Prediction Assignment

Sofia

22 März 2017

Introduction

This assignment is about predicting the type of physical exercises by analysing the data from several accelerometers. The original data can be found here:

Large *.csv file (~ 12 MB)

In this experiment, a certain exercise was performed by several subjects, sometimes in a correct manner (Class A) and sometimes including one of four common mistakes (Class B to E). The goal is to develop an algorithm that is able to predict the type of execution by analysing the accelerometer information.

Import and prepare the data

First, download the data and set the working directory to the correct folder. In this assignment only the "training" data set will be used. The "testing" data set will only be used for the quiz part of the assignment and is not needed here.

```
training_raw <- read.csv("pml-training.csv")
```

The loaded data frame has 160 variables. Most of them contain information from the different accelerator devices (position data, gyroscopic data and so on). There's also some information about the experiment set-up like the name of the executing test subject.

A first inspection of the data shows that there are many NA values, empty cells and "#DIV/0!" values in the training raw data.

```
#Calculate the total number of values in the data frame
no_values <- dim(training_raw)[1] * dim(training_raw)[2]

#Calculate the percentage of NA, empty and "#DIV/0!" values
sum(is.na(training_raw)) / no_values
```

```
## [1] 0.4100856
```

```
sum(training_raw == "", na.rm = TRUE) / no_values
```

```
## [1] 0.2019825
```

```
sum(training_raw == "#DIV/0!", na.rm = TRUE) / no_values
```

```
## [1] 0.001115457
```

You can see that there's all in all more than 60 % of missing values in the data set. Some of the columns contain only very few usable values, so they can be excluded from the analysis.

```r
#For each column: Calculate the percentage of NA, empty values and "#DIV/0!"
exclude_na <- sapply(training_raw, function(x) (sum(is.na(x)) / length(x)))
exclude_empty <- sapply(training_raw, function(x) (sum(x == "", na.rm = TRUE) / length(x)))
exclude_div <- sapply(training_raw, function(x) (sum(x == "#DIV/0!", na.rm = TRUE) / length(x)))

#Find columns with more than 90% of missing data
exclude <- (exclude_na + exclude_empty + exclude_div) > 0.9

#Exclude those colums from the data frame
training_clean <- training_raw[ , !exclude]
```

After you deleted all columns with more than 90 % missing values, you end up with 60 remaining columns. In the next step, you also can delete the first 7 columns because they don't contain information with any value for the prediction, e. g. the name of the executing person or the timestamp of the exercise:

```r
training <- training_clean[ , -c(1:7)]
```

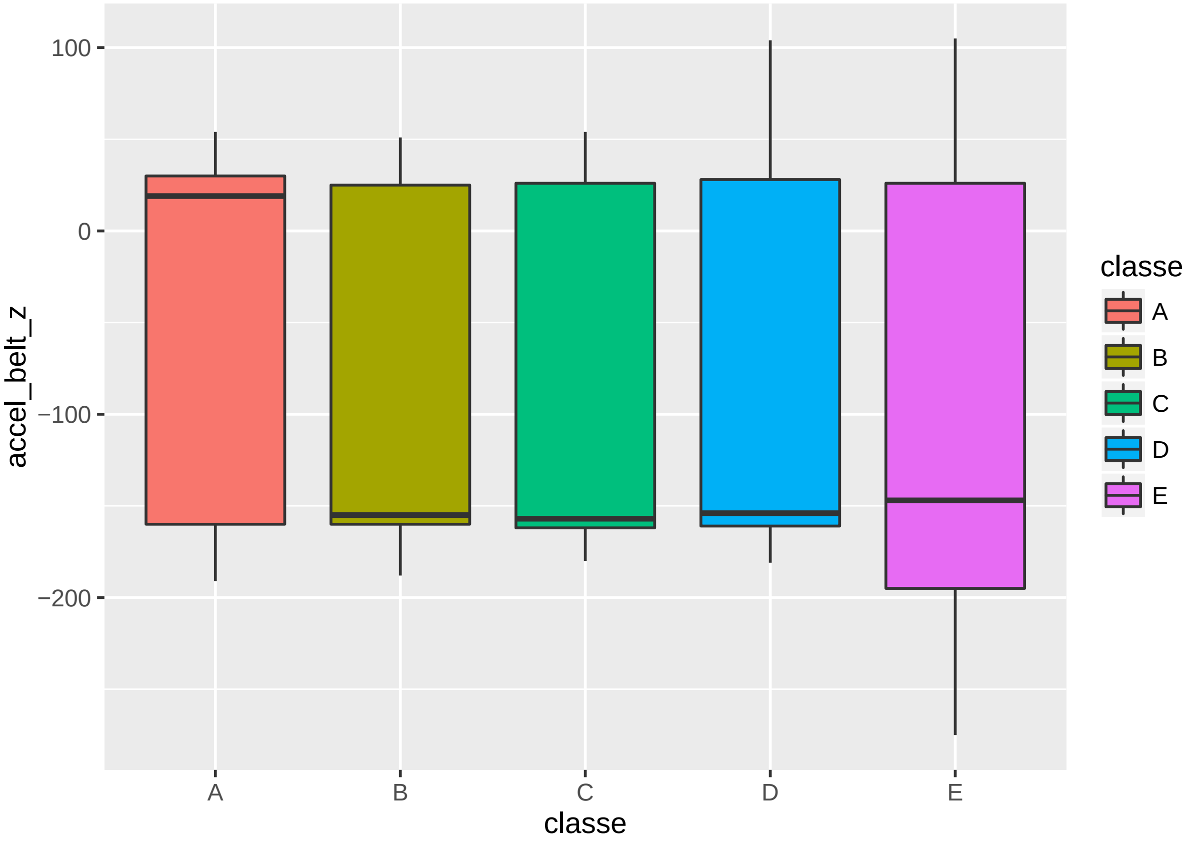
Exploratory analysis

As we have more than 50 remaining variables, it is not possible to show the complete exploratory analysis and all plots here. But as an example we will take a look at the boxplot of one of the variables ("accel_belt_z"):

```r
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 3.1.3
```

```r
qplot(classe, accel_belt_z, data = training, geom = "boxplot", fill = classe)
```



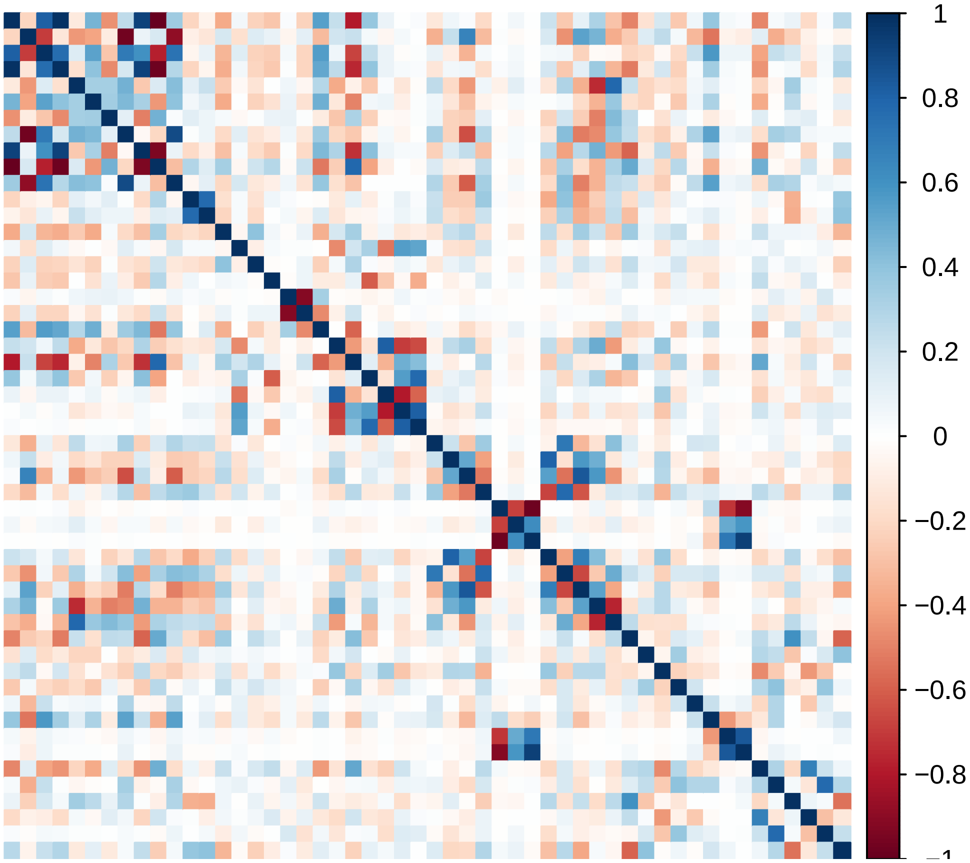
You can see that, although the range does not differ much between the classes, class A has a complementary skew compared to the other classes. We can observe the same pattern for other variables as well. For those variables, the mean value may be a good kind of predictor.

It can also be useful to take a look at the correlation matrix. You can use the package "corrplot" to create an appropriate plot. The column "classe" has to be excluded, because it is non-numerical.

```
library(corrplot)
```

```
## Warning: package 'corrplot' was built under R version 3.1.3
```

```
correlations <- cor(training[-53])
corrplot(correlations, method="color", tl.pos="n")
```



The plot mainly shows light colors. That means that most of the variables are quite independent and it does not seem necessary to try and eliminate unnecessary predictors. However, if we decide to do so, we can use Principal Component Analysis.

Data sclicing and cross validation

Before you start to build the prediction algorithm, you need to think about splitting up our data frame into training and testing data sets. Fortunately, the caret package takes care of data splitting and cross validation automatically. For this project a 10x10 cross validation is used, this means the train() function will create 10 folds and repeat this process 10 times. The calculation may take a while, but it gives us a good estimate of the accuracy.

The cross validation parameter will be set in the trainControl() function.

Model 1: Decision tree

The variable that we are going to predict is the "classe" column. It contains categorical data with the following factor levels:

```
levels(training$classe)
```

```
## [1] "A" "B" "C" "D" "E"
```

This means you need a classification algorithm. The first model will be a **Decision Tree** algorithm. You can use the train() function of the caret package and set the cross validation parameters like explained above:

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 3.1.3

## Loading required package: lattice
```

```
set.seed(3005)
mod1 <- train(classe ~ ., data = training, method = "rpart",
              trControl = trainControl(method = "cv",
                                       number = 10,
                                       repeats = 10))
```

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

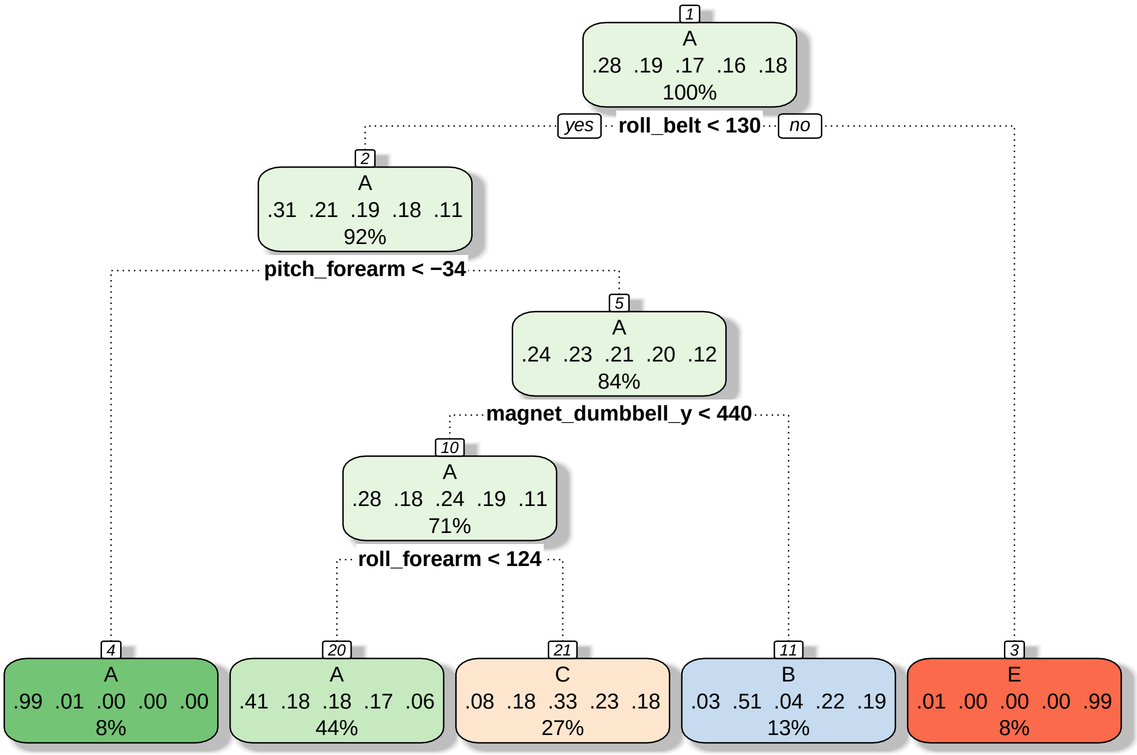
The resulting model can be visualized in a nice way with the fancyRpartPlot() function of the “rattle” package:

```
library(rattle)
```

```
## Warning: package 'rattle' was built under R version 3.1.3

## Rattle: Ein kostenloses grafisches Interface für Data Mining mit R.
## Version 4.1.0 Copyright (c) 2006-2015 Togaware Pty Ltd.
## Geben Sie 'rattle()' ein, um Ihre Daten mischen.
```

```
fancyRpartPlot(mod1$finalModel)
```



Rattle 2017−Apr−02 18:59:00 sofia

All in all, the model is not very good. The overall accuracy is only about 50 %:

```
print(mod1)
```

```
## CART
##
## 19622 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 17661, 17659, 17661, 17660, 17658, 17659, ...
## Resampling results across tuning parameters:
##
##   cp          Accuracy   Kappa
##   0.03567868  0.5092256  0.35911158
##   0.05998671  0.4276257  0.22844357
##   0.11515454  0.3239814  0.06031535
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was cp = 0.03567868.
```

Model 2: Random Forest

In the next step we try another model to see if the accuracy can be improved. The **Random Forest** algorithm combines and averages multiple Decision Trees and correts the Decision Trees' habit for overfitting. You can again use the train() function:

```
mod2 <- train(classe ~ .,
              data = training,
              method = "rf",
              trControl = trainControl(
                  method = "cv",
                  number = 10,
                  repeats = 10))
```

```
## Loading required package: randomForest
```

```
## Warning: package 'randomForest' was built under R version 3.1.3
```

```
## randomForest 4.6-12
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

The resulting model has a much higher accuracy of about 99%. As the train() function used cross validation we can expect a similar accuracy also for predicting on completely unknown data. This means the expected **out-of-sample error** is about 1 %. Details about the model can be printed out by:

```
print(mod2)
```

```
## Random Forest
##
## 19622 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 17659, 17660, 17661, 17659, 17660, 17658, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9954646  0.9942630
##   27    0.9948021  0.9934249
##   52    0.9903172  0.9877503
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 2.
```

Conclusion

In this assignment it could be shown how Random Forests can heavily improve the accuracy compared to Decision Tree algorithms. While the Decision Tree only has an accuracy of about 50 %, the Randon Forest hits values up to more than 99 %. On the other hand, the interpretation of Random Forest algorithms is difficult, while the Decision Tree algorithm can easily be plotted or described with words.