Predict

## Loading required package: strucchange

## Loading required package: zoo

##
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric

## Loading required package: sandwich

##
## Attaching package: 'strucchange'

## The following object is masked from 'package:stringr':
##
```

```
##     boundary
```

```
set.seed(123)
model <- train(
  diabetes ~., data = train.data, method = "ctree2",
  trControl = trainControl("cv", number = 10),
  tuneGrid = expand.grid(maxdepth = 3, mincriterion = 0.95 )
  )
plot(model$finalModel)
```



```
# Make predictions on the test data
predicted.classes <- model %>% predict(test.data)
# Compute model accuracy rate on test data
mean(predicted.classes == test.data$diabetes)
```

```
## [1] 0.7564103
```

The p-value indicates the association between a given predictor variable and the outcome variable. For example, the first decision node at the top shows that glucose is the variable that is most strongly associated with diabetes with a p value < 0.001, and thus is selected as the first node.

This chapter describes how to build classification and regression tree in R. Trees provide a visual tool that are very easy to interpret and to explain to people.

Tree models might be very performant compared to the linear regression model (Chapter @ref(linear-regression)), when there is a highly non-linear and complex relationships between the outcome variable and the predictors.

However, building only one single tree from a training data set might results to a less performant predictive model. A single tree is unstable and the structure might be altered by small changes in the training data.

For example, the exact split point of a given predictor variable and the predictor to be selected at each step of the algorithm are strongly dependent on the training data set. Using a slightly different training data may alter the first variable to split in, and the structure of the tree can be completely modified.

Other machine learning algorithms - including bagging, random forest and boosting - can be used to build multiple different trees from one single data set leading to a better predictive performance. But, with these methods the interpretability observed for a single tree is lost. Note that all these above mentioned strategies are based on the CART algorithm.