Machine Learning Techniques

J.H. van der Zwan

2021-04-19

Contents

Foreword

This book is intended to be a reference for frequently used Machine Learning techniques and how they can be performed in R. Writing this work started in summer 2020 and new chapters are and will be added over time.
It is written with a view to students pursuing a Machine Learning course in a pre-graduate or graduate program. It is not intended to be an exhaustive textbook on all possible Machine Learning techniques.

Chapter 1

Exploring and Preprocessing

Before a Machine Learning model can be generated, an exploratory data analysis (EDA) must be performed. Based on this EDA data is cleaned and prepared for further analysis.
Data cleaning and preparing has, among other issues, to do with missing values, outlier analysis and data formatting.

After importing the data in R first check whether the data type for the different variables is correct.

1.1 Missing values

Missing values are quite common in real world data sets. How to deal with them depends on things like (1) the number of observations with missing values, (2) whether there is pattern in the missing values Different options to deal with missing values:

- remove observations
- numerical variables: impute median (or mean)
- impute value based on knn technique
- categorical variables: impute "NA" value, in other words add a category "NA"

For a more comprehensive treatment of this topic see Van Buuren (2018).

1.1.1 Variables with no or very little variation

Sometimes variables in a data set have no variation at all, in other words are constant. These variables have to be removed before generating a data model. Variables with very little variation also can cause trouble when generating a model. This is especially the case when the data is split in a training and a test set, because it is possible that in the training or in the test set all observations have the same value for this variable.

1.2 Exploratory analysis (1): graphs

Some commonly used graphs:

- numerical variables: histograms; density plots
- categorical variables: bar plots
- relations between two variables
  - two numericals: scatter plot
  - one numerical, one categorical: box plots
  - two categoricals: side-by-side bar plots

1.3 Exploratory analysis (2): Summary statistics

- use summary() function in R

1.4 Outlier analysis

- detect outliers
- analyse these outliers
- decide whether to include them in the further analysis

1.5 Normalization and standardization

For some of the modelling methods numeric features must be normalized or standardised before training the model. This is especially the case when the units used for the features, have impact on the model training process. E.g. models in which distance between observations play an important role, as in KNN models.

Normalization transforms the X-values in the corresponding Z-scores: $z = \frac{x - mean(X)}{sd(X)}$.

Standardization map the X-values on the interval [-1, +1]: $x_standardized = \frac{x - min(X)}{max(X) - min(X)}$.

Normalization and standardization must be performed before splitting the data in training and test data.
The training data are normalized or standardized using the formula above.
Test data are not used to train a model. That is why for normalizing (standardizing) the test data, the mean and standard deviation (minimum and maximum) of the training data are used.
In R one option for normalizing (standardizing) the features is writing a function and aply it on the numeric features. A good alternative is to use the preprocess options in the caret package. See code below.

```
library(caret)
```

```
Loading required package: lattice
```

```
Loading required package: ggplot2
```

```
#example normalizing data, using ISLR::Default data set
df <- ISLR::Default

#split in training and test set; 70% in training set
#use caret::createDataPartition()
set.seed(20210416)
train <- createDataPartition(df$default, p=.7, list=F)

df_train <- df[train,]
df_test <- df[-train,]

#normalize the data in the train set
preproc <- preProcess(df, method=c("center", "scale"))
#outcome is a preProcess model
#apply this model on the train data using predict() function

df_train_preproc <- predict(preproc, newdata=df_train)

#normalize the data in the test set, based on mean and sd in train data
#by applying the preproc preprocessing model on the test set
df_test_preproc <- predict(preproc, newdata=df_test)
```

Chapter 2

Assessing ML models

To assess whether a generated model is performing well or to compare the performance of different models, metrics are needed to measure this performance. This chapter describes the most common metrics used.
It is important to realize that we are mainly interested in how well a model performs out of sample, i.e. outside the data used to generate the model. Comparing the performance within sample with the performance out of sample is also informative. If a model performs well within the sample but not outside the sample, this is generally a signal that it is over fitting. It is quite easy to assess how well a model performs within the sample. The performance of the model out of sample can only be estimated. Commonly used techniques to do just this are discussed in 2.3.

First we discuss the most common model performance metrics in 2.1 and 2.2.

2.1 Assessing Regression Models

The most common metric to assess a regression model is without doubt the Mean Squared Error (MSE) or the Root Mean Squared Error (RMSE). The latter is the square root of the former and is used if we assess one model; if we want to compare a couple of models the MSE is useful as well.

2.2 Assessing Classification Models

When assessing a classification model a justified choice for a metric must be made. The application of the model in its domain determines the choice of a metric. Commonly used metrics are: (1) Accuracy, (2) Sensitivity or True Positive Rate, (3) Specificity or True Negative Rate. See this article on Wikipedia

for an explanation of these and other metrics in case of a binary classification problem.

2.3 Measuring out of sample performance

One pitfall in generating a Machine Learning model is overfitting. The model performs very good on the data used to generate the model, but much worse outside the used sample. When assessing a model, it should be estimated how well the model performs out of sample.
There are different ways to estimate how well a model performs out of the sample data. Commonly used techniques are discussed in the following sections.
The described techniques are used to investigate which model describes the structure in a data set the best, e.g. a linear regression model, or a regression decision tree or some other model. After it is decided which model is most useful in a certain situation, the final model parameters are estimated using the complete available data set.

2.3.1 Training and test data set

The most basic way to estimate out of sample performance is to split the data set at random in a training data set and a test data set. In general the training data set contains 50% or more of all the available data.
The model is generated and trained using the training data set. To estimate the performance out of sample, the performance of the model on the data in the test data set is used.
To improve the estimate of the out of sample error, this procedure can be repeated, e.g. a few hundred times, which leads to a couple of estimates for the out of sample performance. The mean of these estimates can be used as the final estimate for the out of sample performance. This procedure also gives an idea about how accurate this estimate is, or in other words the procedure can be used to construct an interval estimate for the out of sample performance of the model.

2.3.2 Cross validation

In cross-validation, the available data set is randomly divided into k groups of (approximately) equal size. One group is set apart as test set, the other groups are used to train the model. Then the next group is used as test set and so on. So all k-groups are used once as a test set. This procedures leads to k estimates for the out-of-sample error. The average of this estimates is the final estimate for the out-of-sample performance.

2.3.3 Repeated Cross Validation

To come to an even better estimate of the out-of-sample performance, the cross validation procedure can be repeated a couple of times, say t times. This leads to t*k estimates for the out-of sample performance which are averaged to one final estimate.

2.3.3.1 Leave One Out Cross Validation

Leave One Out Cross Validation (LOOCV) is a special case of cross validation. In this case, the test set contains one data point and the model is trained on all data points except the one left out. This leads to n estimates for the out-of-sample performance which are averaged to find the final estimate.

2.3.4 Bootstrapping

Bootstrapping is a state-of-the-art technique to estimate a population parameter. It can be used for point estimates as well as for interval estimates.
The bootstrapping technique is based on one assumption, that is that the sample data is a representative sample of the population being studied as far as the parameter under study is concerned.[1] If that is the case, a simulation of this population can be generated by multiplying the sample many times. From this generated simulated population, many samples with the original sample size can be drawn. Every sample gives an estimate of the parameter studied. The so generated sample estimates together form a good estimate for the sample distribution of the parameter. This sample distribution can be used for point and interval estimates for the parameter.
In stead of multiplying the sample many times and drawing samples from this simulated population, in real, samples are drawn from the sample with replacement and with the same sample size as the original sample
Bootstrapping is used a lot in ML to estimate the out-of-sample error. Bootstrapped samples are used as training data, the data outside the bootstrapped samples as test data. If the generating process makes use of 1000 bootstrapped samples, the output of the process is 1000 estimates for the out-of-sample performance.

[1] if the probability of a particular event equals 1/6, then the odds ratio equals 1/5 or 1 against 5; the probability that the event will not occur is 5 times higher than that it will occur. If for a particular event the odd ratio equals 4 (4 against 1), then the probability the event happens is 4/5 (80%).

Chapter 3

Linear regression models

3.1 Simple linear regression

A simple linear regression model describes the relation between a numeric Y-variable and one numeric X-variable with a linear equation. The general form is:

$Y = \beta_0 + \beta_1 X + \epsilon$ with:

β_0 the intercept
β_1 the slope
ϵ the residual or error term; this term stands for the effect of other variables on Y and for random variation in the Y-variable

Applying this model on a data set means estimating the values of the model parameters, β_0 and β_1, based on the available data.

3.1.1 Assessing a simple linear regression model

To assess the usefulness of an estimated regression model we use the coefficient of determination, R squared, the significance of the relationship between the Y- and the X-variable and the MSE (Mean Squared Error) or the Root Mean Squared Error (RMSE).

3.1.2 R squared (R^2)

The coefficient of determination, R^2, equals 1 minus the quotient of the variation of the Y-values around the regression line and the total variation in the Y-values.

$R^2 = 1 - \frac{\sum(Y-\hat{Y})^2}{\sum(Y-\bar{Y})^2} = 1 - \frac{\sum e_i^2}{\sum(Y-\bar{Y})^2}$

The numerator in the fraction in these formulas, $\sum e_i^2$, measures the variation in the Y-values which is not explained by the regression model. The denominator, $\sum(Y-\bar{Y})^2$, measures the total variation in the Y-values. So the fraction measures the proportion of unexplained variation in the Y-values; 1 minus this fraction measures the proportion of the variation in the Y-values which is explained by the regression model.

For instance, if $R^2 = .65$, 65% of the **variation in the Y-values** is explained by the model; in other words, 65% of the variation in the Y-values can be explained by the variation in the X-values.

3.1.3 Significance of the X-variable in the model

Another way to assess a regression model is testing the significance of the coefficient of the X-variable. This is done by testing H_A_: $\beta_1 \neq 0$. If no support is found in the data for this hypothesis, the X-variable is not a significant variable. Or in other words, the Y-variable doesn't depend significantly on the X-variable. Software packages used to generate a linear regression model, will also report the p-value for this test. A p-value below .05 is commonly interpreted as a significant result.

3.1.4 Mean Squared Error and Root Mean Squared Error

The most used metric to assess a regression model is the MSE (Mean Squared Error) or the square root of the MSE, the RMSE (Root Mean Squared Error):
$MSE = \frac{\sum e_i^2}{N}$
$RMSE = \sqrt{\frac{\sum e_i^2}{N}}$
These metrics measure the average distance of the data points to the regression line, measured in the Y direction. It is a metric which measures the model error.

Remember that in general we are more interested in the out-of-sample performance than in the in-sample performance. In case of a (simple) linear regression model, it can be shown that the in-sample MSE is a good estimate for the out-of-sample MSE. The estimate can even be improved by using N-2 in the denominator in stead of N:
$RMSE = \sqrt{\frac{\sum e_i^2}{N-2}}$.

Of course the commonly used techniques to estimate the out of sample error can be used as wel, such as splitting the data set in a training and a test set, cross validation or bootstrapping. In the example in the next session the different techniques to estimate the out-of-sample performance of a linear rergession model are illustrated, using an example.

LICENSE_PLATE	BRAND	COLOR	ECONOMY_LABEL	FUEL_DESCRIPTION	CILINDER_CONTENT	MASS	CA
SP599G	AUDI	BLACK	A	DIESEL	1,968	1,635	
SG896T	AUDI	BLACK	D	DIESEL	2,967	1,835	
50SNX1	KIA	GRAY	B	PETROL	1,396	1,217	
SX290V	AUDI	GRAY	B	PETROL	1,395	1,390	
GZ086R	CITROEN	GRAY	A	PETROL	1,199	1,215	
TP133J	AUDI	GRAY	C	PETROL	1,984	1,475	
HX408N	KIA	BLACK	D	PETROL	1,396	1,153	
67PNH9	AUDI	GRAY	B	DIESEL	1,968	1,500	
TN023P	AUDI	WHITE	B	PETROL	1,395	1,200	
5SBP96	AUDI	GRAY	B	PETROL	1,984	1,540	

3.2 Example simple linear regression

As an example of applying a linear regression model, we use a dataset with a random sample of Dutch cars from three brands: AUDI, CITROEN and KIA. The data is sampled from the RDW website on August 20, 2020. Data from two databases on this website are joined into one data frame and some data cleaning has been performed.

Table 3.1
First 10 Observations in RDW Data Set

```
library(tidyverse)
library(flextable)
library(kableExtra)
rdw_sample <- read_csv("data/20200820rdw_sample.csv")
flextable(head(rdw_sample, n = 10)) %>% fontsize(size = 7, part = "all")
```

Figure 3.1
Summary of the RDW Dataset

```
summary(rdw_sample)
```

```
 LICENSE_PLATE         BRAND              COLOR           ECONOMY_LABEL
 Length:8341        Length:8341        Length:8341        Length:8341
 Class :character   Class :character   Class :character   Class :character
 Mode  :character   Mode  :character   Mode  :character   Mode  :character
```

```
 FUEL_DESCRIPTION   CILINDER_CONTENT      MASS      CATALOG_PRICE
 Length:8341        Min.   : 998     Min.   : 765   Min.   : 2423
 Class :character   1st Qu.:1197     1st Qu.: 989   1st Qu.:17485
 Mode  :character   Median :1396     Median :1215   Median :25750
                    Mean   :1472     Mean   :1202   Mean   :29271
                    3rd Qu.:1598     3rd Qu.:1402   3rd Qu.:35980
                    Max.   :4163     Max.   :2420   Max.   :99983
  SPECIAL_TAX
 Min.   :    9
 1st Qu.: 1573
 Median : 3084
 Mean   : 4102
 3rd Qu.: 5662
 Max.   :27622
```

The aim is to generate a model that estimates the Catalogue Price (CATALOG_PRICE) given the value of the other variables as predictors, excepted the Special Tax variable (this variable cannot be a predictor for the catalogue price, because this special tax depends on this price).

The first model is a simple linear model with MASS as predictor for CATALOG_PRICE.

Figure 3.2
Output Simple Linear Regression Model in R

```
library(MOTE) #apa() function for apa representation of numbers

linmod_01 <- lm(CATALOG_PRICE ~ MASS, data = rdw_sample)
Rsquared <- apa(summary(linmod_01)$r.squared, 3, FALSE)
rmse <- prettyNum(round(summary(linmod_01)$sigma, 0), big.mark = ",")
mse <- prettyNum(summary(linmod_01)$sigma^2, big.mark = ",")
summary(linmod_01)
```

```
Call:
lm(formula = CATALOG_PRICE ~ MASS, data = rdw_sample)

Residuals:
   Min     1Q Median     3Q    Max
-45568  -5890    475   3837  48936

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -3.125e+04  4.434e+02  -70.47   <2e-16 ***
MASS         5.033e+01  3.599e-01  139.85   <2e-16 ***
---
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 8833 on 8339 degrees of freedom
Multiple R-squared:  0.7011,    Adjusted R-squared:  0.701
F-statistic: 1.956e+04 on 1 and 8339 DF,  p-value: < 2.2e-16
```

In the output can be seen that $R^2 = .701$; in other words 70.1% of the variation in the catalog prices is explained by the model.
MASS is a significant variable in the model, $P\text{-}value < .001$. The RMSE, in the output named as 'Residual standard error', equals 8,833. The MSE is just this number squared, MSE = 78,020,205.

As said before, in case of a (simple) linear regression model, the so calculated MSE is a good estimate for the out-of-sample MSE. This is the case, because with a linear regression model, possible overfitting is not really an issue. However, for more complicated models, it is.
To explain the concepts for estimating out-of-sample errors when using more complex models, in the next seections the MSE is calculated in more general ways.

3.2.1 Estimating out-of-sample error using a training and a test set

The first method splits the data randomly in a training set and a test set. The model parameters are estimated using the training set. The out-of-sample error is estimated using the test set.

```
set.seed(20200823)
prop_train <- .50
train <- sample(1:nrow(rdw_sample), prop_train*nrow(rdw_sample))
rdw_train <- rdw_sample[train,]
rdw_test <- rdw_sample[-train,]

linmod_02 <- lm(CATALOG_PRICE ~ MASS, data = rdw_train)
predict_train <- predict(linmod_02)
MSE_train <- sum((predict_train - rdw_train$CATALOG_PRICE)^2)/(nrow(rdw_train) - 2)
RMSE_train <- prettyNum(sqrt(MSE_train),
                        digits=0,
                        big.mark = ",")
#alternative
#RMSE_train <- summary(linmod_02)$sigma

predict_test <- predict(linmod_02, newdata = rdw_test)
MSE_test <- mean((predict_test - rdw_test$CATALOG_PRICE)^2)
RMSE_test <- prettyNum(sqrt(MSE_test),
                       digits=0, big.mark = ",")
```

The within sample RMSE, i.e. the RMSE calculated based on the training data, equals 8,673. The out-of-sample error estimate equals 8,992.

3.2.2 Estimating out-of-sample error using cross validation

In this section 10-fold cross validation is used to estimate the out-of-sample RMSE.

```
#10-fold CV
library(caret)
library(tidyverse)
library(kableExtra)
set.seed(20200824)
rdw_sample$GROUP <- sample(c(1, rep(1:10, 834)))

MSEs <- NULL

for (t in 1:10) {
  trainset <- filter(rdw_sample, GROUP != t)
  testset <- filter(rdw_sample, GROUP == t)
  linmod <- lm(CATALOG_PRICE ~ MASS, data = trainset)
  predict_test <- predict(linmod, newdata = testset)
  mse <- mean((predict_test - testset$CATALOG_PRICE)^2)
  MSEs[t] = mse
}
mse_estimate <- mean(MSEs)
rmse_estimate <- sqrt(mse_estimate)
```

This procedure leads to 10 estimates for the out-of-sample MSE: 72,139,107, 71,120,651, 78,852,650, 74,810,525, 71,190,694, 77,776,460, 77,709,740, 84,458,930, 84,033,745, 88,307,401. The average of these estimates, 78,039,990, is the CV estimate of the out-of-sample MSE. The RMSE estimate is 8,834.
The variation in the estimates can be used to make a statement about the accuracy of the CV estimate, or to calculate an interval estimate for the out-of-sample error. To improve such an interval estimate, the CV procedure can be repeated dozens or hundreds of times; if this is done this procedure is called Repeated Cross Validation. For instance, if the procedure is repeated 200 times, this leads to 2000 estimates for the out-of-sample MSE, which can be used to calculate an interval estimate for the out-of-sample estimate. To calculate a 95% confidence interval (CI) for the out-of-smaple MSE, the .025th and the .975th quantile of the 2000 estimates are calculated. The probability that the interval between these two values contains the "real" out-of-sample MSE equals 95%.

```
#200 times repeated 10-fold CV
set.seed(20200829)

MSEs <- NULL

for (i in 1:200) {
rdw_sample$GROUP <- sample(c(1, rep(1:10, 834)))
for (t in 1:10) {
  trainset <- filter(rdw_sample, GROUP != t)
  testset <- filter(rdw_sample, GROUP == t)
  linmod <- lm(CATALOG_PRICE ~ MASS, data = trainset)
  predict_test <- predict(linmod, newdata = testset)
  mse <- mean((predict_test - testset$CATALOG_PRICE)^2)
  MSEs <- c(MSEs, mse)
}}

RMSEs <- sqrt(MSEs)
mse_estimate <- mean(MSEs)
mse_estimate_025 <- quantile(MSEs, .025)
mse_estimate_975 <- quantile(MSEs, .975)
rmse_estimate <- mean(sqrt(MSEs))
rmse_estimate_025 <- quantile(sqrt(MSEs), .025)
rmse_estimate_975 <- quantile(sqrt(MSEs), .975)


rmse_plot <- tibble(RMSE = RMSEs) %>%
  ggplot(aes(x=RMSE)) +
  geom_histogram(binwidth = 50, fill = "royalblue") +
  geom_vline(aes(xintercept = rmse_estimate_025), col = "grey") +
  geom_vline(aes(xintercept = rmse_estimate_975), col = "grey") +
  theme_minimal()
```

Figure 3.3
RMSE Estimates Based on 200 times Repeated 10-folds CV

```
rmse_plot
```

Note. The gray lines mark the 95% CI for the RMSE.

In this example 200 times Repeated 10-fold CV, leads to:
point estimate for the out-of-sample MSE: 78,060,204
point estimate for the out-of-sample RMSE: 8,830
95% interval estimate for out-of-sample MSE: <67,379,651; 89,119,641>
95% interval estimate for out-of-sample RMSE: <8,209; 9,440>.

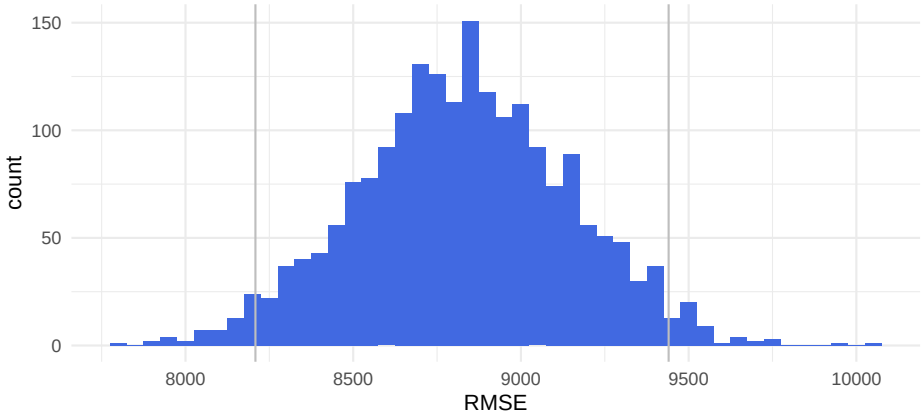


Figure 3.1: rmse_plot

3.2.3 Estimating out-of-sample-error using bootstrapping

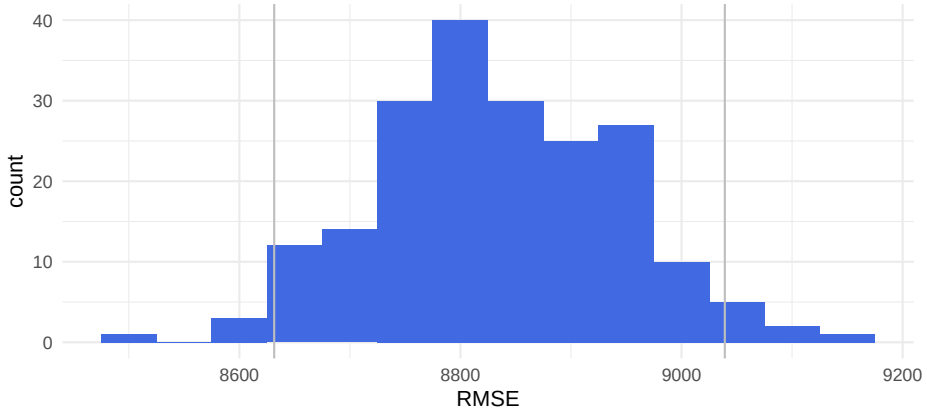
Bootstrapping is a technique of resampling from the available data set. Hundreds or even thousands of samples from the data set are drawn wtih replacement and sample size the number of observartions in the data set. This means that observations can be drwan more than one time in a bootstrap sample.
Bootstrapping is based on just one assumption: the available data form a representative sample of the research population. Under this assumption the population can be assumed to consist of a couple of thousand times this sample. Drawing a sample from this assumed population, is the same as drawing a sample from the available sample data with replacement. Bootstrap samples have the same sample size as the originale available sample data set. Under the mentioned assumption, a bootstrap sample can be seen as a random sample from the population of interest. Drawing 200 bootstrap samples yields 200 possible samples from the research population. These bootstrap samples can be used to estimate unknown population parameters, e.g. an MSE in a simple linear regression problem.

```
#200 bootstrap samples
set.seed(20200829)
MSEs_bootstrap <- NULL
for (i in 1:200) {
  sample <- sample_n(rdw_sample, size = nrow(rdw_sample),replace = TRUE)
  linmod <- lm(CATALOG_PRICE~MASS, data = sample)
  predict_bootstrap <- predict(linmod)
  mse <- mean((predict_bootstrap - sample$CATALOG_PRICE)^2)
  MSEs_bootstrap <- c(MSEs_bootstrap, mse)
}
```

```
RMSEs_bootstrap <- sqrt(MSEs_bootstrap)

rmse_bootstrap_plot <- tibble(RMSE = RMSEs_bootstrap) %>%
  ggplot(aes(x=RMSE)) +
  geom_histogram(binwidth = 50, fill = "royalblue") +
  geom_vline(aes(xintercept = quantile(RMSEs_bootstrap, .025)), col = "grey") +
  geom_vline(aes(xintercept = quantile(RMSEs_bootstrap, .975)), col = "grey") +
  theme_minimal()
```

Figure 3.4
RMSE estimates based on 200 Bootstrap Samples

```
rmse_bootstrap_plot
```



Results bootstrap sampling
Estimate out-of-sample MSE = 78,056,696
Estimate out-of-sample RMSE = 8,834

3.3 Caret Package

The caret package is a usefull R package to apply a Machine Learning (ML) model on a data set. Actually it is not one package, but a combination of dozens of packages which are developed for all kinds of different ML models. The central function to apply a ML model on a sample is the train() function. This section provides the code when using the caret package, for the examples in the former sections.

3.3.1 Estimating MSE using bootstrap

The first example is the use of the bootstrap sampling method. The reason for starting with the bootstrap sample, is because this is the default method in the caret package.

Figure 3.5
The Generated Linear Model with Caret, Default Settings

```
library(caret)
set.seed(20200829)
linmod_caret_bootstrap <-
  train(CATALOG_PRICE~MASS, data = rdw_sample, method = "lm")

linmod_caret_bootstrap
```

```
Linear Regression

8341 samples
   1 predictor

No pre-processing
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 8341, 8341, 8341, 8341, 8341, 8341, ...
Resampling results:

  RMSE      Rsquared   MAE
  8815.187  0.7015241  6444.3

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

Examine the output in Table 3.1 carefully to understand how the caret package generates a linear model. As default caret produces 25 bootstrap samples to estimate the out-of-sample performance. The reported out-of-sample error is the average of 25 estimated out-of-sample errors. The reported RMSE is the bootstrap RMSE, the square root of the average of 25 bootstrapped MSE's.

Figure 3.6
The Summary of the Linear Model

```
summary(linmod_caret_bootstrap)
```

```
Call:
lm(formula = .outcome ~ ., data = dat)

Residuals:
   Min     1Q Median     3Q    Max
```

```
-45568  -5890    475   3837  48936

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -3.125e+04  4.434e+02  -70.47   <2e-16 ***
MASS         5.033e+01  3.599e-01  139.85   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 8833 on 8339 degrees of freedom
Multiple R-squared:  0.7011,    Adjusted R-squared:  0.701
F-statistic: 1.956e+04 on 1 and 8339 DF,  p-value: < 2.2e-16
```

Table 3.2 shows the model that is reported as the final model. This final model is based on all available data. This model will be used if predictions for new data are made with the bootstrap model.

In the default setting 25 bootstrap samples are used to estimate the out-of-sample performance. With the trControl argument of the train() function, this number can be adjusted to another number.

Figure 3.7
Caret Linear Model Based on 200 Bootstrap Samples

```
library(caret)
set.seed(20200829)
linmod_caret_bootstrap_200 <-
  train(CATALOG_PRICE~MASS, data = rdw_sample, method = "lm",
        trControl = trainControl(number = 200))
linmod_caret_bootstrap_200
```

```
Linear Regression

8341 samples
   1 predictor

No pre-processing
Resampling: Bootstrapped (200 reps)
Summary of sample sizes: 8341, 8341, 8341, 8341, 8341, 8341, ...
Resampling results:

  RMSE      Rsquared   MAE
  8834.139  0.7017922  6455.909

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

The summary of this model is based on all the observations in the training set and is the same as the model in Figure 3.2.

Figure 3.8
Summary Caret Linear Model Based on 200 Bootstrap Samples

```
summary(linmod_caret_bootstrap_200)
```

```
Call:
lm(formula = .outcome ~ ., data = dat)

Residuals:
   Min     1Q Median     3Q    Max
-45568  -5890    475   3837  48936

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -3.125e+04  4.434e+02  -70.47   <2e-16 ***
MASS         5.033e+01  3.599e-01  139.85   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 8833 on 8339 degrees of freedom
Multiple R-squared:  0.7011,    Adjusted R-squared:  0.701
F-statistic: 1.956e+04 on 1 and 8339 DF,  p-value: < 2.2e-16
```

3.3.2 Estimating MSE using Repeated Cross Validation

The trControl() argument in the caret::train() funciton can also be used to use another method for estimating the out-of-sample performance. For instance cross validation or repeated cross validation.

Figure 3.8
Caret Linear Model Based on 200 Times Repeated 10-Fold CV

```
set.seed(20200829)
linmod_caret_repeatedcv <-
  train(CATALOG_PRICE~MASS, data = rdw_sample, method = "lm",
        trControl = trainControl(method="repeatedcv", number=10,
                                 repeats = 200))
linmod_caret_repeatedcv
```

```
Linear Regression

8341 samples
   1 predictor

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 200 times)
```

```
Summary of sample sizes: 7506, 7508, 7506, 7508, 7508, 7507, ...
Resampling results:

  RMSE      Rsquared   MAE
  8828.989  0.7015469  6460.448

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

Figure 3.9
Summary Caret Linear Model Based on 200 Times Repeated 10-Fold CV

```
summary(linmod_caret_repeatedcv)
```

```

Call:
lm(formula = .outcome ~ ., data = dat)

Residuals:
   Min     1Q Median     3Q    Max
-45568  -5890    475   3837  48936

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -3.125e+04  4.434e+02  -70.47   <2e-16 ***
MASS         5.033e+01  3.599e-01  139.85   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 8833 on 8339 degrees of freedom
Multiple R-squared:  0.7011,    Adjusted R-squared:  0.701
F-statistic: 1.956e+04 on 1 and 8339 DF,  p-value: < 2.2e-16
```

The output of the train() function is a list with lots of information. One of the components of the list is the 'results' component. Besides the estimated RMSE and R squared value.

Figure 3.10
Results Caret Linear Model Based on 200 Times Repeated 10-Fold CV

```
linmod_caret_repeatedcv$results
```

```
  intercept     RMSE  Rsquared      MAE RMSESD RsquaredSD    MAESD
1      TRUE 8828.989 0.7015469 6460.448 302.76 0.01411291 180.9893
```

If 200 times repeated 10-Fold CV is applied, 2000 linear models are generated and evaluated. The reported out-of-sample performance is the average of 2000 out-of-sample performance estimates. The reported standard deviation (RMSESD) is the standard deviation of these 2000 estimates.

Chapter 4

Logistic regression models

Machine learning assumptions: (i) a dataset is available with observations for a number of explanatory X-variables and the corresponding value of a response Y-variable and (ii) there is a function f that describes the relationship between the X variables and the Y variable: Y = f(X1, X2,..., X3) + ϵ.

Machine Learning objective: to find the function f.

If the Y variable is categorical, we have a classification problem. The aim of constructing a model is to predict, given X values of new observations, the corresponding Y value. In other words to predict to which Y category the observation belongs. Models are usually constructed in such a way that the f-value per category gives a probability that the observation belongs to that category. The user of the model can then classify the observations himself based on these probabilities. This does not necessarily mean that the category with the highest probability is chosen. For example, if a doctor decides on the basis of observations that a patient has a 30% probability of having a certain disease, he may decide to prescribe drugs to combat this disease, although there is a 70% probability that this is unnecessary.

4.1 Binary Y-variable

If the Y-variable is dichotomous, i.e. has two categories, these are often labeled POS and NEG or 1 and 0 and sometimes -1 and +1. The latter is especially the case when a perceptron model is used to distinguish between the two Y-categories.

4.1.1 Linear regression with a binary Y-variable

If the Y-categories are labeled with the number 0 and 1, theoretical a regression model can be used, after all the Y-variable is numeric.
However, this method has some disadvantages:

- assumptions of a linear regression model are violated
- interpretation of predicted values; in a linear regression model, the model Y-value can be interpreted as the average Y-value, givena certain X-value; because the Y-variable is binary, an average Y-value given an X-value should be a value in the interval [0, 1] and can be interpreted as a probability; predicted values based on a linear regression model however, can be far outside this interval.

A model that can be used in case of a dichotomous Y variable, models the probabilities of Y=1 as a function of X: P (Y=1 | X) = f(X), where f is a function that only takes on values in the interval [0, 1]. An example of such a model is the widely used logistic regression model.

4.1.2 Logistic Regression Model

The simple logistic regression model:

P(Y=1 | X) = $\frac{e^{\beta_0+\beta_1 X}}{1+e^{\beta_0+\beta_1 X}}$ or in short: P(X) = $\frac{e^{\beta_0+\beta_1 X}}{1+e^{\beta_0+\beta_1 X}}$
with e Euler's number (2.7182…), known from mathematics.

The formula can be rewritten into: $log(\frac{P(X)}{1-P(X)}) = \beta_0 + \beta_1 X$

The expression between the brackets has the form $\frac{probability}{1-probability}$ which is known as an odds ratio.[1] A logistic regression model is also known as a log-odds-model or logit-model.

If this model is applied, the parameters of the model have to be estimated based on the observations in the training data. For this the so called maximum likelihood criterion is used.

Maximum Likelihood Criterion

Based on n observations for the X-variables, a logit model estimates the probabilities Y belongs to class 1, i.e. P(Y=1 | X).
Assume we have 5 observations, (X1, 1), (X2, 1), (X3, 1), (X4, 0), (X5, 0). Model probabilities are p1, p2, p3, p4, p5. For a model to be a good model, p1,

[1] if the probability of a particular event equals 1/6, then the odds ratio equals 1/5 or 1 against 5; the probability that the event will not occur is 5 times higher than that it will occur. If for a particular event the odd ratio equals 4 (4 against 1), then the probability the event happens is 4/5 (80%).

id	diagnosis	radius_mean	taxture_mean	perimeter_mean	area
842302	M	18	10	123	
842517	M	21	18	133	
84300903	M	20	21	130	
84348301	M	11	20	78	
84358402	M	20	14	135	
843786	M	12	16	83	

p2 and p3 should be close to 1 and p4 and p5 close to 0.
The maximum likelihood criterion for choosing the best model is:
choose the model which maximizes p1 x p2 x p3 x (1-p4) x (1-p5).
This criterion can be easily generalized.

4.1.2.1 Example Logistic Regression Model

The example described in this section makes use of the Breast Cancer Wisconson Data Set, which can be found here.

```
library(tidyverse)
library(caret)
library(GGally) #vanwege ggpairs() plot
library(flextable)

df_read <- read_csv("Data/wbcd.csv")
df_read$id <- as.character(df_read$id)
```

Table
First Six Rows and Six Columns of the WBCD Data Set

```
flextable(head(df_read)[,1:6]) %>% autofit()
```

The first column is a unique identifier, the second column - diagnosis - is the target variable. Thw data set contains 30 X-variables. In this example only the first six X-variables are taken into account.

Figure
Summary of the WBCD Data Set

```
df <- df_read[,2:8]
summary(df)
```

```
  diagnosis          radius_mean      taxture_mean   perimeter_mean
```

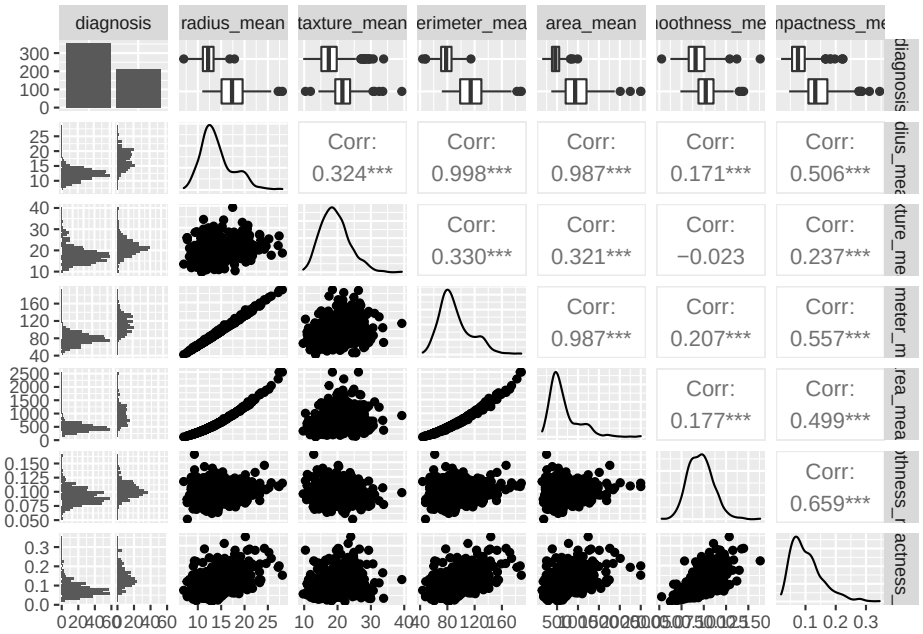
```
 Length:569         Min.   : 6.981   Min.   : 9.71   Min.   : 43.79
 Class :character   1st Qu.:11.700   1st Qu.:16.17   1st Qu.: 75.17
 Mode  :character   Median :13.370   Median :18.84   Median : 86.24
                    Mean   :14.127   Mean   :19.29   Mean   : 91.97
                    3rd Qu.:15.780   3rd Qu.:21.80   3rd Qu.:104.10
                    Max.   :28.110   Max.   :39.28   Max.   :188.50
  area_mean      smoothness_mean   compactness_mean
 Min.   : 143.5   Min.   :0.05263   Min.   :0.01938
 1st Qu.: 420.3   1st Qu.:0.08637   1st Qu.:0.06492
 Median : 551.1   Median :0.09587   Median :0.09263
 Mean   : 654.9   Mean   :0.09636   Mean   :0.10434
 3rd Qu.: 782.7   3rd Qu.:0.10530   3rd Qu.:0.13040
 Max.   :2501.0   Max.   :0.16340   Max.   :0.34540
```

Note. Only 6 of the 30 available X-variables are used in the example and summarized in this overview.

The diagnosis-variable is transformed into a factor variable. The GGally::ggpairs() function is used to examine the correlation between the different variables.

Figure
Correlation Plot Matrix

```
df$diagnosis <- factor(df$diagnosis,
                       levels=c("B", "M"),
                       labels=c("NEG", "POS"))
ggpairs(df)
```



The area_mean variable seems to make a good candidate to distinguish diagnosis M from diagnosis B.
The first logistic regression model usus this variable as the only predictor.

```
logreg_model1 <- glm(diagnosis~area_mean, data = df,
                     family = "binomial")
preds_logreg1 <- predict(logreg_model1, type = 'response')

threshold <- .5
preds_logreg1_diag <- ifelse(preds_logreg1>.5, "POS", "NEG") %>%
  factor()

confusionMatrix(preds_logreg1_diag, df$diagnosis,
                positive = "POS")
```

```
Confusion Matrix and Statistics

          Reference
Prediction NEG POS
       NEG 337  46
       POS  20 166

               Accuracy : 0.884
                 95% CI : (0.8548, 0.9091)
    No Information Rate : 0.6274
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.7456

 Mcnemar's Test P-Value : 0.002089

            Sensitivity : 0.7830
            Specificity : 0.9440
         Pos Pred Value : 0.8925
         Neg Pred Value : 0.8799
             Prevalence : 0.3726
         Detection Rate : 0.2917
   Detection Prevalence : 0.3269
      Balanced Accuracy : 0.8635

       'Positive' Class : POS
```

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --
```

```
## v ggplot2 3.3.3     v purrr   0.3.4
## v tibble  3.0.4     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 ---------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(rpart)
library(rpart.plot)
library(caret)
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

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

```
library(flextable)
```

```
##
## Attaching package: 'flextable'
```

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

Chapter 5

Classification Decision Trees

Decision Trees are probably the most basic models for classification problems. Many websites with an introduction to Decision Trees can be found on the internet, see for instance this website.

5.1 Splitting Criteria

The question what the best choice for a node, i.e. a variable and a splitting criterion, requires a metric to measure how good a possible split is. Commonly choices are (1) Information Gain and (2) Gini Impurity. Another possibility, which was widely used in the past, uses a Chi-Square Criterion.
The concepts are discussed in the next sub sections.

5.1.1 Measuring Information Gain

See https://victorzhou.com/blog/information-gain/.

5.1.2 Gini Impurity

See https://victorzhou.com/blog/gini-impurity/.

	Age	Gender	Polyuria	Polydipsia	sudden weight loss	weakness	Polypha
	40	Male	No	Yes	No	Yes	No
	58	Male	No	No	No	Yes	No
	41	Male	Yes	No	No	Yes	Yes
	45	Male	No	No	Yes	Yes	Yes
	60	Male	Yes	Yes	Yes	Yes	Yes
	55	Male	Yes	Yes	No	Yes	Yes

5.1.3 Chi-Square Criterion

Probably the oldest splitting criterion used when generating a decision tree makes uses of Chi-Square statitics to find out which of the features is strongest correlated to the target function.
However it is no longer used that often, and is beyond the scope of this book.

5.2 Example: Predicting Diabetes

```
df <- read_csv("data/uci_diabetes/diabetes_data_upload.csv")
```

Predicting diabetes
Data set: example from UCI website

The data set consists of 520 observations on 17 features. The target variable is the Class variable which can take on two values, Positive and Negative. All but one of the features are binary. The non binary feature is the Age feature.

Table 1
First Six Observations of the Diabetes Data Set

```
flextable(head(df)) %>%
  fontsize(size = 7, part = "all")
```

Table 2
Summary of the data

```
summary(df)
##       Age           Gender            Polyuria          Polydipsia
##  Min.   :16.00   Length:520         Length:520         Length:520
##  1st Qu.:39.00   Class :character   Class :character   Class :character
##  Median :47.50   Mode  :character   Mode  :character   Mode  :character
```

```
##  Mean   :48.03
##  3rd Qu.:57.00
##  Max.   :90.00
##  sudden weight loss   weakness          Polyphagia        Genital thrush
##  Length:520         Length:520         Length:520         Length:520
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##  visual blurring      Itching           Irritability       delayed healing
##  Length:520         Length:520         Length:520         Length:520
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##  partial paresis    muscle stiffness     Alopecia           Obesity
##  Length:520         Length:520         Length:520         Length:520
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##     class
##  Length:520
##  Class :character
##  Mode  :character
##
##
##
```

Data cleaning and Preprocessing

Correct variable names (no spaces).
Transform class variable in a factor variable with two levels, POS and NEG.

```
df_prep <- df
names(df_prep) <- make.names(names(df_prep)) %>%
  toupper() %>% str_replace_all("[\\.]", "_")

#transform CLASS variable into type factor variable
df_prep$CLASS <- factor(df_prep$CLASS, levels = c("Positive", "Negative"),
                        labels = c("POS", "NEG"))
```

5.2.0.1 Creating a Decision Tree Model using Rpart

As a first step, split the data set in a training and a test set; 70% in training, 30% in test set (other choices are possible as well). The caret::createDataPartition() function, can be used to create a partition such that the proportion Positives and Negatives are the same in the training and in the test set, a so called stratified sample.

```
#split in training end test set
set.seed(20210309)
train <- createDataPartition(df_prep$CLASS, p=0.70, list=FALSE)
df_train <- df_prep[train,]
```

```
## Warning: The `i` argument of ``[`()` can't be a matrix as of tibble 3.0.0.
## Convert to a vector.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```
df_test <- df_prep[-train,]
```

Use rpart::rpart() to create a decision tree.
This funciton uses Gini as default to measure impurity and information gain as alternative option.
See https://www.rdocumentation.org/packages/rpart/versions/4.1-15/topics/rpart

In this example, Accuracy is used as metric to assess the decision tree.

Create decision tree.

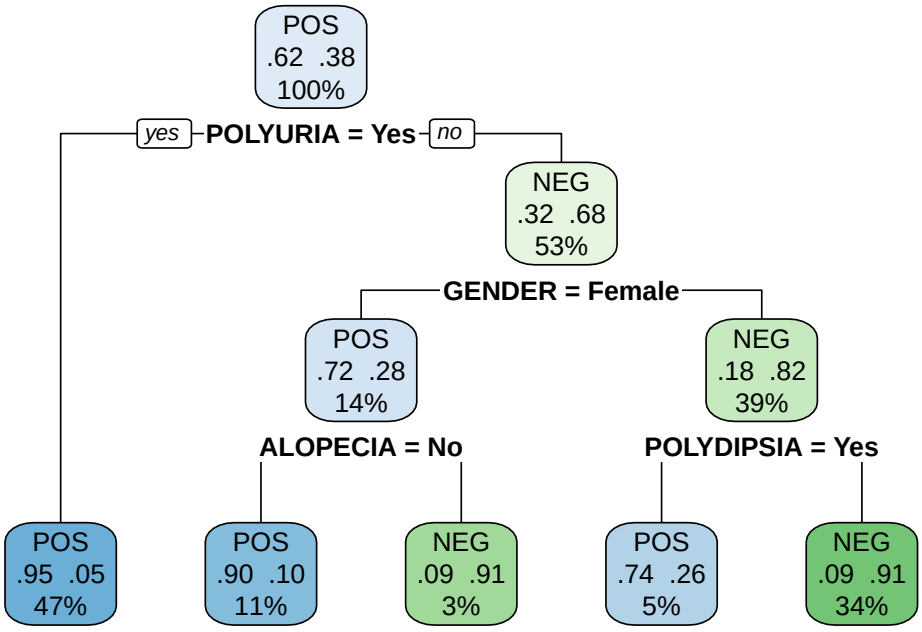
```
tree_01 <- rpart(CLASS~., data = df_train)

#other option
#tree_01 <- rpart(CLASS~., data = df_train,
#                 parms = list(split = "gini"))
```

Plot the decision tree using rpart.plot::rpart.plot() function.

```
library(rpart.plot)
```

```
## Loading required package: rpart
```

```
rpart.plot(tree_01, extra=104)
```




```
#with extra=104 the probs for both classes are printed in the nodes
```

This is the default tree plot made bij the rpart.plot() function. Each node shows (1) the predicted class, (2) the predicted probability of NEG and (3) the percentage of observations in the node. It is possible to change the lay-out of the plots and/or to show other information in the nodes.
For more information about the rpart.plot() function, see the help function and the vignette.
Assessing the decision tree model {-}

First assess the model on the training data:

- use predict() function to make predictions with the tree_01 model
- construct a confusion matrix
- calculate model Accuracy

Table 1

Confusion Matrrix Tree Model on Training Data

```
preds_tree01 <- predict(tree_01, newdata = df_train, type = 'class')
cm <- table(preds_tree01, df_train$CLASS)
cm
```

```
preds_tree01 POS NEG
         POS 212  18
         NEG  12 122
```

The Accuracy on the training data equals 0.918. This is a very high Accuracy, but the Accuracy on the test set is more interesting.

Table 2

Confusion Matrrix Tree Model on Test Data

```
preds_tree01 <- predict(tree_01, newdata = df_test, type = 'class')
cm_test <- table(preds_tree01, df_test$CLASS)
cm_test
```

```
preds_tree01 POS NEG
         POS  92  11
         NEG   4  49
```

The Accuracy on the test data equals 0.904. Although it is a bit lower than the Accuracy on the training data, it is still a high Accuracy.

It seems that the decision tree model performs quit well on this dataset.

The Accuracy on the test data is the estimate for the performance of the model outside the data used to construct the model. This estimates highly depends on the splitting in a training and a test set. Another split could lead to a different estimate. That's why repeating this procedure and using the average of the estimates for the Accuracy as the estimate for the out-of-sample Accuracy is recommended. Cross Validation or Bootstrapping are other options to come to a good estimate for the out-of-sample performance of the model.
The R caret package has built-in possibilities to use one of these techniques to assess the performance of a ML model.

5.2.0.2 Creating a Decision Tree Model using Rpart within caret package

In this section the caret package is used to generate an rpart decision tree model.

The central function in the caret package is the train() function. It can be used to generate a wide variety of models. Actually the caret package is a package which includes a variety of packages to generate ML models. Besides that it provides preprocessing functions and options to use cross validation, bootstrapping, splitting data in training and test data and lots more.

First explore the use of rpart with default settings for most of the options provided.

Table 3
Decision Tree with rpart Method in Caret

```
set.seed(20210316)
```

```
tree_02 <- train(CLASS~., data = df_prep, method = "rpart")

tree_02
```

```
CART

520 samples
 16 predictor
  2 classes: 'POS', 'NEG'

No pre-processing
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 520, 520, 520, 520, 520, 520, ...
Resampling results across tuning parameters:

  cp     Accuracy   Kappa
  0.060  0.8661589  0.7171642
  0.125  0.8260738  0.6477010
  0.540  0.7351813  0.4029460

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was cp = 0.06.
```

Points of attention:

- splitting in training and test data is done by the functions within caret; caret uses 25 times repeated bootstrapping as default method to estimate the performance of the generated model
- caret uses a tuning parameter (cp, complexity parameter) to decide about tree depth; for more information see: ..... ; three different values are tested, for the final model the value with the best performance is used; so actually the total number of decision trees generated equals 75, 25 for each value of the tuning parameter

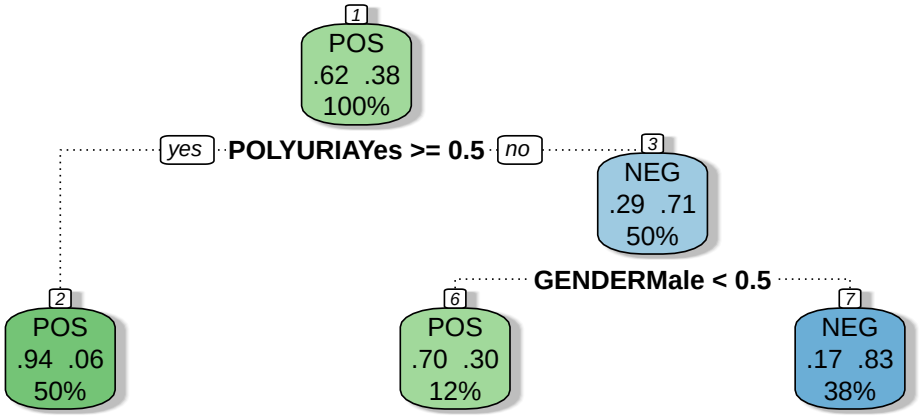
5.2.0.3 Plotting the decision tree

Trees created with the caret package cannot be plotted with the rpart.plot() function. Use rattle::fancyRpartPlot() instead.

```
library(rattle)
```

```
Loading required package: bitops

Rattle: A free graphical interface for data science with R.
Version 5.4.0 Copyright (c) 2006-2020 Togaware Pty Ltd.
Type 'rattle()' to shake, rattle, and roll your data.
```

```
#rpart.plot(tree_02) does not work here
fancyRpartPlot(tree_02$finalModel, sub = NULL)
```



The tree created with the defaults of caret is a pruned version of the tree created with the default settings of rpart. It is possible to create a less pruned tree in caret by tuning the hyper parameters. Rpart also offers options to tune the training process.
In general, it is good practice to start with the default settings and, after some experience, go deeper into model tuning.

5.2.0.4 Model tuning in caret

The strength of the caret package lies in the diversity of ML models that are built into this package and the possibilities to tune the model. To learn and understand the concepts behind an ML model, it is better to use a package developed for the specific ML model. E.g. use rpart (or tree) to better understand the concepts of buidling a decison tree model.

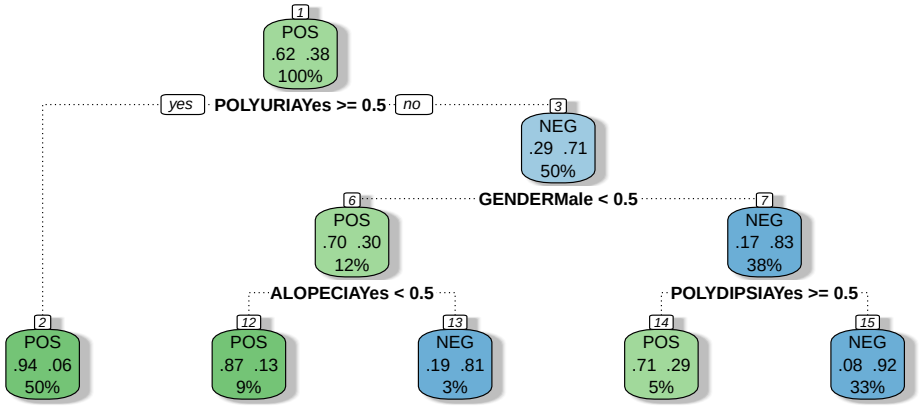
The model in the previous example in caret was built using bootstrapping. It is easy to use another technique instead, e.g. (repeated) Cross Validation. The output showed the cp parameter which was used to tune the model, by default caret tested three different cp values. Testing more cp values is a matter of using extra arguments in the train() function.
See the example below in which Cross Validation is used instead of Bootsttrapping and in which more values for cp are tested.

```
library(rattle)
trctrl <- trainControl(method = "cv", #cross validation
                       number = 10)   #10-fold cross validation
cp_grid <- data.frame(cp = seq(0.02, .2, .02))

tree_03 <- train(CLASS~., data = df_prep, method = 'rpart',
```

```
                 trControl = trctrl,
                 tuneGrid = cp_grid)

tree_03
```

```
CART

520 samples
 16 predictor
  2 classes: 'POS', 'NEG'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 468, 468, 468, 468, 468, 468, ...
Resampling results across tuning parameters:

  cp    Accuracy   Kappa
  0.02  0.9134615  0.8150286
  0.04  0.9076923  0.8007938
  0.06  0.8788462  0.7404646
  0.08  0.8711538  0.7275207
  0.10  0.8711538  0.7275207
  0.12  0.8384615  0.6690638
  0.14  0.8230769  0.6477028
  0.16  0.8230769  0.6477028
  0.18  0.8230769  0.6477028
  0.20  0.8230769  0.6477028

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was cp = 0.02.
```

```
#library(rattle)
#rpart.plot(tree_03)
fancyRpartPlot(tree_03$finalModel, sub = NULL)
```




```
library(tidyverse)
library(openxlsx)
library(kableExtra)
library(flextable)
library(lubridate)
library(ISLR) #Smarket data set
library(class) #knn function
library(caret)
#options(scipen = 999)
```

Chapter 6

KNN Algorithm

The KNN, K Nearest Neighbours, algorithm is an algorithm that can be used for both unsupervised and supervised learning. It can be used in a regression and in a classification context.

6.1 KNN in Classification

The idea behind the KNN algorithm is simple. Suppose a binary classification problem, i.e. the dataset provides a couple of features and a binary target Y-variable. The dataset contains hostorical data.
To predict the value of the target variable for a new observation, based on the values of the features, the distance of the new observation to the observations in the data set is calculated. The K observatioons wtih the lowest distance are filtered. The majority Y-value for these K observations is used as the predicted Y-value for the new observation.

Before this algorithm can be used two important choices must be made: (1) which value of K to use and (2) which distance metric to use.
Besides the choice of a distance metric two important preprocessing steps have to be performed, (1) categorical variables must be transformed into dummy variables and (2) numeric variables must be standardized/ normalized. The first step is necessary, because calculating a distance requires numerical values. The second step is performed, because otherwise the units chosen largely determine the distances.
Different techniques can be used for normalizing the values of a numeric variable. Standardizing the values, i.e. transforming the values in the z-score is an option, z-score $= \frac{x-mean}{SD}$. Another option is standardizing the values using the formula below, $x_{norm} = \frac{max-x}{max-min}$. As in all classification problems, a choice about the metric to use to assess the model must be made.

	Age	Gender	Polyuria	Polydipsia	sudden weight loss	weakness	Polypha
	40	Male	No	Yes	No	Yes	No
	58	Male	No	No	No	Yes	No
	41	Male	Yes	No	No	Yes	Yes
	45	Male	No	No	Yes	Yes	Yes
	60	Male	Yes	Yes	Yes	Yes	Yes
	55	Male	Yes	Yes	No	Yes	Yes

6.1.1 Choosing appropriate K value

There is no general rule for the choice of the best K-value. It is most common to divide the data in a training an a test set, test a number of choices for K and check which choice leads to the best result.

6.1.2 Distance metrics

Commonly used distance metrics are (1) Eucledian distance and (2) Manhattan distance. Be aware that there are lots of other distance metrics as well.

Example 1

```
df <- read_csv("data/uci_diabetes/diabetes_data_upload.csv")
```

Predicting diabetes
Data set: example from UCI website

The data set consists of 520 observations on 17 features. The target variable is the Class variable which can take on two values, Positive and Negative. All but one of the features are binary. The non binary feature is the Age feature.

Table 1
First Six Observations of the Diabetes Data Set

```
flextable(head(df)) %>%
  fontsize(size = 7, part = "all")
```

Table 2
Summary of the data

```
summary(df)
##       Age           Gender            Polyuria          Polydipsia
##  Min.   :16.00   Length:520         Length:520         Length:520
##  1st Qu.:39.00   Class :character   Class :character   Class :character
##  Median :47.50   Mode  :character   Mode  :character   Mode  :character
##  Mean   :48.03
##  3rd Qu.:57.00
##  Max.   :90.00
##  sudden weight loss   weakness          Polyphagia        Genital thrush
##  Length:520         Length:520         Length:520         Length:520
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##  visual blurring      Itching           Irritability       delayed healing
##  Length:520         Length:520         Length:520         Length:520
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##  partial paresis    muscle stiffness     Alopecia           Obesity
##  Length:520         Length:520         Length:520         Length:520
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##     class
##  Length:520
##  Class :character
##  Mode  :character
##
##
##
```

Data cleaning and Preprocessing

Correct variable names (no spaces)
Preprocessing: normalize Age variable
Transform other variables into dummies
Transform class variable into a factor variable, this is a requirement for the caret::knn3() function

Split data set in training and test set; 70% in training, 30% in test set

```
names(df) <- make.names(names(df)) %>%
  toupper() %>% str_replace_all("[\\.]", "_")


df_prep <- df %>%
  mutate(AGE_NORM = (AGE - min(AGE))/(max(AGE)-min(AGE))) %>%
  select(AGE_NORM, everything(), -AGE) %>%
  mutate(GENDER = ifelse(GENDER=="Male", 1, 0))

#variables in column 3-16 are all variables which only take on "Yes" or "No" values; t
df_prep[,3:16] <- apply(df_prep[,3:16], 2, function(x) ifelse(x=="Yes", 1, 0))

#transform CLASS variable into type factor variable
df_prep$CLASS <- factor(df_prep$CLASS, levels = c("Positive", "Negative"),
                        labels = c("POS", "NEG"))

#split in training end test set
set.seed(20210309)
train <- sample(1:nrow(df), size = .7*nrow(df))
df_train <- df_prep[train,]
df_test <- df_prep[-train,]
```

Generate a knn-model. Choice to make beforehand: value of k; first attempt k = 5

```
#generate model using class::knn3() function
#forst attempt with k=5 as an arbitrary choice
knn_model_1 <- knn3(CLASS~., data = df_train, k = 5)
knn_model_1
```

```
5-nearest neighbor model
Training set outcome distribution:

POS NEG
234 130
```

Assess model
Choose metric to assess model: Accuracy
Use model to make predictions for test data
Calculate Accuracy

```
#make predcitions with the generated model
knn_model_1_preds <- predict(knn_model_1, newdata = df_test, type = 'class')

#construct a confusion matrix
cf_1 <- table(knn_model_1_preds, df_test$CLASS)
```

```
#calculate Accuracy
acc_1 <- (cf_1[1,1] + cf_1[2,2]) / sum(cf_1)
```

Table 3
Confusion Matrix knn_1 model on Test Data Set

```
cf_1
```

```
knn_model_1_preds POS NEG
              POS  77   5
              NEG   9  65
```

Accuracy = (77 + 65) / (77 + 5 +9 + 65) = 0.91.

confusionMatrix(knn_model_1_preds, df_test$CLASS)

#knn_model_1 <- train(class~., data = df_prep, method = 'knn') #knn_model_1

Example 1

Predicting credibility bank customers Data set: example from UCI website
The data set contains the following 23 variables as explanatory variables:

- X1: Amount of the given credit (NT dollar): it includes both the individual consumer credit and his/her family (supplementary) credit.
- X2: Gender (1 = male; 2 = female).
- X3: Education (1 = graduate school; 2 = university; 3 = high school; 4 = others).
- X4: Marital status (1 = married; 2 = single; 3 = others).
- X5: Age (year).
- X6 - X11: History of past payment. We tracked the past monthly payment records (from April to September, 2005) as follows: X6 = the repayment status in September, 2005; X7 = the repayment status in August, 2005; . . .;X11 = the repayment status in April, 2005. The measurement scale for the repayment status is: -1 = pay duly; 1 = payment delay for one month; 2 = payment delay for two months; . . .; 8 = payment delay for eight months; 9 = payment delay for nine months and above.
- X12-X17: Amount of bill statement (NT dollar). X12 = amount of bill statement in September, 2005; X13 = amount of bill statement in August, 2005; . . .; X17 = amount of bill statement in April, 2005.
- X18-X23: Amount of previous payment (NT dollar). X18 = amount paid in September, 2005; X19 = amount paid in August, 2005; . . .;X23 = amount paid in April, 2005.

There is one binary variable, default payment (Yes = 1, No = 0), as the response variable.

```
#https://archive.ics.uci.edu/ml/machine-learning-databases/00350/

data <- readxl::read_xls("data/uci_bank/default of credit card clients.xls",
                         col_names = TRUE)
data <- data[-1,]
```

6.1.2.1 Example 1: Predicting stockmarket going up or down

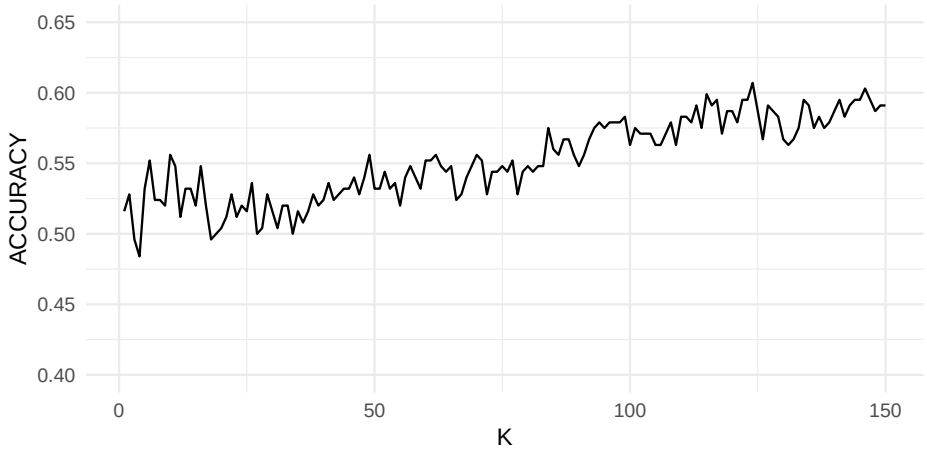
Dataset: ISLR:: Smarket.
See ?ISLR::Smarket for more information.
Responsive variable: Direction; two possible values: "Up", "Down".
Predictors: Lag1, Lag2, Lag3, Lag4, Lag5.
Choices made:
- distance metric: eucledian distance
- k: comparing different values to find the optimal value
- model assessment metric: accuracy

```
df <- ISLR::Smarket
#data from 2001 to 2004 as training data
#data from 2005 as test data
df_train <- filter(df, Year < 2005) %>%
  select(Lag1, Lag2, Lag3, Lag4, Lag5, Direction)
df_test <- filter(df, Year == 2005) %>%
  select(Lag1, Lag2, Lag3, Lag4, Lag5, Direction)

# model 1: k = 1
model_k1 <- knn(train = df_train[, 1:5],
                test = df_test[, 1:5],
                cl = df_train[, 6],
                k = 1)
table(model_k1, df_test$Direction)
```

```
model_k1 Down Up
    Down   55 66
    Up     56 75
```

```
acc_k1 <- round(sum(model_k1 == df_test$Direction)/nrow(df_test), 1)

# accuracy for various k values

accuracies <- data.frame(K = 1:150, ACCURACY = NA)

for (k in 1:150) {
  model_knn <- knn(train = df_train[, 1:5],
```

```
                test = df_test[, 1:5],
                cl = df_train$Direction,
                k = k)
  accuracies[k, 2] <- round(sum(model_knn == df_test$Direction)/nrow(df_test), 3)
}

ggplot(accuracies, aes(x = K, y = ACCURACY)) +
  geom_line() +
  ylim(0.4, 0.65) +
  theme_minimal()
```



Same example using caret.

```
knn_model_caret <- train(Direction~.,
                         data = df[, c("Lag1", "Lag2", "Lag3", "Lag4", "Lag5", "Direction")],
                         method = "knn",
                         tuneGrid = data.frame(k = seq(1, 40, 2)))

preds_caret <- predict(knn_model_caret, newdata = df_test)
confusionMatrix(preds_caret, df_test$Direction)
```

```
Confusion Matrix and Statistics

          Reference
Prediction Down  Up
      Down   39  37
      Up     72 104

               Accuracy : 0.5675
                 95% CI : (0.5038, 0.6295)
    No Information Rate : 0.5595
    P-Value [Acc > NIR] : 0.425500
```

```
                 Kappa : 0.092

 Mcnemar's Test P-Value : 0.001128

            Sensitivity : 0.3514
            Specificity : 0.7376
         Pos Pred Value : 0.5132
         Neg Pred Value : 0.5909
             Prevalence : 0.4405
         Detection Rate : 0.1548
   Detection Prevalence : 0.3016
      Balanced Accuracy : 0.5445

       'Positive' Class : Down
```

```r
library(tidyverse)
library(openxlsx)
library(flextable)
library(lubridate)
library(caret)
library(tm)
library(e1071)
#options(scipen = 999)

df_train <- tibble(Y = c("POS", "POS","POS" ,"POS" ,"POS",
                         "POS", "NEG","NEG", "NEG", "NEG"),
                   X1 = c(1,1,1,1,0,0,1,0,0,0))

df_train_2 <- tibble(Y = c("POS", "POS","POS" ,"POS" ,"POS",
                         "POS", "NEG","NEG", "NEG", "NEG"),
                     X1 = c(1,1,1,1,0,0,1,0,0,0),
                     X2 = c(1,0,1,0,1,0,1,0,1,0))
```

Chapter 7

Bayes Classifier

In a classification setting, the prediction problem can generally be formulated as: given this observation for the predictor variables, which class has the highest probability of containing the observation.

Assume a classification problem with a target variable with two classes, POS and NEG, and one binary predictor variable X1. Given a training set with n observations, this training set can be used to calculate the conditional a posteriori probabilities
P(X1 = 0 | POS),
P(X1 = 1 | POS),
P(X1 = 0 | NEG) and
P(X1 = 1 | NEG).

Table 1
Simple Example Training Data Set

```
flextable(df_train) %>% align(align = "center", part="all")
```

In the example in Table 1,

P(X1 = 0 | Y=POS) = 2/6
P(X1 = 1 | Y=POS) = 4/6
P(X1 = 0 | Y=NEG) = 3/4
P(X1 = 1 | Y=NEG) = 1/4

Given a new observation of the predictor variable, it is intuitively clear what the best prediction is for the Y-variable, based on the information in the training set.
If X1=1, than it is more probable that the observation belongs to the POS class, because P(X1=1 | Y=POS) is higher than P(X1=1 | Y=NEG).

This way to decide uses P(X1=1 | Y=POS) and P(X1=1 | Y=NEG) to predict

Y	X1
POS	1
POS	1
POS	1
POS	1
POS	0
POS	0
NEG	1
NEG	0
NEG	0
NEG	0

the correct class.
In a formal sense these are not the correct probabilities to make this decision. What is actually needed to predict the class based on the observation X1=1 are the probabilities
P(Y=POS | X1=1), which is not the same as P(X1 = 1 | Y=POS), and
P(Y=NEG | X1=1).
Based on the information in Table 1 these probabilities are:
P(Y=POS | X1=1) = 4/5
P(Y=NEG | X1=1) = 1/5

In this simple example, with just one predictor, the needed probabilities can be easily calculated. But in real world problems there are far more predictors than one and the situation becomes more complicated.

The formal way to calculate P(Y=POS | X1=1) which can be generated for more complex problems is:

$$P(Y = POS|X1 = 1) = \frac{P(X1 = 1 \; and \; Y = POS)}{P(X1 = 1)} =$$

$$\frac{P(Y = POS) \; * \; P(X1 = 1|Y = POS)}{P(X1 = 1)} = \frac{6/10 * 4/6}{5/10} = \frac{4}{5}$$

For this example using this formula is not necessary, but in more complicated cases it is.

Table 2 Example with two Features

Y	X1	X2
POS	1	1
POS	1	0
POS	1	1
POS	1	0
POS	0	1
POS	0	0
NEG	1	1
NEG	0	0
NEG	0	1
NEG	0	0

```
flextable(df_train_2) %>% align(align = "center", part="all")
```

Assume for a new observations X1 = 0 and X2 = 1.
The probabilities to calculate to make a prediction for the Y-variable are:
P(Y = NEG | X1 = 0 and X2 = 1) and
P(Y = Pos | X1 = 0 and X2 = 1).

Using the same formule as in the first example:
$P(Y = NEG|X1 = 0\ and\ X2 = 1) = \frac{P(Y=NEG)*P(X1=0\ and\ X2=1|Y=NEG)}{P(X1=0\ and\ X2=1)}$

$P(Y = POS|X1 = 0\ and\ X2 = 1) = \frac{P(Y=POS)*P(X1=0\ and\ X2=1|Y=POS)}{P(X1=0\ and\ X2=1)}$

The denominator of both expressions are the same. The numerator for the first expression can be split up further assuming that X1 and X2 are two conditionally independent variables, which means that the value X1 takes on given a certain condition doesn't depend on the value X2 takes on given this condition and vice versa. Although in a real world problem this assumption would be violated most of the times, using this assumption to calculate the denominator leads to

P(Y = NEG) * P(X1 = 0 and X2 = 1 | Y = NEG) =
= P(Y = NEG) * P(X1 = 0 | Y = NEG) * P(X2 = 1 |Y = NEG)
Using the training data the a posteriori probabilities in this expressesion can easily be calculated.

P(Y = NEG) * P(X1 = 0 | Y = NEG) * P(X2 = 1 |Y = NEG) =
= 4/10 * 3/4 * 2/4

Based on the independency assumption, the numerator can be calculated as

follows:
P(X1 = 0 and X2 = 1) = P(X1 = 0) * P(X2 = 1) etc.

In a real world example the number of features is of course far more, but the principles are the same. The method is named NaiveBayes because it uses Baysian statistics, it is naive because it assumes that the features are mutually independent, which is really a naive assumption.

7.1 A real world example, textmining: sms spam detection

The Naive Bayes model can be used for spam detection. From a training collection of sms text messages, that are labeled Spam or Ham, a set of used words can be extracted.
After a couple of data cleaning operations, see below, it is counted how often words occur in the text messages in the training set. Words with a frequency equal or higher than a chosen threshold, e.g. five, are collected in a dictionary. Only these words are used to distinguish Spam from Ham.
As the next step, each word is converted into a binary variable with the length of the number of messages in the training set, with value 1 in cell j if the word occurs one or more times in the jth text message and 0 otherwise. These binary variables are mutated into factor variables which can be used to generate NaiveBayes model that distinguishes Spam from Ham.

This example uses a dataset from Kaggle.

Read the file with the sms messages as it is published om Kaggle and view the first six messages.

Table 3
Head of the SMS Messages Data Set

```
sms_raw <- read_csv("data/kaggle_sms/sms_spam.csv")

#transform sms_raw$type into a factor variable
sms_raw$type <- factor(sms_raw$type, levels = c("ham", "spam"),
                       labels=c("ham", "spam"))

flextable(head(sms_raw)) %>% autofit()
```

The R package tm (tm stands for text mining) comes with a couple of helpful functions for Text Mining. In order to use the text mining functions, the data to be investigated must be in a so-called Corpus of text documents.
The first step in this analysis is to convert the set of SMS messages into such a corpus. First the text messages are transformed into a vector source this is an R vector that interprets every element as a text document.

type	text
ham	Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there g
ham	Ok lar... Joking wif u oni...
spam	Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to re
ham	U dun say so early hor... U c already then say...
ham	Nah I don't think he goes to usf, he lives around here though
spam	FreeMsg Hey there darling it's been 3 week's now and no word back! I'd like some fun you

Table 4
First Five Elements of SMS message in a Vector Source

```
sms_vector_source <- VectorSource(sms_raw$text)
sms_vector_source[1:5]
```

```
[1] "Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine the
[2] "Ok lar... Joking wif u oni..."
[3] "Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to rece
[4] "U dun say so early hor... U c already then say..."
[5] "Nah I don't think he goes to usf, he lives around here though"
```

As a next step, create a Corpus with all the sms messages as text documents. This is in fact a list of lists, every list contains the text of the document and metadata about the document.

```
sms_corpus <- Corpus(sms_vector_source)
```

Now it is time to perform a couple of text cleaning preparation steps. The tm package has functions for these actions.
(1) Because it is assumed that capitalised letters will not be used to distinguish Ham from Spam, replace all uppercase letters by lowercase letters.
(2) Remove numbers from the text messages. If it is assumed that numbers in the messages can be helpful to diferentiate between Spam and Ham, a more advanced way to deal with numbers is required
(3) Lots of short words - like and, or, if, on, to etc. - are not useful to distinct Ham from Spam; they can be removed from the texts before generating a model.
(4) Remove punctuation.
(5) Remove unnecessary spaces.

Table 5
Example of Cleaned Text Message

```
#transform to lower case text messages
#the tm::tm_map function applies a function on a corpus object
```

```
corpus_clean <- tm_map(sms_corpus, tolower) %>%
#remove numbers
  tm_map(removeNumbers) %>%
#remove stopwords
  tm_map(removeWords, stopwords()) %>%
#remove punctuation
  tm_map(removePunctuation) %>%
#remove additional spaces
  tm_map(stripWhitespace)

corpus_clean[1]$content
```

```
[1] "go jurong point crazy available bugis n great world la e buffet cine got amore wat
```

Assuming that splitting the data in a training and a test set will be used to assess the Naive Bayes model, the next step is splitting the data, e.g. 70% in the training set and 30% in the test set.

```
set.seed(20210332)
train <- sample(1:length(corpus_clean),
                size = .7*length(corpus_clean),
                replace=FALSE)

sms_raw_train <- sms_raw[train,]
sms_raw_test <- sms_raw[-train,]

corpus_train <- corpus_clean[train]
corpus_test <- corpus_clean[-train]

sum(train)
```

```
[1] 10914129
```

The next step is to construct a Document Term Matrix for the training set, this is a matrix in which rows correspond to documents and columns correspond to the terms in the documents. The cells contain the number of times a term occurs in a document.

Table 6
Document Term Matrix

```
dtm_train <- DocumentTermMatrix(corpus_train)
inspect(dtm_train)
```

```
<<DocumentTermMatrix (documents: 3901, terms: 6529)>>
Non-/sparse entries: 29853/25439776
Sparsity           : 100%
Maximal term length: 40
```

```
Weighting          : term frequency (tf)
Sample             :
      Terms
Docs   call can free get got just know ltgt now will
  1219    0   0    0   0   0    0    0    1   0    0
  126     0   0    0   0   0    0    0    1   0    0
  1344    0   0    0   0   0    0    0    0   0    0
  2901    0   0    0   1   0    0    0    0   0    0
  3164    1   0    0   0   0    2    0    0   0    1
  3223    0   0    0   1   0    0    0    3   0    2
  3278    0   0    0   0   0    0    0   18   0    0
  3442    0   0    0   0   0    0    1    0   0    0
  3515    0   0    0   1   0    0    0    0   0   11
  797     0   1    0   0   0    0    0    1   0    0
```

To distinguish Ham from Spam not every word in the corpus are useful. Words must appear in a couple of messages to be useful. A choice must be made for the threshold of the number of messages in which a word appears to be used in the model, e.g. 5 times.
First construct a vector with words with a frequency of at least 5.

Table 8
First 10 Terms with Frequency at Least 5

```
frequent_terms_5 <- findFreqTerms(dtm_train, lowfreq=5)
frequent_terms_5[1:10]
```

```
 [1] "awaiting"   "call"       "collect"    "collection" "currently"
 [6] "just"       "message"    "exam"       "march"      "take"
```

The Naive Bayes model uses as features not the number of times a term appears in a message, but only whether a term appears in a message.
It is possible to construct a Binary DTM in which the cells indicate whether a document contains the term (cell value = 1) or not (cell value = 0). It is this Binary DTM that is used in the Naive Bayes model. The Binary DTM is constructed for the words with frequency at least.

Table 7
Binary Documemt Term Matrix for Training Data

```
#Binary DTM for training data
dtm_train_bin <- DocumentTermMatrix(
                      corpus_train,
                      control=list(weighting=weightBin,
                                   dictionary=frequent_terms_5))

#Binary DTM for test data; needed to asses the model
dtm_test_bin <- DocumentTermMatrix(
                      corpus_test,
```

```
                    control=list(weighting=weightBin,
                                 dictionary=frequent_terms_5))
inspect(dtm_train_bin)
```

```
<<DocumentTermMatrix (documents: 3901, terms: 1148)>>
Non-/sparse entries: 21934/4456414
Sparsity           : 100%
Maximal term length: 19
Weighting          : binary (bin)
Sample             :
      Terms
Docs   call can free get got just know like now will
  1344    0   0    0   0   0    0    0    0   0    0
  1442    0   1    0   0   0    0    1    0   0    0
  1725    0   0    0   0   1    0    0    1   0    0
  2446    0   0    0   0   0    0    1    0   0    0
  3164    1   0    0   0   0    1    0    0   0    1
  3167    1   0    0   1   0    0    0    0   0    0
  3223    0   0    0   1   0    0    0    1   0    1
  3442    0   0    0   0   0    0    1    1   0    0
  3515    0   0    0   1   0    0    0    1   0    1
  797     0   1    0   0   0    0    0    0   0    0
```

The columns in the Binary DTM must be transformed into factor variables to use them in a Naive Bayes model.
Then the Binare DTM is ready to generate a Naive Bayes model.

```
#first use as matrix() to convert DTM matrix from a list into a matrix
dtm_train_bin_matrix <- as.matrix(dtm_train_bin)
dtm_test_bin_matrix <- as.matrix(dtm_test_bin)

#convert the columns into factor
dtm_train_bin_matrix <- apply(dtm_train_bin_matrix, 2, factor)
dtm_test_bin_matrix <- apply(dtm_test_bin_matrix, 2, factor)

#generate model
nb_model <- naiveBayes(x=dtm_train_bin_matrix,
                       y=sms_raw_train$type)

summary(nb_model)
```

```
          Length Class  Mode
apriori      2   table  numeric
tables    1148   -none- list
levels       2   -none- character
isnumeric 1148   -none- logical
```

```
call         3   -none- call
```

Assessing the model:
(1) Use the model to make predictions on the test data
(2) Assess the model using a confusion matrix

```
preds <- predict(nb_model, dtm_test_bin_matrix)

cf <- table(preds, sms_raw_test$type)

cf
```

```
preds   ham spam
  ham  1445   32
  spam    7  189
```

As can be seen in the Confusion Matrix, the model makes a good distinction between Spam and Ham. Only 7 of the 1452 Ham messages (0.5%) are classified as Spam while 32 of the 221 Spam messages (14.5%) are classified as Ham.

The caret::confusionMatrix() function gives a lot of metrics which can be used to assess a classification model. Which metric is most applicable depends on the context of the problem in question.

```
confusionMatrix(preds, sms_raw_test$type)
```

```
Confusion Matrix and Statistics

          Reference
Prediction  ham spam
      ham  1445   32
      spam    7  189

               Accuracy : 0.9767
                 95% CI : (0.9683, 0.9834)
    No Information Rate : 0.8679
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.8932

 Mcnemar's Test P-Value : 0.0001215

            Sensitivity : 0.9952
            Specificity : 0.8552
         Pos Pred Value : 0.9783
         Neg Pred Value : 0.9643
             Prevalence : 0.8679
         Detection Rate : 0.8637
```

```
Detection Prevalence : 0.8828
   Balanced Accuracy : 0.9252

    'Positive' Class : ham
```