Classification

Alfonso R. Reyes

2019-09-18

Contents

Prerequisites

This is a *sample* book written in **Markdown**. You can use anything that Pandoc's Markdown supports, e.g., a math equation $a^2 + b^2 = c^2$.

The **bookdown** package can be installed from CRAN or Github:

```
install.packages("bookdown")
# or the development version
# devtools::install_github("rstudio/bookdown")
```

Remember each Rmd file contains one and only one chapter, and a chapter is defined by the first-level heading `#`.

To compile this example to PDF, you need XeLaTeX. You are recommended to install TinyTeX (which includes XeLaTeX): https://yihui.name/tinytex/.

Chapter 1

A gentle introduction to support vector machines using R

1.1 Introduction

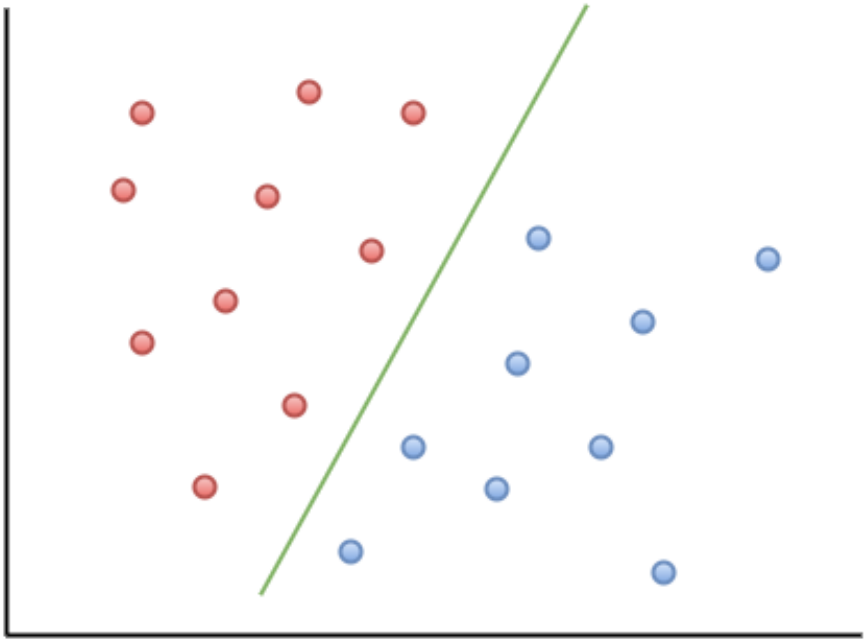
Source: https://eight2late.wordpress.com/2017/02/07/a-gentle-introduction-to-support-vector-machines-using-r/

Most machine learning algorithms involve minimising an error measure of some kind (this measure is often called an objective function or loss function). For example, the error measure in linear regression problems is the famous mean squared error – i.e. the averaged sum of the squared differences between the predicted and actual values. Like the mean squared error, most objective functions depend on all points in the training dataset. In this post, I describe the support vector machine (SVM) approach which focuses instead on finding the optimal separation boundary between datapoints that have different classifications. I'll elaborate on what this means in the next section.

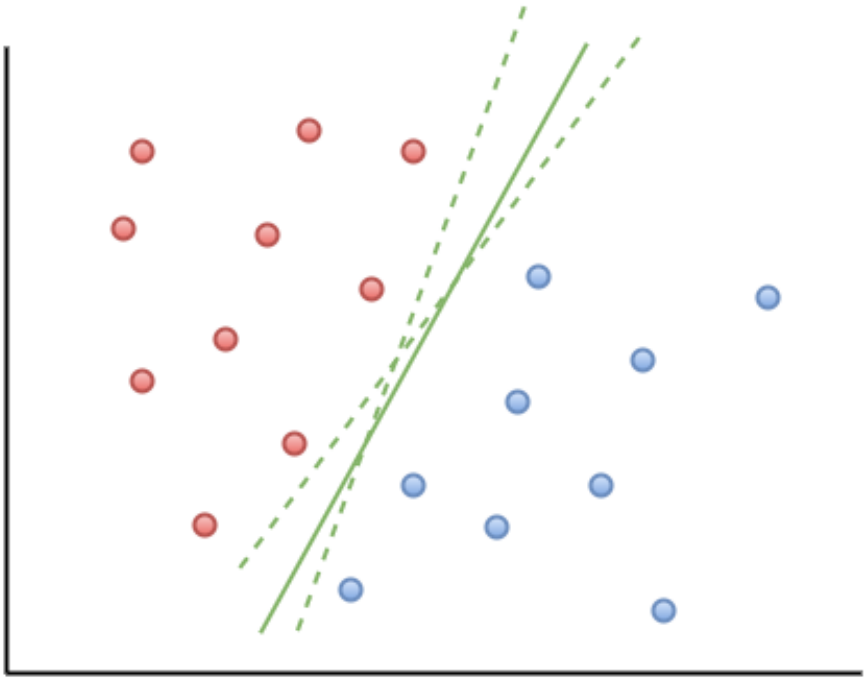
Here's the plan in brief. I'll begin with the rationale behind SVMs using a simple case of a binary (two class) dataset with a simple separation boundary (I'll clarify what "simple" means in a minute). Following that, I'll describe how this can be generalised to datasets with more complex boundaries. Finally, I'll work through a couple of examples in R, illustrating the principles behind SVMs. In line with the general philosophy of my "Gentle Introduction to Data Science Using R" series, the focus is on developing an intuitive understanding of the algorithm along with a practical demonstration of its use through a toy example.

1.2 The rationale

The basic idea behind SVMs is best illustrated by considering a simple case: a set of data points that belong to one of two classes, red and blue, as illustrated in figure 1 below. To make things simpler still, I have assumed that the boundary separating the two classes is a straight line, represented by the solid green line in the diagram. In the technical literature, such datasets are called linearly separable.



In the linearly separable case, there is usually a fair amount of freedom in the way a separating line can be drawn. Figure 2 illustrates this point: the two broken green lines are also valid separation boundaries. Indeed, because there is a non-zero distance between the two closest points between categories, there are an infinite number of possible separation lines. This, quite naturally, raises the question as to whether it is possible to choose a separation boundary that is optimal.



The short answer is, yes there is. One way to do this is to select a boundary line that maximises the margin, i.e. the distance between the separation boundary and the points that are closest to it. Such an optimal boundary is illustrated by the black brace in Figure 3. The really cool thing about this criterion is that the location of the separation boundary depends only on the points that are closest to it. This means, unlike other classification methods, the classifier does not depend on any other points in dataset. The directed

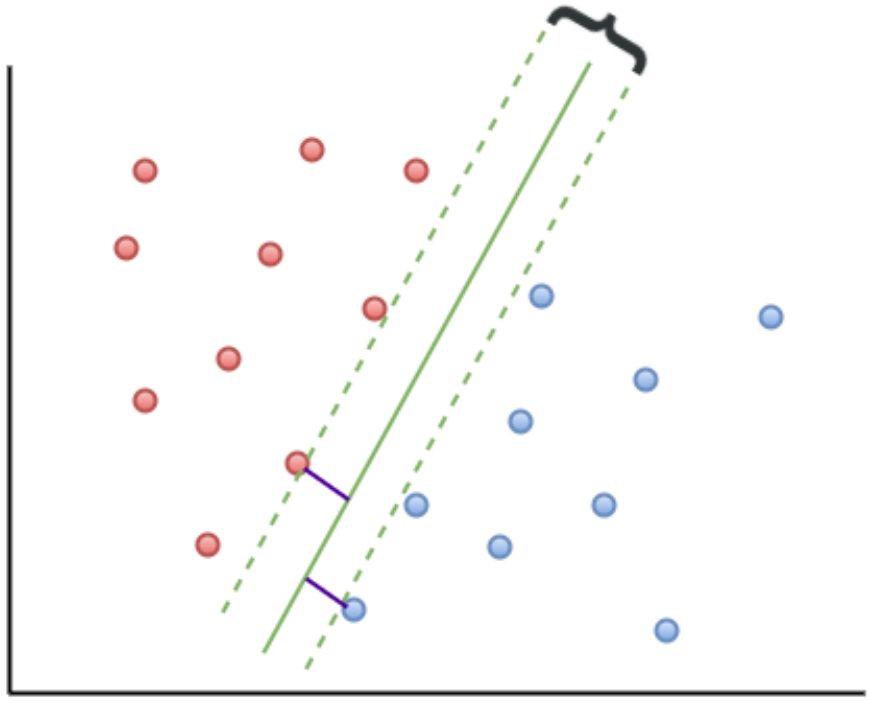


Figure 1.1: Figure 3

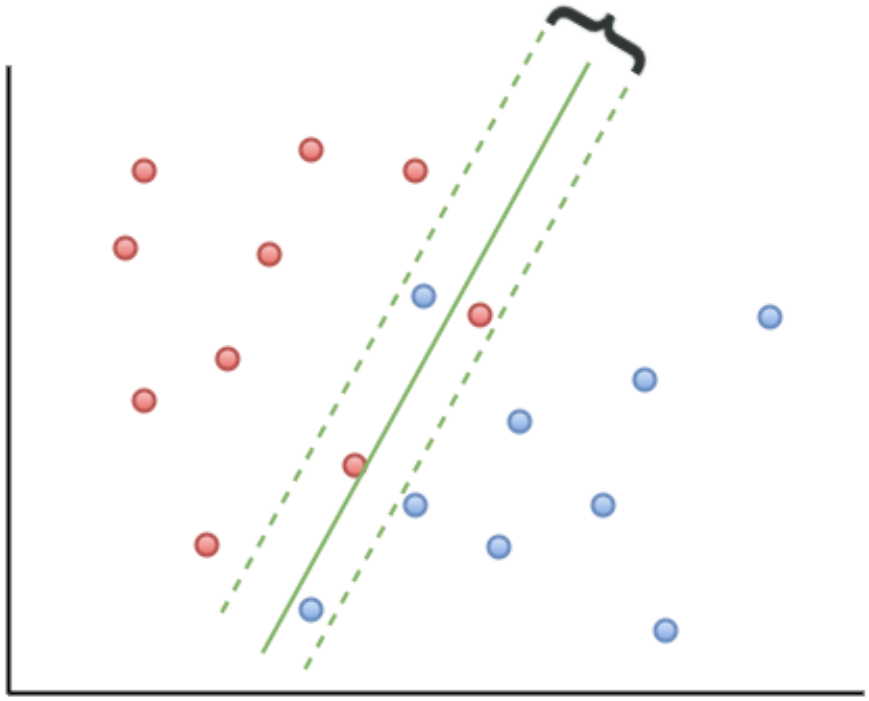
lines between the boundary and the closest points on either side are called support vectors (these are the solid black lines in figure 3). A direct implication of this is that the fewer the support vectors, the better the generalizability of the boundary.

Although the above sounds great, it is of limited practical value because real data sets are seldom (if ever) linearly separable.

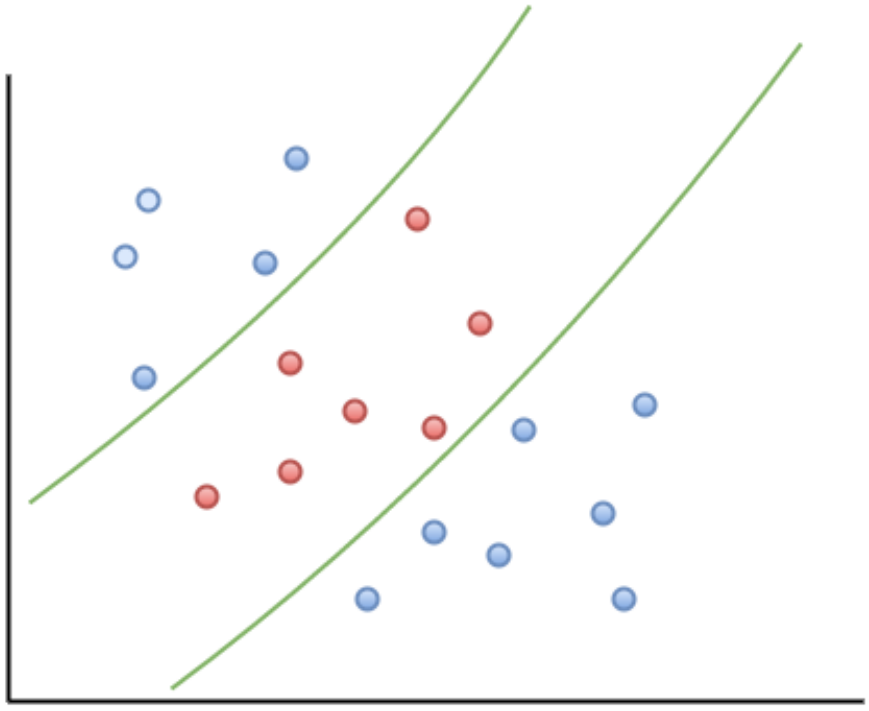
So, what can we do when dealing with real (i.e. non linearly separable) data sets?

A simple approach to tackle small deviations from linear separability is to allow a small number of points (those that are close to the boundary) to be misclassified. The number of possible misclassifications is governed by a free parameter C, which is called the cost. The cost is essentially the penalty associated with making an error: the higher the value of C, the less likely it is that the algorithm will misclassify a point.

This approach – which is called soft margin classification – is illustrated in Figure 4. Note the points on the wrong side of the separation boundary. We will demonstrate soft margin SVMs in the next section. (Note: At the risk of belabouring the obvious, the purely linearly separable case discussed in the previous para is simply is a special case of the soft margin classifier.)



Real life situations are much more complex and cannot be dealt with using soft margin classifiers. For example, as shown in Figure 5, one could have widely separated clusters of points that belong to the same classes. Such situations, which require the use of multiple (and nonlinear) boundaries, can sometimes be dealt with using a clever approach called the kernel trick.



1.3 The kernel trick

Recall that in the linearly separable (or soft margin) case, the SVM algorithm works by finding a separation boundary that maximises the margin, which is the distance between the boundary and the points closest to it. The distance here is the usual straight line distance between the boundary and the closest point(s). This is called the Euclidean distance in honour of the great geometer of antiquity. The point to note is that this process results in a separation boundary that is a straight line, which as Figure 5 illustrates, does not always work. In fact in most cases it won't.

So what can we do? To answer this question, we have to take a bit of a detour…

What if we were able to generalize the notion of distance in a way that generates nonlinear separation boundaries? It turns out that this is possible. To see how, one has to first understand how the notion of distance can be generalized.

The key properties that any measure of distance must satisfy are:

```
Non-negativity - a distance cannot be negative, a point that needs no further explanation I reckon
Symmetry - that is, the distance between point A and point B is the same as the distance between point
Identity- the distance between a point and itself is zero.
Triangle inequality - that is the sum of distances between point A and B and points B and C must be les
```

Any mathematical object that displays the above properties is akin to a distance. Such generalized distances are called metrics and the mathematical space in which they live is called a metric space. Metrics are defined using special mathematical functions designed to satisfy the above conditions. These functions are known as kernels.

The essence of the kernel trick lies in mapping the classification problem to a metric space in which the problem is rendered separable via a separation boundary that is simple in the new space, but complex – as it has to be – in the original one. Generally, the transformed space has a higher dimensionality, with each of the dimensions being (possibly complex) combinations of the original problem variables. However, this is not necessarily a problem because in practice one doesn't actually mess around with transformations, one just tries different kernels (the transformation being implicit in the kernel) and sees which one does the job. The check is simple: we simply test the predictions resulting from using different kernels against a held out subset of the data (as one would for any machine learning algorithm).

It turns out that a particular function – called the radial basis function kernel (RBF kernel) – is very effective in many cases. The RBF kernel is essentially a Gaussian (or Normal) function with the Euclidean distance between pairs of points as the variable (see equation 1 below). The basic rationale behind the RBF kernel is that it creates separation boundaries that it tends to classify points close together (in the Euclidean sense) in the original space in the same way. This is reflected in the fact that the kernel decays (i.e. drops off to zero) as the Euclidean distance between points increases.

The rate at which a kernel decays is governed by the parameter γ – the higher the value of γ, the more rapid the decay. This serves to illustrate that the RBF kernel is extremely flexible….but the flexibility comes at a price – the danger of overfitting for large values of γ . One should choose appropriate values of C and γ so as to ensure that the resulting kernel represents the best possible balance between flexibility and accuracy. We'll discuss how this is done in practice later in this article.

Finally, though it is probably obvious, it is worth mentioning that the separation boundaries for arbitrary kernels are also defined through support vectors as in Figure 3. To reiterate a point made earlier, this means that a solution that has fewer support vectors is likely to be more robust than one with many. Why? Because the data points defining support vectors are ones that are most sensitive to noise- therefore the fewer, the better.

There are many other types of kernels, each with their own pros and cons. However, I'll leave these for adventurous readers to explore by themselves. Finally, for a much more detailed….and dare I say, better… explanation of the kernel trick, I highly recommend this article by Eric Kim.

1.4 Support vector machines in R

In this demo we'll use the svm interface that is implemented in the `e1071` R package. This interface provides R programmers access to the comprehensive `libsvm` library written by Chang and Lin. I'll use two toy datasets: the famous iris dataset available with the base R package and the sonar dataset from the mlbench package. I won't describe details of the datasets as they are discussed at length in the documentation that I have linked to. However, it is worth mentioning the reasons why I chose these datasets:

As mentioned earlier, no real life dataset is linearly separable, but the iris dataset is almost so. Consequently, it is a good illustration of using linear SVMs. Although one almost never uses these in practice, I have illustrated their use primarily for pedagogical reasons. The sonar dataset is a good illustration of the benefits of using RBF kernels in cases where the dataset is hard to visualise (60 variables in this case!). In general, one would almost always use RBF (or other nonlinear) kernels in practice.

With that said, let's get right to it. I assume you have R and RStudio installed. For instructions on how to do this, have a look at the first article in this series. The processing preliminaries – loading libraries, data and creating training and test datasets are much the same as in my previous articles so I won't dwell on these here. For completeness, however, I'll list all the code so you can run it directly in R or R studio (a complete listing of the code can be found here):

1.5 SVM on `iris` dataset

1.5.1 Training and test datasets

```
#load required library
library(e1071)

#load built-in iris dataset
data(iris)

#set seed to ensure reproducible results
set.seed(42)

#split into training and test sets
iris[, "train"] <- ifelse(runif(nrow(iris)) < 0.8, 1, 0)

#separate training and test sets
trainset <- iris[iris$train == 1,]
testset <- iris[iris$train == 0,]

#get column index of train flag
trainColNum <- grep("train", names(trainset))

#remove train flag column from train and test sets
trainset <- trainset[,-trainColNum]
testset <- testset[,-trainColNum]

dim(trainset)
#> [1] 115   5
dim(testset)
#> [1] 35  5
```

1.5.2 Build the SVM model

```
#get column index of predicted variable in dataset
typeColNum <- grep("Species", names(iris))

#build model - linear kernel and C-classification (soft margin) with default cost (C=1)
svm_model <- svm(Species~ ., data = trainset,
                 method = "C-classification",
                 kernel = "linear")
svm_model
#>
#> Call:
#> svm(formula = Species ~ ., data = trainset, method = "C-classification",
#>     kernel = "linear")
#>
#>
#> Parameters:
#>    SVM-Type:  C-classification
#>  SVM-Kernel:  linear
#>        cost:  1
#>       gamma:  0.25
#>
#> Number of Support Vectors:  24
```

The output from the SVM model show that there are 24 support vectors. If desired, these can be examined using the SV variable in the model – i.e via svm_model$SV.

1.5.3 Support Vectors

```
# support vectors
svm_model$SV
#>     Sepal.Length Sepal.Width Petal.Length Petal.Width
#> 19       -0.2564      1.7668       -1.323      -1.305
#> 42       -1.7006     -1.7045       -1.559      -1.305
#> 45       -0.9785      1.7668       -1.205      -1.171
#> 53        1.1878      0.1469        0.568       0.309
#> 55        0.7064     -0.5474        0.390       0.309
#> 57        0.4657      0.6097        0.450       0.443
#> 58       -1.2192     -1.4730       -0.378      -0.364
#> 69        0.3453     -1.9359        0.331       0.309
#> 71       -0.0157      0.3783        0.509       0.712
#> 73        0.4657     -1.2416        0.568       0.309
#> 78        0.9471     -0.0845        0.627       0.578
#> 84        0.1046     -0.7788        0.686       0.443
#> 85       -0.6174     -0.0845        0.331       0.309
#> 86        0.1046      0.8412        0.331       0.443
#> 99       -0.9785     -1.2416       -0.555      -0.229
#> 107      -1.2192     -1.2416        0.331       0.578
#> 111       0.7064      0.3783        0.686       0.981
#> 117       0.7064     -0.0845        0.922       0.712
#> 124       0.4657     -0.7788        0.568       0.712
#> 130       1.5488     -0.0845        1.099       0.443
```

```
#> 138       0.5860      0.1469       0.922       0.712
#> 139       0.1046     -0.0845       0.509       0.712
#> 147       0.4657     -1.2416       0.627       0.847
#> 150      -0.0157     -0.0845       0.686       0.712
```

The test prediction accuracy indicates that the linear performs quite well on this dataset, confirming that it is indeed near linearly separable. To check performance by class, one can create a confusion matrix as described in my post on random forests. I'll leave this as an exercise for you. Another point is that we have used a soft-margin classification scheme with a cost C=1. You can experiment with this by explicitly changing the value of C. Again, I'll leave this for you an exercise.

1.5.4 Predictions on training model

```
# training set predictions
pred_train <- predict(svm_model, trainset)
mean(pred_train == trainset$Species)
#> [1] 0.983
# [1] 0.9826087
```

1.5.5 Predictions on test model

```
# test set predictions
pred_test <-predict(svm_model, testset)
mean(pred_test == testset$Species)
#> [1] 0.914
# [1] 0.9142857
```

1.5.6 Confusion matrix and Accuracy

```
# confusion matrix
cm <- table(pred_test, testset$Species)
cm
#>
#> pred_test    setosa versicolor virginica
#>   setosa         18          0         0
#>   versicolor      0          5         3
#>   virginica       0          0         9
```

```
# accuracy
sum(diag(cm)) / sum(cm)
#> [1] 0.914
```

1.6 SVM with Radial Basis Function kernel. Linear

1.6.1 Training and test sets

```
#load required library (assuming e1071 is already loaded)
library(mlbench)

#load Sonar dataset
data(Sonar)
#set seed to ensure reproducible results
set.seed(42)
#split into training and test sets
Sonar[, "train"] <- ifelse(runif(nrow(Sonar))<0.8,1,0)

#separate training and test sets
trainset <- Sonar[Sonar$train==1,]
testset <- Sonar[Sonar$train==0,]

#get column index of train flag
trainColNum <- grep("train",names(trainset))
#remove train flag column from train and test sets
trainset <- trainset[,-trainColNum]
testset <- testset[,-trainColNum]

#get column index of predicted variable in dataset
typeColNum <- grep("Class",names(Sonar))
```

1.6.2 Predictions on Training model

```
#build model - linear kernel and C-classification with default cost (C=1)
svm_model <- svm(Class~ ., data=trainset,
                 method="C-classification",
                 kernel="linear")

#training set predictions
pred_train <-predict(svm_model,trainset)
mean(pred_train==trainset$Class)
#> [1] 0.97
```

1.6.3 Predictions on test model

```
#test set predictions
pred_test <-predict(svm_model,testset)
mean(pred_test==testset$Class)
#> [1] 0.605
```

I'll leave you to examine the contents of the model. The important point to note here is that the performance of the model with the test set is quite dismal compared to the previous case. This simply indicates that the linear kernel is not appropriate here. Let's take a look at what happens if we use the RBF kernel with default values for the parameters:

1.7 SVM with Radial Basis Function kernel. Non-linear

1.7.1 Predictions on training model

```
#build model: radial kernel, default params
svm_model <- svm(Class~ ., data=trainset,
                 method="C-classification",
                 kernel="radial")
# print params
svm_model$cost
#> [1] 1
svm_model$gamma
#> [1] 0.0167

#training set predictions
pred_train <-predict(svm_model,trainset)
mean(pred_train==trainset$Class)
#> [1] 0.988
```

1.7.2 Predictions on test model

```
#test set predictions
pred_test <-predict(svm_model,testset)
mean(pred_test==testset$Class)
#> [1] 0.767
```

That's a pretty decent improvement from the linear kernel. Let's see if we can do better by doing some parameter tuning. To do this we first invoke tune.svm and use the parameters it gives us in the call to svm:

1.7.3 Tuning of parameters

```
# find optimal parameters in a specified range
tune_out <- tune.svm(x = trainset[,-typeColNum],
                     y = trainset[, typeColNum],
                     gamma = 10^(-3:3),
                     cost = c(0.01, 0.1, 1, 10, 100, 1000),
                     kernel = "radial")

#print best values of cost and gamma
tune_out$best.parameters$cost
#> [1] 10
tune_out$best.parameters$gamma
#> [1] 0.01

#build model
svm_model <- svm(Class~ ., data = trainset,
                 method = "C-classification",
                 kernel = "radial",
                 cost = tune_out$best.parameters$cost,
                 gamma = tune_out$best.parameters$gamma)
```

1.7.4 Prediction on training model with new parameters

```
# training set predictions
pred_train <-predict(svm_model,trainset)
mean(pred_train==trainset$Class)
#> [1] 1
```

1.7.5 Prediction on test model with new parameters

```
# test set predictions
pred_test <-predict(svm_model,testset)
mean(pred_test==testset$Class)
#> [1] 0.814
```

Which is fairly decent improvement on the un-optimised case.

1.8 Wrapping up

This bring us to the end of this introductory exploration of SVMs in R. To recap, the distinguishing feature of SVMs in contrast to most other techniques is that they attempt to construct optimal separation boundaries between different categories.

SVMs are quite versatile and have been applied to a wide variety of domains ranging from chemistry to pattern recognition. They are best used in binary classification scenarios. This brings up a question as to where SVMs are to be preferred to other binary classification techniques such as logistic regression. The honest response is, "it depends" – but here are some points to keep in mind when choosing between the two. A general point to keep in mind is that SVM algorithms tend to be expensive both in terms of memory and computation, issues that can start to hurt as the size of the dataset increases.

Given all the above caveats and considerations, the best way to figure out whether an SVM approach will work for your problem may be to do what most machine learning practitioners do: try it out!

Chapter 2

Classification with SVM. Social Network dataset

2.1 Introduction

Source: https://www.geeksforgeeks.org/classifying-data-using-support-vector-machinessvms-in-r/

2.2 Data Operations

2.2.1 Load libraries

```
# load packages
library(dplyr)
library(caTools)
library(e1071)
library(ElemStatLearn)
```

2.2.2 Importing dataset

```
# Importing the dataset
dataset = read.csv(file.path(data_raw_dir, 'Social_Network_Ads.csv'))
dplyr::glimpse(dataset)
#> Observations: 400
#> Variables: 5
#> $ User.ID         <int> 15624510, 15810944, 15668575, 15603246, 158040...
#> $ Gender          <fct> Male, Male, Female, Female, Male, Male, Female...
#> $ Age             <int> 19, 35, 26, 27, 19, 27, 27, 32, 25, 35, 26, 26...
#> $ EstimatedSalary <int> 19000, 20000, 43000, 57000, 76000, 58000, 8400...
#> $ Purchased       <int> 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0...
```

```
tibble::as_tibble(dataset)
#> # A tibble: 400 x 5
#>    User.ID Gender   Age EstimatedSalary Purchased
#>      <int> <fct>  <int>           <int>     <int>
```

```
#> 1 15624510 Male      19           19000         0
#> 2 15810944 Male      35           20000         0
#> 3 15668575 Female    26           43000         0
#> 4 15603246 Female    27           57000         0
#> 5 15804002 Male      19           76000         0
#> 6 15728773 Male      27           58000         0
#> # ... with 394 more rows
```

```
# Taking columns 3-5
dataset = dataset[3:5]
tibble::as_tibble(dataset)
#> # A tibble: 400 x 3
#>     Age EstimatedSalary Purchased
#>   <int>           <int>     <int>
#> 1    19           19000         0
#> 2    35           20000         0
#> 3    26           43000         0
#> 4    27           57000         0
#> 5    19           76000         0
#> 6    27           58000         0
#> # ... with 394 more rows
```

```
# Encoding the target feature as factor
dataset$Purchased = factor(dataset$Purchased, levels = c(0, 1))
str(dataset)
#> 'data.frame':    400 obs. of  3 variables:
#>  $ Age            : int  19 35 26 27 19 27 27 32 25 35 ...
#>  $ EstimatedSalary: int  19000 20000 43000 57000 76000 58000 84000 150000 33000 65000 ...
#>  $ Purchased      : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 2 1 1 ...
```

```
# Splitting the dataset into the Training set and Test set
set.seed(123)
split = sample.split(dataset$Purchased, SplitRatio = 0.75)

training_set = subset(dataset, split == TRUE)
test_set = subset(dataset, split == FALSE)
```

```
dim(training_set)
#> [1] 300   3
dim(test_set)
#> [1] 100   3
```

```
# Feature Scaling
training_set[-3] = scale(training_set[-3])
test_set[-3] = scale(test_set[-3])
```

```
# Fitting SVM to the Training set
classifier = svm(formula = Purchased ~ .,
                 data = training_set,
                 type = 'C-classification',
                 kernel = 'linear')
```

```
classifier
#>
#> Call:
#> svm(formula = Purchased ~ ., data = training_set, type = "C-classification",
```

```
#>     kernel = "linear")
#>
#>
#> Parameters:
#>    SVM-Type:  C-classification
#>  SVM-Kernel:  linear
#>        cost:  1
#>       gamma:  0.5
#>
#> Number of Support Vectors:  116
```

```
summary(classifier)
#>
#> Call:
#> svm(formula = Purchased ~ ., data = training_set, type = "C-classification",
#>     kernel = "linear")
#>
#>
#> Parameters:
#>    SVM-Type:  C-classification
#>  SVM-Kernel:  linear
#>        cost:  1
#>       gamma:  0.5
#>
#> Number of Support Vectors:  116
#>
#>  ( 58 58 )
#>
#>
#> Number of Classes:  2
#>
#> Levels:
#>  0 1
```

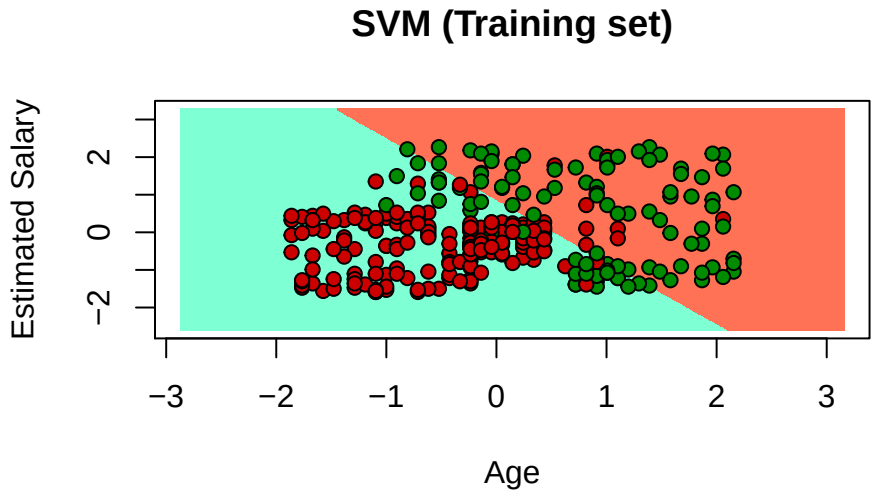
```
# Predicting the Test set results
y_pred = predict(classifier, newdata = test_set[-3])
y_pred
#>   2   4   5   9  12  18  19  20  22  29  32  34  35  38  45  46  48  52
#>   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
#>  66  69  74  75  82  84  85  86  87  89 103 104 107 108 109 117 124 126
#>   0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0
#> 127 131 134 139 148 154 156 159 162 163 170 175 176 193 199 200 208 213
#>   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   1
#> 224 226 228 229 230 234 236 237 239 241 255 264 265 266 273 274 281 286
#>   1   0   1   0   1   1   1   0   1   1   1   0   1   1   1   1   1   0
#> 292 299 302 305 307 310 316 324 326 332 339 341 343 347 353 363 364 367
#>   1   1   1   0   1   0   0   0   0   1   0   1   0   1   1   0   1   1
#> 368 369 372 373 380 383 389 392 395 400
#>   1   0   1   0   1   1   0   0   0   0
#> Levels: 0 1
```

```
# Making the Confusion Matrix
cm = table(test_set[, 3], y_pred)
cm
```

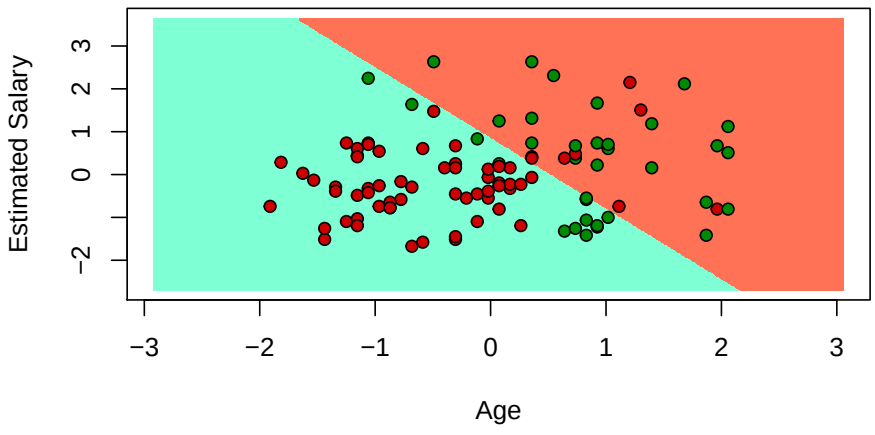
```
#>    y_pred
#>      0  1
#>   0 57  7
#>   1 13 23
```

```
xtable::xtable(cm)
#> % latex table generated in R 3.6.0 by xtable 1.8-4 package
#> % Wed Sep 18 15:11:17 2019
#> \begin{table}[ht]
#> \centering
#> \begin{tabular}{rrr}
#>   \hline
#>  & 0 & 1 \\
#>   \hline
#> 0 &  57 &   7 \\
#>   1 &  13 &  23 \\
#>    \hline
#> \end{tabular}
#> \end{table}
```

```
# installing library ElemStatLearn
# library(ElemStatLearn)

# Plotting the training data set results
set = training_set
X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)

grid_set = expand.grid(X1, X2)
colnames(grid_set) = c('Age', 'EstimatedSalary')
y_grid = predict(classifier, newdata = grid_set)

plot(set[, -3],
     main = 'SVM (Training set)',
     xlab = 'Age', ylab = 'Estimated Salary',
     xlim = range(X1), ylim = range(X2))

contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)

points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'coral1', 'aquamarine'))

points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))
```




```
set = test_set
X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)

grid_set = expand.grid(X1, X2)
colnames(grid_set) = c('Age', 'EstimatedSalary')
y_grid = predict(classifier, newdata = grid_set)

plot(set[, -3], main = 'SVM (Test set)',
     xlab = 'Age', ylab = 'Estimated Salary',
     xlim = range(X1), ylim = range(X2))

contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)

points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'coral1', 'aquamarine'))

points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))
```



Chapter 3

Broad view of SVM

3.1 Introduction

Source: http://uc-r.github.io/svm

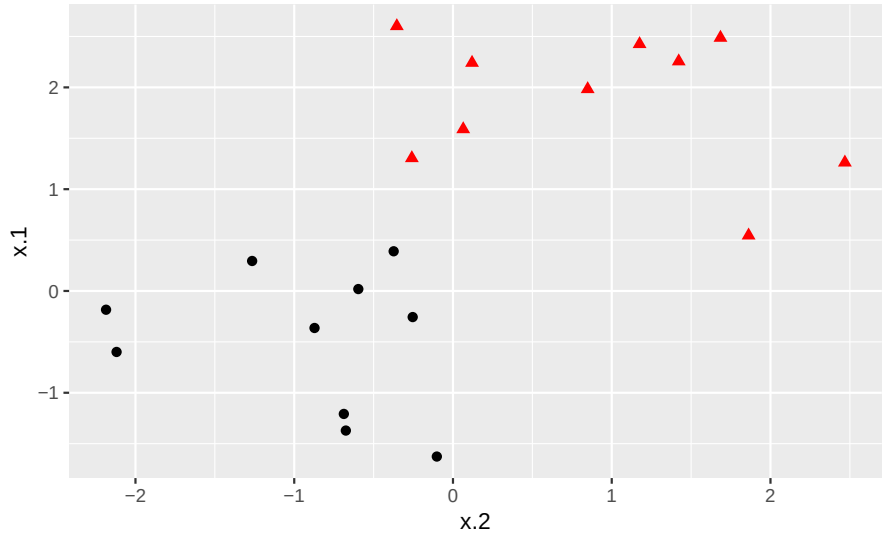
```
# set pseudorandom number generator
set.seed(10)

# Attach Packages
library(tidyverse)    # data manipulation and visualization
#> Registered S3 methods overwritten by 'ggplot2':
#>   method         from
#>   [.quosures     rlang
#>   c.quosures     rlang
#>   print.quosures rlang
#> Registered S3 method overwritten by 'rvest':
#>   method            from
#>   read_xml.response xml2
#> -- Attaching packages ---------------------------------------- tidyverse 1.2.1 --
#> v ggplot2 3.1.1       v purrr   0.3.2
#> v tibble  2.1.1       v dplyr   0.8.0.1
#> v tidyr   0.8.3       v stringr 1.4.0
#> v readr   1.3.1       v forcats 0.4.0
#> -- Conflicts ------------------------------------------- tidyverse_conflicts() --
#> x dplyr::filter() masks stats::filter()
#> x dplyr::lag()    masks stats::lag()
library(kernlab)      # SVM methodology
#>
#> Attaching package: 'kernlab'
#> The following object is masked from 'package:purrr':
#>
#>     cross
#> The following object is masked from 'package:ggplot2':
#>
#>     alpha
library(e1071)        # SVM methodology
library(ISLR)         # contains example data set "Khan"
library(RColorBrewer) # customized coloring of plots
```

The data sets used in the tutorial (with the exception of Khan) will be generated using built-in R commands. The Support Vector Machine methodology is sound for any number of dimensions, but becomes difficult to visualize for more than 2. As previously mentioned, SVMs are robust for any number of classes, but we will stick to no more than 3 for the duration of this tutorial.

3.2 Maximal Margin Classifier

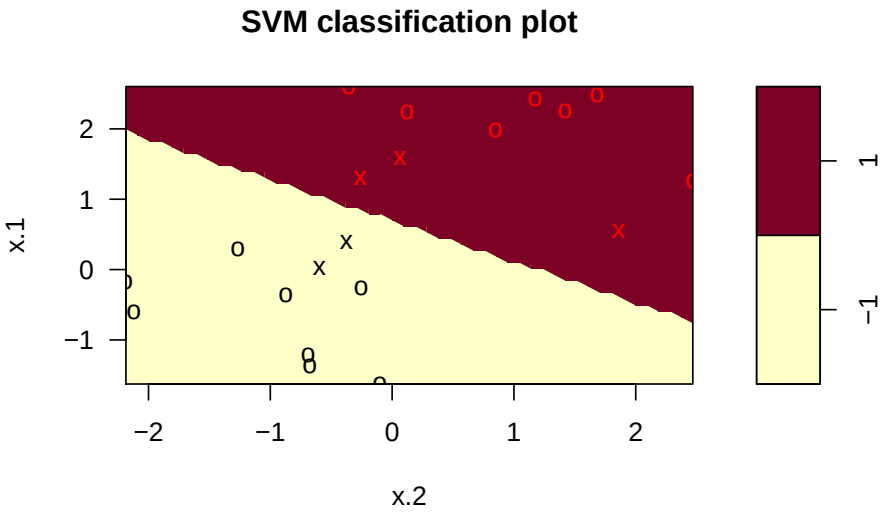
If the classes are separable by a linear boundary, we can use a Maximal Margin Classifier to find the classification boundary. To visualize an example of separated data, we generate 40 random observations and assign them to two classes. Upon visual inspection, we can see that infinitely many lines exist that split the two classes.

```
# Construct sample data set - completely separated
x <- matrix(rnorm(20*2), ncol = 2)
y <- c(rep(-1,10), rep(1,10))
x[y==1,] <- x[y==1,] + 3/2
dat <- data.frame(x=x, y=as.factor(y))

# Plot data
ggplot(data = dat, aes(x = x.2, y = x.1, color = y, shape = y)) +
  geom_point(size = 2) +
  scale_color_manual(values=c("#000000", "#FF0000")) +
  theme(legend.position = "none")
```



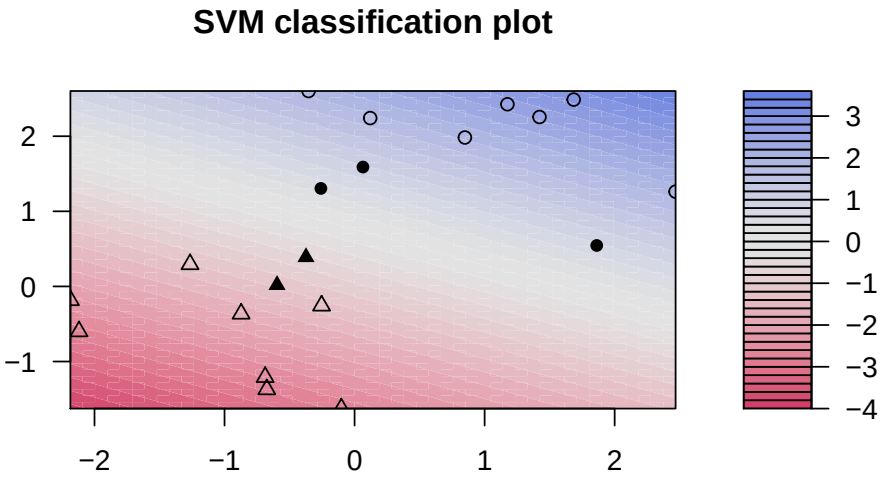
The goal of the maximal margin classifier is to identify the linear boundary that maximizes the total distance between the line and the closest point in each class. We can use the svm() function in the e1071 package to find this boundary.

```
# Fit Support Vector Machine model to data set
svmfit <- svm(y~., data = dat, kernel = "linear", scale = FALSE)
# Plot Results
plot(svmfit, dat)
```



In the plot, points that are represented by an "X" are the support vectors, or the points that directly affect the classification line. The points marked with an "o" are the other points, which don't affect the calculation of the line. This principle will lay the foundation for support vector machines. The same plot can be generated using the kernlab package, with the following results:

```
# fit model and produce plot
kernfit <- ksvm(x, y, type = "C-svc", kernel = 'vanilladot')
#>  Setting default kernel parameters
plot(kernfit, data = x)
```



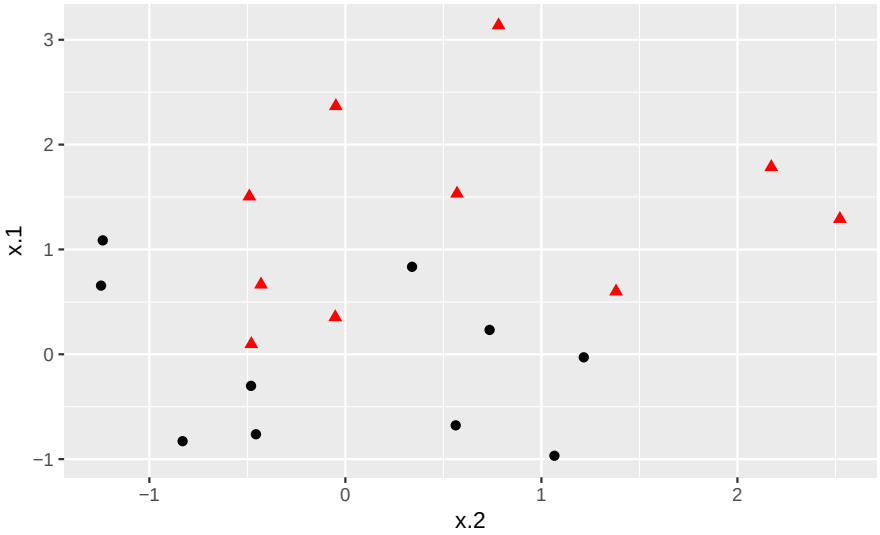
`kernlab` shows a little more detail than e1071, showing a color gradient that indicates how confidently a new point would be classified based on its features. Just as in the first plot, the support vectors are marked, in this case as filled-in points, while the classes are denoted by different shapes.

3.3 Support Vector Classifiers

As convenient as the maximal marginal classifier is to understand, most real data sets will not be fully separable by a linear boundary. To handle such data, we must use modified methodology. We simulate a new data set where the classes are more mixed.

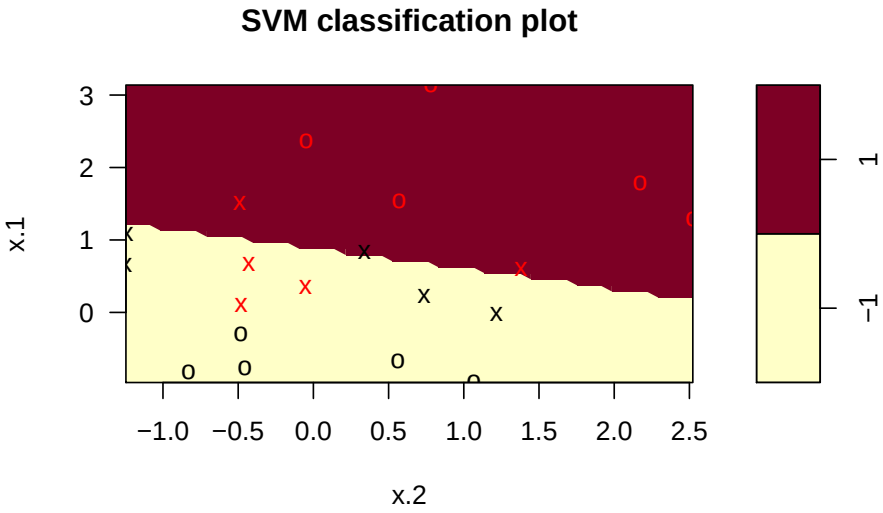
```
# Construct sample data set - not completely separated
x <- matrix(rnorm(20*2), ncol = 2)
```

```
y <- c(rep(-1,10), rep(1,10))
x[y==1,] <- x[y==1,] + 1
dat <- data.frame(x=x, y=as.factor(y))

# Plot data set
ggplot(data = dat, aes(x = x.2, y = x.1, color = y, shape = y)) +
  geom_point(size = 2) +
  scale_color_manual(values=c("#000000", "#FF0000")) +
  theme(legend.position = "none")
```



Whether the data is separable or not, the `svm()` command syntax is the same. In the case of data that is not linearly separable, however, the cost = argument takes on real importance. This quantifies the penalty associated with having an observation on the wrong side of the classification boundary. We can plot the fit in the same way as the completely separable case. We first use `e1071`:

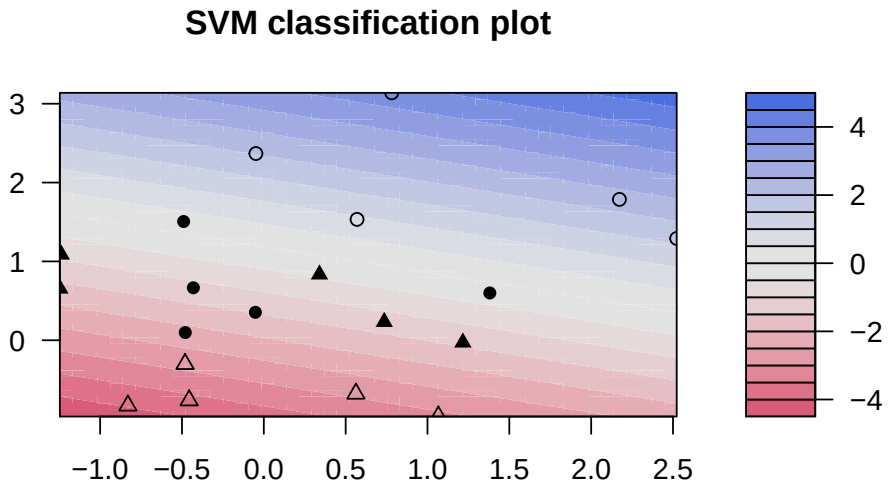
```
# Fit Support Vector Machine model to data set
svmfit <- svm(y~., data = dat, kernel = "linear", cost = 10)
# Plot Results
plot(svmfit, dat)
```



By upping the cost of misclassification from 10 to 100, you can see the difference in the classification line.

We repeat the process of plotting the SVM using the `kernlab` package:

```
# Fit Support Vector Machine model to data set
kernfit <- ksvm(x,y, type = "C-svc", kernel = 'vanilladot', C = 100)
#>  Setting default kernel parameters
# Plot results
plot(kernfit, data = x)
```



But how do we decide how costly these misclassifications actually are? Instead of specifying a cost up front, we can use the tune() function from e1071 to test various costs and identify which value produces the best fitting model.

```
# find optimal cost of misclassification
tune.out <- tune(svm, y~., data = dat, kernel = "linear",
                 ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100)))
# extract the best model
(bestmod <- tune.out$best.model)
#>
#> Call:
#> best.tune(method = svm, train.x = y ~ ., data = dat, ranges = list(cost = c(0.001,
#>     0.01, 0.1, 1, 5, 10, 100)), kernel = "linear")
#>
#>
#> Parameters:
#>    SVM-Type:  C-classification
#>  SVM-Kernel:  linear
#>        cost:  0.1
#>       gamma:  0.5
#>
#> Number of Support Vectors:  16
```

For our data set, the optimal cost (from amongst the choices we provided) is calculated to be 0.1, which doesn't penalize the model much for misclassified observations. Once this model has been identified, we can construct a table of predicted classes against true classes using the predict() command as follows:

```
# Create a table of misclassified observations
ypred <- predict(bestmod, dat)
(misclass <- table(predict = ypred, truth = dat$y))
#>        truth
#> predict -1 1
#>      -1  9 3
```

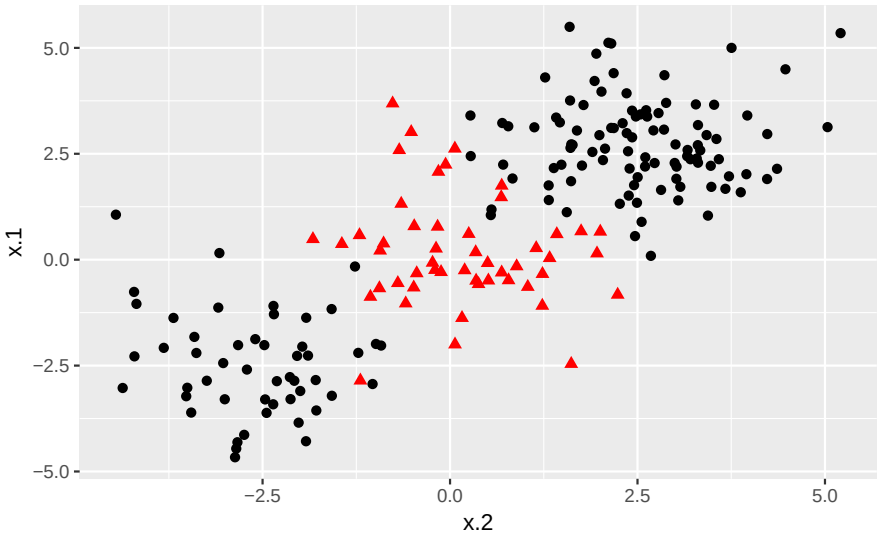
```
#>      1   1 7
```

Using this support vector classifier, 80% of the observations were correctly classified, which matches what we see in the plot. If we wanted to test our classifier more rigorously, we could split our data into training and testing sets and then see how our SVC performed with the observations not used to construct the model. We will use this training-testing method later in this tutorial to validate our SVMs.

3.4 Support Vector Machines

Support Vector Classifiers are a subset of the group of classification structures known as Support Vector Machines. Support Vector Machines can construct classification boundaries that are nonlinear in shape. The options for classification structures using the svm() command from the e1071 package are linear, polynomial, radial, and sigmoid. To demonstrate a nonlinear classification boundary, we will construct a new data set.

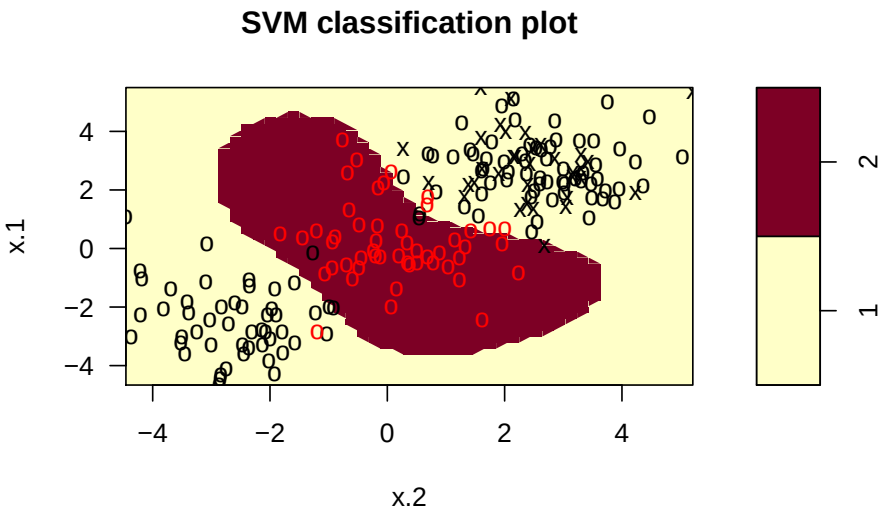
```
# construct larger random data set
x <- matrix(rnorm(200*2), ncol = 2)
x[1:100,] <- x[1:100,] + 2.5
x[101:150,] <- x[101:150,] - 2.5
y <- c(rep(1,150), rep(2,50))
dat <- data.frame(x=x,y=as.factor(y))

# Plot data
ggplot(data = dat, aes(x = x.2, y = x.1, color = y, shape = y)) +
  geom_point(size = 2) +
  scale_color_manual(values=c("#000000", "#FF0000")) +
  theme(legend.position = "none")
```



Notice that the data is not linearly separable, and furthermore, isn't all clustered together in a single group. There are two sections of class 1 observations with a cluster of class 2 observations in between. To demonstrate the power of SVMs, we'll take 100 random observations from the set and use them to construct our boundary. We set kernel = "radial" based on the shape of our data and plot the results.

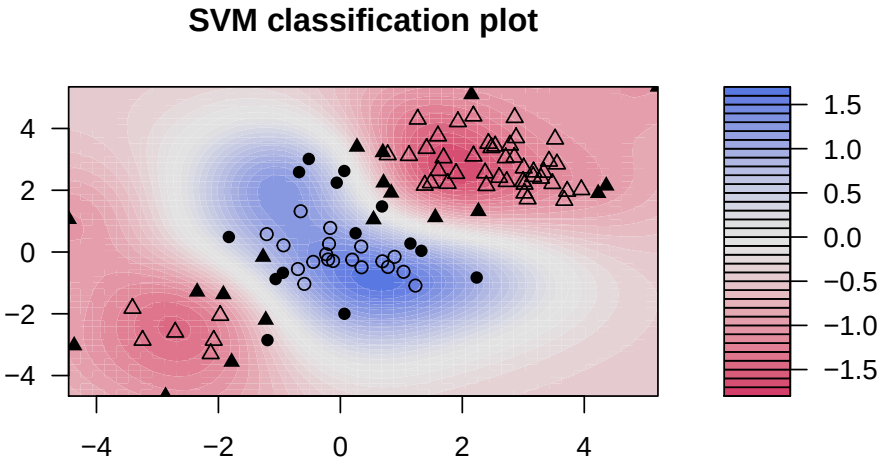
```
# set pseudorandom number generator
set.seed(123)
# sample training data and fit model
train <- base::sample(200,100, replace = FALSE)
svmfit <- svm(y~., data = dat[train,], kernel = "radial", gamma = 1, cost = 1)
```

```
# plot classifier
plot(svmfit, dat)
```



The same procedure can be run using the kernlab package, which has far more kernel options than the corresponding function in e1071. In addition to the four choices in e1071, this package allows use of a hyperbolic tangent, Laplacian, Bessel, Spline, String, or ANOVA RBF kernel. To fit this data, we set the cost to be the same as it was before, 1.

```
# Fit radial-based SVM in kernlab
kernfit <- ksvm(x[train,],y[train], type = "C-svc", kernel = 'rbfdot', C = 1, scaled = c())
# Plot training data
plot(kernfit, data = x[train,])
```



We see that, at least visually, the SVM does a reasonable job of separating the two classes. To fit the model, we used `cost = 1`, but as mentioned previously, it isn't usually obvious which cost will produce the optimal classification boundary. We can use the tune() command to try several different values of cost as well as several different values of γ, a scaling parameter used to fit nonlinear boundaries.

```
# tune model to find optimal cost, gamma values
tune.out <- tune(svm, y~., data = dat[train,], kernel = "radial",
                 ranges = list(cost = c(0.1,1,10,100,1000),
                 gamma = c(0.5,1,2,3,4)))
# show best model
tune.out$best.model
```

```
#>
#> Call:
#> best.tune(method = svm, train.x = y ~ ., data = dat[train, ],
#>     ranges = list(cost = c(0.1, 1, 10, 100, 1000), gamma = c(0.5,
#>         1, 2, 3, 4)), kernel = "radial")
#>
#>
#> Parameters:
#>    SVM-Type:  C-classification
#>  SVM-Kernel:  radial
#>        cost:  1
#>       gamma:  0.5
#>
#> Number of Support Vectors:  30
```

The model that reduces the error the most in the training data uses a cost of 1 and γ value of 0.5. We can now see how well the SVM performs by predicting the class of the 100 testing observations:

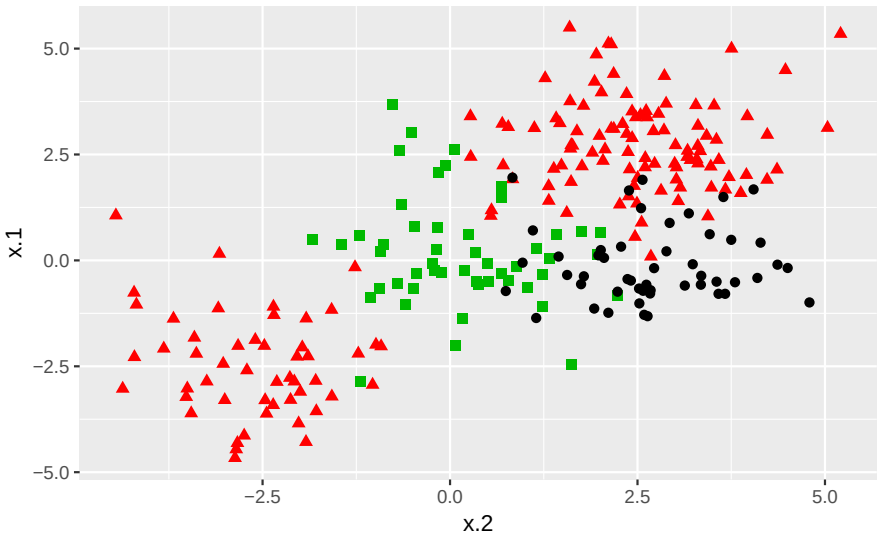
```
# validate model performance
(valid <- table(true = dat[-train,"y"], pred = predict(tune.out$best.model,
                                                       newx = dat[-train,])))
#>     pred
#> true  1  2
#>    1 55 28
#>    2 12  5
##     pred
## true  1  2
##    1 58 19
##    2 16  7
```

Our best-fitting model produces 65% accuracy in identifying classes. For such a complicated shape of observations, this performed reasonably well. We can challenge this method further by adding additional classes of observations.

3.5 SVMs for Multiple Classes

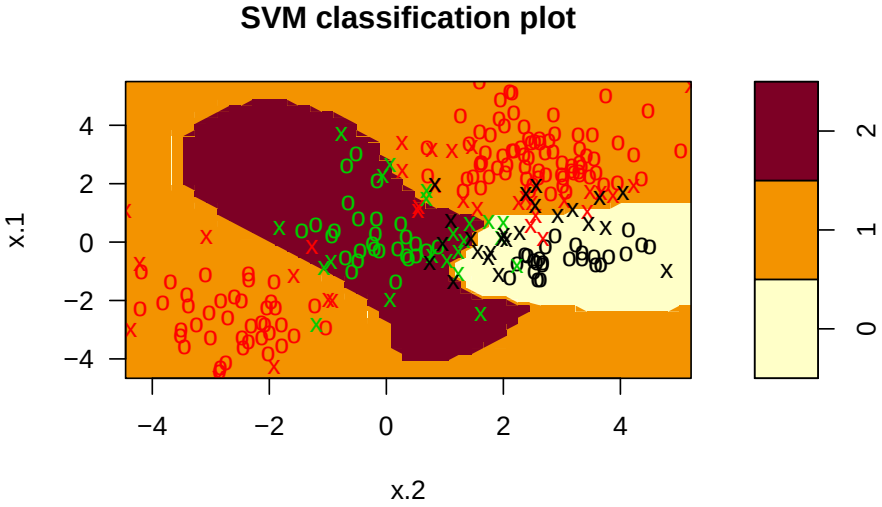
The procedure does not change for data sets that involve more than two classes of observations. We construct our data set the same way as we have previously, only now specifying three classes instead of two:

```
# construct data set
x <- rbind(x, matrix(rnorm(50*2), ncol = 2))
y <- c(y, rep(0,50))
x[y==0,2] <- x[y==0,2] + 2.5
dat <- data.frame(x=x, y=as.factor(y))
# plot data set
ggplot(data = dat, aes(x = x.2, y = x.1, color = y, shape = y)) +
  geom_point(size = 2) +
  scale_color_manual(values=c("#000000","#FF0000","#00BA00")) +
  theme(legend.position = "none")
```



The commands don't change for the e1071 package. We specify a cost and tuning parameter γ and fit a support vector machine. The results and interpretation are similar to two-class classification.

```
# fit model
svmfit <- svm(y~., data = dat, kernel = "radial", cost = 10, gamma = 1)
# plot results
plot(svmfit, dat)
```



We can check to see how well our model fit the data by using the `predict()` command, as follows:

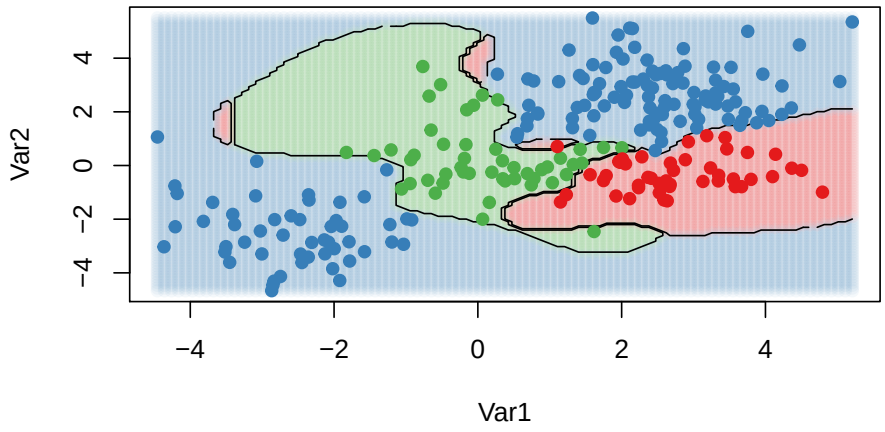
```
#construct table
ypred <- predict(svmfit, dat)
(misclass <- table(predict = ypred, truth = dat$y))
#>         truth
#> predict   0   1   2
#>       0  38   2   5
#>       1   7 145   2
#>       2   5   3  43
##         truth
## predict   0   1   2
##       0  38   2   4
##       1   8 143   4
```

```
##       2   4   5  42
```

As shown in the resulting table, 89% of our training observations were correctly classified. However, since we didn't break our data into training and testing sets, we didn't truly validate our results.

The kernlab package, on the other hand, can fit more than 2 classes, but cannot plot the results. To visualize the results of the ksvm function, we take the steps listed below to create a grid of points, predict the value of each point, and plot the results:

```
# fit and plot
kernfit <- ksvm(as.matrix(dat[,2:1]),dat$y, type = "C-svc", kernel = 'rbfdot',
                C = 100, scaled = c())

# Create a fine grid of the feature space
x.1 <- seq(from = min(dat$x.1), to = max(dat$x.1), length = 100)
x.2 <- seq(from = min(dat$x.2), to = max(dat$x.2), length = 100)
x.grid <- expand.grid(x.2, x.1)

# Get class predictions over grid
pred <- predict(kernfit, newdata = x.grid)

# Plot the results
cols <- brewer.pal(3, "Set1")
plot(x.grid, pch = 19, col = adjustcolor(cols[pred], alpha.f = 0.05))

classes <- matrix(pred, nrow = 100, ncol = 100)
contour(x = x.2, y = x.1, z = classes, levels = 1:3, labels = "", add = TRUE)

points(dat[, 2:1], pch = 19, col = cols[predict(kernfit)])
```



3.6 Application

The Khan data set contains data on 83 tissue samples with 2308 gene expression measurements on each sample. These were split into 63 training observations and 20 testing observations, and there are four distinct classes in the set. It would be impossible to visualize such data, so we choose the simplest classifier (linear) to construct our model. We will use the svm command from `e1071` to conduct our analysis.

```
# fit model
dat <- data.frame(x = Khan$xtrain, y=as.factor(Khan$ytrain))
(out <- svm(y~., data = dat, kernel = "linear", cost=10))
```

```
#>
#> Call:
#> svm(formula = y ~ ., data = dat, kernel = "linear", cost = 10)
#>
#>
#> Parameters:
#>    SVM-Type:  C-classification
#>  SVM-Kernel:  linear
#>        cost:  10
#>       gamma:  0.000433
#>
#> Number of Support Vectors:  58
```

First of all, we can check how well our model did at classifying the training observations. This is usually high, but again, doesn't validate the model. If the model doesn't do a very good job of classifying the training set, it could be a red flag. In our case, all 63 training observations were correctly classified.

```
# check model performance on training set
table(out$fitted, dat$y)
#>
#>      1  2  3  4
#>   1  8  0  0  0
#>   2  0 23  0  0
#>   3  0  0 12  0
#>   4  0  0  0 20
```

To perform validation, we can check how the model performs on the testing set:

```
# validate model performance
dat.te <- data.frame(x=Khan$xtest, y=as.factor(Khan$ytest))
pred.te <- predict(out, newdata=dat.te)
table(pred.te, dat.te$y)
#>
#> pred.te 1 2 3 4
#>       1 3 0 0 0
#>       2 0 6 2 0
#>       3 0 0 4 0
#>       4 0 0 0 5
```

The model correctly identifies 18 of the 20 testing observations. SVMs and the boundaries they impose are more difficult to interpret at higher dimensions, but these results seem to suggest that our model is a good classifier for the gene data.

Chapter 4

Sonar Standalone Model with Random Forest

Classification problem

4.1 Introduction

- `mtry`: Number of variables randomly sampled as candidates at each split.
- `ntree`: Number of trees to grow.

4.2 Load libraries

```
# load packages
library(caret)
library(mlbench)
library(randomForest)
library(tictoc)

# load dataset
data(Sonar)
set.seed(7)
```

4.3 Explore data

```
dplyr::glimpse(Sonar)
#> Observations: 208
#> Variables: 61
#> $ V1    <dbl> 0.0200, 0.0453, 0.0262, 0.0100, 0.0762, 0.0286, 0.0317, ...
#> $ V2    <dbl> 0.0371, 0.0523, 0.0582, 0.0171, 0.0666, 0.0453, 0.0956, ...
#> $ V3    <dbl> 0.0428, 0.0843, 0.1099, 0.0623, 0.0481, 0.0277, 0.1321, ...
#> $ V4    <dbl> 0.0207, 0.0689, 0.1083, 0.0205, 0.0394, 0.0174, 0.1408, ...
#> $ V5    <dbl> 0.0954, 0.1183, 0.0974, 0.0205, 0.0590, 0.0384, 0.1674, ...
```

```
#> $ V6    <dbl> 0.0986, 0.2583, 0.2280, 0.0368, 0.0649, 0.0990, 0.1710, ...
#> $ V7    <dbl> 0.1539, 0.2156, 0.2431, 0.1098, 0.1209, 0.1201, 0.0731, ...
#> $ V8    <dbl> 0.1601, 0.3481, 0.3771, 0.1276, 0.2467, 0.1833, 0.1401, ...
#> $ V9    <dbl> 0.3109, 0.3337, 0.5598, 0.0598, 0.3564, 0.2105, 0.2083, ...
#> $ V10   <dbl> 0.2111, 0.2872, 0.6194, 0.1264, 0.4459, 0.3039, 0.3513, ...
#> $ V11   <dbl> 0.1609, 0.4918, 0.6333, 0.0881, 0.4152, 0.2988, 0.1786, ...
#> $ V12   <dbl> 0.1582, 0.6552, 0.7060, 0.1992, 0.3952, 0.4250, 0.0658, ...
#> $ V13   <dbl> 0.2238, 0.6919, 0.5544, 0.0184, 0.4256, 0.6343, 0.0513, ...
#> $ V14   <dbl> 0.0645, 0.7797, 0.5320, 0.2261, 0.4135, 0.8198, 0.3752, ...
#> $ V15   <dbl> 0.0660, 0.7464, 0.6479, 0.1729, 0.4528, 1.0000, 0.5419, ...
#> $ V16   <dbl> 0.2273, 0.9444, 0.6931, 0.2131, 0.5326, 0.9988, 0.5440, ...
#> $ V17   <dbl> 0.3100, 1.0000, 0.6759, 0.0693, 0.7306, 0.9508, 0.5150, ...
#> $ V18   <dbl> 0.300, 0.887, 0.755, 0.228, 0.619, 0.902, 0.426, 0.120, ...
#> $ V19   <dbl> 0.508, 0.802, 0.893, 0.406, 0.203, 0.723, 0.202, 0.668, ...
#> $ V20   <dbl> 0.4797, 0.7818, 0.8619, 0.3973, 0.4636, 0.5122, 0.4233, ...
#> $ V21   <dbl> 0.578, 0.521, 0.797, 0.274, 0.415, 0.207, 0.772, 0.783, ...
#> $ V22   <dbl> 0.507, 0.405, 0.674, 0.369, 0.429, 0.399, 0.974, 0.535, ...
#> $ V23   <dbl> 0.433, 0.396, 0.429, 0.556, 0.573, 0.589, 0.939, 0.681, ...
#> $ V24   <dbl> 0.555, 0.391, 0.365, 0.485, 0.540, 0.287, 0.556, 0.917, ...
#> $ V25   <dbl> 0.671, 0.325, 0.533, 0.314, 0.316, 0.204, 0.527, 0.761, ...
#> $ V26   <dbl> 0.641, 0.320, 0.241, 0.533, 0.229, 0.578, 0.683, 0.822, ...
#> $ V27   <dbl> 0.7104, 0.3271, 0.5070, 0.5256, 0.6995, 0.5389, 0.5713, ...
#> $ V28   <dbl> 0.8080, 0.2767, 0.8533, 0.2520, 1.0000, 0.3750, 0.5429, ...
#> $ V29   <dbl> 0.6791, 0.4423, 0.6036, 0.2090, 0.7262, 0.3411, 0.2177, ...
#> $ V30   <dbl> 0.3857, 0.2028, 0.8514, 0.3559, 0.4724, 0.5067, 0.2149, ...
#> $ V31   <dbl> 0.131, 0.379, 0.851, 0.626, 0.510, 0.558, 0.581, 0.132, ...
#> $ V32   <dbl> 0.2604, 0.2947, 0.5045, 0.7340, 0.5459, 0.4778, 0.6323, ...
#> $ V33   <dbl> 0.512, 0.198, 0.186, 0.612, 0.288, 0.330, 0.296, 0.099, ...
#> $ V34   <dbl> 0.7547, 0.2341, 0.2709, 0.3497, 0.0981, 0.2198, 0.1873, ...
#> $ V35   <dbl> 0.8537, 0.1306, 0.4232, 0.3953, 0.1951, 0.1407, 0.2969, ...
#> $ V36   <dbl> 0.851, 0.418, 0.304, 0.301, 0.418, 0.286, 0.516, 0.105, ...
#> $ V37   <dbl> 0.669, 0.384, 0.612, 0.541, 0.460, 0.381, 0.615, 0.192, ...
#> $ V38   <dbl> 0.6097, 0.1057, 0.6756, 0.8814, 0.3217, 0.4158, 0.4283, ...
#> $ V39   <dbl> 0.4943, 0.1840, 0.5375, 0.9857, 0.2828, 0.4054, 0.5479, ...
#> $ V40   <dbl> 0.2744, 0.1970, 0.4719, 0.9167, 0.2430, 0.3296, 0.6133, ...
#> $ V41   <dbl> 0.0510, 0.1674, 0.4647, 0.6121, 0.1979, 0.2707, 0.5017, ...
#> $ V42   <dbl> 0.2834, 0.0583, 0.2587, 0.5006, 0.2444, 0.2650, 0.2377, ...
#> $ V43   <dbl> 0.2825, 0.1401, 0.2129, 0.3210, 0.1847, 0.0723, 0.1957, ...
#> $ V44   <dbl> 0.4256, 0.1628, 0.2222, 0.3202, 0.0841, 0.1238, 0.1749, ...
#> $ V45   <dbl> 0.2641, 0.0621, 0.2111, 0.4295, 0.0692, 0.1192, 0.1304, ...
#> $ V46   <dbl> 0.1386, 0.0203, 0.0176, 0.3654, 0.0528, 0.1089, 0.0597, ...
#> $ V47   <dbl> 0.1051, 0.0530, 0.1348, 0.2655, 0.0357, 0.0623, 0.1124, ...
#> $ V48   <dbl> 0.1343, 0.0742, 0.0744, 0.1576, 0.0085, 0.0494, 0.1047, ...
#> $ V49   <dbl> 0.0383, 0.0409, 0.0130, 0.0681, 0.0230, 0.0264, 0.0507, ...
#> $ V50   <dbl> 0.0324, 0.0061, 0.0106, 0.0294, 0.0046, 0.0081, 0.0159, ...
#> $ V51   <dbl> 0.0232, 0.0125, 0.0033, 0.0241, 0.0156, 0.0104, 0.0195, ...
#> $ V52   <dbl> 0.0027, 0.0084, 0.0232, 0.0121, 0.0031, 0.0045, 0.0201, ...
#> $ V53   <dbl> 0.0065, 0.0089, 0.0166, 0.0036, 0.0054, 0.0014, 0.0248, ...
#> $ V54   <dbl> 0.0159, 0.0048, 0.0095, 0.0150, 0.0105, 0.0038, 0.0131, ...
#> $ V55   <dbl> 0.0072, 0.0094, 0.0180, 0.0085, 0.0110, 0.0013, 0.0070, ...
#> $ V56   <dbl> 0.0167, 0.0191, 0.0244, 0.0073, 0.0015, 0.0089, 0.0138, ...
#> $ V57   <dbl> 0.0180, 0.0140, 0.0316, 0.0050, 0.0072, 0.0057, 0.0092, ...
#> $ V58   <dbl> 0.0084, 0.0049, 0.0164, 0.0044, 0.0048, 0.0027, 0.0143, ...
```

```
#> $ V59   <dbl> 0.0090, 0.0052, 0.0095, 0.0040, 0.0107, 0.0051, 0.0036, ...
#> $ V60   <dbl> 0.0032, 0.0044, 0.0078, 0.0117, 0.0094, 0.0062, 0.0103, ...
#> $ Class <fct> R, R, R, R, R, R, R, R, R, R, R, R, R, R, R, R, R, R, R,...
```

```
tibble::as_tibble(Sonar)
#> # A tibble: 208 x 61
#>       V1     V2     V3     V4     V5     V6     V7     V8     V9   V10    V11
#>    <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl> <dbl> <dbl>  <dbl> <dbl>  <dbl>
#> 1 0.02   0.0371 0.0428 0.0207 0.0954 0.0986 0.154 0.160 0.311  0.211 0.161
#> 2 0.0453 0.0523 0.0843 0.0689 0.118  0.258  0.216 0.348 0.334  0.287 0.492
#> 3 0.0262 0.0582 0.110  0.108  0.0974 0.228  0.243 0.377 0.560  0.619 0.633
#> 4 0.01   0.0171 0.0623 0.0205 0.0205 0.0368 0.110 0.128 0.0598 0.126 0.0881
#> 5 0.0762 0.0666 0.0481 0.0394 0.059  0.0649 0.121 0.247 0.356  0.446 0.415
#> 6 0.0286 0.0453 0.0277 0.0174 0.0384 0.099  0.120 0.183 0.210  0.304 0.299
#> # ... with 202 more rows, and 50 more variables: V12 <dbl>, V13 <dbl>,
#> #   V14 <dbl>, V15 <dbl>, V16 <dbl>, V17 <dbl>, V18 <dbl>, V19 <dbl>,
#> #   V20 <dbl>, V21 <dbl>, V22 <dbl>, V23 <dbl>, V24 <dbl>, V25 <dbl>,
#> #   V26 <dbl>, V27 <dbl>, V28 <dbl>, V29 <dbl>, V30 <dbl>, V31 <dbl>,
#> #   V32 <dbl>, V33 <dbl>, V34 <dbl>, V35 <dbl>, V36 <dbl>, V37 <dbl>,
#> #   V38 <dbl>, V39 <dbl>, V40 <dbl>, V41 <dbl>, V42 <dbl>, V43 <dbl>,
#> #   V44 <dbl>, V45 <dbl>, V46 <dbl>, V47 <dbl>, V48 <dbl>, V49 <dbl>,
#> #   V50 <dbl>, V51 <dbl>, V52 <dbl>, V53 <dbl>, V54 <dbl>, V55 <dbl>,
#> #   V56 <dbl>, V57 <dbl>, V58 <dbl>, V59 <dbl>, V60 <dbl>, Class <fct>
```

```
# create 80%/20% for training and validation datasets
validationIndex <- createDataPartition(Sonar$Class, p=0.80, list=FALSE)
validation <- Sonar[-validationIndex,]
training   <- Sonar[validationIndex,]
```

```
tic()
# train a model and summarize model
set.seed(7)
trainControl <- trainControl(method="repeatedcv", number=10, repeats=3)
fit.rf <- train(Class~., data=training,
                method = "rf",
                metric = "Accuracy",
                trControl = trainControl,
                ntree = 2000)
toc()
#> 118.619 sec elapsed
print(fit.rf)
#> Random Forest
#>
#> 167 samples
#>  60 predictor
#>   2 classes: 'M', 'R'
#>
#> No pre-processing
#> Resampling: Cross-Validated (10 fold, repeated 3 times)
#> Summary of sample sizes: 150, 150, 150, 151, 151, 150, ...
#> Resampling results across tuning parameters:
#>
#>   mtry  Accuracy  Kappa
#>    2    0.845     0.682
```

```
#>   31    0.828     0.651
#>   60    0.808     0.611
#>
#> Accuracy was used to select the optimal model using the largest value.
#> The final value used for the model was mtry = 2.
print(fit.rf$finalModel)
#>
#> Call:
#>  randomForest(x = x, y = y, ntree = 2000, mtry = param$mtry)
#>                Type of random forest: classification
#>                      Number of trees: 2000
#> No. of variables tried at each split: 2
#>
#>         OOB estimate of  error rate: 14.4%
#> Confusion matrix:
#>    M  R class.error
#> M 84  5      0.0562
#> R 19 59      0.2436
```

Accuracy: 85.26% at mtry=2

4.4 Apply tuning parameters for final model

```
# create standalone model using all training data
set.seed(7)
finalModel <- randomForest(Class~., training, mtry=2, ntree=2000)

# make a predictions on "new data" using the final model
finalPredictions <- predict(finalModel, validation[,1:60])
confusionMatrix(finalPredictions, validation$Class)
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction  M  R
#>          M 20  4
#>          R  2 15
#>
#>                Accuracy : 0.854
#>                  95% CI : (0.708, 0.944)
#>     No Information Rate : 0.537
#>     P-Value [Acc > NIR] : 1.88e-05
#>
#>                   Kappa : 0.704
#>
#>  Mcnemar's Test P-Value : 0.683
#>
#>             Sensitivity : 0.909
#>             Specificity : 0.789
#>          Pos Pred Value : 0.833
#>          Neg Pred Value : 0.882
#>              Prevalence : 0.537
#>          Detection Rate : 0.488
```

```
#>    Detection Prevalence : 0.585
#>       Balanced Accuracy : 0.849
#>
#>        'Positive' Class : M
#>
```

Accuracy: 82.93%

4.5 Save model

```
# save the model to disk
saveRDS(finalModel, file.path(model_out_dir, "sonar-finalModel.rds"))
```

4.6 Use the saved model

```
# load the model
superModel <- readRDS(file.path(model_out_dir, "sonar-finalModel.rds"))
print(superModel)
#>
#> Call:
#>  randomForest(formula = Class ~ ., data = training, mtry = 2,      ntree = 2000)
#>                Type of random forest: classification
#>                      Number of trees: 2000
#> No. of variables tried at each split: 2
#>
#>         OOB estimate of  error rate: 16.2%
#> Confusion matrix:
#>    M  R class.error
#> M 81  8      0.0899
#> R 19 59      0.2436
```

4.7 Make prediction with new data

```
# make a predictions on "new data" using the final model
finalPredictions <- predict(superModel, validation[,1:60])
confusionMatrix(finalPredictions, validation$Class)
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction  M  R
#>          M 20  4
#>          R  2 15
#>
#>                Accuracy : 0.854
#>                  95% CI : (0.708, 0.944)
#>     No Information Rate : 0.537
#>     P-Value [Acc > NIR] : 1.88e-05
```

```
#>
#>                  Kappa : 0.704
#>
#>  Mcnemar's Test P-Value : 0.683
#>
#>             Sensitivity : 0.909
#>             Specificity : 0.789
#>          Pos Pred Value : 0.833
#>          Neg Pred Value : 0.882
#>              Prevalence : 0.537
#>          Detection Rate : 0.488
#>    Detection Prevalence : 0.585
#>       Balanced Accuracy : 0.849
#>
#>        'Positive' Class : M
#>
```

Chapter 5

Glass classification

https://cran.r-project.org/web/packages/e1071/vignettes/svmdoc.pdf

In this example, we use the glass data from the UCI Repository of Machine Learning Databases for classification. The task is to predict the type of a glass on basis of its chemical analysis. We start by splitting the data into a train and test set:

```
library(caret)
#> Loading required package: lattice
#> Loading required package: ggplot2
#> Registered S3 methods overwritten by 'ggplot2':
#>   method         from
#>   [.quosures     rlang
#>   c.quosures     rlang
#>   print.quosures rlang
library(e1071)
library(rpart)

data(Glass, package="mlbench")
str(Glass)
#> 'data.frame':    214 obs. of  10 variables:
#>  $ RI  : num  1.52 1.52 1.52 1.52 1.52 ...
#>  $ Na  : num  13.6 13.9 13.5 13.2 13.3 ...
#>  $ Mg  : num  4.49 3.6 3.55 3.69 3.62 3.61 3.6 3.61 3.58 3.6 ...
#>  $ Al  : num  1.1 1.36 1.54 1.29 1.24 1.62 1.14 1.05 1.37 1.36 ...
#>  $ Si  : num  71.8 72.7 73 72.6 73.1 ...
#>  $ K   : num  0.06 0.48 0.39 0.57 0.55 0.64 0.58 0.57 0.56 0.57 ...
#>  $ Ca  : num  8.75 7.83 7.78 8.22 8.07 8.07 8.17 8.24 8.3 8.4 ...
#>  $ Ba  : num  0 0 0 0 0 0 0 0 0 0 ...
#>  $ Fe  : num  0 0 0 0 0 0.26 0 0 0 0.11 ...
#>  $ Type: Factor w/ 6 levels "1","2","3","5",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```
## split data into a train and test set
index <- 1:nrow(Glass)
testindex <- sample(index, trunc(length(index)/3))
testset  <- Glass[testindex,]
trainset <- Glass[-testindex,]
```

Both for the SVM and the partitioning tree (via `rpart()`), we fit the model and try to predict the test set values:

```
## svm
svm.model <- svm(Type ~ ., data = trainset, cost = 100, gamma = 1)
svm.pred  <- predict(svm.model, testset[,-10])
```

(The dependent variable, Type, has column number 10. cost is a general penalizing parameter for C-classification and `gamma` is the radial basis function-specific `kernel` parameter.)

```
## rpart
rpart.model <- rpart(Type ~ ., data = trainset)
rpart.pred <- predict(rpart.model, testset[,-10], type = "class")
```

A cross-tabulation of the true versus the predicted values yields:

```
## compute svm confusion matrix
table(pred = svm.pred, true = testset[,10])
#>      true
#> pred  1  2  3  5  6  7
#>    1 20  3  3  0  0  0
#>    2  6 13  5  4  2  4
#>    3  2  1  0  0  0  0
#>    5  0  0  0  1  0  0
#>    6  0  0  0  0  0  0
#>    7  0  0  0  0  0  7
```

```
## compute rpart confusion matrix
table(pred = rpart.pred, true = testset[,10])
#>      true
#> pred  1  2  3  5  6  7
#>    1 22  0  3  0  0  0
#>    2  5 12  4  0  0  0
#>    3  0  2  1  0  0  0
#>    5  0  2  0  5  2  1
#>    6  0  0  0  0  0  0
#>    7  1  1  0  0  0 10
```

5.0.1 Comparison test sets

```
confusionMatrix(svm.pred, testset$Type)
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction  1  2  3  5  6  7
#>          1 20  3  3  0  0  0
#>          2  6 13  5  4  2  4
#>          3  2  1  0  0  0  0
#>          5  0  0  0  1  0  0
#>          6  0  0  0  0  0  0
#>          7  0  0  0  0  0  7
#>
#> Overall Statistics
#>
#>                Accuracy : 0.577
#>                  95% CI : (0.454, 0.694)
#>     No Information Rate : 0.394
```

```
#>     P-Value [Acc > NIR] : 0.00137
#>
#>                   Kappa : 0.413
#>
#>  Mcnemar's Test P-Value : NA
#>
#> Statistics by Class:
#>
#>                      Class: 1 Class: 2 Class: 3 Class: 5 Class: 6 Class: 7
#> Sensitivity             0.714    0.765   0.0000   0.2000   0.0000   0.6364
#> Specificity             0.860    0.611   0.9524   1.0000   1.0000   1.0000
#> Pos Pred Value          0.769    0.382   0.0000   1.0000      NaN   1.0000
#> Neg Pred Value          0.822    0.892   0.8824   0.9429   0.9718   0.9375
#> Prevalence              0.394    0.239   0.1127   0.0704   0.0282   0.1549
#> Detection Rate          0.282    0.183   0.0000   0.0141   0.0000   0.0986
#> Detection Prevalence    0.366    0.479   0.0423   0.0141   0.0000   0.0986
#> Balanced Accuracy       0.787    0.688   0.4762   0.6000   0.5000   0.8182
```

```
confusionMatrix(rpart.pred, testset$Type)
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction  1  2  3  5  6  7
#>          1 22  0  3  0  0  0
#>          2  5 12  4  0  0  0
#>          3  0  2  1  0  0  0
#>          5  0  2  0  5  2  1
#>          6  0  0  0  0  0  0
#>          7  1  1  0  0  0 10
#>
#> Overall Statistics
#>
#>                Accuracy : 0.704
#>                  95% CI : (0.584, 0.807)
#>     No Information Rate : 0.394
#>     P-Value [Acc > NIR] : 1.23e-07
#>
#>                   Kappa : 0.605
#>
#>  Mcnemar's Test P-Value : NA
#>
#> Statistics by Class:
#>
#>                      Class: 1 Class: 2 Class: 3 Class: 5 Class: 6 Class: 7
#> Sensitivity             0.786    0.706   0.1250   1.0000   0.0000    0.909
#> Specificity             0.930    0.833   0.9683   0.9242   1.0000    0.967
#> Pos Pred Value          0.880    0.571   0.3333   0.5000      NaN    0.833
#> Neg Pred Value          0.870    0.900   0.8971   1.0000   0.9718    0.983
#> Prevalence              0.394    0.239   0.1127   0.0704   0.0282    0.155
#> Detection Rate          0.310    0.169   0.0141   0.0704   0.0000    0.141
#> Detection Prevalence    0.352    0.296   0.0423   0.1408   0.0000    0.169
#> Balanced Accuracy       0.858    0.770   0.5466   0.9621   0.5000    0.938
```

5.0.2 Comparison with resamples

Finally, we compare the performance of the two methods by computing the respective accuracy rates and the kappa indices (as computed by `classAgreement()` also contained in package `e1071`). In Table 1, we summarize the results of 10 replications—Support Vector Machines show better results.

```
set.seed(1234567)

# SVM
fit.svm <- train(Type ~., data = trainset,
                 method = "svmRadial")

# Random Forest
fit.rpart <- train(Type ~., data = trainset,
                method="rpart")

# collect resamples
results <- resamples(list(svm = fit.svm,
                          rpart  = fit.rpart))

summary(results)
#>
#> Call:
#> summary.resamples(object = results)
#>
#> Models: svm, rpart
#> Number of resamples: 25
#>
#> Accuracy
#>        Min. 1st Qu. Median  Mean 3rd Qu.  Max. NA's
#> svm   0.510   0.565  0.600 0.599   0.625 0.704    0
#> rpart 0.462   0.519  0.554 0.558   0.600 0.660    0
#>
#> Kappa
#>        Min. 1st Qu. Median  Mean 3rd Qu.  Max. NA's
#> svm   0.267   0.376  0.406 0.410   0.446 0.559    0
#> rpart 0.135   0.299  0.358 0.363   0.443 0.545    0
```

Chapter 6

Ozone SVM

https://cran.r-project.org/web/packages/e1071/vignettes/svmdoc.pdf

```
library(e1071)
library(rpart)

data(Ozone, package="mlbench")
## split data into a train and test set
index <- 1:nrow(Ozone)
testindex <- sample(index, trunc(length(index)/3))
testset <- na.omit(Ozone[testindex,-3])
trainset <- na.omit(Ozone[-testindex,-3])
```

```
## svm
svm.model <- svm(V4 ~ ., data = trainset, cost = 1000, gamma = 0.0001)
svm.pred  <- predict(svm.model, testset[,-3])
crossprod(svm.pred - testset[,3]) / length(testindex)
#>      [,1]
#> [1,] 10.7
```

```
## rpart
rpart.model <- rpart(V4 ~ ., data = trainset)
rpart.pred  <- predict(rpart.model, testset[,-3])
crossprod(rpart.pred - testset[,3]) / length(testindex)
#>      [,1]
#> [1,] 11.9
```

Chapter 7

A gentle introduction to support vector machines using R

https://eight2late.wordpress.com/2017/02/07/a-gentle-introduction-to-support-vector-machines-using-r/

7.1 Support vector machines in R

In this demo we'll use the svm interface that is implemented in the `e1071` R package. This interface provides R programmers access to the comprehensive `libsvm` library written by Chang and Lin. I'll use two toy datasets: the famous iris dataset available with the base R package and the sonar dataset from the mlbench package. I won't describe details of the datasets as they are discussed at length in the documentation that I have linked to. However, it is worth mentioning the reasons why I chose these datasets:

As mentioned earlier, no real life dataset is linearly separable, but the iris dataset is almost so. Consequently, it is a good illustration of using linear SVMs. Although one almost never uses these in practice, I have illustrated their use primarily for pedagogical reasons. The sonar dataset is a good illustration of the benefits of using RBF kernels in cases where the dataset is hard to visualise (60 variables in this case!). In general, one would almost always use RBF (or other nonlinear) kernels in practice.

With that said, let's get right to it. I assume you have R and RStudio installed. For instructions on how to do this, have a look at the first article in this series. The processing preliminaries – loading libraries, data and creating training and test datasets are much the same as in my previous articles so I won't dwell on these here. For completeness, however, I'll list all the code so you can run it directly in R or R studio (a complete listing of the code can be found here):

7.2 SVM on `iris` dataset

7.2.1 Training and test datasets

```
#load required library
library(e1071)

#load built-in iris dataset
data(iris)
```

```
#set seed to ensure reproducible results
set.seed(42)

#split into training and test sets
iris[, "train"] <- ifelse(runif(nrow(iris)) < 0.8, 1, 0)

#separate training and test sets
trainset <- iris[iris$train == 1,]
testset <- iris[iris$train == 0,]

#get column index of train flag
trainColNum <- grep("train", names(trainset))

#remove train flag column from train and test sets
trainset <- trainset[,-trainColNum]
testset <- testset[,-trainColNum]

dim(trainset)
#> [1] 115   5
dim(testset)
#> [1] 35  5
```

7.2.2 Build the SVM model

```
#get column index of predicted variable in dataset
typeColNum <- grep("Species", names(iris))

#build model - linear kernel and C-classification (soft margin) with default cost (C=1)
svm_model <- svm(Species~ ., data = trainset,
                 method = "C-classification",
                 kernel = "linear")
svm_model
#>
#> Call:
#> svm(formula = Species ~ ., data = trainset, method = "C-classification",
#>     kernel = "linear")
#>
#>
#> Parameters:
#>    SVM-Type:  C-classification
#>  SVM-Kernel:  linear
#>        cost:  1
#>       gamma:  0.25
#>
#> Number of Support Vectors:  24
```

The output from the SVM model show that there are 24 support vectors. If desired, these can be examined using the SV variable in the model – i.e via svm_model$SV.

7.2.3 Support Vectors

```
# support vectors
svm_model$SV
#>     Sepal.Length Sepal.Width Petal.Length Petal.Width
#> 19       -0.2564      1.7668       -1.323      -1.305
#> 42       -1.7006     -1.7045       -1.559      -1.305
#> 45       -0.9785      1.7668       -1.205      -1.171
#> 53        1.1878      0.1469        0.568       0.309
#> 55        0.7064     -0.5474        0.390       0.309
#> 57        0.4657      0.6097        0.450       0.443
#> 58       -1.2192     -1.4730       -0.378      -0.364
#> 69        0.3453     -1.9359        0.331       0.309
#> 71       -0.0157      0.3783        0.509       0.712
#> 73        0.4657     -1.2416        0.568       0.309
#> 78        0.9471     -0.0845        0.627       0.578
#> 84        0.1046     -0.7788        0.686       0.443
#> 85       -0.6174     -0.0845        0.331       0.309
#> 86        0.1046      0.8412        0.331       0.443
#> 99       -0.9785     -1.2416       -0.555      -0.229
#> 107      -1.2192     -1.2416        0.331       0.578
#> 111       0.7064      0.3783        0.686       0.981
#> 117       0.7064     -0.0845        0.922       0.712
#> 124       0.4657     -0.7788        0.568       0.712
#> 130       1.5488     -0.0845        1.099       0.443
#> 138       0.5860      0.1469        0.922       0.712
#> 139       0.1046     -0.0845        0.509       0.712
#> 147       0.4657     -1.2416        0.627       0.847
#> 150      -0.0157     -0.0845        0.686       0.712
```

The test prediction accuracy indicates that the linear performs quite well on this dataset, confirming that it is indeed near linearly separable. To check performance by class, one can create a confusion matrix as described in my post on random forests. I'll leave this as an exercise for you. Another point is that we have used a soft-margin classification scheme with a cost C=1. You can experiment with this by explicitly changing the value of C. Again, I'll leave this for you an exercise.

7.2.4 Predictions on training model

```
# training set predictions
pred_train <- predict(svm_model, trainset)
mean(pred_train == trainset$Species)
#> [1] 0.983
# [1] 0.9826087
```

7.2.5 Predictions on test model

```
# test set predictions
pred_test <-predict(svm_model, testset)
mean(pred_test == testset$Species)
#> [1] 0.914
# [1] 0.9142857
```

7.2.6 Confusion matrix and Accuracy

```
# confusion matrix
cm <- table(pred_test, testset$Species)
cm
#>
#> pred_test    setosa versicolor virginica
#>   setosa         18          0         0
#>   versicolor      0          5         3
#>   virginica       0          0         9
```

```
# accuracy
sum(diag(cm)) / sum(cm)
#> [1] 0.914
```

7.3 SVM with Radial Basis Function kernel. Linear

7.3.1 Training and test sets

```
#load required library (assuming e1071 is already loaded)
library(mlbench)

#load Sonar dataset
data(Sonar)
#set seed to ensure reproducible results
set.seed(42)
#split into training and test sets
Sonar[, "train"] <- ifelse(runif(nrow(Sonar))<0.8,1,0)

#separate training and test sets
trainset <- Sonar[Sonar$train==1,]
testset <- Sonar[Sonar$train==0,]

#get column index of train flag
trainColNum <- grep("train",names(trainset))
#remove train flag column from train and test sets
trainset <- trainset[,-trainColNum]
testset <- testset[,-trainColNum]

#get column index of predicted variable in dataset
typeColNum <- grep("Class",names(Sonar))
```

7.3.2 Predictions on Training model

```
#build model - linear kernel and C-classification with default cost (C=1)
svm_model <- svm(Class~ ., data=trainset,
                 method="C-classification",
                 kernel="linear")
```

```
#training set predictions
pred_train <-predict(svm_model,trainset)
mean(pred_train==trainset$Class)
#> [1] 0.97
```

7.3.3 Predictions on test model

```
#test set predictions
pred_test <-predict(svm_model,testset)
mean(pred_test==testset$Class)
#> [1] 0.605
```

I'll leave you to examine the contents of the model. The important point to note here is that the performance of the model with the test set is quite dismal compared to the previous case. This simply indicates that the linear kernel is not appropriate here. Let's take a look at what happens if we use the RBF kernel with default values for the parameters:

7.4 SVM with Radial Basis Function kernel. Non-linear

7.4.1 Predictions on training model

```
#build model: radial kernel, default params
svm_model <- svm(Class~ ., data=trainset,
                 method="C-classification",
                 kernel="radial")
# print params
svm_model$cost
#> [1] 1
svm_model$gamma
#> [1] 0.0167

#training set predictions
pred_train <-predict(svm_model,trainset)
mean(pred_train==trainset$Class)
#> [1] 0.988
```

7.4.2 Predictions on test model

```
#test set predictions
pred_test <-predict(svm_model,testset)
mean(pred_test==testset$Class)
#> [1] 0.767
```

That's a pretty decent improvement from the linear kernel. Let's see if we can do better by doing some parameter tuning. To do this we first invoke tune.svm and use the parameters it gives us in the call to svm:

7.4.3 Tuning of parameters

```
# find optimal parameters in a specified range
tune_out <- tune.svm(x = trainset[,-typeColNum],
                     y = trainset[, typeColNum],
                     gamma = 10^(-3:3),
                     cost = c(0.01, 0.1, 1, 10, 100, 1000),
                     kernel = "radial")

#print best values of cost and gamma
tune_out$best.parameters$cost
#> [1] 10
tune_out$best.parameters$gamma
#> [1] 0.01

#build model
svm_model <- svm(Class~ ., data = trainset,
                 method = "C-classification",
                 kernel = "radial",
                 cost = tune_out$best.parameters$cost,
                 gamma = tune_out$best.parameters$gamma)
```

7.4.4 Prediction on training model with new parameters

```
# training set predictions
pred_train <-predict(svm_model,trainset)
mean(pred_train==trainset$Class)
#> [1] 1
```

7.4.5 Prediction on test model with new parameters

```
# test set predictions
pred_test <-predict(svm_model,testset)
mean(pred_test==testset$Class)
#> [1] 0.814
```

Which is fairly decent improvement on the un-optimised case.

7.5 Wrapping up

This bring us to the end of this introductory exploration of SVMs in R. To recap, the distinguishing feature of SVMs in contrast to most other techniques is that they attempt to construct optimal separation boundaries between different categories.

SVMs are quite versatile and have been applied to a wide variety of domains ranging from chemistry to pattern recognition. They are best used in binary classification scenarios. This brings up a question as to where SVMs are to be preferred to other binary classification techniques such as logistic regression. The honest response is, "it depends" – but here are some points to keep in mind when choosing between the two. A general point to keep in mind is that SVM algorithms tend to be expensive both in terms of memory and computation, issues that can start to hurt as the size of the dataset increases.

Given all the above caveats and considerations, the best way to figure out whether an SVM approach will work for your problem may be to do what most machine learning practitioners do: try it out!

Chapter 8

SMS spam. Naive Bayes. Classification

Dataset: https://github.com/stedy/Machine-Learning-with-R-datasets/blob/master/sms_spam.csv

Instructions: Machine Learning with R. Page 104.

```
library(tictoc)
```

```
sms_raw <- read.csv(file.path(data_raw_dir, "sms_spam.csv"), stringsAsFactors = FALSE)
```

```
str(sms_raw)
#> 'data.frame':    5574 obs. of  2 variables:
#>  $ type: chr  "ham" "ham" "spam" "ham" ...
#>  $ text: chr  "Go until jurong point, crazy.. Available only in bugis n great world la e buffet... C
```

8.0.1 convert type to a factor

```
sms_raw$type <- factor(sms_raw$type)
```

```
str(sms_raw$type)
#>  Factor w/ 2 levels "ham","spam": 1 1 2 1 1 2 1 1 2 2 ...
```

How many email of type ham or spam:

```
table(sms_raw$type)
#>
#>  ham spam
#> 4827  747
```

Create the corpus:

```
library(tm)
#> Loading required package: NLP

sms_corpus <- VCorpus(VectorSource(sms_raw$text))
print(sms_corpus)
#> <<VCorpus>>
#> Metadata:  corpus specific: 0, document level (indexed): 0
#> Content:  documents: 5574
```

Let's see a couple of documents:

```
inspect(sms_corpus[1:2])
#> <<VCorpus>>
#> Metadata:  corpus specific: 0, document level (indexed): 0
#> Content:  documents: 2
#>
#> [[1]]
#> <<PlainTextDocument>>
#> Metadata:  7
#> Content:  chars: 111
#>
#> [[2]]
#> <<PlainTextDocument>>
#> Metadata:  7
#> Content:  chars: 29
```

```
# show some text
as.character(sms_corpus[[1]])
#> [1] "Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there
```

```
# show three documents
lapply(sms_corpus[1:3], as.character)
#> $`1`
#> [1] "Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there
#>
#> $`2`
#> [1] "Ok lar... Joking wif u oni..."
#>
#> $`3`
#> [1] "Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive
```

8.1 Some conversion

```
# convert to lowercase
sms_corpus_clean <- tm_map(sms_corpus, content_transformer(tolower))
```

```
as.character(sms_corpus[[1]])
#> [1] "Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there
```

```
# converted to lowercase
as.character(sms_corpus_clean[[1]])
#> [1] "go until jurong point, crazy.. available only in bugis n great world la e buffet... cine there
```

```
# remove numbers
sms_corpus_clean <- tm_map(sms_corpus_clean, removeNumbers)
```

What transformations are available

```
# what transformations are available
getTransformations()
#> [1] "removeNumbers"     "removePunctuation" "removeWords"
#> [4] "stemDocument"      "stripWhitespace"
```

```
# remove stop words
sms_corpus_clean <- tm_map(sms_corpus_clean, removeWords, stopwords())
```

```
# remove punctuation
sms_corpus_clean <- tm_map(sms_corpus_clean, removePunctuation)
```

Stemming:

```
library(SnowballC)
wordStem(c("learn", "learned", "learning", "learns"))
#> [1] "learn" "learn" "learn" "learn"
```

```
# stemming corpus
sms_corpus_clean <- tm_map(sms_corpus_clean, stemDocument)
```

```
# remove white spaces
sms_corpus_clean <- tm_map(sms_corpus_clean, stripWhitespace)
```

Show what we've got so far

```
# show what we've got so far
lapply(sms_corpus[1:3], as.character)
#> $`1`
#> [1] "Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there
#>
#> $`2`
#> [1] "Ok lar... Joking wif u oni..."
#>
#> $`3`
#> [1] "Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive

lapply(sms_corpus_clean[1:3], as.character)
#> $`1`
#> [1] "go jurong point crazi avail bugi n great world la e buffet cine got amor wat"
#>
#> $`2`
#> [1] "ok lar joke wif u oni"
#>
#> $`3`
#> [1] "free entri wkli comp win fa cup final tkts st may text fa receiv entri questionstd txt ratetc a
```

8.2 Convert to Document Term Matrix (dtm

)

```
sms_dtm <- DocumentTermMatrix(sms_corpus_clean)
sms_dtm
#> <<DocumentTermMatrix (documents: 5574, terms: 6592)>>
#> Non-/sparse entries: 42608/36701200
#> Sparsity           : 100%
#> Maximal term length: 40
#> Weighting          : term frequency (tf)
```

8.3 split in training and test datasets

```
sms_dtm_train <- sms_dtm[1:4169, ]
sms_dtm_test  <- sms_dtm[4170:5559, ]
```

8.3.1 separate the labels

```
sms_train_labels <- sms_raw[1:4169, ]$type
sms_test_labels  <- sms_raw[4170:5559, ]$type
```

```
prop.table(table(sms_train_labels))
#> sms_train_labels
#>   ham  spam
#> 0.865 0.135
```

```
prop.table(table(sms_test_labels))
#> sms_test_labels
#>  ham spam
#> 0.87 0.13
```

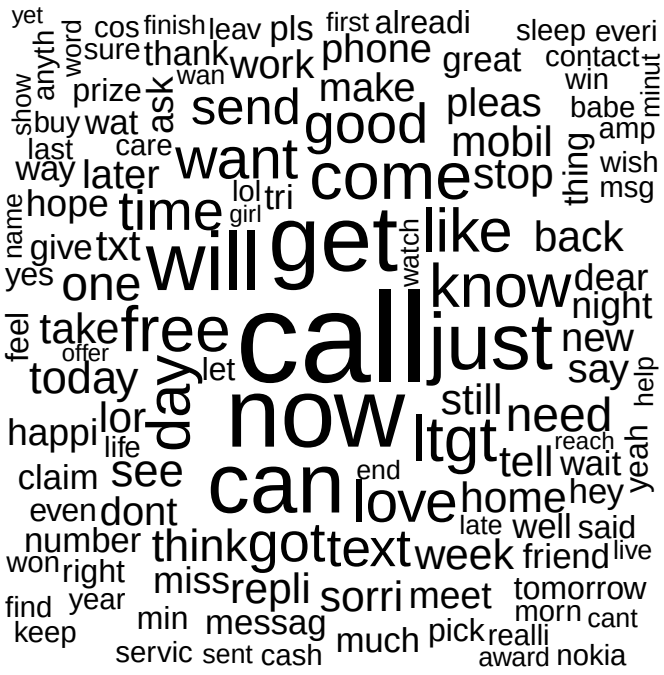
```
# convert dtm to matrix
sms_mat_train <- as.matrix(t(sms_dtm_train))
dtm.rs <- sort(rowSums(sms_mat_train), decreasing=TRUE)

# dataframe with word-frequency
dtm.df <- data.frame(word = names(dtm.rs), freq = as.integer(dtm.rs),
                     stringsAsFactors = FALSE)
```

8.4 plot wordcloud

```
library(wordcloud)
#> Loading required package: RColorBrewer
wordcloud(sms_corpus_clean, min.freq = 50, random.order = FALSE)
#> Warning in wordcloud(sms_corpus_clean, min.freq = 50, random.order =
#> FALSE): tone could not be fit on page. It will not be plotted.
#> Warning in wordcloud(sms_corpus_clean, min.freq = 50, random.order =
#> FALSE): also could not be fit on page. It will not be plotted.
#> Warning in wordcloud(sms_corpus_clean, min.freq = 50, random.order =
#> FALSE): look could not be fit on page. It will not be plotted.
#> Warning in wordcloud(sms_corpus_clean, min.freq = 50, random.order =
#> FALSE): start could not be fit on page. It will not be plotted.
#> Warning in wordcloud(sms_corpus_clean, min.freq = 50, random.order =
#> FALSE): smile could not be fit on page. It will not be plotted.
#> Warning in wordcloud(sms_corpus_clean, min.freq = 50, random.order =
#> FALSE): urgent could not be fit on page. It will not be plotted.
#> Warning in wordcloud(sms_corpus_clean, min.freq = 50, random.order =
#> FALSE): use could not be fit on page. It will not be plotted.
#> Warning in wordcloud(sms_corpus_clean, min.freq = 50, random.order =
#> FALSE): someth could not be fit on page. It will not be plotted.
#> Warning in wordcloud(sms_corpus_clean, min.freq = 50, random.order =
```

```
#> FALSE): place could not be fit on page. It will not be plotted.
#> Warning in wordcloud(sms_corpus_clean, min.freq = 50, random.order =
#> FALSE): gud could not be fit on page. It will not be plotted.
#> Warning in wordcloud(sms_corpus_clean, min.freq = 50, random.order =
#> FALSE): guy could not be fit on page. It will not be plotted.
#> Warning in wordcloud(sms_corpus_clean, min.freq = 50, random.order =
#> FALSE): custom could not be fit on page. It will not be plotted.
#> Warning in wordcloud(sms_corpus_clean, min.freq = 50, random.order =
#> FALSE): next could not be fit on page. It will not be plotted.
#> Warning in wordcloud(sms_corpus_clean, min.freq = 50, random.order =
#> FALSE): person could not be fit on page. It will not be plotted.
#> Warning in wordcloud(sms_corpus_clean, min.freq = 50, random.order =
#> FALSE): someon could not be fit on page. It will not be plotted.
#> Warning in wordcloud(sms_corpus_clean, min.freq = 50, random.order =
#> FALSE): tonight could not be fit on page. It will not be plotted.
#> Warning in wordcloud(sms_corpus_clean, min.freq = 50, random.order =
#> FALSE): went could not be fit on page. It will not be plotted.
#> Warning in wordcloud(sms_corpus_clean, min.freq = 50, random.order =
#> FALSE): around could not be fit on page. It will not be plotted.
#> Warning in wordcloud(sms_corpus_clean, min.freq = 50, random.order =
#> FALSE): talk could not be fit on page. It will not be plotted.
#> Warning in wordcloud(sms_corpus_clean, min.freq = 50, random.order =
#> FALSE): chat could not be fit on page. It will not be plotted.
#> Warning in wordcloud(sms_corpus_clean, min.freq = 50, random.order =
#> FALSE): money could not be fit on page. It will not be plotted.
#> Warning in wordcloud(sms_corpus_clean, min.freq = 50, random.order =
#> FALSE): collect could not be fit on page. It will not be plotted.
#> Warning in wordcloud(sms_corpus_clean, min.freq = 50, random.order =
#> FALSE): mani could not be fit on page. It will not be plotted.
#> Warning in wordcloud(sms_corpus_clean, min.freq = 50, random.order =
#> FALSE): per could not be fit on page. It will not be plotted.
#> Warning in wordcloud(sms_corpus_clean, min.freq = 50, random.order =
#> FALSE): soon could not be fit on page. It will not be plotted.
#> Warning in wordcloud(sms_corpus_clean, min.freq = 50, random.order =
#> FALSE): gonna could not be fit on page. It will not be plotted.
#> Warning in wordcloud(sms_corpus_clean, min.freq = 50, random.order =
#> FALSE): plan could not be fit on page. It will not be plotted.
#> Warning in wordcloud(sms_corpus_clean, min.freq = 50, random.order =
#> FALSE): alway could not be fit on page. It will not be plotted.
#> Warning in wordcloud(sms_corpus_clean, min.freq = 50, random.order =
#> FALSE): nice could not be fit on page. It will not be plotted.
#> Warning in wordcloud(sms_corpus_clean, min.freq = 50, random.order =
#> FALSE): check could not be fit on page. It will not be plotted.
#> Warning in wordcloud(sms_corpus_clean, min.freq = 50, random.order =
#> FALSE): dun could not be fit on page. It will not be plotted.
#> Warning in wordcloud(sms_corpus_clean, min.freq = 50, random.order =
#> FALSE): special could not be fit on page. It will not be plotted.
#> Warning in wordcloud(sms_corpus_clean, min.freq = 50, random.order =
#> FALSE): told could not be fit on page. It will not be plotted.
#> Warning in wordcloud(sms_corpus_clean, min.freq = 50, random.order =
#> FALSE): box could not be fit on page. It will not be plotted.
#> Warning in wordcloud(sms_corpus_clean, min.freq = 50, random.order =
#> FALSE): lot could not be fit on page. It will not be plotted.
```

```
#> Warning in wordcloud(sms_corpus_clean, min.freq = 50, random.order =
#> FALSE): shop could not be fit on page. It will not be plotted.
#> Warning in wordcloud(sms_corpus_clean, min.freq = 50, random.order =
#> FALSE): hello could not be fit on page. It will not be plotted.
#> Warning in wordcloud(sms_corpus_clean, min.freq = 50, random.order =
#> FALSE): hour could not be fit on page. It will not be plotted.
#> Warning in wordcloud(sms_corpus_clean, min.freq = 50, random.order =
#> FALSE): mean could not be fit on page. It will not be plotted.
#> Warning in wordcloud(sms_corpus_clean, min.freq = 50, random.order =
#> FALSE): month could not be fit on page. It will not be plotted.
#> Warning in wordcloud(sms_corpus_clean, min.freq = 50, random.order =
#> FALSE): guarante could not be fit on page. It will not be plotted.
#> Warning in wordcloud(sms_corpus_clean, min.freq = 50, random.order =
#> FALSE): peopl could not be fit on page. It will not be plotted.
#> Warning in wordcloud(sms_corpus_clean, min.freq = 50, random.order =
#> FALSE): happen could not be fit on page. It will not be plotted.
#> Warning in wordcloud(sms_corpus_clean, min.freq = 50, random.order =
#> FALSE): thk could not be fit on page. It will not be plotted.
```



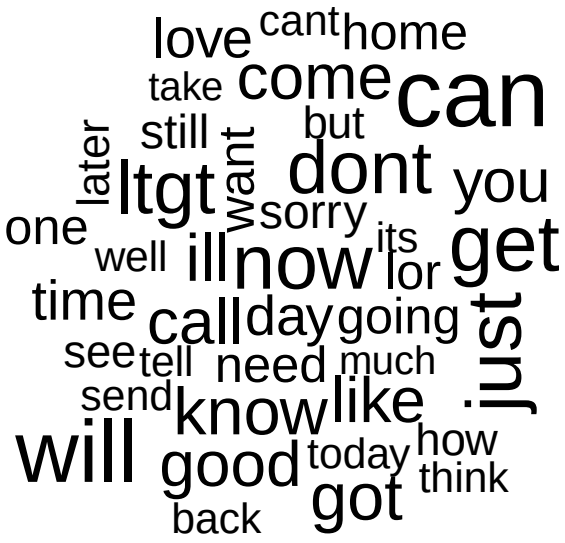

```
spam <- subset(sms_raw, type == "spam")
ham  <- subset(sms_raw, type == "ham")
```

Words related to **spam**

```
wordcloud(spam$text, max.words = 40, scale = c(3, 0.5))
#> Warning in tm_map.SimpleCorpus(corpus, tm::removePunctuation):
#> transformation drops documents
#> Warning in tm_map.SimpleCorpus(corpus, function(x) tm::removeWords(x,
#> tm::stopwords())): transformation drops documents
```

Words related to **ham**

```
wordcloud(ham$text, max.words = 40, scale = c(3, 0.5))
#> Warning in tm_map.SimpleCorpus(corpus, tm::removePunctuation):
#> transformation drops documents
#> Warning in tm_map.SimpleCorpus(corpus, function(x) tm::removeWords(x,
#> tm::stopwords())): transformation drops documents
```



8.5 Limit Frequent words

```
# words that appear at least in 5 messages
sms_freq_words <- findFreqTerms(sms_dtm_train, 6)
```

```
str(sms_freq_words)
#>  chr [1:997] "abiola" "abl" "abt" "accept" "access" "account" "across" ...
```

8.5.1 get only frequent words

```
sms_dtm_freq_train<- sms_dtm_train[ , sms_freq_words]
sms_dtm_freq_test <- sms_dtm_test[ , sms_freq_words]
```

8.5.2 function to change value to Yes/No

```
convert_counts <- function(x) {
    x <- ifelse(x > 0, "Yes", "No")
  }
```

```
# change from number to Yes/No
# also the result returns a matrix
sms_train <- apply(sms_dtm_freq_train, MARGIN = 2,
                                        convert_counts)
sms_test  <- apply(sms_dtm_freq_test, MARGIN = 2,
                                       convert_counts)
```

```
# matrix of
# 4169 documents as rows
# 1159 terms as columns
dim(sms_train)
#> [1] 4169  997
length(sms_train_labels)
#> [1] 4169
```

```
# this is how the matrix looks
sms_train[1:10, 10:15]
#>     Terms
#> Docs add  address admir advanc aft  afternoon
#>   1  "No" "No"    "No"  "No"   "No" "No"
#>   2  "No" "No"    "No"  "No"   "No" "No"
#>   3  "No" "No"    "No"  "No"   "No" "No"
#>   4  "No" "No"    "No"  "No"   "No" "No"
#>   5  "No" "No"    "No"  "No"   "No" "No"
#>   6  "No" "No"    "No"  "No"   "No" "No"
#>   7  "No" "No"    "No"  "No"   "No" "No"
#>   8  "No" "No"    "No"  "No"   "No" "No"
#>   9  "No" "No"    "No"  "No"   "No" "No"
#>   10 "No" "No"    "No"  "No"   "No" "No"
```

```
library(e1071)
sms_classifier <- naiveBayes(sms_train, sms_train_labels)
```

```
tic()
sms_test_pred <- predict(sms_classifier, sms_test)
toc()
#> 20.346 sec elapsed
```

```
library(gmodels)
CrossTable(sms_test_pred, sms_test_labels,
    prop.chisq = FALSE, prop.t = FALSE,
    dnn = c('predicted', 'actual'))
#>
#>
#>    Cell Contents
#> |-------------------------|
#> |                       N |
#> |           N / Row Total |
#> |           N / Col Total |
```

```
#> |-------------------------|
#>
#>
#> Total Observations in Table:  1390
#>
#>
#>              | actual
#>    predicted |       ham |      spam | Row Total |
#> -------------|-----------|-----------|-----------|
#>          ham |      1202 |        21 |      1223 |
#>              |     0.983 |     0.017 |     0.880 |
#>              |     0.994 |     0.116 |           |
#> -------------|-----------|-----------|-----------|
#>         spam |         7 |       160 |       167 |
#>              |     0.042 |     0.958 |     0.120 |
#>              |     0.006 |     0.884 |           |
#> -------------|-----------|-----------|-----------|
#> Column Total |      1209 |       181 |      1390 |
#>              |     0.870 |     0.130 |           |
#> -------------|-----------|-----------|-----------|
#>
#>
```

Misclassified: 20+9 (frequency = 5) 25+7 (freq=4) 23+7 (freq=3) 25+8 (freq=2) 21+7 (freq=6)

Decreasing the minimum word frequency doesn't make the model better.

8.6 Improve model performance

```
sms_classifier2 <- naiveBayes(sms_train, sms_train_labels,
                              laplace = 1)
```

```
tic()
sms_test_pred2 <- predict(sms_classifier2, sms_test)
toc()
#> 20.755 sec elapsed
```

```
CrossTable(sms_test_pred2, sms_test_labels,
    prop.chisq = FALSE, prop.t = FALSE, prop.r = FALSE,
    dnn = c('predicted', 'actual'))
#>
#>
#>    Cell Contents
#> |-------------------------|
#> |                       N |
#> |           N / Col Total |
#> |-------------------------|
#>
#>
#> Total Observations in Table:  1390
#>
#>
#>              | actual
```

```
#>    predicted |       ham |      spam | Row Total |
#> -------------|-----------|-----------|-----------|
#>          ham |      1203 |        28 |      1231 |
#>              |     0.995 |     0.155 |           |
#> -------------|-----------|-----------|-----------|
#>         spam |         6 |       153 |       159 |
#>              |     0.005 |     0.845 |           |
#> -------------|-----------|-----------|-----------|
#> Column Total |      1209 |       181 |      1390 |
#>              |     0.870 |     0.130 |           |
#> -------------|-----------|-----------|-----------|
#>
#>
```

Misclassified: 28+7

Chapter 9

Classification Tree: Vehicle example

- Dataset: Vehicle (mlbench)
- Instructions: book "Applied Predictive Modeling Techniques", Lewis, N.D.

9.1 Load packages

```
library(tree)
library(mlbench)

data(Vehicle)
str(Vehicle)
#> 'data.frame':    846 obs. of  19 variables:
#>  $ Comp        : num  95 91 104 93 85 107 97 90 86 93 ...
#>  $ Circ        : num  48 41 50 41 44 57 43 43 34 44 ...
#>  $ D.Circ      : num  83 84 106 82 70 106 73 66 62 98 ...
#>  $ Rad.Ra      : num  178 141 209 159 205 172 173 157 140 197 ...
#>  $ Pr.Axis.Ra  : num  72 57 66 63 103 50 65 65 61 62 ...
#>  $ Max.L.Ra    : num  10 9 10 9 52 6 6 9 7 11 ...
#>  $ Scat.Ra     : num  162 149 207 144 149 255 153 137 122 183 ...
#>  $ Elong       : num  42 45 32 46 45 26 42 48 54 36 ...
#>  $ Pr.Axis.Rect: num  20 19 23 19 19 28 19 18 17 22 ...
#>  $ Max.L.Rect  : num  159 143 158 143 144 169 143 146 127 146 ...
#>  $ Sc.Var.Maxis: num  176 170 223 160 241 280 176 162 141 202 ...
#>  $ Sc.Var.maxis: num  379 330 635 309 325 957 361 281 223 505 ...
#>  $ Ra.Gyr      : num  184 158 220 127 188 264 172 164 112 152 ...
#>  $ Skew.Maxis  : num  70 72 73 63 127 85 66 67 64 64 ...
#>  $ Skew.maxis  : num  6 9 14 6 9 5 13 3 2 4 ...
#>  $ Kurt.maxis  : num  16 14 9 10 11 9 1 3 14 14 ...
#>  $ Kurt.Maxis  : num  187 189 188 199 180 181 200 193 200 195 ...
#>  $ Holl.Ra     : num  197 199 196 207 183 183 204 202 208 204 ...
#>  $ Class       : Factor w/ 4 levels "bus","opel","saab",..: 4 4 3 4 1 1 1 4 4 3 ...
```

```
summary(Vehicle[1])
#>       Comp
#>  Min.   : 73.0
#>  1st Qu.: 87.0
#>  Median : 93.0
```

```
#>  Mean   : 93.7
#>  3rd Qu.:100.0
#>  Max.   :119.0
```

```
summary(Vehicle[2])
#>       Circ
#>  Min.   :33.0
#>  1st Qu.:40.0
#>  Median :44.0
#>  Mean   :44.9
#>  3rd Qu.:49.0
#>  Max.   :59.0
```

```
attributes(Vehicle$Class)
#> $levels
#> [1] "bus"  "opel" "saab" "van"
#>
#> $class
#> [1] "factor"
```

9.2 Prepare data

```
set.seed(107)
N = nrow(Vehicle)
train <- sample(1:N, 500, FALSE)
```

```
# training and test sets
trainset <- Vehicle[train,]
testset  <- Vehicle[-train,]
```

9.3 Estimate the decision tree

```
fit <- tree(Class ~., data = trainset, split = "deviance")
fit
#> node), split, n, deviance, yval, (yprob)
#>       * denotes terminal node
#>
#>   1) root 500 1000 opel ( 0 0 0 0 )
#>     2) Elong < 41.5 215  500 saab ( 0 0 0 0 )
#>       4) Max.L.Ra < 7.5 51   50 bus ( 1 0 0 0 )
#>         8) Comp < 93.5 12   20 bus ( 0 0 0 0 )
#>          16) Pr.Axis.Ra < 67.5 7    8 saab ( 0 0 1 0 ) *
#>          17) Pr.Axis.Ra > 67.5 5    0 bus ( 1 0 0 0 ) *
#>         9) Comp > 93.5 39    9 bus ( 1 0 0 0 ) *
#>       5) Max.L.Ra > 7.5 164  200 opel ( 0 1 0 0 )
#>        10) Sc.Var.maxis < 723 149  200 saab ( 0 0 1 0 )
#>          20) Comp < 109.5 137  200 opel ( 0 1 0 0 ) *
#>          21) Comp > 109.5 12    0 saab ( 0 0 1 0 ) *
#>        11) Sc.Var.maxis > 723 15    7 opel ( 0 1 0 0 ) *
#>     3) Elong > 41.5 285  700 van ( 0 0 0 0 )
```

```
#>       6) Sc.Var.maxis < 305.5 116  200 van ( 0 0 0 1 )
#>        12) Max.L.Rect < 128.5 40   90 saab ( 0 0 0 0 )
#>          24) Scat.Ra < 120.5 15   30 van ( 0 0 0 1 ) *
#>          25) Scat.Ra > 120.5 25   30 saab ( 0 0 1 0 ) *
#>        13) Max.L.Rect > 128.5 76   90 van ( 0 0 0 1 )
#>          26) Max.L.Rect < 138.5 38   60 van ( 0 0 0 1 )
#>            52) Circ < 37.5 17   10 van ( 0 0 0 1 ) *
#>            53) Circ > 37.5 21   40 opel ( 0 0 0 0 ) *
#>          27) Max.L.Rect > 138.5 38   20 van ( 0 0 0 1 ) *
#>       7) Sc.Var.maxis > 305.5 169  400 bus ( 0 0 0 0 )
#>        14) Max.L.Ra < 8.5 116  200 bus ( 1 0 0 0 )
#>          28) D.Circ < 76.5 97  100 bus ( 1 0 0 0 )
#>            56) Skew.maxis < 10.5 87   70 bus ( 1 0 0 0 )
#>             112) Max.L.Rect < 134.5 12   20 bus ( 0 0 0 0 ) *
#>             113) Max.L.Rect > 134.5 75   20 bus ( 1 0 0 0 ) *
#>            57) Skew.maxis > 10.5 10   20 opel ( 0 0 0 0 ) *
#>          29) D.Circ > 76.5 19   30 opel ( 0 1 0 0 ) *
#>        15) Max.L.Ra > 8.5 53   20 van ( 0 0 0 1 ) *
```

```
# fit <- tree(Class ~., data = Vehicle[train,], split ="deviance")
# fit
```

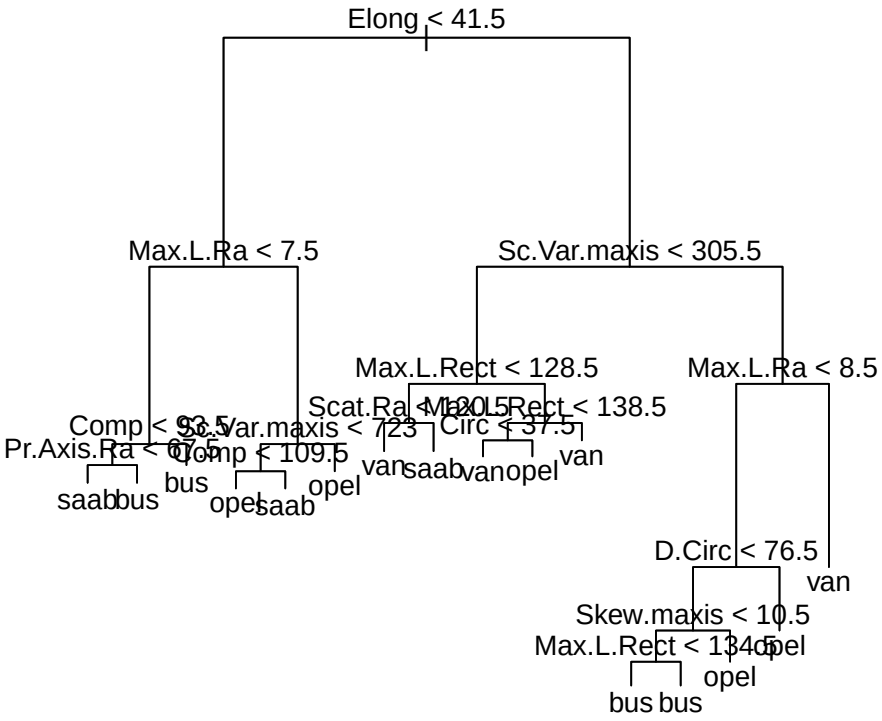
We use deviance as the splitting criteria, a common alternative is to use split="gini".

At each branch of the tree (after root) we see in order: 1. The branch number (e.g. in this case 1,2,14 and 15); 2. the split (e.g. Elong < 41.5); 3. the number of samples going along that split (e.g. 229); 4. the deviance associated with that split (e.g. 489.1); 5. the predicted class (e.g. opel); 6. the associated probabilities (e.g. (0.222707 0.410480 0.366812 0.000000); 7. and for a terminal node (or leaf), the symbol "*".

```
summary(fit)
#>
#> Classification tree:
#> tree(formula = Class ~ ., data = trainset, split = "deviance")
#> Variables actually used in tree construction:
#>  [1] "Elong"        "Max.L.Ra"     "Comp"         "Pr.Axis.Ra"
#>  [5] "Sc.Var.maxis" "Max.L.Rect"   "Scat.Ra"      "Circ"
#>  [9] "D.Circ"       "Skew.maxis"
#> Number of terminal nodes:  16
#> Residual mean deviance:  0.943 = 456 / 484
#> Misclassification error rate: 0.252 = 126 / 500
```

Notice that summary(fit) shows: 1. The type of tree, in this case a Classification tree; 2. the formula used to fit the tree; 3. the variables used to fit the tree; 4. the number of terminal nodes in this case 15; 5. the residual mean deviance - 0.9381; 6. the misclassification error rate 0.232 or 23.2%.

```
plot(fit); text(fit)
```



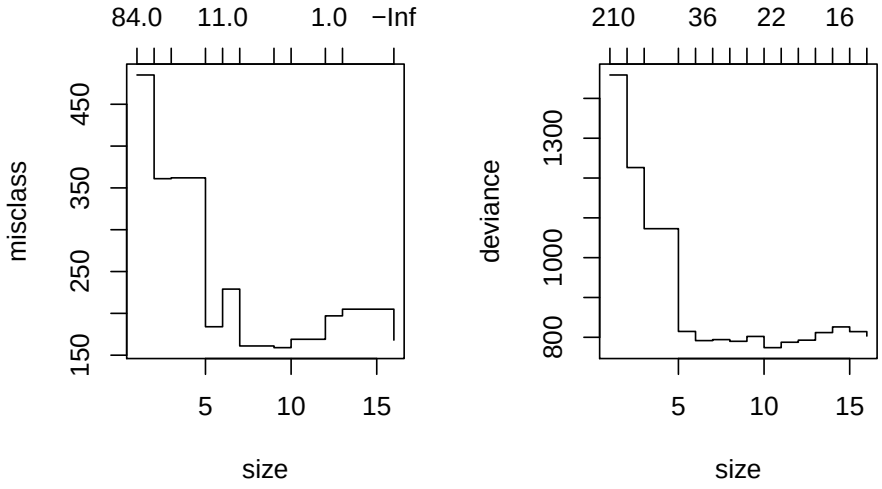
9.4 Assess model

Unfortunately, classification trees have a tendency to overfit the data. One approach to reduce this risk is to use cross-validation. For each hold out sample we fit the model and note at what level the tree gives the best results (using deviance or the misclassification rate). Then we hold out a different sample and repeat. This can be carried out using the `cv.tree()` function. We use a leave-one-out cross-validation using the misclassification rate and deviance (`FUN=prune.misclass`, followed by `FUN=prune.tree`).

```
fitM.cv <- cv.tree(fit, K=346, FUN = prune.misclass)
fitP.cv <- cv.tree(fit, K=346, FUN = prune.tree)
```

The results are plotted out side by side in Figure 1.2. The jagged lines shows where the minimum deviance / misclassification occurred with the cross-validated tree. Since the cross validated misclassification and deviance both reach their minimum close to the number of branches in the original fitted tree there is little to be gained from pruning this tree

```
par(mfrow = c(1, 2))
plot(fitM.cv)
plot(fitP.cv)
```



9.5 Make predictions

We use the validation data set and the fitted decision tree to predict vehicle classes; then we display the confusion matrix and calculate the error rate of the fitted tree. Overall, the model has an error rate of 32%.

```
testLabels <- Vehicle$Class[-train]
testLabels
#>   [1] van  bus  bus  van  van  bus  bus  saab opel bus  van  saab van  saab
#>  [15] saab van  saab opel van  saab saab saab bus  bus  saab opel bus  opel
#>  [29] bus  opel van  opel opel saab saab bus  bus  bus  van  van  saab opel
#>  [43] bus  opel van  opel saab bus  van  bus  opel van  saab bus  opel bus
#>  [57] opel opel van  bus  van  saab opel bus  van  saab opel opel saab saab
#>  [71] saab opel bus  van  bus  opel bus  saab bus  bus  bus  opel opel van
#>  [85] saab bus  bus  bus  van  saab opel van  van  bus  bus  opel bus  opel
#>  [99] saab opel bus  opel bus  saab van  van  saab saab bus  van  opel van
#> [113] saab opel saab saab van  van  van  van  bus  bus  opel bus  bus  van
#> [127] saab bus  opel bus  bus  bus  bus  opel van  saab saab bus  opel van
#> [141] bus  saab bus  van  bus  opel van  saab opel saab opel van  saab van
#> [155] saab opel bus  van  bus  saab saab opel opel bus  bus  opel van  van
#> [169] bus  van  van  saab bus  saab opel saab opel bus  bus  bus  saab bus
#> [183] opel opel saab saab saab van  van  opel opel van  van  opel bus  saab
#> [197] bus  van  opel opel bus  bus  bus  opel saab opel van  bus  opel opel
#> [211] saab opel bus  opel opel opel van  opel van  saab saab van  saab saab
#> [225] saab saab van  van  van  saab bus  van  van  bus  saab opel saab saab
#> [239] opel saab saab saab saab van  saab opel bus  saab bus  opel opel opel
#> [253] saab bus  van  opel saab opel bus  bus  saab van  opel bus  saab van
#> [267] opel saab saab saab saab van  opel bus  bus  bus  opel saab saab saab
#> [281] van  saab bus  opel saab van  opel bus  saab saab opel opel van  saab
#> [295] bus  opel bus  van  van  opel bus  bus  saab bus  van  saab bus  van
#> [309] saab van  opel bus  bus  opel saab opel bus  bus  saab van  saab saab
#> [323] bus  opel opel opel bus  saab bus  van  bus  van  saab opel saab van
#> [337] opel opel van  bus  saab saab van  saab opel saab
#> Levels: bus opel saab van
```

```
# Confusion Matrix
pred <- predict(fit, newdata = testset)
# find column whih has the maximum of all rows
```

```
pred.class <- colnames(pred)[max.col(pred, ties.method = c("random"))]
cm <- table(testLabels, pred.class,
      dnn = c("Observed Class", "Predicted Class"))
cm
#>                Predicted Class
#> Observed Class bus opel saab van
#>           bus   85    1    1   5
#>           opel   3   70   10   2
#>           saab   7   67   14   7
#>           van    1    4    5  64
```

```
# Sensitivity
sum(diag(cm)) / sum(cm)
#> [1] 0.673
```

```
# pred <- predict(fit, newdata = Vehicle[-train,])
# pred.class <- colnames(pred)[max.col(pred, ties.method = c("random"))]
# table(Vehicle$Class[-train], pred.class,
#       dnn = c("Observed Class", "Predicted Class"))
```

```
error_rate = (1 - sum(pred.class == testset) / nrow(testset))
round(error_rate, 3)
#> [1] 0.327
```

```
# error_rate = (1 - sum(pred.class == Vehicle$Class[-train])/346)
# round(error_rate,3)
```

Chapter 10

Bike sharing demand

```
#loading the required libraries
library(rpart)
library(rattle)
#> Rattle: A free graphical interface for data science with R.
#> Version 5.2.0 Copyright (c) 2006-2018 Togaware Pty Ltd.
#> Type 'rattle()' to shake, rattle, and roll your data.
library(rpart.plot)
library(RColorBrewer)
library(randomForest)
#> randomForest 4.6-14
#> Type rfNews() to see new features/changes/bug fixes.
#>
#> Attaching package: 'randomForest'
#> The following object is masked from 'package:rattle':
#>
#>     importance
library(corrplot)
#> corrplot 0.84 loaded
library(dplyr)
#>
#> Attaching package: 'dplyr'
#> The following object is masked from 'package:randomForest':
#>
#>     combine
#> The following objects are masked from 'package:stats':
#>
#>     filter, lag
#> The following objects are masked from 'package:base':
#>
#>     intersect, setdiff, setequal, union
library(tictoc)
```

Source: https://www.analyticsvidhya.com/blog/2015/06/solution-kaggle-competition-bike-sharing-demand/

10.1 Hypothesis Generation

Before exploring the data to understand the relationship between variables, I'd recommend you to focus on hypothesis generation first. Now, this might sound counter-intuitive for solving a data science problem, but if there is one thing I have learnt over years, it is this. Before exploring data, you should spend some time thinking about the business problem, gaining the domain knowledge and may be gaining first hand experience of the problem (only if I could travel to North America!)

How does it help? This practice usually helps you form better features later on, which are not biased by the data available in the dataset. At this stage, you are expected to posses structured thinking i.e. a thinking process which takes into consideration all the possible aspects of a particular problem.

Here are some of the hypothesis which I thought could influence the demand of bikes:

- **Hourly trend**: There must be high demand during office timings. Early morning and late evening can have different trend (cyclist) and low demand during 10:00 pm to 4:00 am.
- **Daily Trend**: Registered users demand more bike on weekdays as compared to weekend or holiday.
- **Rain**: The demand of bikes will be lower on a rainy day as compared to a sunny day. Similarly, higher humidity will cause to lower the demand and vice versa.
- **Temperature**: Would high or low temperature encourage or disencourage bike riding?
- **Pollution**: If the pollution level in a city starts soaring, people may start using Bike (it may be influenced by government / company policies or increased awareness).
- **Time**: Total demand should have higher contribution of registered user as compared to casual because registered user base would increase over time.
- **Traffic**: It can be positively correlated with Bike demand. Higher traffic may force people to use bike as compared to other road transport medium like car, taxi etc

10.2 Understanding the Data Set

The dataset shows hourly rental data for two years (2011 and 2012). The training data set is for the **first 19 days of each month**. The test dataset is from **20th day to month's end**. We are required to predict the total count of bikes rented during each hour covered by the test set.

In the training data set, they have separately given bike demand by registered, casual users and sum of both is given as count.

Training data set has 12 variables (see below) and Test has 9 (excluding registered, casual and count).

10.2.1 Independent variables

```
datetime:   date and hour in "mm/dd/yyyy hh:mm" format
season:     Four categories-> 1 = spring, 2 = summer, 3 = fall, 4 = winter
holiday:    whether the day is a holiday or not (1/0)
workingday: whether the day is neither a weekend nor holiday (1/0)
weather:    Four Categories of weather
            1-> Clear, Few clouds, Partly cloudy, Partly cloudy
            2-> Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
            3-> Light Snow and Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
            4-> Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
temp:       hourly temperature in Celsius
atemp:      "feels like" temperature in Celsius
```

```
humidity:   relative humidity
windspeed:  wind speed
```

10.2.2 Dependent variables

```
registered: number of registered user
casual:     number of non-registered user
count:      number of total rentals (registered + casual)
```

10.3 Importing the dataset and Data Exploration

For this solution, I have used R (R Studio 0.99.442) in Windows Environment.

Below are the steps to import and perform data exploration. If you are new to this concept, you can refer this guide on Data Exploration in R

1. Import Train and Test Data Set

```
# https://www.kaggle.com/c/bike-sharing-demand/data
train = read.csv(file.path(data_raw_dir, "bike_train.csv"))
test = read.csv(file.path(data_raw_dir, "bike_test.csv"))
```

```
glimpse(train)
#> Observations: 10,886
#> Variables: 12
#> $ datetime   <fct> 2011-01-01 00:00:00, 2011-01-01 01:00:00, 2011-01-0...
#> $ season     <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
#> $ holiday    <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
#> $ workingday <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
#> $ weather    <int> 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, ...
#> $ temp       <dbl> 9.84, 9.02, 9.02, 9.84, 9.84, 9.84, 9.02, 8.20, 9.8...
#> $ atemp      <dbl> 14.4, 13.6, 13.6, 14.4, 14.4, 12.9, 13.6, 12.9, 14....
#> $ humidity   <int> 81, 80, 80, 75, 75, 75, 80, 86, 75, 76, 76, 81, 77,...
#> $ windspeed  <dbl> 0, 0, 0, 0, 0, 6, 0, 0, 0, 0, 17, 19, 19, 20, 19, 2...
#> $ casual     <int> 3, 8, 5, 3, 0, 0, 2, 1, 1, 8, 12, 26, 29, 47, 35, 4...
#> $ registered <int> 13, 32, 27, 10, 1, 1, 0, 2, 7, 6, 24, 30, 55, 47, 7...
#> $ count      <int> 16, 40, 32, 13, 1, 1, 2, 3, 8, 14, 36, 56, 84, 94, ...
glimpse(test)
#> Observations: 6,493
#> Variables: 9
#> $ datetime   <fct> 2011-01-20 00:00:00, 2011-01-20 01:00:00, 2011-01-2...
#> $ season     <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
#> $ holiday    <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
#> $ workingday <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
#> $ weather    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 2, 2, 2, 2, 2, ...
#> $ temp       <dbl> 10.66, 10.66, 10.66, 10.66, 10.66, 9.84, 9.02, 9.02...
#> $ atemp      <dbl> 11.4, 13.6, 13.6, 12.9, 12.9, 11.4, 10.6, 10.6, 10....
#> $ humidity   <int> 56, 56, 56, 56, 56, 60, 60, 55, 55, 52, 48, 45, 42,...
#> $ windspeed  <dbl> 26, 0, 0, 11, 11, 15, 15, 15, 19, 15, 20, 11, 0, 7,...
```

2. Combine both Train and Test Data set (to understand the distribution of independent variable together).

```
# add variables to test dataset before merging
test$registered=0
test$casual=0
test$count=0

data = rbind(train,test)
```

3. Variable Type Identification

```
str(data)
#> 'data.frame':    17379 obs. of  12 variables:
#>  $ datetime  : Factor w/ 17379 levels "2011-01-01 00:00:00",..: 1 2 3 4 5 6 7 8 9 10 ...
#>  $ season    : int  1 1 1 1 1 1 1 1 1 1 ...
#>  $ holiday   : int  0 0 0 0 0 0 0 0 0 0 ...
#>  $ workingday: int  0 0 0 0 0 0 0 0 0 0 ...
#>  $ weather   : int  1 1 1 1 1 2 1 1 1 1 ...
#>  $ temp      : num  9.84 9.02 9.02 9.84 9.84 ...
#>  $ atemp     : num  14.4 13.6 13.6 14.4 14.4 ...
#>  $ humidity  : int  81 80 80 75 75 75 80 86 75 76 ...
#>  $ windspeed : num  0 0 0 0 0 ...
#>  $ casual    : num  3 8 5 3 0 0 2 1 1 8 ...
#>  $ registered: num  13 32 27 10 1 1 0 2 7 6 ...
#>  $ count     : num  16 40 32 13 1 1 2 3 8 14 ...
```

4. Find missing values in the dataset if any

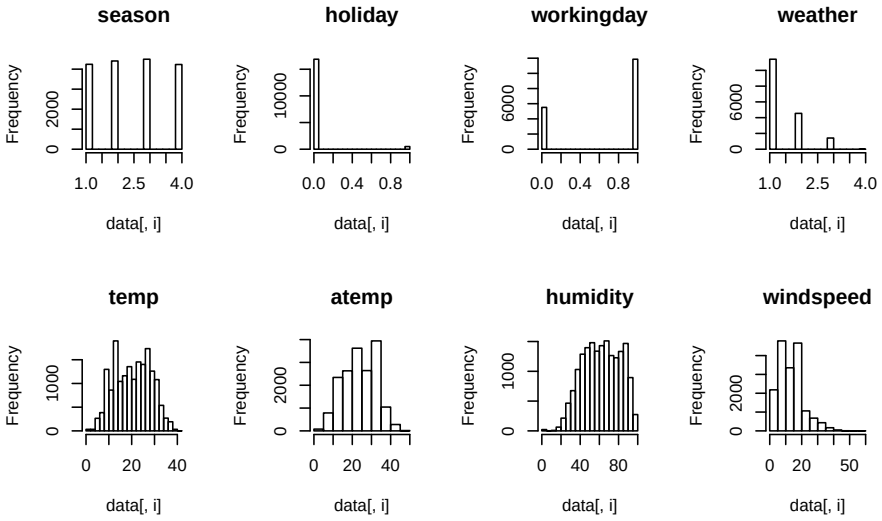
```
table(is.na(data))
#>
#>   FALSE
#> 208548
```

No NAs in the dataset.

5. Understand the distribution of numerical variables and generate a frequency table for numeric variables. Analyze the distribution.

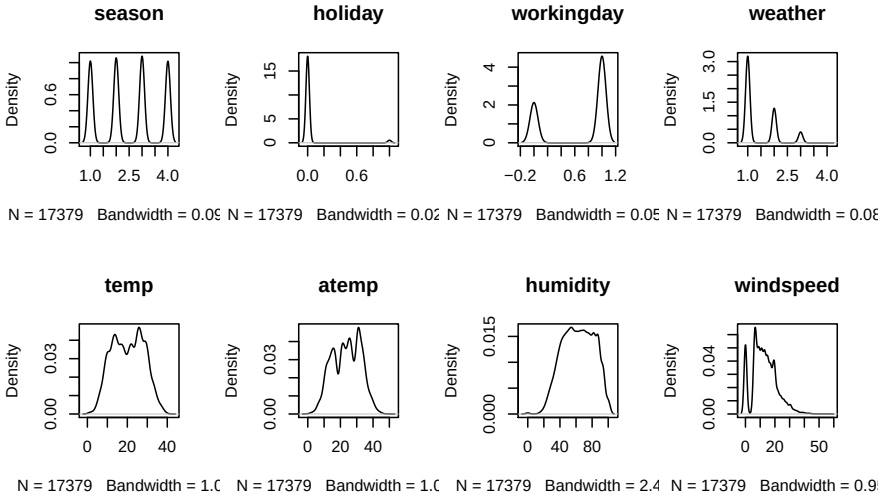
10.3.1 histograms

```
# histograms each attribute
par(mfrow=c(2,4))
for(i in 2:9) {
    hist(data[,i], main = names(data)[i])
}
```



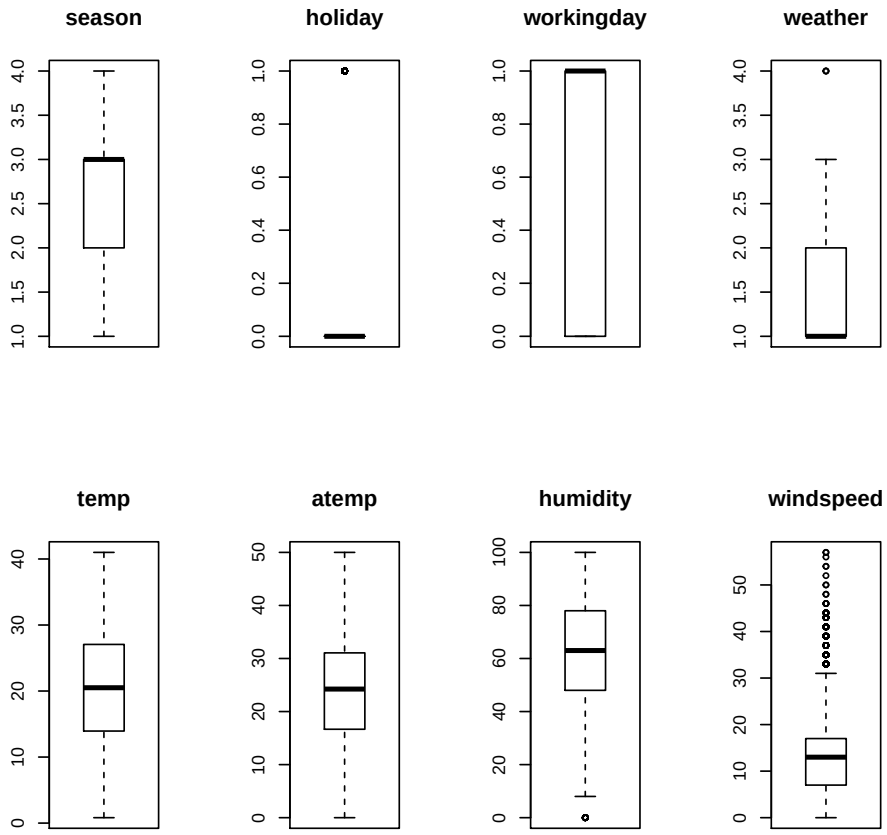
10.3.2 density plots

```
# density plot for each attribute
par(mfrow=c(2,4))
for(i in 2:9) {
    plot(density(data[,i]), main=names(data)[i])
}
```



10.3.3 boxplots

```
# boxplots for each attribute
par(mfrow=c(2,4))
for(i in 2:9) {
    boxplot(data[,i], main=names(data)[i])
}
```



10.3.4 Unique values of discrete variables

```
# the discrete variables in this case are integers
ints <- unlist(lapply(data, is.integer))
names(data)[ints]
#> [1] "season"     "holiday"    "workingday" "weather"    "humidity"
```

Humidity should not be an integer or discrete variable; it is a continuous or numeric variable.

```
# convert humidity to numeric
data$humidity <- as.numeric(data$humidity)
```

```
# list unique values of integer variables
ints <- unlist(lapply(data, is.integer))
int_vars <- names(data)[ints]

sapply(int_vars, function(x) unique(data[x]))
#> $season.season
#> [1] 1 2 3 4
#>
#> $holiday.holiday
#> [1] 0 1
#>
#> $workingday.workingday
#> [1] 0 1
#>
#> $weather.weather
```

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

10.3.5 Inferences

1. The variables `season`, `holiday`, `workingday` and `weather` are discrete (integer).
2. Activity is even through all seasons.
3. Most of the activitity happens during non-holidays.
4. Activity doubles during the working days.
5. Activity happens mostly during clear (1) weather.
6. temp, atemp and humidity are continuous variables (numeric).

10.4 Hypothesis Testing (using multivariate analysis)

Till now, we have got a fair understanding of the data set. Now, let's test the hypothesis which we had generated earlier. Here I have added some additional hypothesis from the dataset. Let's test them one by one:

10.4.1 Hourly trend

> *There must be high demand during office timings. Early morning and late evening can have different trend (cyclist) and low demand during 10:00 pm to 4:00 am.*

We don't have the variable 'hour' with us. But we can extract it using the datetime column.

```
head(data$datetime)
#> [1] 2011-01-01 00:00:00 2011-01-01 01:00:00 2011-01-01 02:00:00
#> [4] 2011-01-01 03:00:00 2011-01-01 04:00:00 2011-01-01 05:00:00
#> 17379 Levels: 2011-01-01 00:00:00 2011-01-01 01:00:00 ... 2012-12-31 23:00:00
```

```
class(data$datetime)
#> [1] "factor"
```

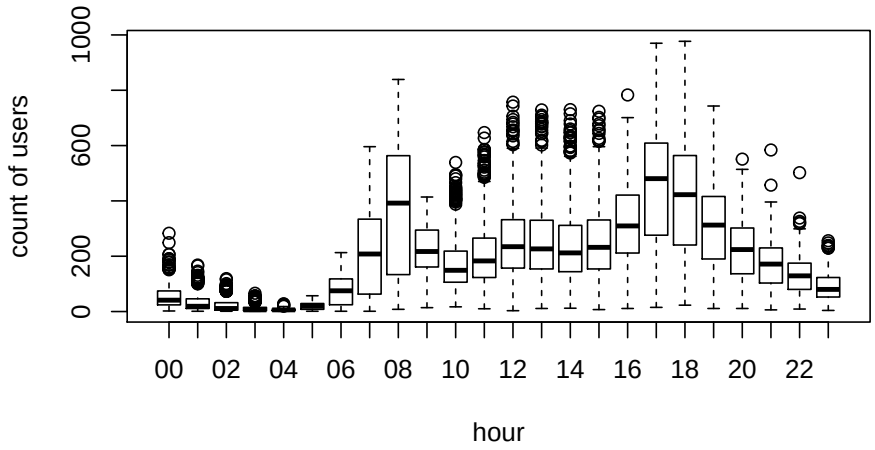
```
# show hour and day from the variable datetime
head(substr(data$datetime, 12, 13))  # hour
#> [1] "00" "01" "02" "03" "04" "05"
head(substr(data$datetime, 9, 10))   # day
#> [1] "01" "01" "01" "01" "01" "01"
```

```
# extracting hour
data$hour = substr(data$datetime,12,13)
data$hour = as.factor(data$hour)
head(data$hour)
#> [1] 00 01 02 03 04 05
#> 24 Levels: 00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 ... 23
```

```
### dividing again in train and test
# the train dataset is for the first 19 days
train = data[as.integer(substr(data$datetime, 9, 10)) < 20,]

# the test dataset is from day 20 to the end of the month
test = data[as.integer(substr(data$datetime, 9, 10)) > 19,]
```

10.4.2 boxplot count vs hour in training set

```
boxplot(train$count ~ train$hour, xlab="hour", ylab="count of users")
```

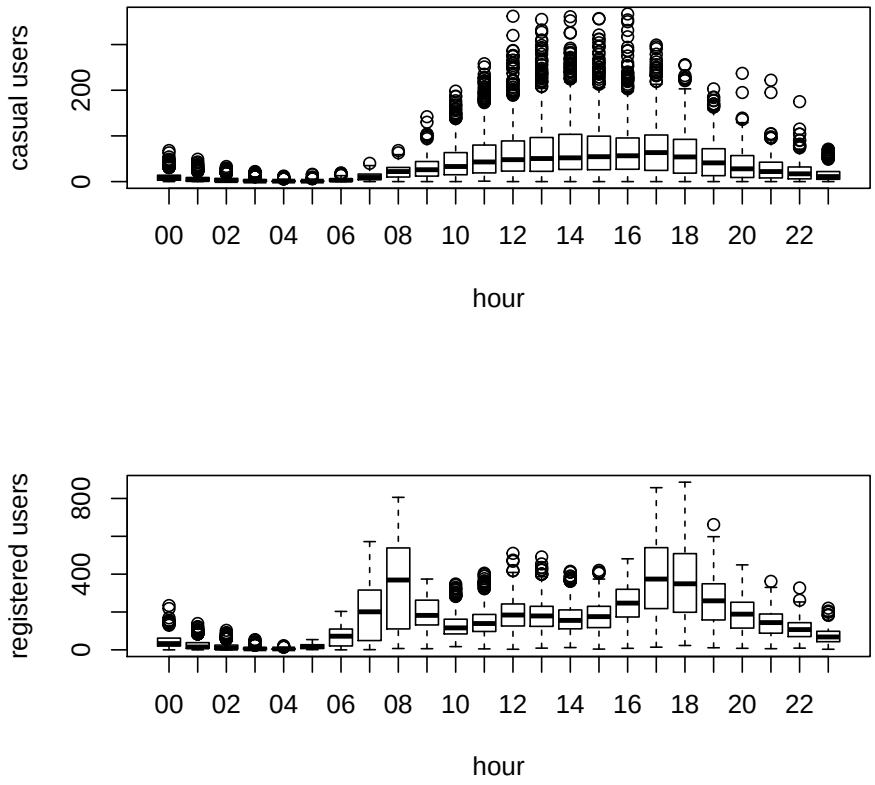


Rides increase from 6 am to 6pm, during office hours.

```
# casual users
casual <- data[data$casual > 0, ]
registered <- data[data$registered > 0, ]

dim(casual)
#> [1] 9900   13
dim(registered)
#> [1] 10871    13
```

10.4.3 Boxplot hourly: casual vs registered users in the training set

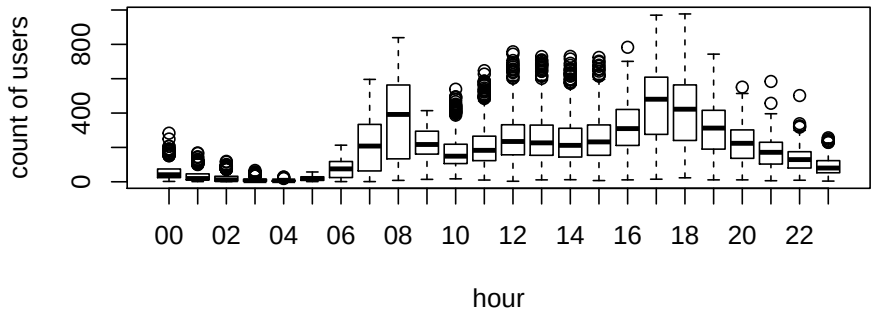
```
# by hour: casual vs registered users
par(mfrow=c(2,1))
boxplot(train$casual ~ train$hour, xlab="hour", ylab="casual users")
boxplot(train$registered ~ train$hour, xlab="hour", ylab="registered users")
```

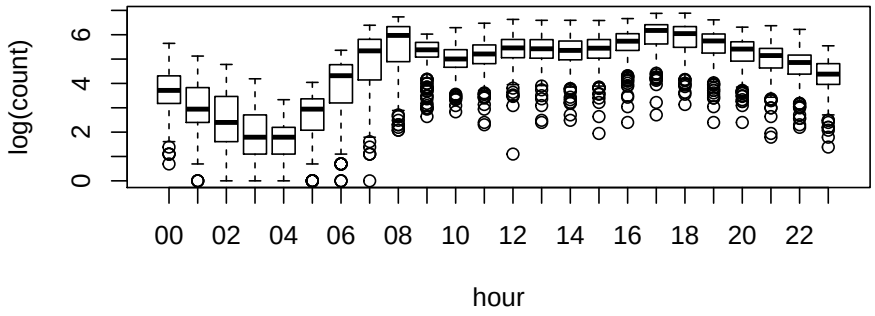


Casual and Registered users have different distributions. Casual users tend to rent more during office hours.

10.4.4 outliers in the training set

```
par(mfrow=c(2,1))
boxplot(train$count ~ train$hour, xlab="hour", ylab="count of users")
boxplot(log(train$count) ~ train$hour,xlab="hour",ylab="log(count)")
```





10.4.5 Daily trend

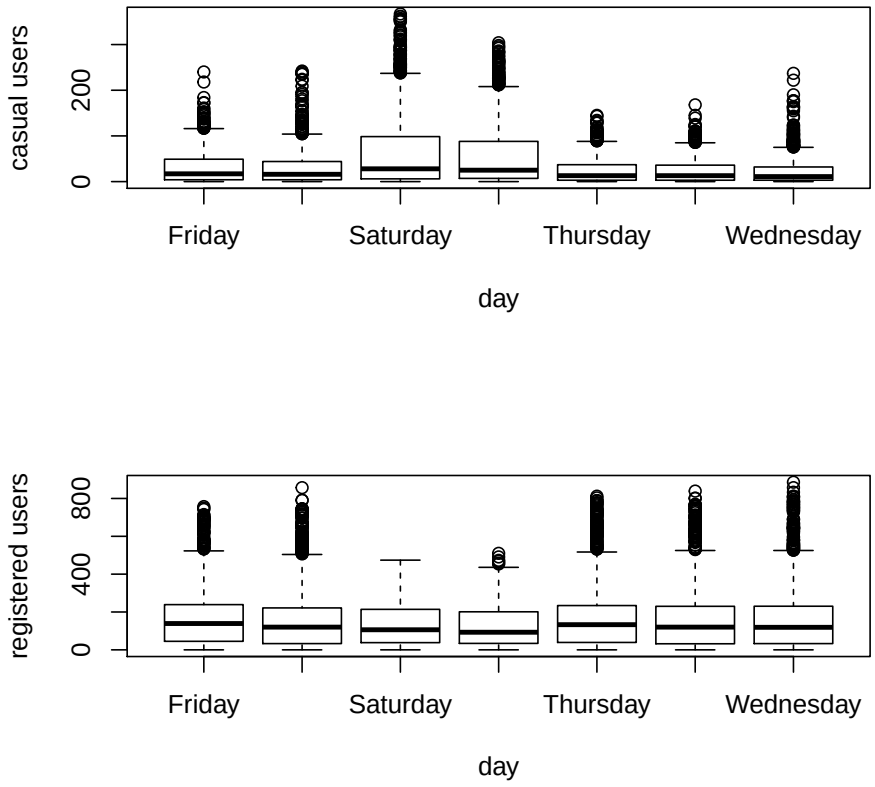
Registered users demand more bike on weekdays as compared to weekend or holiday.

```
# extracting days of week
date <- substr(data$datetime, 1, 10)
days <- weekdays(as.Date(date))
data$day <- days
```

```
# split the dataset again at day 20 of the month, before and after
train = data[as.integer(substr(data$datetime,9,10)) < 20,]
test  = data[as.integer(substr(data$datetime,9,10)) > 19,]
```

10.4.6 Boxplot daily trend: casual vs registered users, training set

```
# creating boxplots for rentals with different variables to see the variation
par(mfrow=c(2,1))
boxplot(train$casual ~ train$day, xlab="day", ylab="casual users")
boxplot(train$registered ~ train$day, xlab="day", ylab="registered users")
```



Demand of casual users increases during the weekend, contrary of registered users.

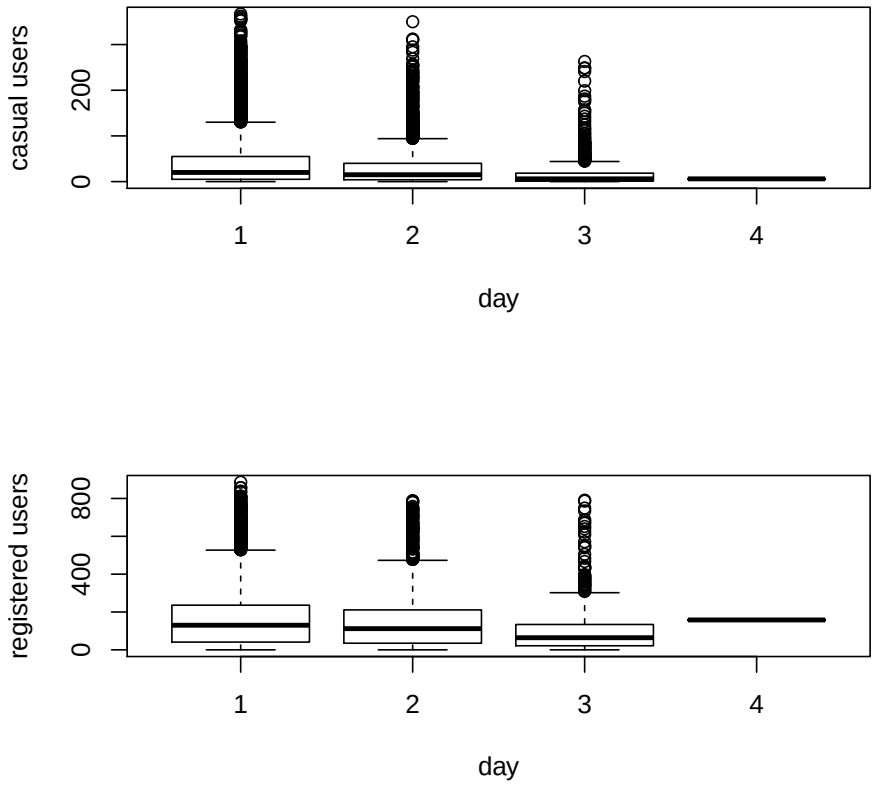
10.4.7 Rain

The demand of bikes will be lower on a rainy day as compared to a sunny day. Similarly, higher humidity will cause to lower the demand and vice versa.

We use the variable weather (1 to 4) to analyze riding under rain conditions.

10.4.7.1 Boxplot of rain effect on bike riding, training set

```
par(mfrow=c(2,1))
boxplot(train$casual ~ train$weather, xlab="day", ylab="casual users")
boxplot(train$registered ~ train$weather, xlab="day", ylab="registered users")
```



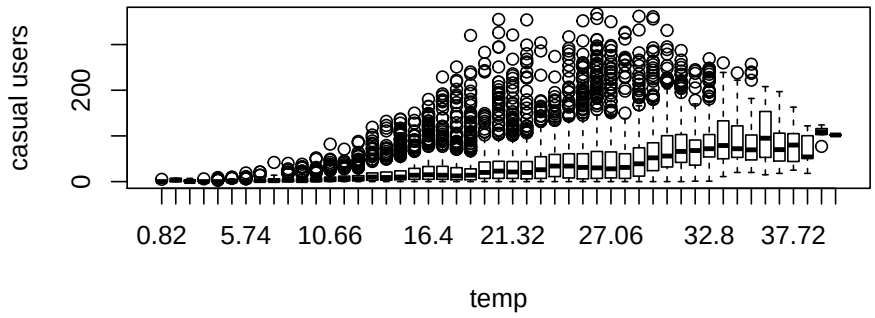
Registered used tend to ride even with rain.

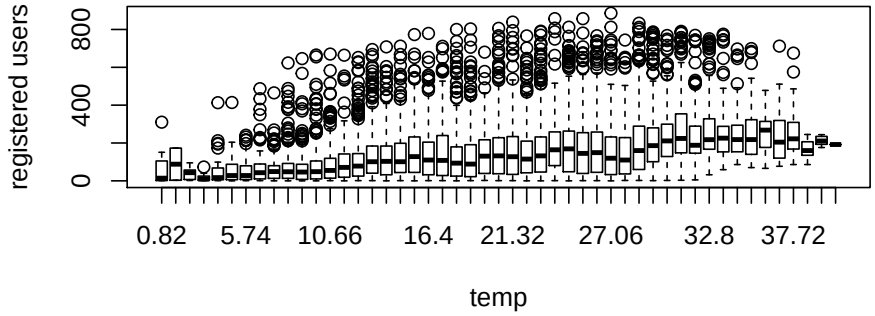
10.4.8 Temperature

Would high or low temperature encourage or disencourage bike riding?

10.4.8.1 boxplot of temperature effect, training set

```
par(mfrow=c(2,1))
boxplot(train$casual ~ train$temp, xlab="temp", ylab="casual users")
boxplot(train$registered ~ train$temp, xlab="temp", ylab="registered users")
```



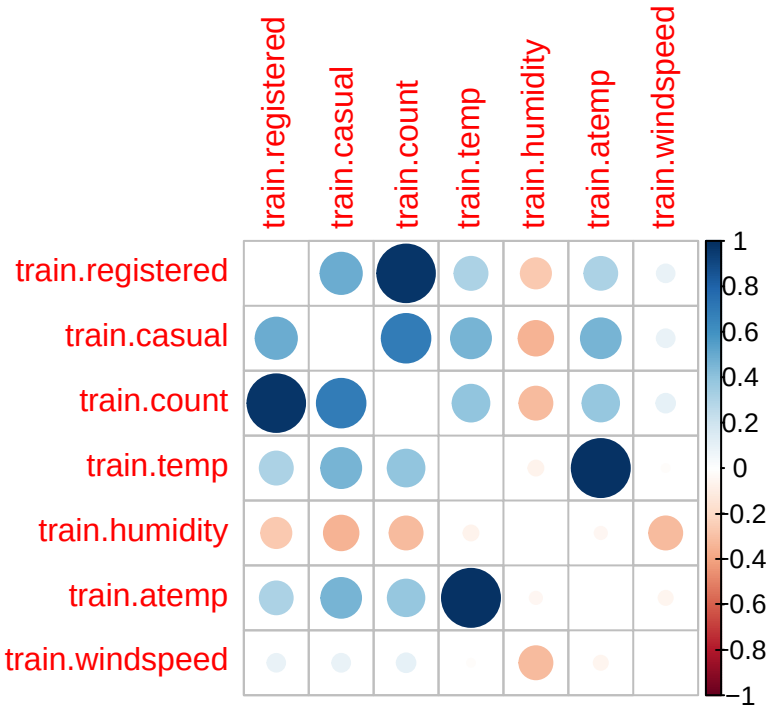


Casual users tend to ride with milder temperatures while registered users ride even at low temperatures.

10.4.9 Correlation

```
sub = data.frame(train$registered, train$casual, train$count, train$temp,
                 train$humidity, train$atemp, train$windspeed)
cor(sub)
#>                  train.registered train.casual train.count train.temp
#> train.registered           1.0000       0.4972       0.971     0.3186
#> train.casual               0.4972       1.0000       0.690     0.4671
#> train.count                0.9709       0.6904       1.000     0.3945
#> train.temp                 0.3186       0.4671       0.394     1.0000
#> train.humidity            -0.2655      -0.3482      -0.317    -0.0649
#> train.atemp                0.3146       0.4621       0.390     0.9849
#> train.windspeed            0.0911       0.0923       0.101    -0.0179
#>                  train.humidity train.atemp train.windspeed
#> train.registered        -0.2655      0.3146          0.0911
#> train.casual            -0.3482      0.4621          0.0923
#> train.count             -0.3174      0.3898          0.1014
#> train.temp              -0.0649      0.9849         -0.0179
#> train.humidity           1.0000     -0.0435         -0.3186
#> train.atemp             -0.0435      1.0000         -0.0575
#> train.windspeed         -0.3186     -0.0575          1.0000
```

```
# do not show the diagonal
corrplot(cor(sub), diag = FALSE)
```



1. correlation between `casual` and atemp, temp.
2. Strong correlation between temp and atemp.

10.4.10 Activity by year

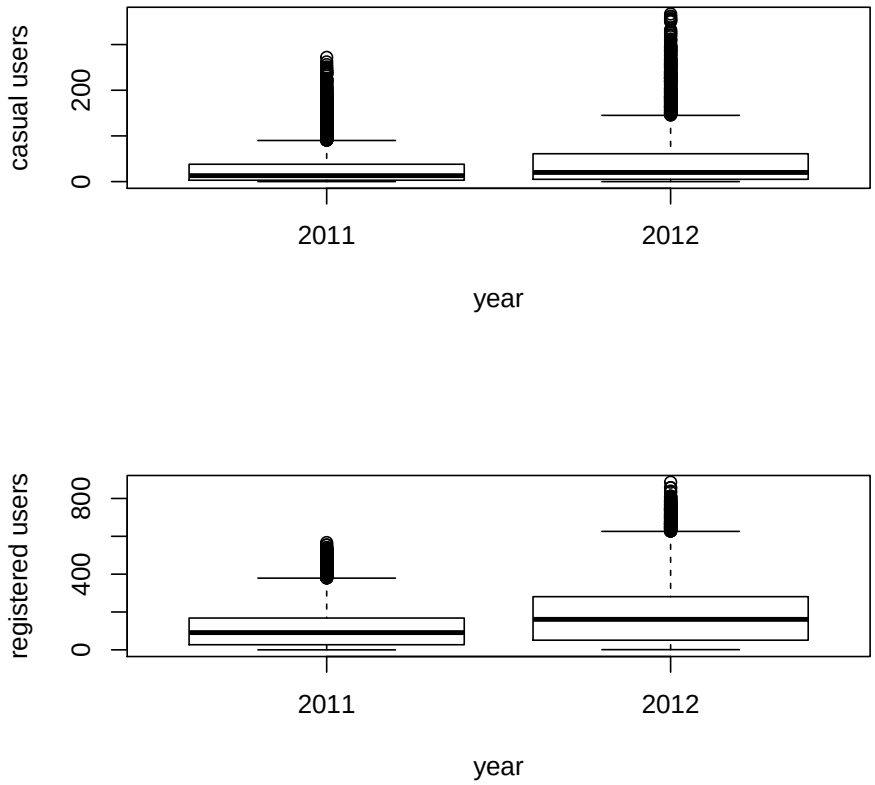
10.4.10.1 Year extraction

```
# extracting year
data$year = substr(data$datetime, 1, 4)
data$year = as.factor(data$year)
```

```
# ignore the division of data again and again, this could have been done together also
train = data[as.integer(substr(data$datetime,9,10)) < 20,]
test = data[as.integer(substr(data$datetime,9,10)) > 19,]
```

10.4.10.2 Trend by year, training set

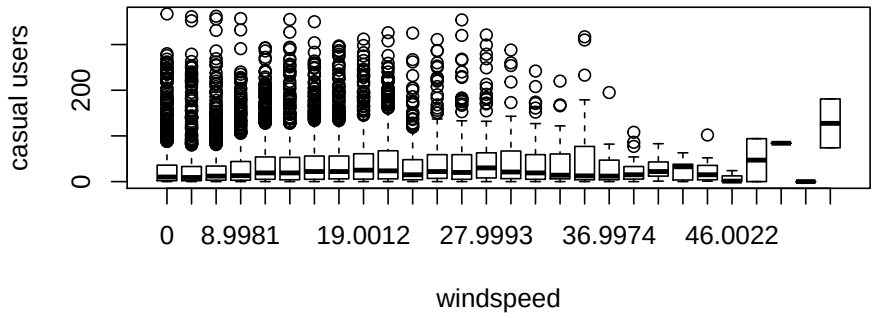
```
par(mfrow=c(2,1))
# again some boxplots with different variables
# these boxplots give important information about the dependent variable with respect to the independen
boxplot(train$casual ~ train$year, xlab="year", ylab="casual users")
boxplot(train$registered ~ train$year, xlab="year", ylab="registered users")
```

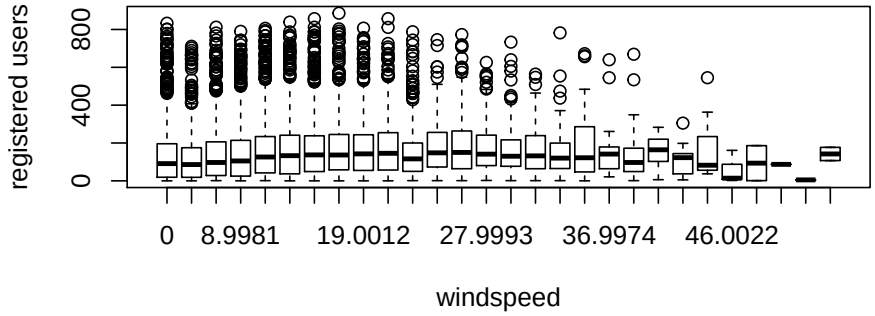


Activity increased in 2012.

10.4.10.3 trend by windspeed, training set

```
par(mfrow=c(2,1))
boxplot(train$casual ~ train$windspeed, xlab="windspeed", ylab="casual users")
boxplot(train$registered ~ train$windspeed, xlab="windspeed", ylab="registered users")
```

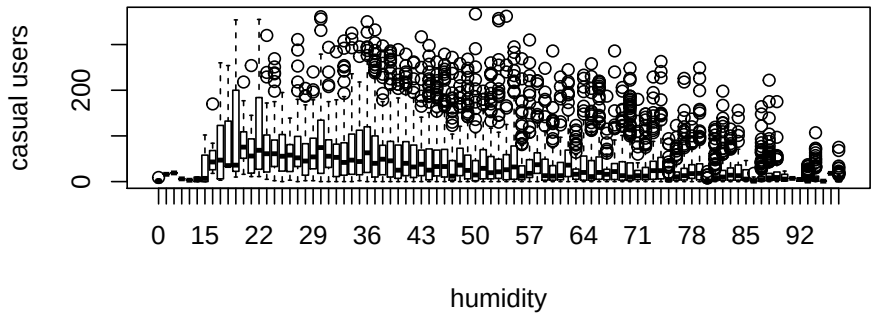


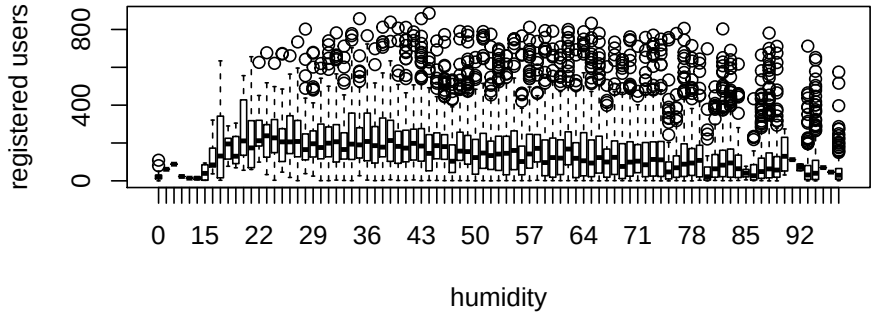


Casual users ride even with stron winds.

10.4.10.4 trend by humidity, training set

```
par(mfrow=c(2,1))
boxplot(train$casual ~ train$humidity, xlab="humidity", ylab="casual users")
boxplot(train$registered ~ train$humidity, xlab="humidity", ylab="registered users")
```





Casual users prefer not to ride with humid weather.

10.5 Feature Engineering

10.5.1 Prepare data

```
# factoring some variables from integer
data$season     <- as.factor(data$season)
data$weather    <- as.factor(data$weather)
data$holiday    <- as.factor(data$holiday)
data$workingday <- as.factor(data$workingday)
```

```
# new column
data$hour <- as.integer(data$hour)
```

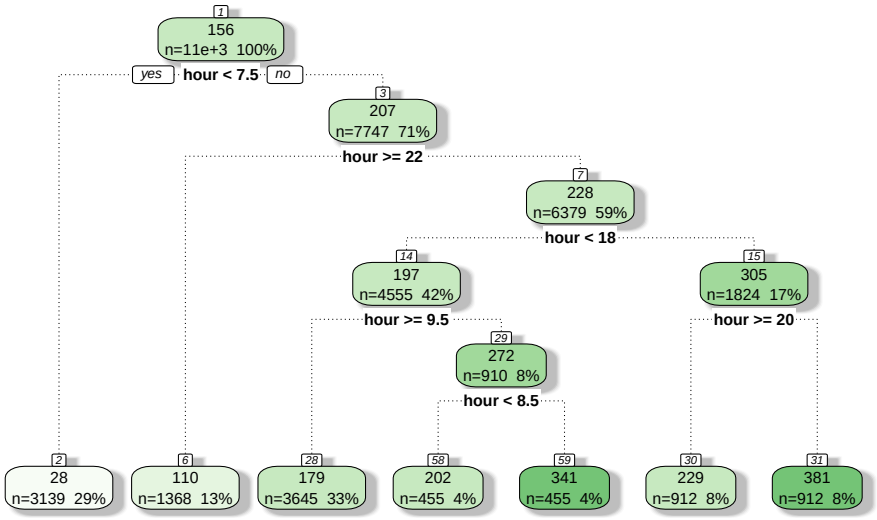
```
# created this variable to divide a day into parts, but did not finally use it
data$day_part <- 0
```

```
# split in training and test sets again
train <- data[as.integer(substr(data$datetime, 9, 10)) < 20,]
test  <- data[as.integer(substr(data$datetime, 9, 10)) > 19,]
```

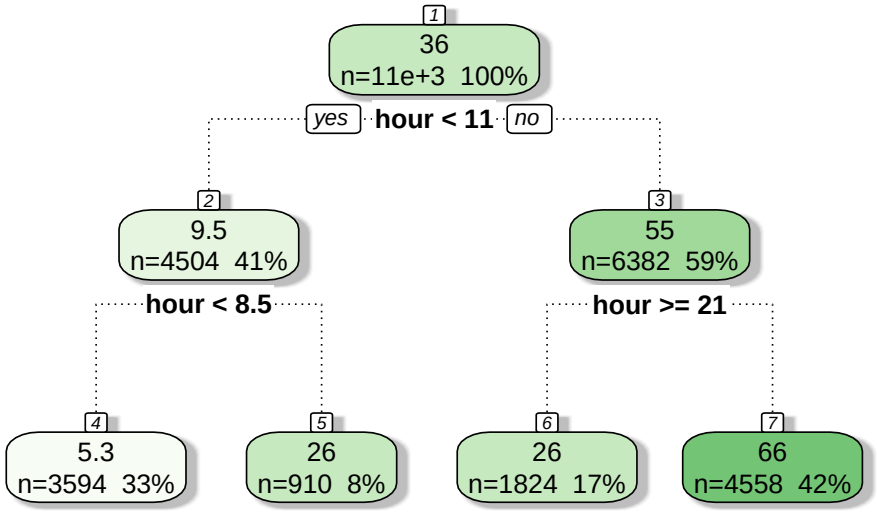
```
# combine the sets
data <- rbind(train, test)
```

10.5.2 Build hour bins

```
# for registered users
d = rpart(registered ~ hour, data = train)
fancyRpartPlot(d)
```




```
# for casual users
d = rpart(casual ~ hour, data = train)
fancyRpartPlot(d)
```



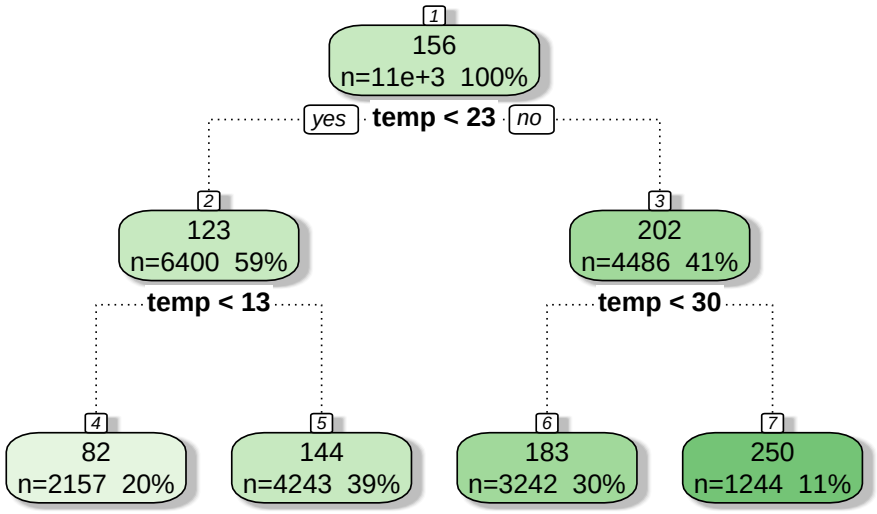

```
# Assign the timings according to tree
# fill the hour bins
data = rbind(train,test)

# create hour buckets for registered users
# 0,1,2,3,4,5,6,7 < 7.5
# 22,23,24 >=22
# 10,11,12,13,14,15,16,17: h>=9.5 & h<18
```

```
# h<9.5 & h<8.5 : 8
# h<9.5 & h>=8.5 : 9
# h>=20: 20,21
# h < 20: 18,19

data$dp_reg = 0
data$dp_reg[data$hour < 8] = 1
data$dp_reg[data$hour >= 22] = 2
data$dp_reg[data$hour > 9 & data$hour < 18] = 3
data$dp_reg[data$hour == 8] = 4
data$dp_reg[data$hour == 9] = 5
data$dp_reg[data$hour == 20 | data$hour == 21] = 6
data$dp_reg[data$hour == 19 | data$hour == 18] = 7
```

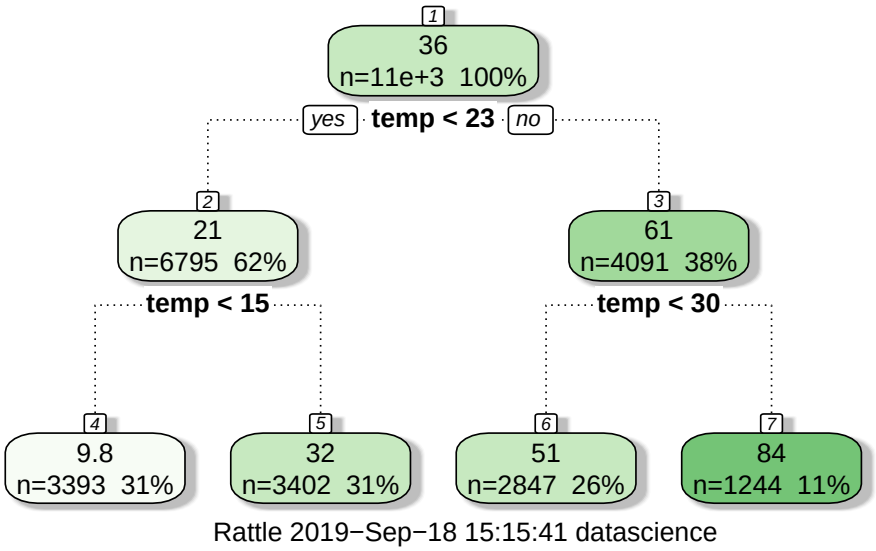
```
# casual users
# h<11, h<8.5: 0,1,2,3,4,5,6,7,8
# h>=8.5 & h<11: 9, 10
# h >=11 & h>=21: 21,22,23,24
# h >=11 & h<21: 11,12,13,14,15,16,17,18,19,20
data$dp_cas = 0
data$dp_cas[data$hour < 11 & data$hour >= 8] = 1
data$dp_cas[data$hour == 9 | data$hour == 10] = 2
data$dp_cas[data$hour >= 11 & data$hour < 21] = 3
data$dp_cas[data$hour >= 21] = 4
```

10.5.3 Temperature bins

```
# partition the data by temperature, registered users
f = rpart(registered ~ temp, data=train)
fancyRpartPlot(f)
```




```
# partition the data by temperature,, casual users
f=rpart(casual ~ temp, data=train)
fancyRpartPlot(f)
```



10.5.3.1 Assign temperature ranges accoding to trees

```
data$temp_reg = 0
data$temp_reg[data$temp < 13] = 1
data$temp_reg[data$temp >= 13 & data$temp < 23] = 2
data$temp_reg[data$temp >= 23 & data$temp < 30] = 3
data$temp_reg[data$temp >= 30] = 4
```

```
data$temp_cas = 0
data$temp_cas[data$temp < 15] = 1
data$temp_cas[data$temp >= 15 & data$temp < 23] = 2
data$temp_cas[data$temp >= 23 & data$temp < 30] = 3
data$temp_cas[data$temp >= 30] = 4
```

10.5.4 Year bins by quarter

```
# add new variable with the month number
data$month <- substr(data$datetime, 6, 7)
data$month <- as.integer(data$month)
```

```
# bin by quarter manually
data$year_part[data$year=='2011']                = 1
data$year_part[data$year=='2011' & data$month>3] = 2
data$year_part[data$year=='2011' & data$month>6] = 3
data$year_part[data$year=='2011' & data$month>9] = 4
data$year_part[data$year=='2012']                = 5
data$year_part[data$year=='2012' & data$month>3] = 6
data$year_part[data$year=='2012' & data$month>6] = 7
data$year_part[data$year=='2012' & data$month>9] = 8
table(data$year_part)
#>
#>    1    2    3    4    5    6    7    8
#> 2067 2183 2192 2203 2176 2182 2208 2168
```

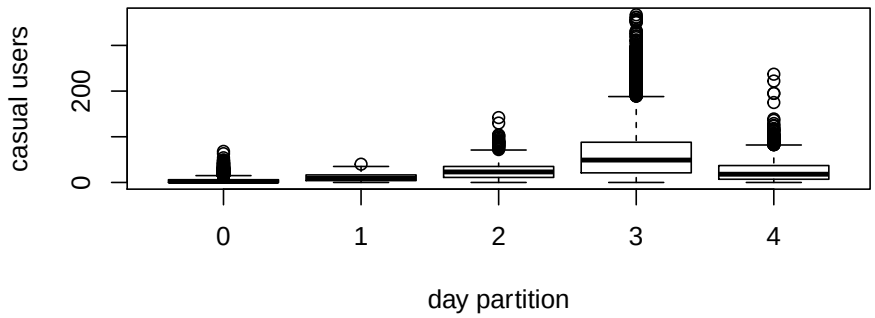
10.5.5 Day Type

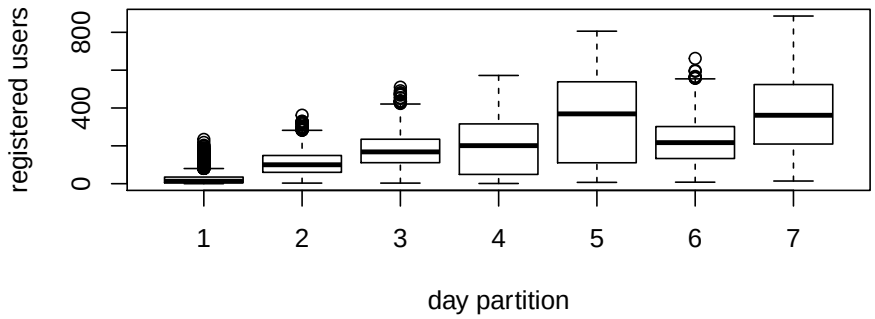
Created a variable having categories like “weekday”, “weekend” and “holiday”.

```
# creating another variable day_type which may affect our accuracy as weekends and weekdays are importa
data$day_type = 0
data$day_type[data$holiday==0 & data$workingday==0] = "weekend"
data$day_type[data$holiday==1]                      = "holiday"
data$day_type[data$holiday==0 & data$workingday==1] = "working day"
```

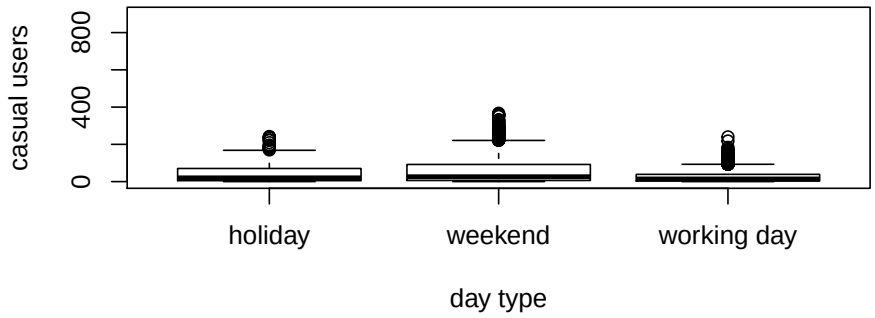
```
# split dataset again
train = data[as.integer(substr(data$datetime,9,10)) < 20,]
test = data[as.integer(substr(data$datetime,9,10)) > 19,]
```

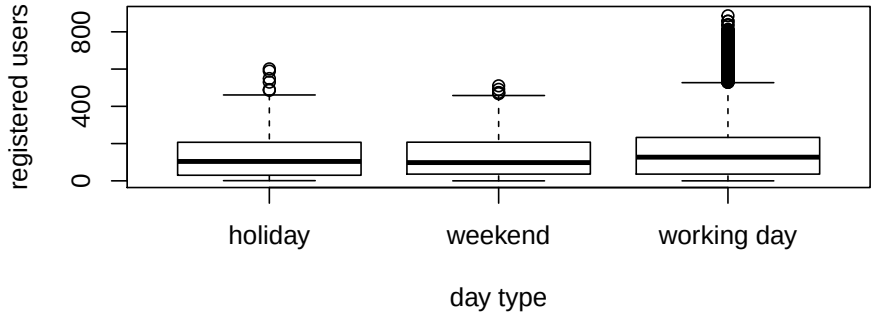
```
par(mfrow=c(2,1))
boxplot(train$casual ~ train$dp_cas, xlab = "day partition", ylab="casual users")
boxplot(train$registered ~ train$dp_reg, xlab = "day partition", ylab="registered users")
```



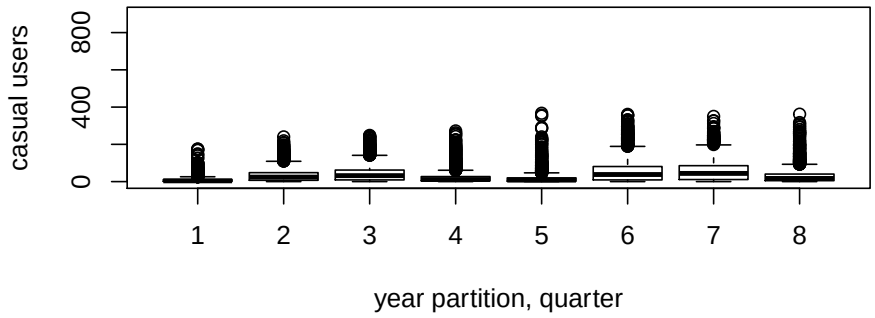


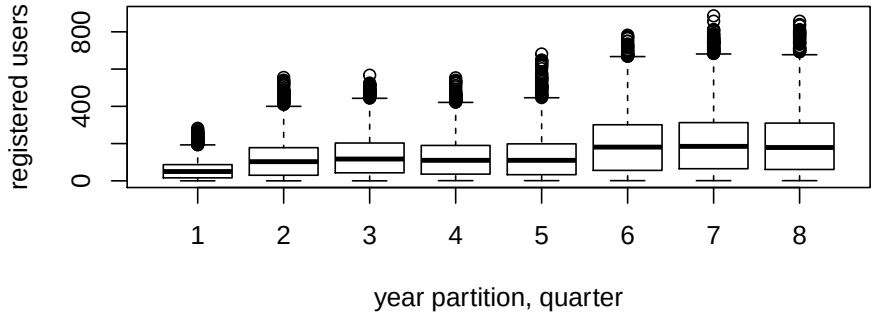

```
par(mfrow=c(2,1))
boxplot(train$casual ~ train$day_type, xlab = "day type",
        ylab="casual users", ylim = c(0,900))
boxplot(train$registered ~ train$day_type, xlab = "day type",
        ylab="registered users", ylim = c(0,900))
```





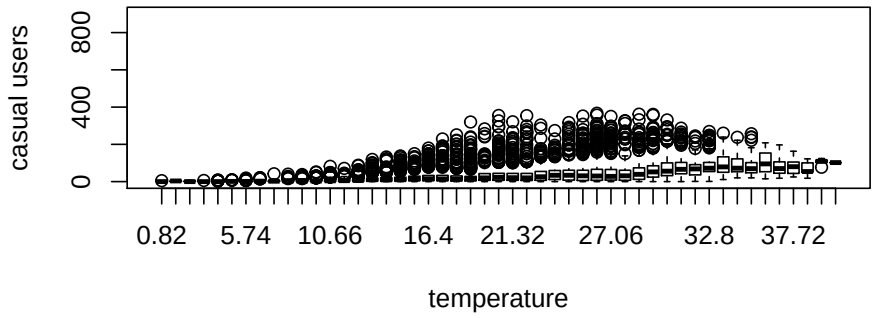

```
par(mfrow=c(2,1))
boxplot(train$casual ~ train$year_part, xlab = "year partition, quarter",
        ylab="casual users", ylim = c(0,900))
boxplot(train$registered ~ train$year_part, xlab = "year partition, quarter",
        ylab="registered users", ylim = c(0,900))
```

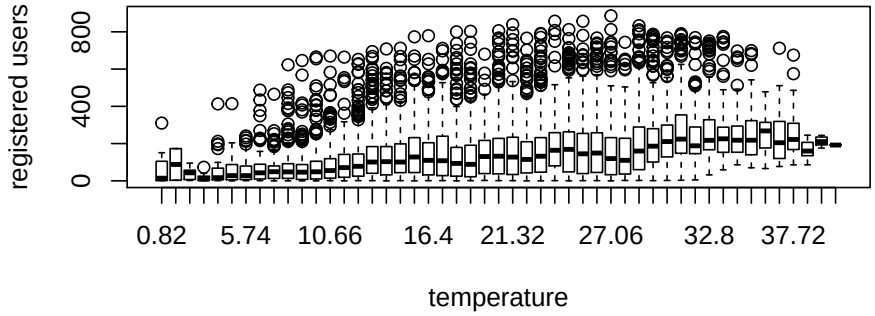




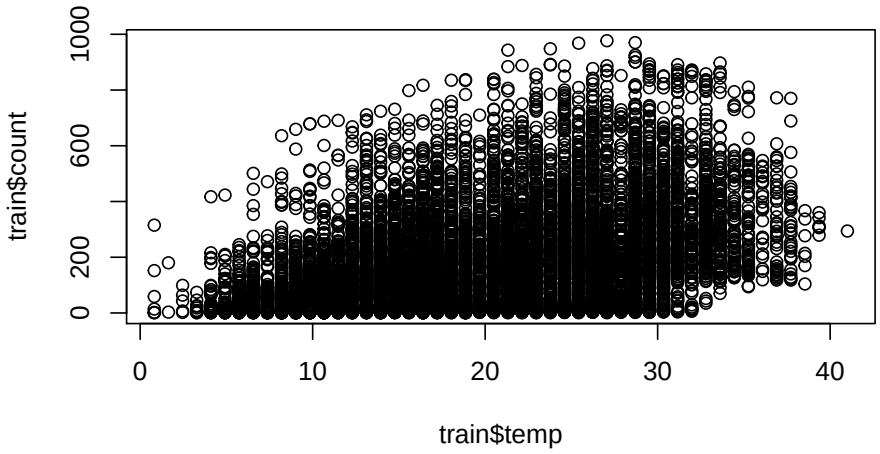
10.5.6 Temperatures

```
par(mfrow=c(2,1))
boxplot(train$casual ~ train$temp, xlab = "temperature",
        ylab="casual users", ylim = c(0,900))
boxplot(train$registered ~ train$temp, xlab = "temperature",
        ylab="registered users", ylim = c(0,900))
```






```
plot(train$temp, train$count)
data <- rbind(train, test)
# data$month <- substr(data$datetime, 6, 7)
# data$month <- as.integer(data$month)
```



10.5.7 Imputting missing data to wind speed

```
# dividing total data depending on windspeed to impute/predict the missing values
table(data$windspeed == 0)
#>
#> FALSE  TRUE
#> 15199  2180
    # FALSE  TRUE
```

```
    # 15199  2180

k = data$windspeed == 0

wind_0 = subset(data, k)    # windspeed is zero
wind_1 = subset(data, !k)   # windspeed not zero
```

```
tic()
# predicting missing values in windspeed using a random forest model
# this is a different approach to impute missing values rather than
# just using the mean or median or some other statistic for imputation

set.seed(415)
fit <- randomForest(windspeed ~ season + weather + humidity + month + temp +
                        year + atemp,
                    data = wind_1,
                    importance = TRUE,
                    ntree = 250)

pred = predict(fit, wind_0)
wind_0$windspeed = pred       # fill with wind speed predictions
toc()
#> 63.95 sec elapsed
```

```
# recompose the whole dataset
data = rbind(wind_0, wind_1)
```

```
# how many zero values now?
sum(data$windspeed == 0)
#> [1] 0
```

10.5.8 Weekend variable

Created a separate variable for weekend (0/1)

```
data$weekend = 0
data$weekend[data$day=="Sunday" | data$day=="Saturday" ] = 1
```

10.6 Model Building

As this was our first attempt, we applied decision tree, conditional inference tree and random forest algorithms and found that random forest is performing the best. You can also go with regression, boosted regression, neural network and find which one is working well for you.

Before executing the random forest model code, I have followed following steps:

Convert discrete variables into factor (weather, season, hour, holiday, working day, month, day)

```
str(data)
#> 'data.frame':    17379 obs. of  24 variables:
#>  $ datetime  : Factor w/ 17379 levels "2011-01-01 00:00:00",..: 1 2 3 4 5 7 8 9 10 65 ...
#>  $ season    : Factor w/ 4 levels "1","2","3","4": 1 1 1 1 1 1 1 1 1 1 ...
#>  $ holiday   : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
```

```
#>  $ workingday: Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 2 ...
#>  $ weather   : Factor w/ 4 levels "1","2","3","4": 1 1 1 1 1 1 1 1 1 1 ...
#>  $ temp      : num  9.84 9.02 9.02 9.84 9.84 ...
#>  $ atemp     : num  14.4 13.6 13.6 14.4 14.4 ...
#>  $ humidity  : num  81 80 80 75 75 80 86 75 76 47 ...
#>  $ windspeed : num  9.03 9.05 9.05 9.15 9.15 ...
#>  $ casual    : num  3 8 5 3 0 2 1 1 8 8 ...
#>  $ registered: num  13 32 27 10 1 0 2 7 6 102 ...
#>  $ count     : num  16 40 32 13 1 2 3 8 14 110 ...
#>  $ hour      : int  1 2 3 4 5 7 8 9 10 20 ...
#>  $ day       : chr  "Saturday" "Saturday" "Saturday" "Saturday" ...
#>  $ year      : Factor w/ 2 levels "2011","2012": 1 1 1 1 1 1 1 1 1 1 ...
#>  $ day_part  : num  0 0 0 0 0 0 0 0 0 0 ...
#>  $ dp_reg    : num  1 1 1 1 1 1 4 5 3 6 ...
#>  $ dp_cas    : num  0 0 0 0 0 0 1 2 2 3 ...
#>  $ temp_reg  : num  1 1 1 1 1 1 1 1 2 1 ...
#>  $ temp_cas  : num  1 1 1 1 1 1 1 1 1 1 ...
#>  $ month     : int  1 1 1 1 1 1 1 1 1 1 ...
#>  $ year_part : num  1 1 1 1 1 1 1 1 1 1 ...
#>  $ day_type  : chr  "weekend" "weekend" "weekend" "weekend" ...
#>  $ weekend   : num  1 1 1 1 1 1 1 1 1 0 ...
```

10.6.1 Convert variables to factors

```
# converting all relevant categorical variables into factors to feed to our random forest model
data$season     = as.factor(data$season)
data$holiday    = as.factor(data$holiday)
data$workingday = as.factor(data$workingday)
data$weather    = as.factor(data$weather)
data$hour       = as.factor(data$hour)
data$month      = as.factor(data$month)
data$day_part   = as.factor(data$dp_cas)
data$day_type   = as.factor(data$dp_reg)
data$day        = as.factor(data$day)
data$temp_cas   = as.factor(data$temp_cas)
data$temp_reg   = as.factor(data$temp_reg)
```

```
str(data)
#> 'data.frame':    17379 obs. of  24 variables:
#>  $ datetime  : Factor w/ 17379 levels "2011-01-01 00:00:00",..: 1 2 3 4 5 7 8 9 10 65 ...
#>  $ season    : Factor w/ 4 levels "1","2","3","4": 1 1 1 1 1 1 1 1 1 1 ...
#>  $ holiday   : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
#>  $ workingday: Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 2 ...
#>  $ weather   : Factor w/ 4 levels "1","2","3","4": 1 1 1 1 1 1 1 1 1 1 ...
#>  $ temp      : num  9.84 9.02 9.02 9.84 9.84 ...
#>  $ atemp     : num  14.4 13.6 13.6 14.4 14.4 ...
#>  $ humidity  : num  81 80 80 75 75 80 86 75 76 47 ...
#>  $ windspeed : num  9.03 9.05 9.05 9.15 9.15 ...
#>  $ casual    : num  3 8 5 3 0 2 1 1 8 8 ...
#>  $ registered: num  13 32 27 10 1 0 2 7 6 102 ...
#>  $ count     : num  16 40 32 13 1 2 3 8 14 110 ...
#>  $ hour      : Factor w/ 24 levels "1","2","3","4",..: 1 2 3 4 5 7 8 9 10 20 ...
```

```
#>  $ day       : Factor w/ 7 levels "Friday","Monday",..: 3 3 3 3 3 3 3 3 3 2 ...
#>  $ year      : Factor w/ 2 levels "2011","2012": 1 1 1 1 1 1 1 1 1 1 ...
#>  $ day_part  : Factor w/ 5 levels "0","1","2","3",..: 1 1 1 1 1 1 2 3 3 4 ...
#>  $ dp_reg    : num  1 1 1 1 1 1 4 5 3 6 ...
#>  $ dp_cas    : num  0 0 0 0 0 0 1 2 2 3 ...
#>  $ temp_reg  : Factor w/ 4 levels "1","2","3","4": 1 1 1 1 1 1 1 1 2 1 ...
#>  $ temp_cas  : Factor w/ 4 levels "1","2","3","4": 1 1 1 1 1 1 1 1 1 1 ...
#>  $ month     : Factor w/ 12 levels "1","2","3","4",..: 1 1 1 1 1 1 1 1 1 1 ...
#>  $ year_part : num  1 1 1 1 1 1 1 1 1 1 ...
#>  $ day_type  : Factor w/ 7 levels "1","2","3","4",..: 1 1 1 1 1 1 4 5 3 6 ...
#>  $ weekend   : num  1 1 1 1 1 1 1 1 1 0 ...
```

- As we know that dependent variables have natural outliers so we will predict log of dependent variables.

- Predict bike demand registered and casual users separately. $y1 = \log(casual + 1)$ and $y2 = \log(registered + 1)$, Here we have added 1 to deal with zero values in the casual and registered columns.

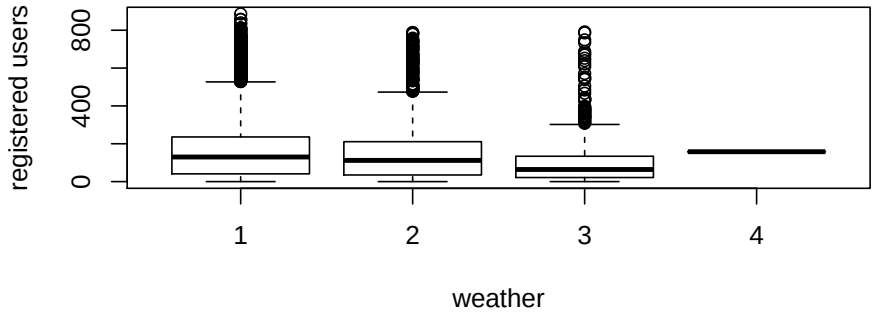
```
# separate again as train and test set
train = data[as.integer(substr(data$datetime, 9, 10)) < 20,]
test = data[as.integer(substr(data$datetime, 9, 10)) > 19,]
```

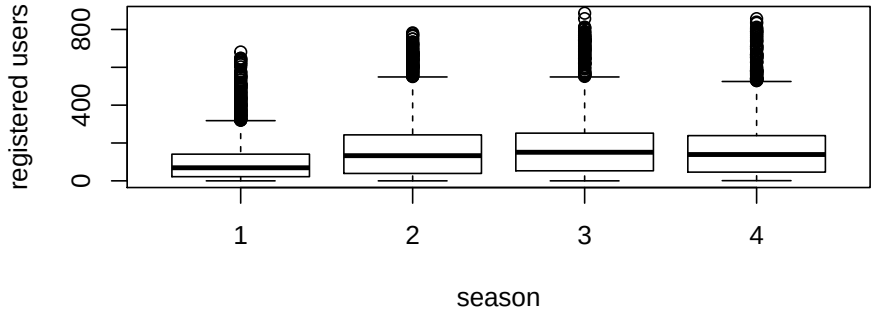
10.6.2 Log transform

```
# log transformation for some skewed variables,
# which can be seen from their distribution
train$reg1   = train$registered + 1
train$cas1   = train$casual + 1
train$logcas = log(train$cas1)
train$logreg = log(train$reg1)
test$logreg  = 0
test$logcas  = 0
```

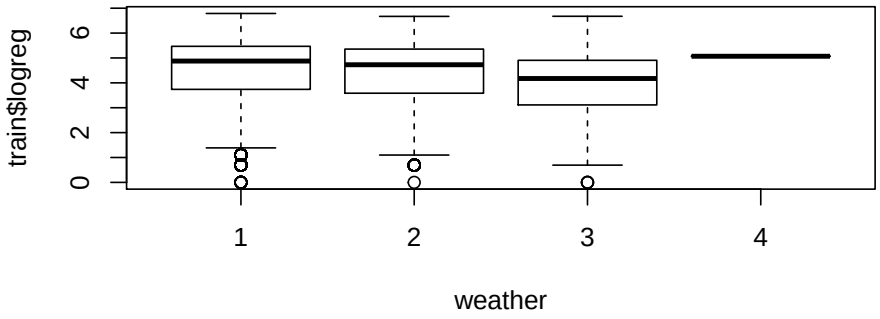
10.6.2.1 Plot by weather, by season

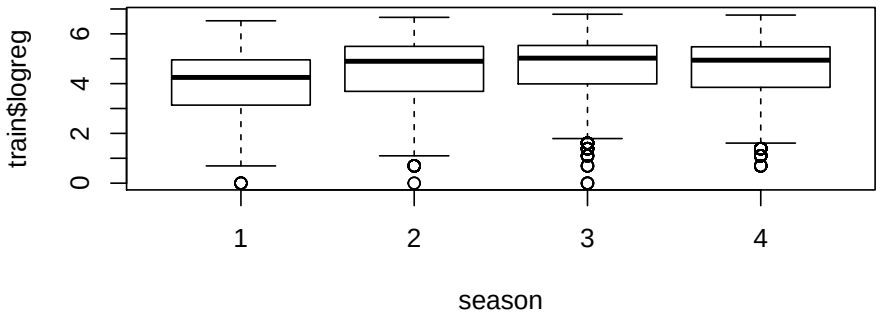
```
# cartesian plot
par(mfrow=c(2,1))
boxplot(train$registered ~ train$weather, xlab="weather", ylab="registered users")
boxplot(train$registered ~ train$season, xlab="season", ylab="registered users")
```






```
# semilog plot
par(mfrow=c(2,1))
boxplot(train$logreg ~ train$weather, xlab = "weather")
boxplot(train$logreg ~ train$season, xlab = "season")
```





10.6.3 Predicting for registered and casual users, test dataset

```
tic()
# final model building using random forest
# note that we build different models for predicting for
# registered and casual users
# this was seen as giving best result after a lot of experimentation
set.seed(415)
fit1 <- randomForest(logreg ~ hour + workingday + day + holiday + day_type +
                         temp_reg + humidity + atemp + windspeed + season +
                         weather + dp_reg + weekend + year + year_part,
                     data = train,
                     importance = TRUE,
                     ntree = 250)

pred1 = predict(fit1, test)
test$logreg = pred1
toc()
#> 160.988 sec elapsed
```

```
# casual users
set.seed(415)
fit2 <- randomForest(logcas ~ hour + day_type + day + humidity + atemp +
                         temp_cas + windspeed + season + weather + holiday +
                         workingday + dp_cas + weekend + year + year_part,
                     data = train, importance = TRUE, ntree = 250)

pred2 = predict(fit2, test)
test$logcas = pred2
```

10.6.4 Preparing and exporting results

```
# creating the final submission file
# reverse log conversion
test$registered <- exp(test$logreg) - 1
test$casual     <- exp(test$logcas) - 1
test$count      <- test$casual + test$registered

r <- data.frame(datetime = test$datetime,
                casual = test$casual,
                registered = test$registered)

print(sum(r$casual))
#> [1] 205804
print(sum(r$registered))
#> [1] 962834

s <- data.frame(datetime = test$datetime, count = test$count)
write.csv(s, file =file.path(data_out_dir, "bike-submit.csv"), row.names = FALSE)

# sum(cas+reg) = 1168638
# month number now is correct
```

After following the steps mentioned above, you can score 0.38675 on Kaggle leaderboard i.e. top 5 percentile of total participants. As you might have seen, we have not applied any extraordinary science in getting to this level. But, the real competition starts here. I would like to see, if I can improve this further by use of more features and some more advanced modeling techniques.

10.7 End Notes

In this article, we have looked at structured approach of problem solving and how this method can help you to improve performance. I would recommend you to generate hypothesis before you deep dive in the data set as this technique will not limit your thought process. You can improve your performance by applying advanced techniques (or ensemble methods) and understand your data trend better.

You can find the complete solution here : GitHub Link

```
# this is the older submission. months were incomplete
old <- read.csv(file = file.path(data_raw_dir, "bike-submit-old.csv"))
sum(old$count)
#> [1] 1164829
```

Chapter 11

Breast Cancer Wisconsin

Source: https://shiring.github.io/machine_learning/2017/01/15/rfe_ga_post

11.1 Read and process the data

```
bc_data <- read.table(file.path(data_raw_dir, "breast-cancer-wisconsin.data"),
                      header = FALSE, sep = ",")
```

```
# assign the column names
colnames(bc_data) <- c("sample_code_number", "clump_thickness",
                       "uniformity_of_cell_size", "uniformity_of_cell_shape",
                       "marginal_adhesion", "single_epithelial_cell_size",
                       "bare_nuclei", "bland_chromatin", "normal_nucleoli",
                       "mitosis", "classes")

# change classes from numeric to character
bc_data$classes <- ifelse(bc_data$classes == "2", "benign",
                          ifelse(bc_data$classes == "4", "malignant", NA))

# if query sign make NA
bc_data[bc_data == "?"] <- NA

# how many NAs are in the data
length(which(is.na(bc_data)))
#> [1] 16
```

```
names(bc_data)
#>  [1] "sample_code_number"          "clump_thickness"
#>  [3] "uniformity_of_cell_size"     "uniformity_of_cell_shape"
#>  [5] "marginal_adhesion"           "single_epithelial_cell_size"
#>  [7] "bare_nuclei"                 "bland_chromatin"
#>  [9] "normal_nucleoli"             "mitosis"
#> [11] "classes"
```

11.1.1 Missing data

```
# impute missing data
library(mice)
#> Loading required package: lattice
#>
#> Attaching package: 'mice'
#> The following objects are masked from 'package:base':
#>
#>     cbind, rbind

# skip these columns: sample_code_number and classes
# convert to numeric
bc_data[,2:10] <- apply(bc_data[, 2:10], 2, function(x) as.numeric(as.character(x)))

# impute but mute
dataset_impute <- mice(bc_data[, 2:10],  print = FALSE)

# bind "classes" with the rest. skip "sample_code_number"
bc_data <- cbind(bc_data[, 11, drop = FALSE],
                 mice::complete(dataset_impute, action =1))

bc_data$classes <- as.factor(bc_data$classes)

# how many benign and malignant cases are there?
summary(bc_data$classes)
#>    benign malignant
#>       458       241
```

```
# confirm NAs have been removed
length(which(is.na(bc_data)))
#> [1] 0
```

```
str(bc_data)
#> 'data.frame':    699 obs. of  10 variables:
#>  $ classes                    : Factor w/ 2 levels "benign","malignant": 1 1 1 1 1 2 1 1 1 1 ...
#>  $ clump_thickness            : num  5 5 3 6 4 8 1 2 2 4 ...
#>  $ uniformity_of_cell_size    : num  1 4 1 8 1 10 1 1 1 2 ...
#>  $ uniformity_of_cell_shape   : num  1 4 1 8 1 10 1 2 1 1 ...
#>  $ marginal_adhesion          : num  1 5 1 1 3 8 1 1 1 1 ...
#>  $ single_epithelial_cell_size: num  2 7 2 3 2 7 2 2 2 2 ...
#>  $ bare_nuclei                : num  1 10 2 4 1 10 10 1 1 1 ...
#>  $ bland_chromatin            : num  3 3 3 3 3 9 3 3 1 2 ...
#>  $ normal_nucleoli            : num  1 2 1 7 1 7 1 1 1 1 ...
#>  $ mitosis                    : num  1 1 1 1 1 1 1 1 5 1 ...
```

11.2 Principal Component Analysis (PCA)

To get an idea about the dimensionality and variance of the datasets, I am first looking at PCA plots for samples and features. The first two principal components (PCs) show the two components that explain the majority of variation in the data.

After defining my custom `ggplot2` theme, I am creating a function that performs the PCA (using the

`pcaGoPromoter` package), calculates ellipses of the data points (with the `ellipse` package) and produces the plot with `ggplot2`. Some of the features in datasets 2 and 3 are not very distinct and overlap in the PCA plots, therefore I am also plotting hierarchical clustering dendrograms.

11.2.0.1 theme

```
# plotting theme

library(ggplot2)
#> Registered S3 methods overwritten by 'ggplot2':
#>   method         from
#>   [.quosures     rlang
#>   c.quosures     rlang
#>   print.quosures rlang

my_theme <- function(base_size = 12, base_family = "sans"){
  theme_minimal(base_size = base_size, base_family = base_family) +
  theme(
    axis.text = element_text(size = 12),
    axis.text.x = element_text(angle = 0, vjust = 0.5, hjust = 0.5),
    axis.title = element_text(size = 14),
    panel.grid.major = element_line(color = "grey"),
    panel.grid.minor = element_blank(),
    panel.background = element_rect(fill = "aliceblue"),
    strip.background = element_rect(fill = "navy", color = "navy", size = 1),
    strip.text = element_text(face = "bold", size = 12, color = "white"),
    legend.position = "right",
    legend.justification = "top",
    legend.background = element_blank(),
    panel.border = element_rect(color = "grey", fill = NA, size = 0.5)
  )
}

theme_set(my_theme())
```

11.2.0.2 PCA function

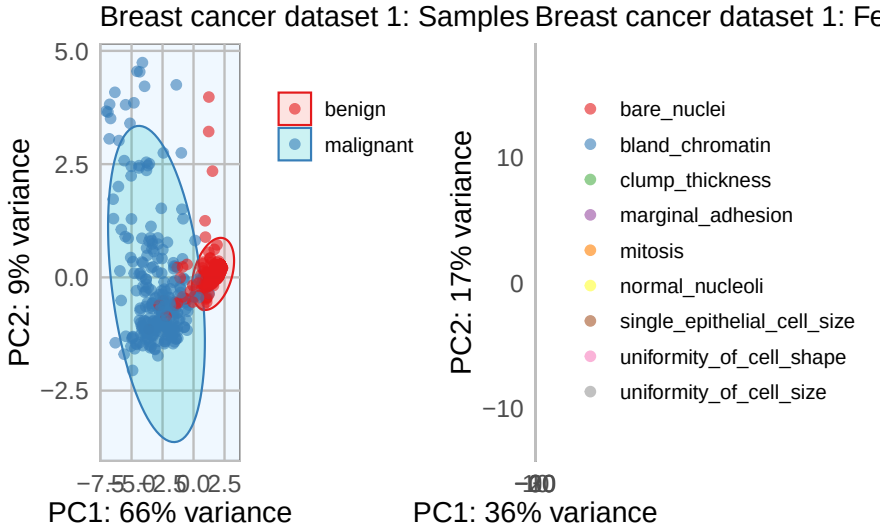
```
# function for PCA plotting
library(pcaGoPromoter)                  # install from BioConductor
#> Loading required package: ellipse
#>
#> Attaching package: 'ellipse'
#> The following object is masked from 'package:graphics':
#>
#>     pairs
#> Loading required package: Biostrings
#> Loading required package: BiocGenerics
#> Loading required package: parallel
#>
#> Attaching package: 'BiocGenerics'
#> The following objects are masked from 'package:parallel':
```

```
#>
#>     clusterApply, clusterApplyLB, clusterCall, clusterEvalQ,
#>     clusterExport, clusterMap, parApply, parCapply, parLapply,
#>     parLapplyLB, parRapply, parSapply, parSapplyLB
#> The following objects are masked from 'package:mice':
#>
#>     cbind, rbind
#> The following objects are masked from 'package:stats':
#>
#>     IQR, mad, sd, var, xtabs
#> The following objects are masked from 'package:base':
#>
#>     anyDuplicated, append, as.data.frame, basename, cbind,
#>     colnames, dirname, do.call, duplicated, eval, evalq, Filter,
#>     Find, get, grep, grepl, intersect, is.unsorted, lapply, Map,
#>     mapply, match, mget, order, paste, pmax, pmax.int, pmin,
#>     pmin.int, Position, rank, rbind, Reduce, rownames, sapply,
#>     setdiff, sort, table, tapply, union, unique, unsplit, which,
#>     which.max, which.min
#> Loading required package: S4Vectors
#> Loading required package: stats4
#>
#> Attaching package: 'S4Vectors'
#> The following object is masked from 'package:base':
#>
#>     expand.grid
#> Loading required package: IRanges
#> Loading required package: XVector
#>
#> Attaching package: 'Biostrings'
#> The following object is masked from 'package:base':
#>
#>     strsplit
library(ellipse)

pca_func <- function(data, groups, title, print_ellipse = TRUE) {

  # perform pca and extract scores for all principal components: PC1:PC9
  pcaOutput <- pca(data, printDropped = FALSE, scale = TRUE, center = TRUE)
  pcaOutput2 <- as.data.frame(pcaOutput$scores)

  # define groups for plotting. will group the classes
  pcaOutput2$groups <- groups

  # when plotting samples calculate ellipses for plotting
  # (when plotting features, there are no replicates)
  if (print_ellipse) {
    # group and summarize by classes: benign, malignant
    # centroids w/3 columns: groups, PC1, PC2
    centroids <- aggregate(cbind(PC1, PC2) ~ groups, pcaOutput2, mean)
    # bind for the two groups (classes)
    # conf.rgn w/3 columns: groups, PC1, PC2
    conf.rgn  <- do.call(rbind, lapply(unique(pcaOutput2$groups), function(t)
```

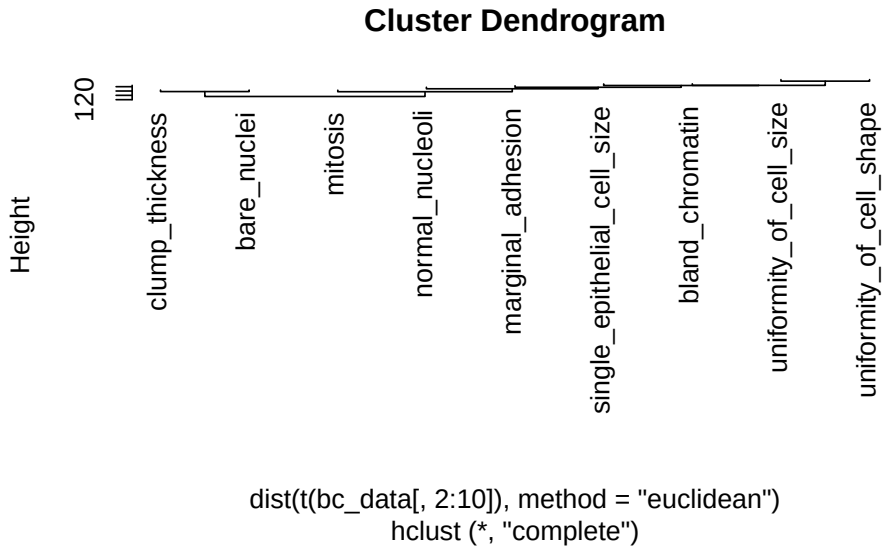
```
    data.frame(groups = as.character(t),
               # ellipse data for PC1 and PC2
               ellipse(cov(pcaOutput2[pcaOutput2$groups == t, 1:2]),
                     centre = as.matrix(centroids[centroids$groups == t, 2:3]),
                     level = 0.95),
               stringsAsFactors = FALSE)))

  plot <- ggplot(data = pcaOutput2, aes(x = PC1, y = PC2,
                                        group = groups,
                                        color = groups)) +
    geom_polygon(data = conf.rgn, aes(fill = groups), alpha = 0.2) + # ellipses
    geom_point(size = 2, alpha = 0.6) +
    scale_color_brewer(palette = "Set1") +
    labs(title = title,
         color = "",
         fill = "",
         x = paste0("PC1: ", round(pcaOutput$pov[1], digits = 2) * 100,
                    "% variance"),
         y = paste0("PC2: ", round(pcaOutput$pov[2], digits = 2) * 100,
                    "% variance"))

} else {

  # if < 10 groups (e.g. the predictor classes) have colors from RColorBrewer
  if (length(unique(pcaOutput2$groups)) <= 10) {

    plot <- ggplot(data = pcaOutput2, aes(x = PC1, y = PC2,
                                          group = groups,
                                          color = groups)) +
      geom_point(size = 2, alpha = 0.6) +
      scale_color_brewer(palette = "Set1") +
      labs(title = title,
           color = "",
           fill = "",
           x = paste0("PC1: ", round(pcaOutput$pov[1], digits = 2) * 100,
                      "% variance"),
           y = paste0("PC2: ", round(pcaOutput$pov[2], digits = 2) * 100,
                      "% variance"))

  } else {
    # otherwise use the default rainbow colors
    plot <- ggplot(data = pcaOutput2, aes(x = PC1, y = PC2,
                                          group = groups, color = groups)) +
      geom_point(size = 2, alpha = 0.6) +
      labs(title = title,
           color = "",
           fill = "",
           x = paste0("PC1: ", round(pcaOutput$pov[1], digits = 2) * 100,
                      "% variance"),
           y = paste0("PC2: ", round(pcaOutput$pov[2], digits = 2) * 100,
                      "% variance"))
  }
}
```

```
  return(plot)

}
```

```
library(gridExtra)
#>
#> Attaching package: 'gridExtra'
#> The following object is masked from 'package:BiocGenerics':
#>
#>     combine
library(grid)

# plot all data. one row is a feature
p1 <- pca_func(data = t(bc_data[, 2:10]),
               groups = as.character(bc_data$classes),
               title = "Breast cancer dataset 1: Samples")

# plot features only. features as columns
p2 <- pca_func(data = bc_data[, 2:10],
               groups = as.character(colnames(bc_data[, 2:10])),
               title = "Breast cancer dataset 1: Features", print_ellipse = FALSE)

grid.arrange(p1, p2, ncol = 2)
```

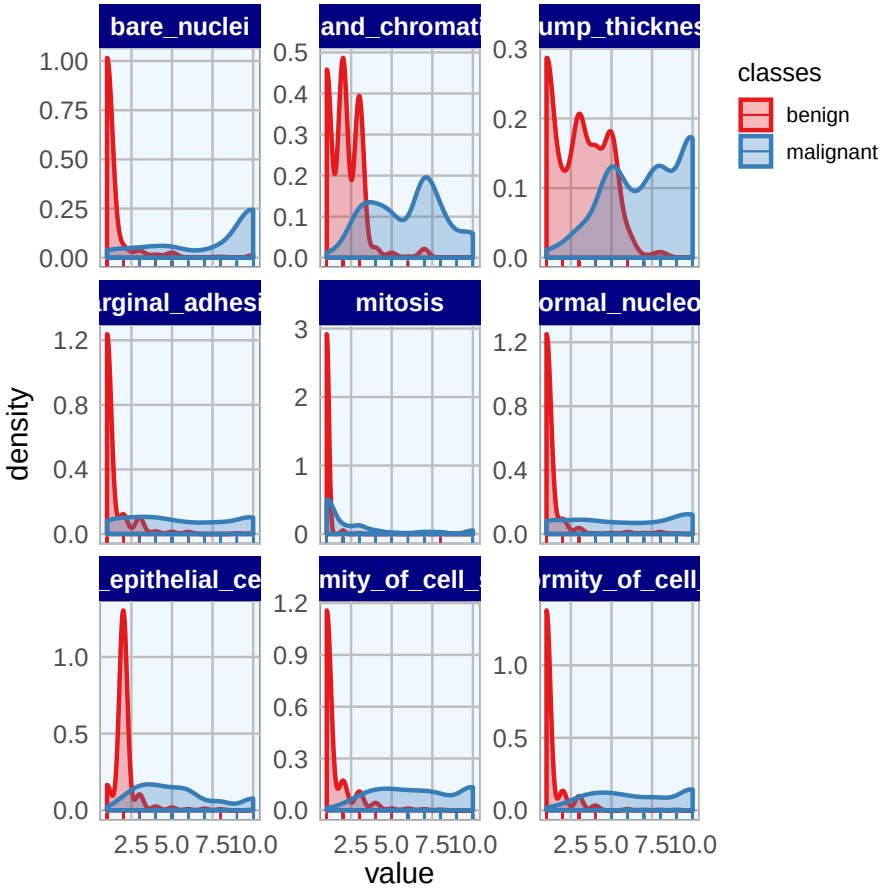



```
h_1 <- hclust(dist(t(bc_data[, 2:10]), method = "euclidean"), method = "complete")
plot(h_1)
```



11.2.1 density plots vs class

```
# density plot showing the feature vs classes
library(tidyr)
#>
#> Attaching package: 'tidyr'
#> The following object is masked from 'package:S4Vectors':
#>
#>     expand
#> The following object is masked from 'package:mice':
#>
#>     complete

# gather data. from column clump_thickness to mitosis
bc_data_gather <- bc_data %>%
  gather(measure, value, clump_thickness:mitosis)

ggplot(data = bc_data_gather, aes(x = value, fill = classes, color = classes)) +
  geom_density(alpha = 0.3, size = 1) +
  geom_rug() +
  scale_fill_brewer(palette = "Set1") +
  scale_color_brewer(palette = "Set1") +
  facet_wrap( ~ measure, scales = "free_y", ncol = 3)
```



11.3 Feature importance

To get an idea about the feature's respective importances, I'm running Random Forest models with 10 x 10 cross validation using the `caret` package. If I wanted to use feature importance to select features for modeling, I would need to perform it on the training data instead of on the complete dataset. But here, I only want to use it to get acquainted with my data. I am again defining a function that estimates the feature importance and produces a plot.

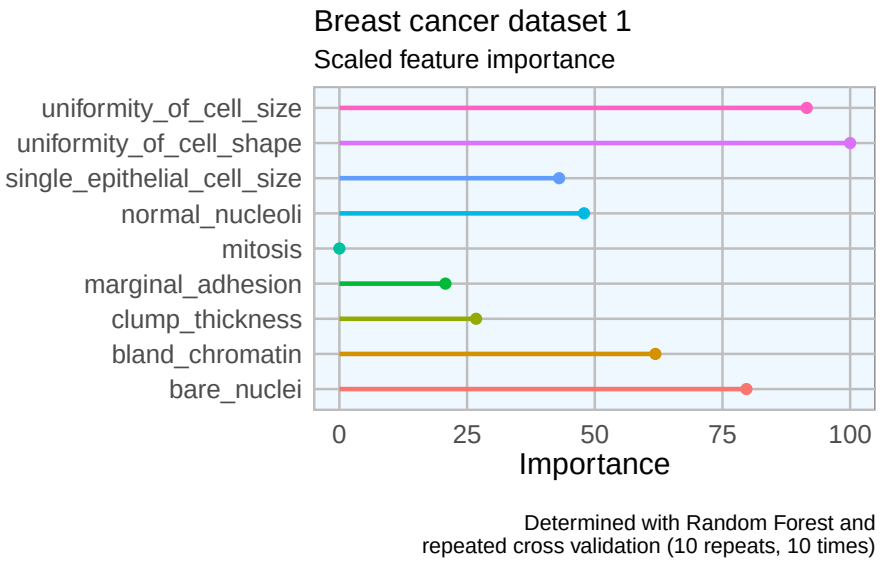
```
library(caret)
# library(doParallel) # parallel processing
# registerDoParallel()

# prepare training scheme
control <- trainControl(method = "repeatedcv", number = 10, repeats = 10)
```

```
feature_imp <- function(model, title) {
  # estimate variable importance
  importance <- varImp(model, scale = TRUE)
  # prepare dataframes for plotting
  importance_df_1 <- importance$importance
  importance_df_1$group <- rownames(importance_df_1)

  importance_df_2 <- importance_df_1
  importance_df_2$Overall <- 0
```

```
  importance_df <- rbind(importance_df_1, importance_df_2)

  plot <- ggplot() +
    geom_point(data = importance_df_1, aes(x = Overall,
                                           y = group,
                                           color = group), size = 2) +
    geom_path(data = importance_df, aes(x = Overall,
                                        y = group,
                                        color = group,
                                        group = group), size = 1) +
    theme(legend.position = "none") +
    labs(
      x = "Importance",
      y = "",
      title = title,
      subtitle = "Scaled feature importance",
      caption = "\nDetermined with Random Forest and
      repeated cross validation (10 repeats, 10 times)"
    )
  return(plot)
}
```

```
# train the model
set.seed(27)
imp_1 <- train(classes ~ ., data = bc_data, method = "rf",
               preProcess = c("scale", "center"),
               trControl = control)
```

```
p1 <- feature_imp(imp_1, title = "Breast cancer dataset 1")
p1
```



11.4 Feature Selection

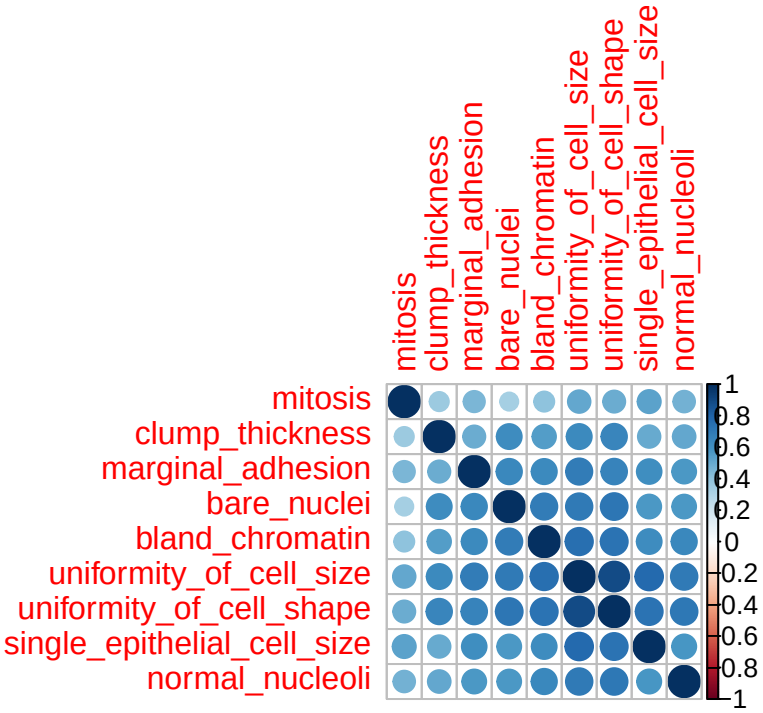
1. By correlation

2. By Recursive Feature Elimination
3. By Genetic Algorithm

```
set.seed(27)
bc_data_index <- createDataPartition(bc_data$classes, p = 0.7, list = FALSE)
bc_data_train <- bc_data[bc_data_index, ]
bc_data_test  <- bc_data[-bc_data_index, ]
```

11.4.1 Correlation

```
library(corrplot)
#> corrplot 0.84 loaded

# calculate correlation matrix
corMatMy <- cor(bc_data_train[, -1])
corrplot(corMatMy, order = "hclust")
```




```
# Apply correlation filter at 0.70,
highlyCor <- colnames(bc_data_train[, -1])[findCorrelation(corMatMy,
                                                           cutoff = 0.7,
                                                           verbose = TRUE)]
#> Compare row 2  and column  3 with corr  0.9
#>   Means:  0.709 vs 0.595 so flagging column 2
#> Compare row 3  and column  7 with corr  0.737
#>   Means:  0.674 vs 0.572 so flagging column 3
#> All correlations <= 0.7
```

```
# which variables are flagged for removal?
highlyCor
#> [1] "uniformity_of_cell_size"  "uniformity_of_cell_shape"
```

```
# then we remove these variables
bc_data_cor <- bc_data_train[, which(!colnames(bc_data_train) %in% highlyCor)]
names(bc_data_cor)
#> [1] "classes"                    "clump_thickness"
#> [3] "marginal_adhesion"          "single_epithelial_cell_size"
#> [5] "bare_nuclei"                "bland_chromatin"
#> [7] "normal_nucleoli"            "mitosis"
```

```
# confirm features were removed
outersect <- function(x, y) {
  sort(c(setdiff(x, y),
         setdiff(y, x)))
}

outersect(names(bc_data_cor), names(bc_data_train))
#> [1] "uniformity_of_cell_shape" "uniformity_of_cell_size"
```

Four features removed

11.4.2 Recursive Feature Elimination (RFE)

```
# ensure the results are repeatable
set.seed(7)

# define the control using a random forest selection function with cross validation
control <- rfeControl(functions = rfFuncs, method = "cv", number = 10)

# run the RFE algorithm
results_1 <- rfe(x = bc_data_train[, -1],
                 y = bc_data_train$classes,
                 sizes = c(1:9),
                 rfeControl = control)

# chosen features
predictors(results_1)
#> [1] "bare_nuclei"                "clump_thickness"
#> [3] "normal_nucleoli"            "uniformity_of_cell_size"
#> [5] "uniformity_of_cell_shape"   "single_epithelial_cell_size"
#> [7] "bland_chromatin"            "marginal_adhesion"
```

```
# subset the chosen features
sel_cols <- which(colnames(bc_data_train) %in% predictors(results_1))
bc_data_rfe <- bc_data_train[, c(1, sel_cols)]
names(bc_data_rfe)
#> [1] "classes"                    "clump_thickness"
#> [3] "uniformity_of_cell_size"    "uniformity_of_cell_shape"
#> [5] "marginal_adhesion"          "single_epithelial_cell_size"
#> [7] "bare_nuclei"                "bland_chromatin"
#> [9] "normal_nucleoli"
```

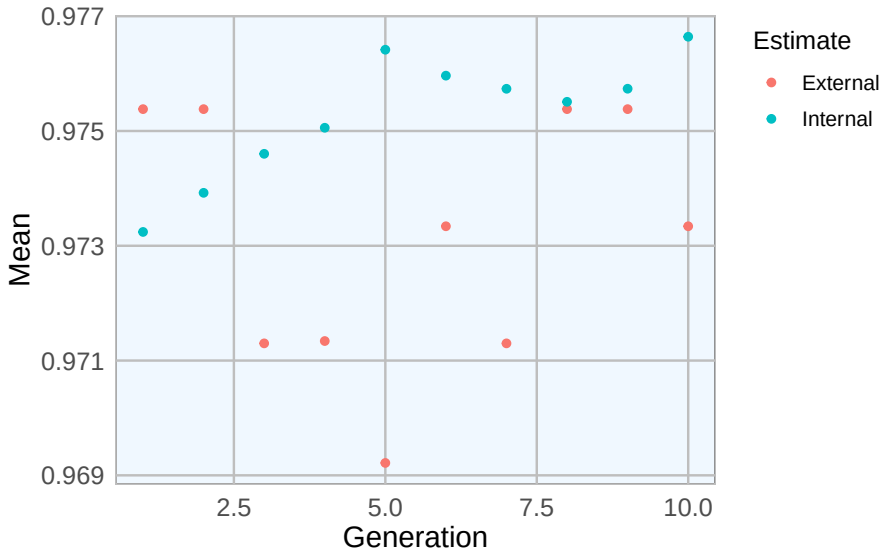
```
# confirm features removed by RFE
outersect(names(bc_data_rfe), names(bc_data_train))
#> [1] "mitosis"
```

No features removed with RFE

11.4.3 Genetic Algorithm (GA)

```
library(dplyr)
#>
#> Attaching package: 'dplyr'
#> The following object is masked from 'package:gridExtra':
#>
#>     combine
#> The following objects are masked from 'package:Biostrings':
#>
#>     collapse, intersect, setdiff, setequal, union
#> The following object is masked from 'package:XVector':
#>
#>     slice
#> The following objects are masked from 'package:IRanges':
#>
#>     collapse, desc, intersect, setdiff, slice, union
#> The following objects are masked from 'package:S4Vectors':
#>
#>     first, intersect, rename, setdiff, setequal, union
#> The following objects are masked from 'package:BiocGenerics':
#>
#>     combine, intersect, setdiff, union
#> The following objects are masked from 'package:stats':
#>
#>     filter, lag
#> The following objects are masked from 'package:base':
#>
#>     intersect, setdiff, setequal, union

ga_ctrl <- gafsControl(functions = rfGA, # Assess fitness with RF
                       method = "cv",    # 10 fold cross validation
                       genParallel = TRUE, # Use parallel programming
                       allowParallel = TRUE)

lev <- c("malignant", "benign")     # Set the levels

set.seed(27)
model_1 <- gafs(x = bc_data_train[, -1], y = bc_data_train$classes,
                  iters = 10, # generations of algorithm
                  popSize = 5, # population size for each generation
                  levels = lev,
                  gafsControl = ga_ctrl)
#>
#> Attaching package: 'recipes'
#> The following object is masked from 'package:stats':
#>
#>     step
```

```
plot(model_1) # Plot mean fitness (AUC) by generation
```




```
# features
model_1$ga$final
#> [1] "clump_thickness"          "uniformity_of_cell_size"
#> [3] "uniformity_of_cell_shape" "marginal_adhesion"
#> [5] "bare_nuclei"              "bland_chromatin"
#> [7] "normal_nucleoli"
```

```
# select features
sel_cols_ga <- which(colnames(bc_data_train) %in% model_1$ga$final)
bc_data_ga <- bc_data_train[, c(1, sel_cols_ga)]
names(bc_data_ga)
#> [1] "classes"                  "clump_thickness"
#> [3] "uniformity_of_cell_size"  "uniformity_of_cell_shape"
#> [5] "marginal_adhesion"        "bare_nuclei"
#> [7] "bland_chromatin"          "normal_nucleoli"
```

```
# features removed GA
outersect(names(bc_data_ga), names(bc_data_train))
#> [1] "mitosis"                    "single_epithelial_cell_size"
```

Two features removed with GA.

11.5 Model comparison

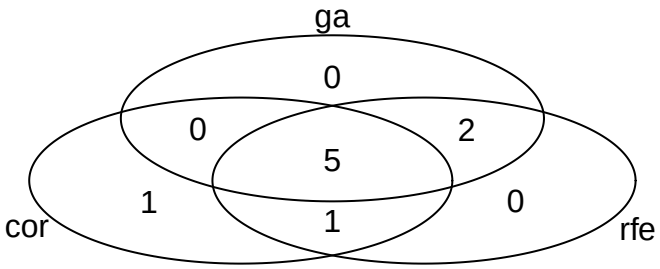
11.5.1 Using all features

```
set.seed(27)
model_bc_data_all <- train(classes ~ .,
                           data = bc_data_train,
                           method = "rf",
                           preProcess = c("scale", "center"),
                           trControl = trainControl(method = "repeatedcv",
                                                    number = 5, repeats = 10,
                                                    verboseIter = FALSE))
```

```
# confusion matrix
cm_all_1 <- confusionMatrix(predict(model_bc_data_all, bc_data_test[, -1]), bc_data_test$classes)
cm_all_1
#> Confusion Matrix and Statistics
#>
#>            Reference
#> Prediction  benign malignant
#>   benign       131         2
#>   malignant      6        70
#>
#>                Accuracy : 0.962
#>                  95% CI : (0.926, 0.983)
#>     No Information Rate : 0.656
#>     P-Value [Acc > NIR] : <2e-16
#>
#>                   Kappa : 0.916
#>
#>  Mcnemar's Test P-Value : 0.289
#>
#>             Sensitivity : 0.956
#>             Specificity : 0.972
#>          Pos Pred Value : 0.985
#>          Neg Pred Value : 0.921
#>              Prevalence : 0.656
#>          Detection Rate : 0.627
#>    Detection Prevalence : 0.636
#>       Balanced Accuracy : 0.964
#>
#>        'Positive' Class : benign
#>
```

11.5.2 Compare selection methods

```
# compare features selected by the three methods
library(gplots)
#>
#> Attaching package: 'gplots'
#> The following object is masked from 'package:IRanges':
#>
#>     space
#> The following object is masked from 'package:S4Vectors':
#>
#>     space
#> The following object is masked from 'package:stats':
#>
#>     lowess

venn_list <- list(cor = colnames(bc_data_cor)[-1],
                  rfe = colnames(bc_data_rfe)[-1],
                  ga  = colnames(bc_data_ga)[-1])

venn <- venn(venn_list)
```




```
venn
#>     num cor rfe ga
#> 000   0   0   0  0
#> 001   0   0   0  1
#> 010   0   0   1  0
#> 011   2   0   1  1
#> 100   1   1   0  0
#> 101   0   1   0  1
#> 110   1   1   1  0
#> 111   5   1   1  1
#> attr(,"intersections")
#> attr(,"intersections")$`cor:rfe:ga`
#> [1] "clump_thickness"   "marginal_adhesion" "bare_nuclei"
#> [4] "bland_chromatin"   "normal_nucleoli"
#>
#> attr(,"intersections")$cor
#> [1] "mitosis"
#>
#> attr(,"intersections")$`rfe:ga`
#> [1] "uniformity_of_cell_size"  "uniformity_of_cell_shape"
#>
#> attr(,"intersections")$`cor:rfe`
#> [1] "single_epithelial_cell_size"
#>
#> attr(,"class")
#> [1] "venn"
```

4 out of 10 features were chosen by all three methods; the biggest overlap is seen between GA and RFE with 7 features. RFE and GA both retained 8 features for modeling, compared to only 5 based on the correlation method.

11.5.3 Correlation

```
# correlation
set.seed(127)
model_bc_data_cor <- train(classes ~ .,
                 data = bc_data_cor,
                 method = "rf",
                 preProcess = c("scale", "center"),
                 trControl = trainControl(method = "repeatedcv", number = 5, repeats = 10, verboseIter =
```

```
cm_cor_1 <- confusionMatrix(predict(model_bc_data_cor, bc_data_test[, -1]), bc_data_test$classes)
cm_cor_1
#> Confusion Matrix and Statistics
#>
```

```
#>            Reference
#> Prediction  benign malignant
#>   benign       130         4
#>   malignant      7        68
#>
#>                Accuracy : 0.947
#>                  95% CI : (0.908, 0.973)
#>     No Information Rate : 0.656
#>     P-Value [Acc > NIR] : <2e-16
#>
#>                   Kappa : 0.885
#>
#>  Mcnemar's Test P-Value : 0.546
#>
#>             Sensitivity : 0.949
#>             Specificity : 0.944
#>          Pos Pred Value : 0.970
#>          Neg Pred Value : 0.907
#>              Prevalence : 0.656
#>          Detection Rate : 0.622
#>    Detection Prevalence : 0.641
#>       Balanced Accuracy : 0.947
#>
#>        'Positive' Class : benign
#>
```

11.5.4 Recursive Feature Elimination

```
set.seed(127)
model_bc_data_rfe <- train(classes ~ .,
                           data = bc_data_rfe,
                           method = "rf",
                           preProcess = c("scale", "center"),
                           trControl = trainControl(method = "repeatedcv",
                                                    number = 5, repeats = 10,
                                                    verboseIter = FALSE))
```

```
cm_rfe_1 <- confusionMatrix(predict(model_bc_data_rfe, bc_data_test[, -1]), bc_data_test$classes)
cm_rfe_1
#> Confusion Matrix and Statistics
#>
#>            Reference
#> Prediction  benign malignant
#>   benign       130         3
#>   malignant      7        69
#>
#>                Accuracy : 0.952
#>                  95% CI : (0.914, 0.977)
#>     No Information Rate : 0.656
#>     P-Value [Acc > NIR] : <2e-16
#>
#>                   Kappa : 0.895
```

```
#>
#>  Mcnemar's Test P-Value : 0.343
#>
#>             Sensitivity : 0.949
#>             Specificity : 0.958
#>          Pos Pred Value : 0.977
#>          Neg Pred Value : 0.908
#>              Prevalence : 0.656
#>          Detection Rate : 0.622
#>    Detection Prevalence : 0.636
#>       Balanced Accuracy : 0.954
#>
#>        'Positive' Class : benign
#>
```

11.5.5 GA

```
set.seed(127)
model_bc_data_ga <- train(classes ~ .,
                          data = bc_data_ga,
                          method = "rf",
                          preProcess = c("scale", "center"),
                          trControl = trainControl(method = "repeatedcv",
                                                   number = 5, repeats = 10,
                                                   verboseIter = FALSE))
```

```
cm_ga_1 <- confusionMatrix(predict(model_bc_data_ga, bc_data_test[, -1]), bc_data_test$classes)
cm_ga_1
#> Confusion Matrix and Statistics
#>
#>            Reference
#> Prediction  benign malignant
#>   benign       131         2
#>   malignant      6        70
#>
#>                Accuracy : 0.962
#>                  95% CI : (0.926, 0.983)
#>     No Information Rate : 0.656
#>     P-Value [Acc > NIR] : <2e-16
#>
#>                   Kappa : 0.916
#>
#>  Mcnemar's Test P-Value : 0.289
#>
#>             Sensitivity : 0.956
#>             Specificity : 0.972
#>          Pos Pred Value : 0.985
#>          Neg Pred Value : 0.921
#>              Prevalence : 0.656
#>          Detection Rate : 0.627
#>    Detection Prevalence : 0.636
#>       Balanced Accuracy : 0.964
```

```
#>
#>        'Positive' Class : benign
#>
```

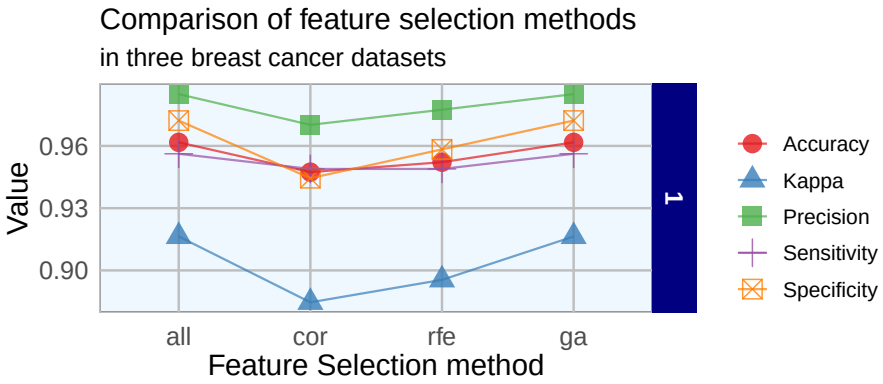
11.6 Create comparison tables

```
# take "overall" variable only from Confusion Matrix
overall <- data.frame(dataset = 1,
           model = rep(c("all", "cor", "rfe", "ga"), 1),
           rbind(cm_all_1$overall,
                 cm_cor_1$overall,
                 cm_rfe_1$overall,
                 cm_ga_1$overall)
)

# convert to tidy data
library(tidyr)
overall_gather <- overall[, 1:4] %>%     # take the first columns:
  gather(measure, value, Accuracy:Kappa) # dataset, model, Accuracy and Kappa
```

```
# take "byClass" variable only from Confusion Matrix
byClass <- data.frame(dataset = 1,
                      model = rep(c("all", "cor", "rfe", "ga"), 1),
                      rbind(cm_all_1$byClass,
                      cm_cor_1$byClass,
                      cm_rfe_1$byClass,
                      cm_ga_1$byClass)
)

# convert to tidy data
byClass_gather <- byClass[, c(1:4, 7)] %>%      # select columns: dataset, model
  gather(measure, value, Sensitivity:Precision) # Sensitiv, Specific, Precis
```

```
# join the two tables
overall_byClass_gather <- rbind(overall_gather, byClass_gather)
overall_byClass_gather <- within(
  overall_byClass_gather, model <- factor(model,
                                          levels = c("all", "cor", "rfe", "ga")))
                                          # convert to factor

ggplot(overall_byClass_gather, aes(x = model, y = value, color = measure,
                                   shape = measure, group = measure)) +
  geom_point(size = 4, alpha = 0.8) +
  geom_path(alpha = 0.7) +
  scale_colour_brewer(palette = "Set1") +
  facet_grid(dataset ~ ., scales = "free_y") +
  labs(
    x = "Feature Selection method",
    y = "Value",
    color = "",
    shape = "",
    title = "Comparison of feature selection methods",
```

```
    subtitle = "in three breast cancer datasets",
    caption = "\nBreast Cancer Wisconsin (Diagnostic) Data Sets: 1, 2 & 3
    Street et al., 1993;
    all: no feature selection
    cor: features with correlation > 0.7 removed
    rfe: Recursive Feature Elimination
    ga: Genetic Algorithm"
  )
```



Breast Cancer Wisconsin (Diagnostic) Data Sets: 1, 2 & 3
Street et al., 1993;
all: no feature selection
cor: features with correlation > 0.7 removed
rfe: Recursive Feature Elimination
ga: Genetic Algorithm

1. Less accurate: selection of features by correlation
2. More accurate: genetic algorithm
3. Including all features is more accurate to removing features by correlation.

11.7 Notes

`pcaGoPromoter` is a BioConductor package. Its dependencies are `BioGenerics`, `AnnotationDbi` and `BioStrings`, which at their turn require `DBI` and `RSQLite` packages from CRAN. Install first those from CRAN, and then move to install `pcaGoPromoter`.

Chapter 12

Titanic with Naive-Bayes Classifier

The Titanic dataset in R is a table for about 2200 passengers summarised according to four factors – economic status ranging from 1st class, 2nd class, 3rd class and crew; gender which is either male or female; Age category which is either Child or Adult and whether the type of passenger survived. For each combination of Age, Gender, Class and Survived status, the table gives the number of passengers who fall into the combination. We will use the Naive Bayes Technique to classify such passengers and check how well it performs.

```
#Getting started with Naive Bayes
#Install the package
#install.packages("e1071")
#Loading the library
library(e1071)

#Next load the Titanic dataset
data("Titanic")
#Save into a data frame and view it
Titanic_df = as.data.frame(Titanic)
```

We see that there are 32 observations which represent all possible combinations of Class, Sex, Age and Survived with their frequency. Since it is summarised, this table is not suitable for modelling purposes. We need to expand the table into individual rows. Let's create a repeating sequence of rows based on the frequencies in the table

```
#Creating data from table
repeating_sequence=rep.int(seq_len(nrow(Titanic_df)), Titanic_df$Freq) #This will repeat each combinati

# Create the dataset by row repetition created
Titanic_dataset=Titanic_df[repeating_sequence,]

# We no longer need the frequency, drop the feature
Titanic_dataset$Freq=NULL
```

The data is now ready for Naive Bayes to process. Let's fit the model

```
# Fitting the Naive Bayes model
Naive_Bayes_Model=naiveBayes(Survived ~., data=Titanic_dataset)

# What does the model say? Print the model summary
Naive_Bayes_Model
#>
```

```
#> Naive Bayes Classifier for Discrete Predictors
#>
#> Call:
#> naiveBayes.default(x = X, y = Y, laplace = laplace)
#>
#> A-priori probabilities:
#> Y
#>    No   Yes
#> 0.677 0.323
#>
#> Conditional probabilities:
#>      Class
#> Y        1st    2nd    3rd   Crew
#>   No  0.0819 0.1121 0.3544 0.4517
#>   Yes 0.2855 0.1660 0.2504 0.2982
#>
#>      Sex
#> Y       Male Female
#>   No  0.9154 0.0846
#>   Yes 0.5162 0.4838
#>
#>      Age
#> Y      Child  Adult
#>   No  0.0349 0.9651
#>   Yes 0.0802 0.9198
```

The model creates the conditional probability for each feature separately. We also have the a-priori probabilities which indicates the distribution of our data. Let's calculate how we perform on the data.

```
# Prediction on the dataset
NB_Predictions=predict(Naive_Bayes_Model,Titanic_dataset)
# Confusion matrix to check accuracy
table(NB_Predictions,Titanic_dataset$Survived)
#>
#> NB_Predictions   No  Yes
#>            No  1364  362
#>            Yes  126  349
```

We have the results! We are able to classify 1364 out of 1490 "No" cases correctly and 349 out of 711 "Yes" cases correctly. This means the ability of Naive Bayes algorithm to predict "No" cases is about 91.5% but it falls down to only 49% of the "Yes" cases resulting in an overall accuracy of 77.8%

Chapter 13

Can we Do any Better?

Naive Bayes is a parametric algorithm which implies that you cannot perform differently in different runs as long as the data remains the same. We will, however, learn another implementation of Naive Bayes algorithm using the 'mlr' package. Assuming the same session is going on for the readers, I will install and load the package and start fitting a model

```
# Getting started with Naive Bayes in mlr
# install.packages("mlr")
# Loading the library
library(mlr)
#> Loading required package: ParamHelpers
#> Registered S3 methods overwritten by 'ggplot2':
#>   method         from
#>   [.quosures     rlang
#>   c.quosures     rlang
#>   print.quosures rlang
#>
#> Attaching package: 'mlr'
#> The following object is masked from 'package:e1071':
#>
#>     impute
```

The mlr package consists of a lot of models and works by creating tasks and learners which are then trained. Let's create a classification task using the titanic dataset and fit a model with the naive bayes algorithm.

```
# Create a classification task for learning on Titanic Dataset and specify the target feature
task = makeClassifTask(data = Titanic_dataset, target = "Survived")

# Initialize the Naive Bayes classifier
selected_model = makeLearner("classif.naiveBayes")

# Train the model
NB_mlr = train(selected_model, task)
```

The summary of the model which was printed in e3071 package is stored in learner model. Let's print it and compare

```
# Read the model learned
NB_mlr$learner.model
#>
#> Naive Bayes Classifier for Discrete Predictors
```

```
#>
#> Call:
#> naiveBayes.default(x = X, y = Y, laplace = laplace)
#>
#> A-priori probabilities:
#> Y
#>    No   Yes
#> 0.677 0.323
#>
#> Conditional probabilities:
#>      Class
#> Y        1st    2nd    3rd   Crew
#>   No  0.0819 0.1121 0.3544 0.4517
#>   Yes 0.2855 0.1660 0.2504 0.2982
#>
#>      Sex
#> Y       Male Female
#>   No  0.9154 0.0846
#>   Yes 0.5162 0.4838
#>
#>      Age
#> Y      Child  Adult
#>   No  0.0349 0.9651
#>   Yes 0.0802 0.9198
```

The a-priori probabilities and the conditional probabilities for the model are similar to the one calculated by e3071 package as was expected. This means that our predictions will also be the same.

```
# Predict on the dataset without passing the target feature
predictions_mlr = as.data.frame(predict(NB_mlr, newdata = Titanic_dataset[,1:3]))

## Confusion matrix to check accuracy
table(predictions_mlr[,1],Titanic_dataset$Survived)
#>
#>         No  Yes
#>   No  1364  362
#>   Yes  126  349
```

As we see, the predictions are exactly same. The only way to improve is to have more features or more data. Perhaps, if we have more features such as the exact age, size of family, number of parents in the ship and siblings then we may arrive at a better model using Naive Bayes. In essence, Naive Bayes has an advantage of a strong foundation build and is very robust. I know of the 'caret' package which also consists of Naive Bayes function but it will also give us the same predictions and probability.

Chapter 14

Building a Naive Bayes Classifier in R

https://www.machinelearningplus.com/predictive-modeling/how-naive-bayes-algorithm-works-with-example-and-full-code/

14.1 8. Building a Naive Bayes Classifier in R

Understanding Naive Bayes was the (slightly) tricky part. Implementing it is fairly straightforward.

In R, Naive Bayes classifier is implemented in packages such as `e1071`, `klaR` and `bnlearn`. In Python, it is implemented in `scikit-learn`.

For sake of demonstration, let's use the standard iris dataset to predict the Species of flower using 4 different features: Sepal.Length, Sepal.Width, Petal.Length, Petal.Width

```
# Import Data
training <- read.csv('https://raw.githubusercontent.com/selva86/datasets/master/iris_train.csv')
test <- read.csv('https://raw.githubusercontent.com/selva86/datasets/master/iris_test.csv')
```

The training data is now contained in training and test data in test dataframe. Lets load the klaR package and build the naive bayes model.

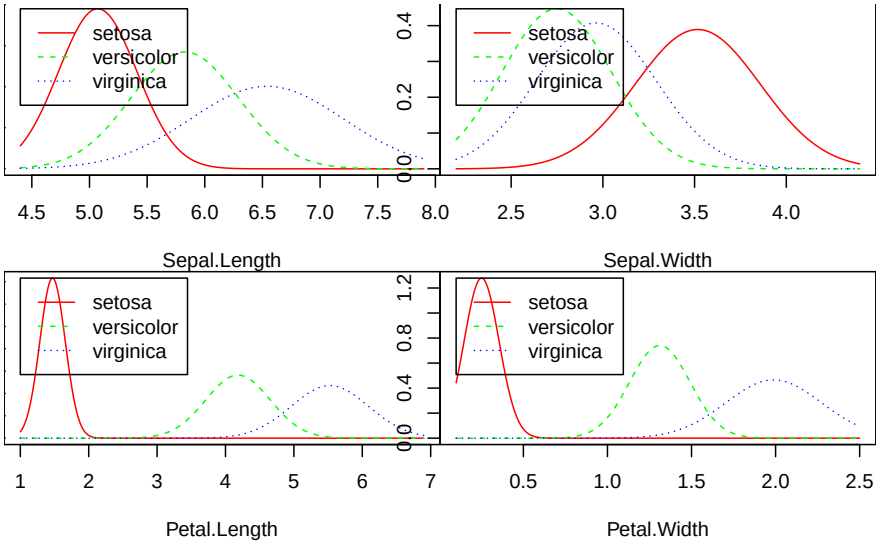
```
# Using klaR for Naive Bayes
library(klaR)
#> Loading required package: MASS
nb_mod <- NaiveBayes(Species ~ ., data=training)
pred <- predict(nb_mod, test)
```

Lets see the confusion matrix.

```
# Confusion Matrix
tab <- table(pred$class, test$Species)
caret::confusionMatrix(tab)
#> Registered S3 methods overwritten by 'ggplot2':
#>   method         from
#>   [.quosures     rlang
#>   c.quosures     rlang
#>   print.quosures rlang
#> Confusion Matrix and Statistics
#>
#>
#>              setosa versicolor virginica
```

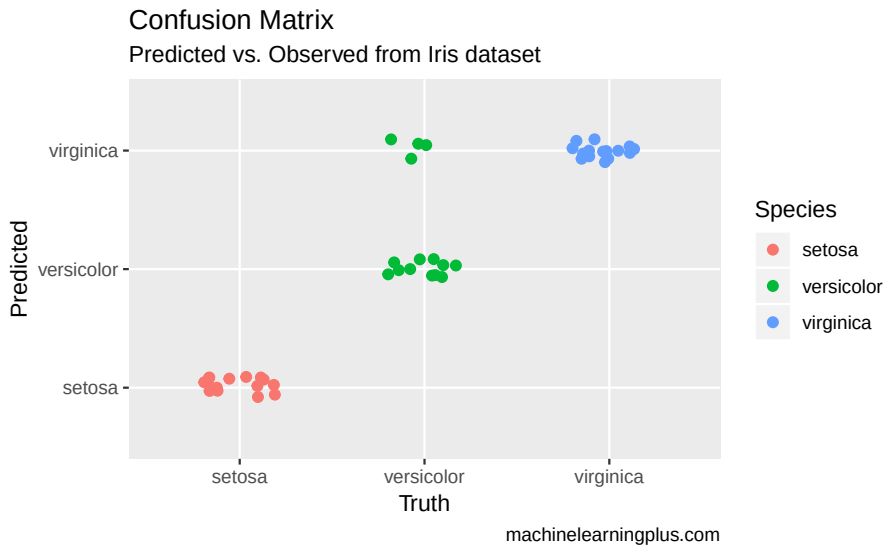
```
#>   setosa          15          0         0
#>   versicolor       0         11         0
#>   virginica        0          4        15
#>
#> Overall Statistics
#>
#>                Accuracy : 0.911
#>                  95% CI : (0.788, 0.975)
#>     No Information Rate : 0.333
#>     P-Value [Acc > NIR] : 8.47e-16
#>
#>                   Kappa : 0.867
#>
#>  Mcnemar's Test P-Value : NA
#>
#> Statistics by Class:
#>
#>                      Class: setosa Class: versicolor Class: virginica
#> Sensitivity                 1.000             0.733            1.000
#> Specificity                 1.000             1.000            0.867
#> Pos Pred Value              1.000             1.000            0.789
#> Neg Pred Value              1.000             0.882            1.000
#> Prevalence                  0.333             0.333            0.333
#> Detection Rate              0.333             0.244            0.333
#> Detection Prevalence        0.333             0.244            0.422
#> Balanced Accuracy           1.000             0.867            0.933
```

```
# Plot density of each feature using nb_mod
opar = par(mfrow=c(2, 2), mar=c(4,0,0,0))
plot(nb_mod, main="")
par(opar)
```




```
# Plot the Confusion Matrix
library(ggplot2)
test$pred <- pred$class
ggplot(test, aes(Species, pred, color = Species)) +
```

```
  geom_jitter(width = 0.2, height = 0.1, size=2) +
  labs(title="Confusion Matrix",
       subtitle="Predicted vs. Observed from Iris dataset",
       y="Predicted",
       x="Truth",
       caption="machinelearningplus.com")
```



Chapter 15

Logistic Regression. Diabetes

15.1 Introduction

Source: https://github.com/AntoineGuillot2/Logistic-Regression-R/blob/master/script.R Source: http://enhancedatascience.com/2017/04/26/r-basics-logistic-regression-with-r/ Data: https://www.kaggle.com/uciml/pima-indians-diabetes-database

The goal of logistic regression is to predict whether an outcome will be positive (aka 1) or negative (i.e: 0). Some real life example could be:

- Will Emmanuel Macron win the French Presidential election or will he lose?
- Does Mr.X has this illness or not?
- Will this visitor click on my link or not?

So, logistic regression can be used in a lot of binary classification cases and will often be run before more advanced methods. For this tutorial, we will use the diabetes detection dataset from Kaggle.

This dataset contains data from Pima Indians Women such as the number of pregnancies, the blood pressure, the skin thickness, … the goal of the tutorial is to be able to detect diabetes using only these measures.

15.2 Exploring the data

As usual, first, let's take a look at our data. You can download the data here then please put the file diabetes.csv in your working directory. With the summary function, we can easily summarise the different variables:

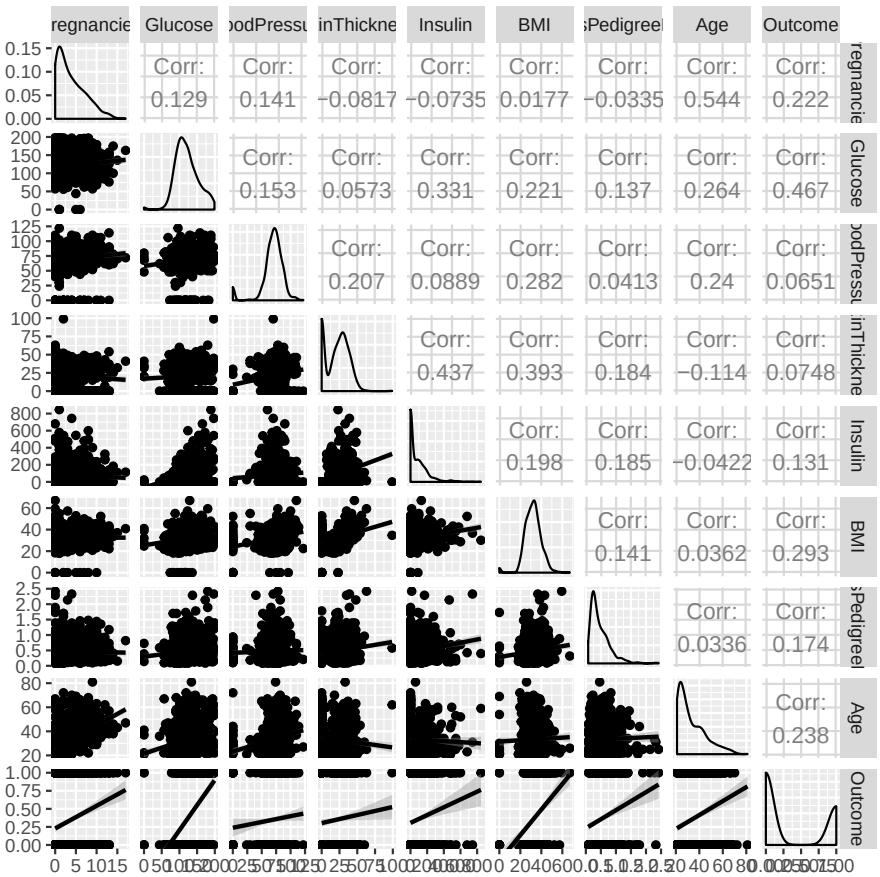
```
library(ggplot2)
#> Registered S3 methods overwritten by 'ggplot2':
#>   method         from
#>   [.quosures     rlang
#>   c.quosures     rlang
#>   print.quosures rlang
library(data.table)

DiabetesData <- data.table(read.csv(file.path(data_raw_dir, 'diabetes.csv')))

# Quick data summary
summary(DiabetesData)
#>   Pregnancies       Glucose    BloodPressure   SkinThickness
```

```
#>  Min.   : 0.00   Min.   :  0   Min.   :  0.0   Min.   : 0.0
#>  1st Qu.: 1.00   1st Qu.: 99   1st Qu.: 62.0   1st Qu.: 0.0
#>  Median : 3.00   Median :117   Median : 72.0   Median :23.0
#>  Mean   : 3.85   Mean   :121   Mean   : 69.1   Mean   :20.5
#>  3rd Qu.: 6.00   3rd Qu.:140   3rd Qu.: 80.0   3rd Qu.:32.0
#>  Max.   :17.00   Max.   :199   Max.   :122.0   Max.   :99.0
#>     Insulin         BMI       DiabetesPedigreeFunction      Age
#>  Min.   :  0   Min.   : 0.0   Min.   :0.078            Min.   :21.0
#>  1st Qu.:  0   1st Qu.:27.3   1st Qu.:0.244            1st Qu.:24.0
#>  Median : 30   Median :32.0   Median :0.372            Median :29.0
#>  Mean   : 80   Mean   :32.0   Mean   :0.472            Mean   :33.2
#>  3rd Qu.:127   3rd Qu.:36.6   3rd Qu.:0.626            3rd Qu.:41.0
#>  Max.   :846   Max.   :67.1   Max.   :2.420            Max.   :81.0
#>     Outcome
#>  Min.   :0.000
#>  1st Qu.:0.000
#>  Median :0.000
#>  Mean   :0.349
#>  3rd Qu.:1.000
#>  Max.   :1.000
```

The mean of the outcome is 0.35 which shows an imbalance between the classes. However, the imbalance should not be too strong to be a problem.

To understand the relationship between variables, a Scatter Plot Matrix will be used. To plot it, the GGally package was used.

```
# Scatter plot matrix
library(GGally)
#> Registered S3 method overwritten by 'GGally':
#>   method from
#>   +.gg   ggplot2
ggpairs(DiabetesData, lower = list(continuous='smooth'))
```



The correlations between explanatory variables do not seem too strong. Hence the model is not likely to suffer from multicollinearity. All explanatory variable are correlated with the Outcome. At first sight, glucose rate is the most important factor to detect the outcome.

15.3 Logistic regression with R

After variable exploration, a first model can be fitted using the glm function. With stargazer, it is easy to get nice output in ASCII or even Latex.

```
# first model: all features
glm1 = glm(Outcome~.,
           DiabetesData,
           family = binomial(link="logit"))

summary(glm1)
#>
#> Call:
#> glm(formula = Outcome ~ ., family = binomial(link = "logit"),
#>     data = DiabetesData)
#>
#> Deviance Residuals:
#>    Min      1Q  Median      3Q     Max
#> -2.557  -0.727  -0.416   0.727   2.930
#>
#> Coefficients:
```

```
#>                            Estimate Std. Error z value Pr(>|z|)
#> (Intercept)               -8.404696   0.716636  -11.73  < 2e-16 ***
#> Pregnancies                0.123182   0.032078    3.84  0.00012 ***
#> Glucose                    0.035164   0.003709    9.48  < 2e-16 ***
#> BloodPressure             -0.013296   0.005234   -2.54  0.01107 *
#> SkinThickness              0.000619   0.006899    0.09  0.92852
#> Insulin                   -0.001192   0.000901   -1.32  0.18607
#> BMI                        0.089701   0.015088    5.95  2.8e-09 ***
#> DiabetesPedigreeFunction  0.945180   0.299147    3.16  0.00158 **
#> Age                        0.014869   0.009335    1.59  0.11119
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> (Dispersion parameter for binomial family taken to be 1)
#>
#>     Null deviance: 993.48  on 767  degrees of freedom
#> Residual deviance: 723.45  on 759  degrees of freedom
#> AIC: 741.4
#>
#> Number of Fisher Scoring iterations: 5
require(stargazer)
#> Loading required package: stargazer
#>
#> Please cite as:
#>  Hlavac, Marek (2018). stargazer: Well-Formatted Regression and Summary Statistics Tables.
#>  R package version 5.2.2. https://CRAN.R-project.org/package=stargazer
stargazer(glm1,type='text')
#>
#> ================================================
#>                              Dependent variable:
#>                            ---------------------
#>                                     Outcome
#> ------------------------------------------------
#> Pregnancies                        0.123***
#>                                     (0.032)
#>
#> Glucose                            0.035***
#>                                     (0.004)
#>
#> BloodPressure                      -0.013**
#>                                     (0.005)
#>
#> SkinThickness                       0.001
#>                                     (0.007)
#>
#> Insulin                            -0.001
#>                                     (0.001)
#>
#> BMI                                0.090***
#>                                     (0.015)
#>
#> DiabetesPedigreeFunction           0.945***
#>                                     (0.299)
```

```
#>
#> Age                                0.015
#>                                   (0.009)
#>
#> Constant                         -8.400***
#>                                   (0.717)
#>
#> ------------------------------------------------
#> Observations                        768
#> Log Likelihood                   -362.000
#> Akaike Inf. Crit.                 741.000
#> ================================================
#> Note:                  *p<0.1; **p<0.05; ***p<0.01
```

The overall model is significant. As expected the glucose rate has the lowest p-value of all the variables. However, Age, Insulin and Skin Thickness are not good predictors of Diabetes.

15.4 A second model

Since some variables are not significant, removing them is a good way to improve model robustness. In the second model, SkinThickness, Insulin, and Age are removed.

```
# second model: selected features
glm2 = glm(Outcome~.,
         data = DiabetesData[,c(1:3,6:7,9), with=F],
         family = binomial(link="logit"))

summary(glm2)
#>
#> Call:
#> glm(formula = Outcome ~ ., family = binomial(link = "logit"),
#>     data = DiabetesData[, c(1:3, 6:7, 9), with = F])
#>
#> Deviance Residuals:
#>    Min      1Q  Median      3Q     Max
#> -2.793  -0.736  -0.419   0.725   2.955
#>
#> Coefficients:
#>                           Estimate Std. Error z value Pr(>|z|)
#> (Intercept)               -7.95495    0.67582  -11.77  < 2e-16 ***
#> Pregnancies                0.15349    0.02784    5.51  3.5e-08 ***
#> Glucose                    0.03466    0.00339   10.21  < 2e-16 ***
#> BloodPressure             -0.01201    0.00503   -2.39    0.017 *
#> BMI                        0.08483    0.01412    6.01  1.9e-09 ***
#> DiabetesPedigreeFunction   0.91063    0.29403    3.10    0.002 **
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> (Dispersion parameter for binomial family taken to be 1)
#>
#>     Null deviance: 993.48  on 767  degrees of freedom
#> Residual deviance: 728.56  on 762  degrees of freedom
#> AIC: 740.6
```

```
#>
#> Number of Fisher Scoring iterations: 5
```

15.5 Classification rate and confusion matrix

Now that we have our model, let's access its performance.

```
# Correctly classified observations
mean((glm2$fitted.values>0.5)==DiabetesData$Outcome)
#> [1] 0.775
```

Around 77.4% of all observations are correctly classified. Due to class imbalance, we need to go further with a confusion matrix.

```
###Confusion matrix count
RP=sum((glm2$fitted.values>=0.5)==DiabetesData$Outcome & DiabetesData$Outcome==1)
FP=sum((glm2$fitted.values>=0.5)!=DiabetesData$Outcome & DiabetesData$Outcome==0)
RN=sum((glm2$fitted.values>=0.5)==DiabetesData$Outcome & DiabetesData$Outcome==0)
FN=sum((glm2$fitted.values>=0.5)!=DiabetesData$Outcome & DiabetesData$Outcome==1)
confMat<-matrix(c(RP,FP,FN,RN),ncol = 2)
colnames(confMat)<-c("Pred Diabetes",'Pred no diabetes')
rownames(confMat)<-c("Real Diabetes",'Real no diabetes')
confMat
#>                  Pred Diabetes Pred no diabetes
#> Real Diabetes              154              114
#> Real no diabetes            59              441
```

The model is good to detect people who do not have diabetes. However, its performance on ill people is not great (only 154 out of 268 have been correctly classified).

You can also get the percentage of Real/False Positive/Negative:

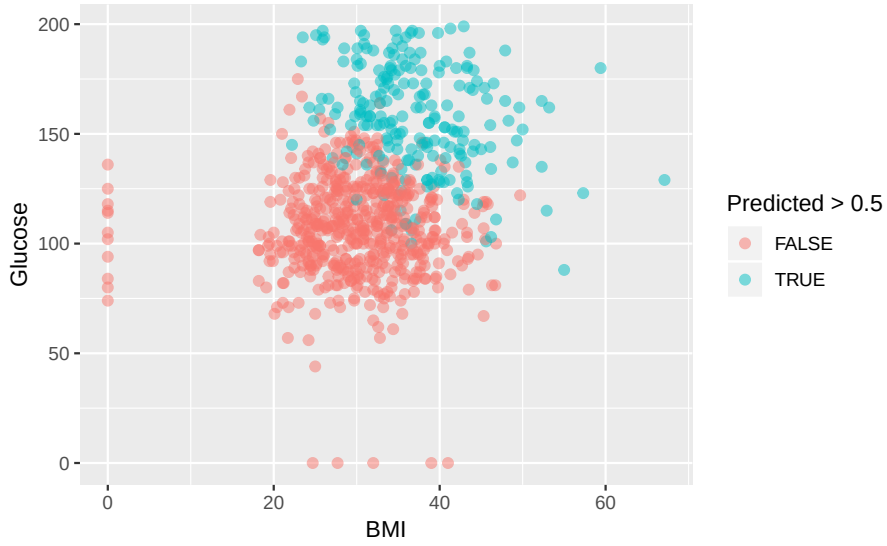
```
# Confusion matrix proportion
RPR=RP/sum(DiabetesData$Outcome==1)*100
FNR=FN/sum(DiabetesData$Outcome==1)*100
FPR=FP/sum(DiabetesData$Outcome==0)*100
RNR=RN/sum(DiabetesData$Outcome==0)*100
confMat<-matrix(c(RPR,FPR,FNR,RNR),ncol = 2)
colnames(confMat)<-c("Pred Diabetes",'Pred no diabetes')
rownames(confMat)<-c("Real Diabetes",'Real no diabetes')
confMat
#>                  Pred Diabetes Pred no diabetes
#> Real Diabetes             57.5             42.5
#> Real no diabetes          11.8             88.2
```

And here is the matrix, 57.5% of people with diabetes are correctly classified. A way to improve the false negative rate would lower the detection threshold. However, as a consequence, the false positive rate would increase.

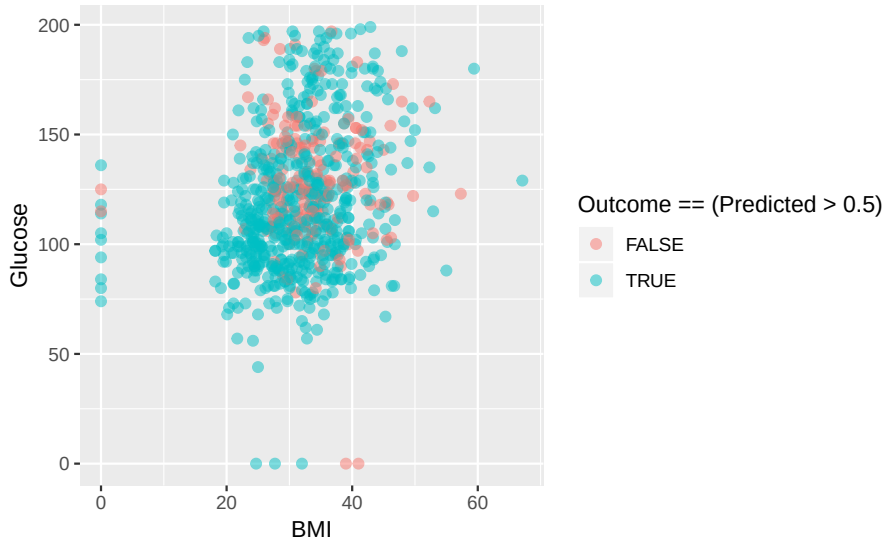
15.6 Plots and decision boundaries

The two strongest predictors of the outcome are Glucose rate and BMI. High glucose rate and high BMI are strong indicators of Diabetes.

```
# Plot and decision boundaries
DiabetesData$Predicted <- glm2$fitted.values

ggplot(DiabetesData, aes(x = BMI, y = Glucose, color = Predicted > 0.5)) +
    geom_point(size=2, alpha=0.5)
```




```
ggplot(DiabetesData, aes(x=BMI, y = Glucose, color=Outcome == (Predicted > 0.5))) +
    geom_point(size=2, alpha=0.5)
```

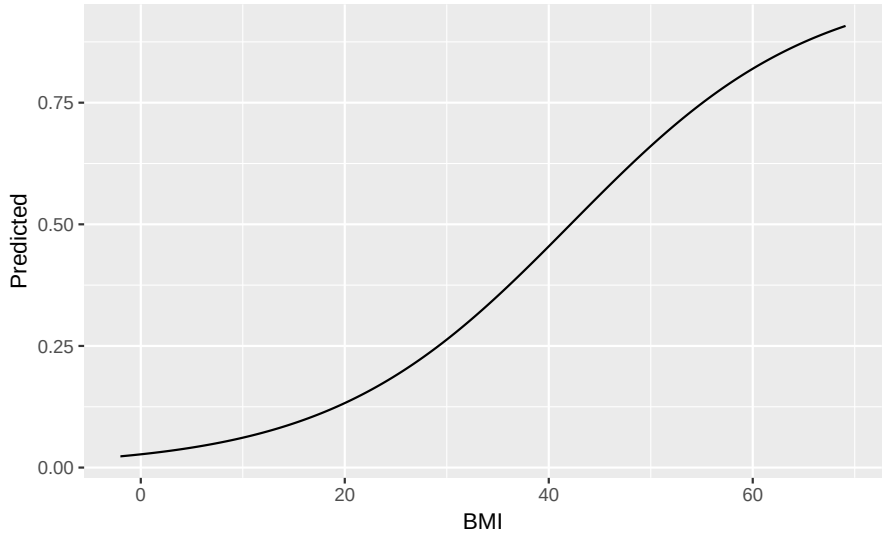


We can also plot both BMI and glucose against the outcomes, the other variables are taken at their mean level.

```
range(DiabetesData$BMI)
#> [1]  0.0 67.1
```

```
# BMI vs predicted
BMI_plot = data.frame(BMI = ((min(DiabetesData$BMI-2)*100):
                               (max(DiabetesData$BMI+2)*100))/100,
                      Glucose = mean(DiabetesData$Glucose),
                      Pregnancies = mean(DiabetesData$Pregnancies),
```

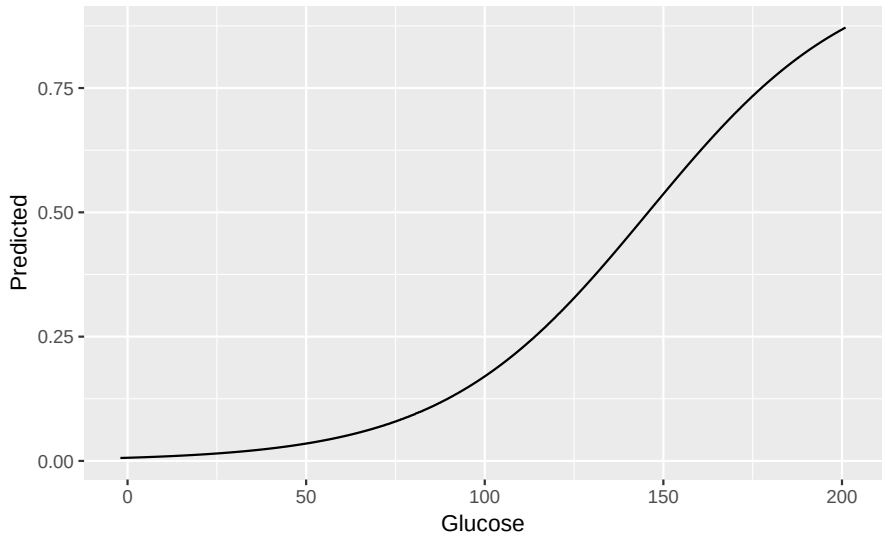
```
                    BloodPressure = mean(DiabetesData$BloodPressure),
                    DiabetesPedigreeFunction = mean(DiabetesData$DiabetesPedigreeFunction))

BMI_plot$Predicted = predict(glm2, newdata = BMI_plot, type = 'response')
ggplot(BMI_plot, aes(x = BMI, y = Predicted)) +
    geom_line()
```




```
range(BMI_plot$BMI)
#> [1] -2.0 69.1
```

```
range(DiabetesData$Glucose)
#> [1]   0 199
```

```
# Glucose vs predicted
Glucose_plot=data.frame(Glucose =
                        ((min(DiabetesData$Glucose-2)*100):
                             (max(DiabetesData$Glucose+2)*100))/100,
                    BMI=mean(DiabetesData$BMI),
                    Pregnancies=mean(DiabetesData$Pregnancies),
                    BloodPressure=mean(DiabetesData$BloodPressure),
                    DiabetesPedigreeFunction=mean(DiabetesData$DiabetesPedigreeFunction))

Glucose_plot$Predicted = predict(glm2, newdata = Glucose_plot, type = 'response')
ggplot(Glucose_plot, aes(x = Glucose, y = Predicted)) +
    geom_line()
```




```
range(Glucose_plot$Glucose)
#> [1]  -2 201
```