# Practical Machine Learning: Prediction Assignment

*Robert Emerencia*

*October 7, 2016*

## Contents

# Executive summary

A dataset consisting of various accelerometer measurements is used to predict 5 different ways to perform a weight lifting exercise. The received dataset is cleaned by removing all missing and erroneous values. Additional preprocessing is done by removing outliers and irrelevant features.

Three different classifiers are trained after splitting the dataset into a training set (for training a model) and a validation set (for testing the classifier on new observations). The random forest classifier yields the best results with a 99.5% accuracy on the validation set. This model is applied to 20 different test cases with an obtained accuracy of 100%.

# Introduction

In this analysis the Weight Lifting Exercises (WLE) dataset is analyzed and used to train and evaluate different machine learning models. The dataset consists of measurements from accelerometers on the belt, forearm, arm, and dumbell. Six participants were asked to perform barbell lifts correctly and incorrectly in 5 different ways (classes A-E).

In Section 2 the dataset is downloaded and cleaned. In Section 3 an exploratory data analysis (EDA) is performed to get acquainted with the data.

Based on the EDA a set of preprocessing operations is performed in Section 4. Different classifiers are trained and evaluated in Section 5, after which a final model is chosen and applied to a small test set of 20 observations.

# Data cleaning

## Download and missing values

The training and test datasets are downloaded and loaded into R.

```
# function to download data
getData <- function(downloadURL, dsName) {
    if (!(file.exists(dsName))) {
        download.file(downloadURL, dsName)
        downloadDate <- date()
        print(paste("File", dsName, "downloaded. Date:", downloadDate))
    } else {
        print(paste("File", dsName, "found in working directory."))
    }
}
# get training and test sets
dsTrain <- "./data/pml-training.csv"
dsTest <- "./data/pml-testing.csv"
urlData <- "https://d396qusza40orc.cloudfront.net/predmachlearn/"
urlTrain <- paste(urlData, "pml-training.csv", sep = "")
urlTest <- paste(urlData, "pml-testing.csv", sep = "")
getData(urlTrain, dsTrain)
```

```
## [1] "File ./data/pml-training.csv found in working directory."
```

```
getData(urlTest, dsTest)
```

```
## [1] "File ./data/pml-testing.csv found in working directory."
```

```
# load into R
pmlTrain <- read.csv(dsTrain)
pmlTest <- read.csv(dsTest)
dim(pmlTrain)
```

```
## [1] 19622   160
```

```
dim(pmlTest)
```

```
## [1]  20 160
```

The training set contains 19622 observations, 159 features and a single outcome `classe`. The test set contains 19622 observations, 159 features and an identification feature `problem_id`.

The datasets are briefly opened in a spreadspreet program to check for obvious errors. There appears to be a large amount of missing values (`NA`) as well as blanks and erroneous values (`#DIV/0!`). Blanks and erroneous values are converted to `NA` after temporarily merging the training and test sets.

```
# temporarily merge the datasets (excluding the outcome)
pmlTrain$train <- 1
pmlTest$train <- 0
keepTrain <- !(colnames(pmlTrain) %in% "classe")
keepTest <- !(colnames(pmlTest) %in% "problem_id")
pmlData <- rbind(pmlTrain[, keepTrain], pmlTest[, keepTest])
# replace some erroneous values by NA
pmlData[pmlData == ""] <- NA
pmlData[pmlData == "#DIV/0!"] <- NA
```

In the merged dataset the total amount of missing values is calculated for each feature.

```
# features with missing values
featMissing <- sapply(pmlData, function(x) sum(is.na(x)))
table(featMissing[featMissing != 0])
```

```
##
## 19236 19237 19238 19240 19241 19245 19246 19247 19268 19313 19314 19316
##    67     1     1     1     4     1     4     2     2     1     1     2
## 19319 19320 19321 19642
##     1     4     2     6
```

For 100/159 (excluding `train`) features there are missing values, varying between 19236 and 19642 observations or 98 to 100 percent.

Imputing all these missing values wouldn't lead to good results which is why all features with missing values are removed from the dataset.

```
# only keep features without missing values
pmlData <- pmlData[, featMissing == 0]
```

### Irrelevant features

Some of the features are irrelevant for this analysis such as the time-related features `raw_timestamp_part_1`, `raw_timestamp_part_2`, `cvtd_timestamp`, `new_window` and `num_window`. Other irrelevant features are the row number `X` and user name `user_name`. These features are removed from the dataset.

```
pmlData$raw_timestamp_part_1 <- NULL
pmlData$raw_timestamp_part_2 <- NULL
pmlData$cvtd_timestamp <- NULL
pmlData$new_window <- NULL
pmlData$num_window <- NULL
pmlData$X <- NULL
pmlData$user_name <- NULL
```

### Cleaned datasets

The merged dataset is split again in a cleaned training and test set.

```
pmlTrain2 <- subset(pmlData, train == 1)
pmlTrain2$classe <- pmlTrain$classe
pmlTrain2$train <- NULL
pmlTest2 <- subset(pmlData, train == 0)
pmlTest2$train <- NULL
pmlTest2$problem_id <- pmlTest$problem_id
```

The cleaned datasets contain 52 (quantitative) features.

## Exploratory data analysis

To explore the training data `pmlTrain2` descriptive statistics (5-number summaries) and exploratory plots (boxplots and histograms) were generated for each of the 52 features.

Relationships between the (quantitative) features and the (categorical) outcome `classe` were investigated by scatter plots (feature vs. feature) and side-by-side boxplots (feature vs. `classe`).

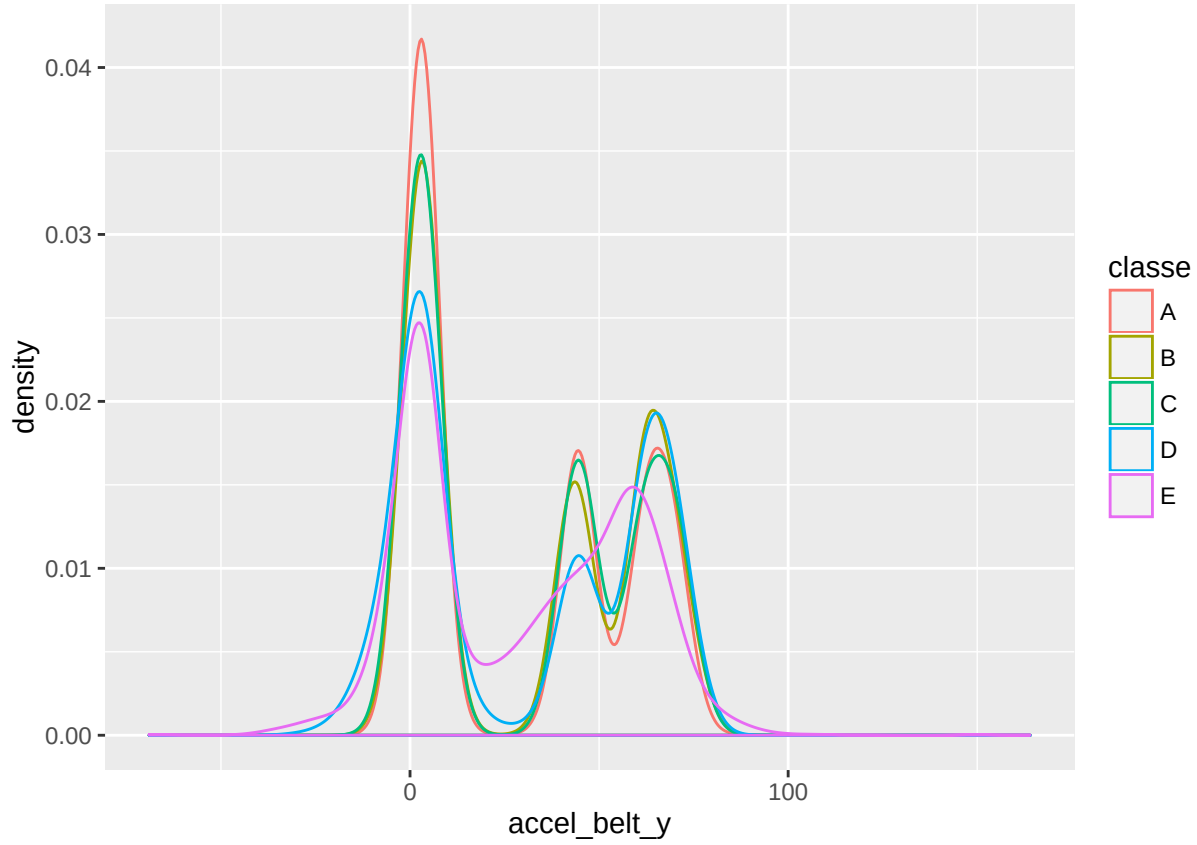
This section describes the three main findings of the EDA.

### Normality of the features

Many of the **features don't follow a normal distribution** or can easily be normalized. The distributions in general are complex and multi-modal which could be related to specific circumstances related to the experiments and/or the participants. There however is not enough information to explore this further.

The figure below shows an example for the `accel_belt_y` feature.

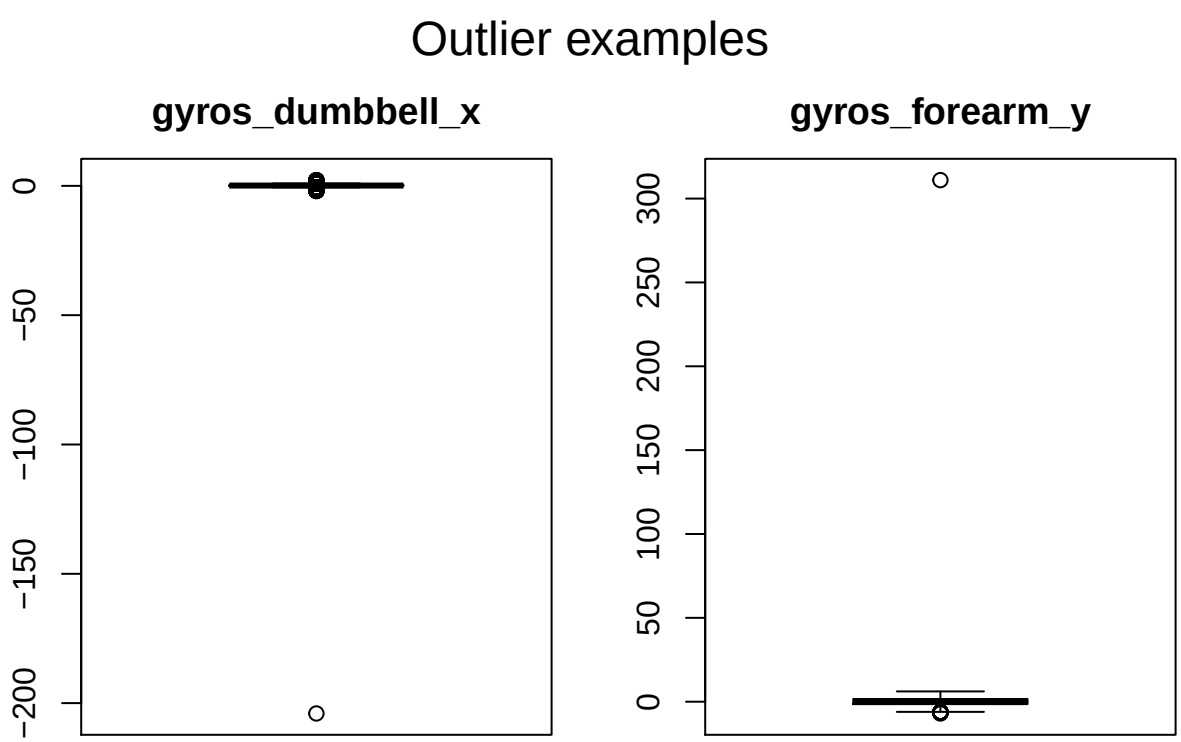
```
library(ggplot2)
ggplot(pmlTrain2, aes(x = accel_belt_y, color = classe)) + geom_density()
```




## Outliers

The training set contains a clear **outlier on the 5373th row** that should be removed. The figure below shows the outlier values for the features `gyros_dumbbell_x` and `gyros_forearm_y`.

```
par(mfrow=c(1,2), oma = c(0, 0, 2, 0), mar = c(3, 3, 2.5, 1))
boxplot(pmlTrain2$gyros_dumbbell_x, main = "gyros_dumbbell_x")
boxplot(pmlTrain2$gyros_forearm_y, main = "gyros_forearm_y")
mtext("Outlier examples", outer = TRUE, cex = 1.5)
```




The outlier has extreme values for 8 different features.

```
# show outlier values compared to mean & median values
colSelect <- c("total_accel_dumbbell", "gyros_dumbbell_x",
               "gyros_dumbbell_y", "gyros_dumbbell_z",
               "total_accel_forearm", "gyros_forearm_x",
               "gyros_forearm_y", "gyros_forearm_z")
xOutlier <- pmlTrain2[5373, colSelect]
xAvg <- round(sapply(pmlTrain2[, colSelect], mean), 2)
xMedian <- round(sapply(pmlTrain2[, colSelect], median), 2)
xComp <- t(rbind(xOutlier, xAvg, xMedian))
colnames(xComp) <- c("outlier", "average", "median")
xComp
```

```
##                      outlier average median
## total_accel_dumbbell      58   13.72  10.00
## gyros_dumbbell_x        -204    0.16   0.13
## gyros_dumbbell_y          52    0.05   0.03
## gyros_dumbbell_z         317   -0.13  -0.13
## total_accel_forearm      108   34.72  36.00
## gyros_forearm_x          -22    0.16   0.05
## gyros_forearm_y          311    0.08   0.03
## gyros_forearm_z          231    0.15   0.08
```

### Variability of the features

There are **no features with low variability** that should be removed from the training (and test) set.

```
# show features with near-zero variability
suppressPackageStartupMessages(library(caret))
```

```
## Warning: package 'caret' was built under R version 3.2.5
```

```
dfNZV <- nearZeroVar(pmlTrain2, saveMetrics = TRUE)
subset(dfNZV, nzv == TRUE)
```

```
## [1] freqRatio     percentUnique zeroVar       nzv
## <0 rows> (or 0-length row.names)
```

## Preprocessing

The main conclusion of the EDA was that many of the features don't follow a normal distribution. Therefore only machine learning **algorithms which do not assume normality** will be considered, more specifically the tree-based algorithms: classification trees, random forests and gradient boosting.

Because these methods don't require centering, scaling and/or normalization of the features the preprocessing is limited to the **outlier removal** and **data slicing** of the training set.

### Outlier removal

The outlier on the 5373th row, identified during the EDA, is removed from the training set.

```
pmlTrain2 <- pmlTrain2[-5373, ]
```

### Data slicing

The training set `pmlTrain2` is split into a smaller training set `training` (70% for training the classifier) and a validation set `validating` (30% for testing the model on new data).

```
set.seed(483)
inTrain <- createDataPartition(y = pmlTrain2$classe, p = 0.7, list = FALSE)
training <- pmlTrain2[inTrain, ]
validating <- pmlTrain2[-inTrain, ]
```

# Model selection

As discussed earlier three different tree-based models will be trained and evaluated: **classification trees** (CART), **random forests** (RF) and **gradient boosting** (GBM). At the end of the section a note is included about combining the models.

For evaluating the performance of a model two functions will be used:

```
# training set performance (accuracy)
trainingPerformance <- function(model) {
    pred <- predict(model)
    acc <- confusionMatrix(pred, training$classe)$overall[1]
    100*acc
}
# test set performance (accuracy)
validatingPerformance <- function(model, x) {
    pred <- predict(model, newdata = x)
    acc <- confusionMatrix(pred, validating$classe)$overall[1]
    100*acc
}
```

## Classification tree

Classification trees have 1 parameter `cp` (the complexity of the tree):

```
getModelInfo()$rpart$parameters
```

```
##   parameter   class                label
## 1        cp numeric Complexity Parameter
```

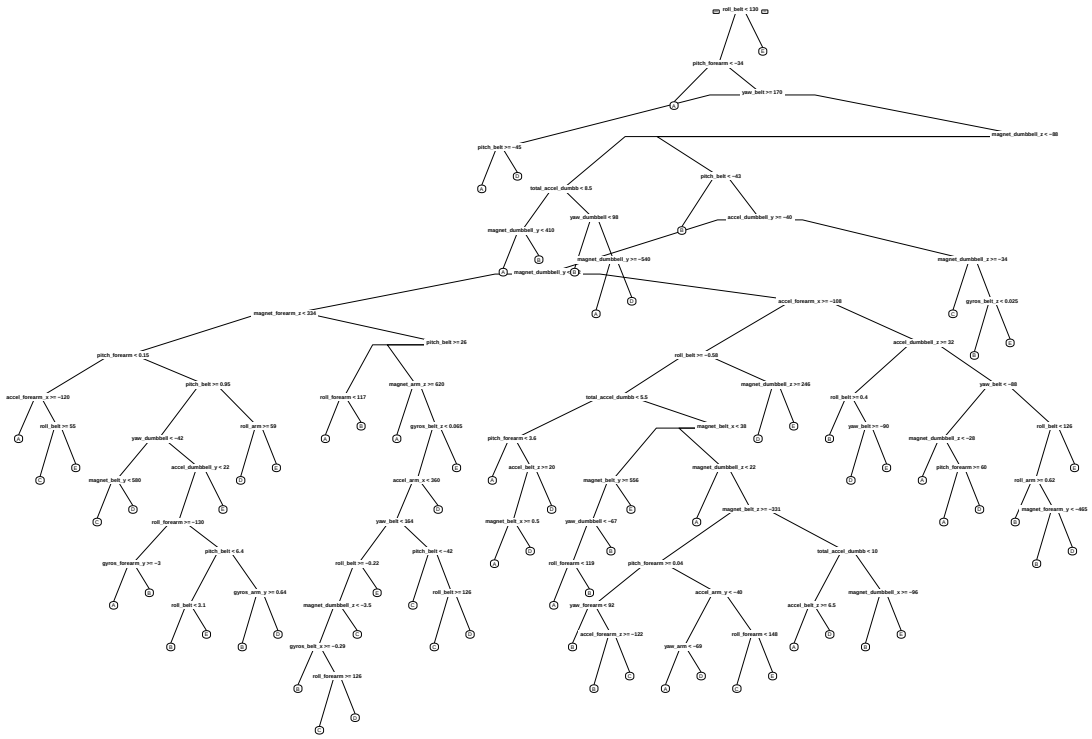
The default `cp` value is 0.01 (see `rpart` documentation) but can be tuned using (for example) a 10-fold cross-validation on the training set `training`. A smaller `cp` value in general leads to more accurate but much less interpretable (more complex) models.

```
set.seed(264)
fitControl <- trainControl(method = "cv", number = 10)
cartGrid <- expand.grid(.cp = seq(0.002, 0.1, 0.002))
trainCART <- train(classe ~ ., data = training,
                   method = "rpart",
                   trControl = fitControl, tuneGrid = cartGrid)
bestCP <- trainCART$bestTune
trainRes <- trainCART$results
trainRes[trainRes$cp %in% c(0.01, bestCP), c("cp", "Accuracy")]
```

```
##      cp  Accuracy
## 1 0.002 0.8659835
## 5 0.010 0.7345160
```

The best classification tree has a `cp` value of 0.002 which is much smaller than the default value. There is a large increase in accuracy compared to the result with the default `cp` value. The tree plot below shows that the resulting tree however is very hard to interpret (as expected).

```
library(rpart.plot)
prp(trainCART$finalModel)
```



```
accTrainCART <- trainingPerformance(trainCART)
accValCART <- validatingPerformance(trainCART, validating)
c(accTrainCART, accValCART)
```

```
## Accuracy Accuracy
## 87.59554 85.36710
```

The final classification tree has a training set accuracy of 87.6% and a validation set accuracy of 85.4%. This could be improved by lowering the `cp` value but will lead to even more uninterpretable models. Other techniques will probably be more useful.

## Random forest

For random forests the parameter `mtry` (the number of variables randomly sampled as candidates at each split) can be tuned, similar to the `cp` parameter for classification trees.

```
getModelInfo()$rf$parameters
```

```
##   parameter   class                         label
## 1      mtry numeric #Randomly Selected Predictors
```

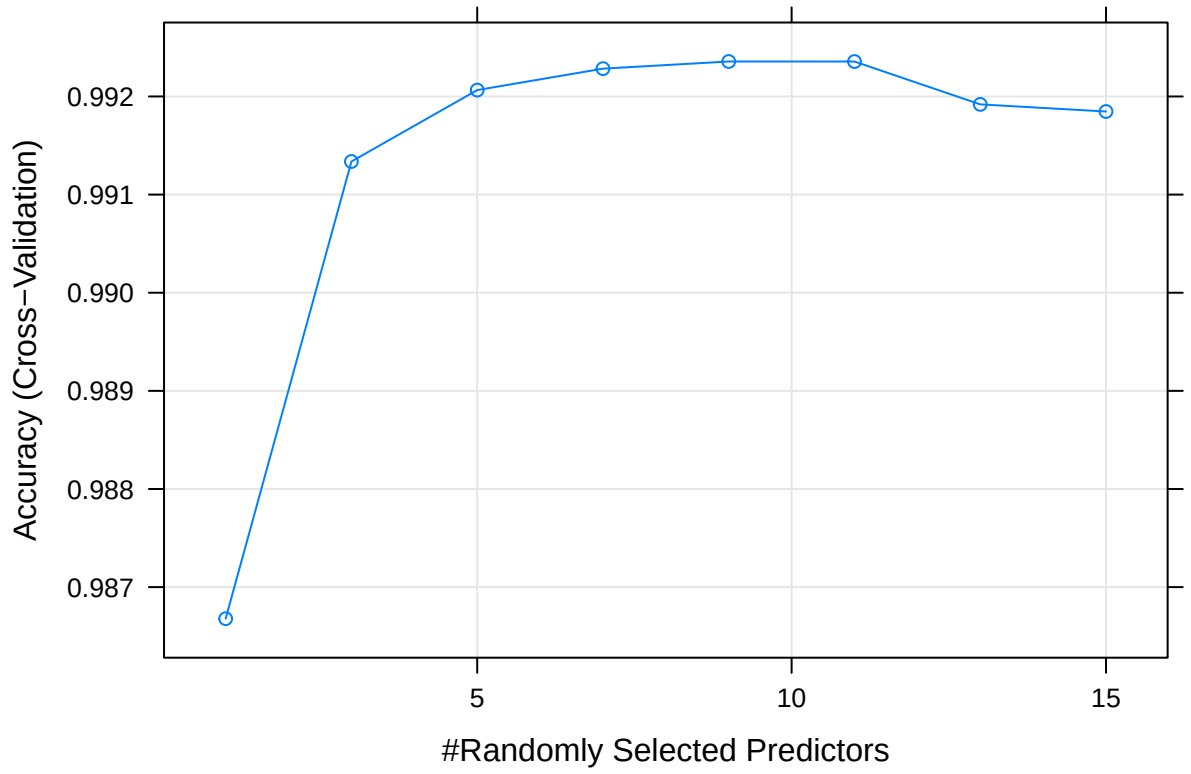
The default `mtry` value is equal to `floor(sqrt(ncol(training)))` = 7 (see `randomForest` documentation). A range of values between 1 and 15 is tried out. To decrease the total computation time the number of folds in the cross-validation is decreased from 10 to 5.

```
set.seed(264)
fitControl <- trainControl(method = "cv", number = 5)
rfGrid = expand.grid(.mtry = seq(1, 15, 2))
trainRF <- train(classe ~ ., data = training, method = "rf",
                 trControl = fitControl, tuneGrid = rfGrid)
trainRF$bestTune
```

```
##   mtry
## 5    9
```

The best random forest model has a `mtry` value of 9. This is also shown in the accuracy plot below.

```
plot(trainRF)
```




The tuned random forest model has a training set accuracy of 100% and a validation set accuracy of 99.5%. This is quite a big improvement over the classification tree model.

```
accTrainRF <- trainingPerformance(trainRF)
accValRF <- validatingPerformance(trainRF, validating)
c(accTrainRF, accValRF)
```

```
##  Accuracy  Accuracy
## 100.00000  99.54113
```

## Gradient boosting

The gradient boosting algorithm has 4 tuning parameters.

```
getModelInfo()$gbm$parameters
```

```
##           parameter   class                     label
## 1           n.trees numeric   # Boosting Iterations
## 2 interaction.depth numeric          Max Tree Depth
## 3         shrinkage numeric               Shrinkage
## 4    n.minobsinnode numeric Min. Terminal Node Size
```

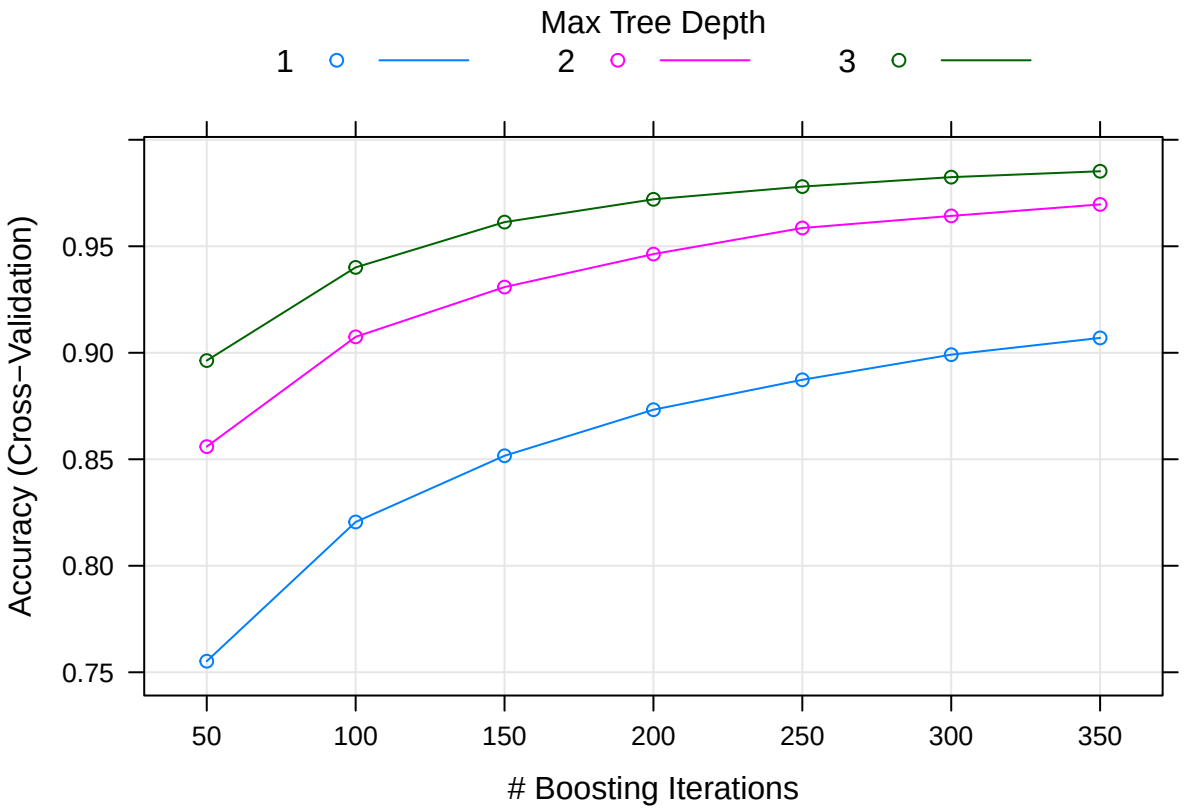
The default parameters are `interaction.depth` = 1, `n.trees` = 100, `shrinkage` = 0.1 and `n.minobsinnode` = 10. In this analysis `shrinkage` and `n.minobsinnode` are kept fixed while a (limited) set of values of `interaction.depth` and `n.trees` are used for tuning, again using a 5-fold cross-validation.

```
set.seed(264)
fitControl <- trainControl(method = "cv", number = 5)
gbmGrid <-  expand.grid(.interaction.depth = c(1:3),
                        .n.trees = (1:7)*50,
                        .shrinkage = 0.1,
                        .n.minobsinnode = 10)
trainGBM <- train(classe ~ ., method = "gbm", data = training,
                  trControl = fitControl, tuneGrid = gbmGrid, verbose = FALSE)
trainGBM$bestTune
```

```
##    n.trees interaction.depth shrinkage n.minobsinnode
## 21     350                 3       0.1             10
```

The best gradient boosting model has an interaction depth of 3 and 350 trees. The plot below shows the training accuracy for the different combinations of parameters. Larger values for both parameters will improve the model slightly but greatly increase the total computation time.

```
plot(trainGBM)
```




The final model has a training set accuracy of 99.7% and a validation set accuracy of 98.8%, which is a bit worse than the random forest model.

```
accTrainGBM <- trainingPerformance(trainGBM)
accValGBM <- validatingPerformance(trainGBM, validating)
c(accTrainGBM, accValGBM)
```

```
## Accuracy Accuracy
## 99.73793 98.79334
```

### Combined model?

Because of the high accuracy of the random forest model, a blended model (e.g. a simple weighted majority vote ensemble classifier) is unlikely to improve the results significantly. Therefore it will not be investigated further in this analysis.

## Final model and prediction

The table below shows an overview of the training and validation set accuracies for the three different trained classifiers.

| method | training accuracy (%) | validation accuracy (%) | estimated out-of-sample error (%) |
|---|---|---|---|
| `rpart` | 87.60 | 85.37 | 14.63 |
| `rf` | 100.00 | 99.54 | 0.46 |
| `gbm` | 99.74 | 98.79 | 1.21 |

The random forest model has the best accuracy and estimated out-of-sample error and is chosen as final model to predict the 20 different test set cases. The obtained accuracy is 100% after submitting the results to the Course Project Prediction Quiz.

```
predict(trainRF, newdata = pmlTest2)
```

```
##  [1] B A B A A E D B A A B C B A E E A B B B
## Levels: A B C D E
```