# Practical Machine Learning - Course Project

Francesco Aldo Tucci

4/25/2021

## Table of Contents[1]

## 1. Overview

The data for this assignment comes from the Human Activity Recognition project[2]. It offers a huge array of measurements[3] related to physical activity and movements performed by 6 subjects as registered by devices such as *Jawbone Up*, *Nike FuelBand*, and *Fitbit*, classifying the resulting *quality* of the exercise (how well it was performed compared to theoretical execution) into 5 distinct categories[4], recorded by the levels of the outcome factor variable *classe*.

Using the data provided, I build a model that allows reliable prediction of the "classe" outcome, explaining model choice based on measured accuracy of the trained model, using cross-validation to select the model parameters tuning and an independent hold-out data set for final validation before choice. I compute the out-of-sample error rate and finally apply the selected model to the 20 observations in the "pml-testing" data set to answer the assignment quiz by predicting the classe outcome.

## 2. Data Processing

Data are correctly downloaded, read & loaded into RStudio. I get rid of metadata (columns 1 to 7 included) since they are uninformative[5]. Missing values pester a relatively large subset of the recorded variables; since there are a huge number of variables with lots of NAs (more than 90%), while the other variables have no missing values at all, I employ the simple[6] approach of dropping variables with more than 50% of missing values[7].

[1]Please note that blank space and division in sections were used to make the document more readable; overall, the page lengths requirements are (more or less) met!

[2]Velloso, E., Bulling, A., Gellersen, H., Ugulino, W. & Fuks, H. (2013), *Qualitative Activity Recognition of Weight Lifting Exercises*, Proceedings of 4th Augmented Human (AH) International Conference in cooperation with ACM SIGCHI (Augmented Human'13), Stuttgart, Germany: ACM SIGCHI 2013. See http://groupware.les.inf.puc-rio.br/work.jsf?p1=11201#ixzz4T79Uh6w7 for further details.

[3]A grand total of 153 different variables, as recorded by the sensors, excluding 7 variables which only record metadata.

[4]For clarity and to be pedantic, factor level A corresponds to doing the exercise exactly as intended; B is for throwing the elbows to the front; C records *lifting* the dumbbell only halfway; D records *lowering* the dumbbell halfway; and finally E is for the case "throwing the hips to the front".

[5]Well, they might be informative, because e.g. subject X might be on average better at performing the exercises than subject Y, and we have the name of the subject available, but we want our prediction to rely only on the measurements recorded by the sensors in the devices.

[6]Brutish, yet perfectly justifiable in this type of situation, when imputing missing values is either impossible or nonsensical.

[7]To clarify further, the variables thus selected are the same for all thresholds from 50% (sometimes used by practitioners) to 90% and beyond (I believe it's mandatory at that point!).

Further inspection of the data shows that there are some highly correlated variables. While some methods are (more) robust to potential multicollinearity, I use two different approaches, which are directly incorporated into the analysis and thus are discussed below.

## 3. Analysis

For all models, 3-fold Cross-Validation is used. Instead of a full tuning grid, a tuneLength input (set equal to 5) is specified for number of combinations for parameters selection. Since the steps in the analysis (see code) try to follow as close as possible those outlined in the semi-automated process proposed by Kuhn (2008)[8], the 'caret' package is used extensively, especially for training the models through the 'train' function.

The first model trained is a decision tree, fitted by applying the 'rpart' method of the 'train' function. Its performance is limited by its inherent simplicity to a very modest 0.5524 accuracy. A second, natural candidate is then random forest.

As summarized in a nice yet disparaging table by Mentor Leonard Greski[9] in the Discussion section of the Course, to get all of the 20 predictions in the hold-out "quiz" data set right with at least 90% probability, one needs at least 0.995 accuracy, even accounting for Šidák correction for multiple tests[10].

Since the outcome variable is categorical, I use the proportion of classes predicted incorrectly (one minus accuracy of the model in predicting the test data) as the out-of-sample[11] error measure for model choice.

Model 2, Random forest, is thus already a winner, with 0.9963 accuracy (.9943, .9977 95% Confidence Interval), corresponding to a meager 0.37% classes predicted incorrectly[12].

As an additional check to avoid overfitting and/or pernicious multicollinearity, model 3 consists of training a random forest model on the first 30 principal components (which together explain more than 97.5% of the variance in the features) of the PCA[13]. The accuracy on the test set is slightly lower[14] (0.9798).

Thus, model 2 is the choice for answering the quiz. Results in terms of the *roaring twenties*, ehm, predictions, are reported after the code for the model. Confusion matrices are reported in the section named as such.

[8]Kuhn, M. (2008), *Building predictive models in R using the caret package*, Journal of Statistical Software, 28(5), 1-26.

[9]Aka Len Igreski. See https://github.com/lgreski/datasciencectacontent/blob/master/markdown/pml-requiredModelAccuracy.md for reference.

[10]See http://bit.ly/2DuPwlq (reference cited in the above).

[11]As seen during the course, out-of-sample error, or "generalization" error, is more appropriate than in-sample error measures, due to overfitting.

[12]Again, since *repetita iuvant*, this refers to the test data, *not* the data used to train aka estimate the model!

[13]PCA is *Principal Component Analysis*, ça va sans dire.

[14]As expected I would say, since by using PCA one "collapses" the information by using only part of it (a part big enough to explain most of the features' variance, as said, but still).

## 4. R Code

### 4.1 Miscellanea

```r
library(readr)
library(dplyr)
library(caret)
library(corrplot)
library(rattle)
library(RColorBrewer)
library(randomForest)
library(doMC)
```

### Load & Read Datasets

```r
# Data downloaded in the same folder as the .Rproj this .Rmd is in
# urlTrain<-"https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
# urlTest<-"https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"
pml_training <- read.csv("pml-training.csv", na.strings = c('#DIV/0', '', 'NA'))
pml_training <- as.data.frame(pml_training) # 19,622 x 160 data frame
pml_testing  <- read.csv("pml-testing.csv", na.strings = c('#DIV/0', '', 'NA'))
pml_testing  <- as.data.frame(pml_testing)  # 20 x 160 data frame
```

### "Quiz" data (for final prediction) & Training data

```r
quizData <- pml_testing[, -c(1:7)]
quizData <- quizData[, colMeans(is.na(quizData)) < 0.5] # 20 x 53
training <- pml_training[, -c(1:7)]
training <- training[, colMeans(is.na(training)) < 0.5] # 19,622 x 53
```

### Correlation matrix & Corrplot

```r
corr.mat  <- cor(training[, -53]) # See plot as Fig. 1
```

### Split training data btw train (CV will also be applied on them) & test data

```r
set.seed(42)
inTrain <- createDataPartition(training.nohc$classe, p = 0.7, list = FALSE)
trainData <- training.nohc[ inTrain, ] # 13,737 x 46
testData  <- training.nohc[-inTrain, ] # 5885 x 46
```

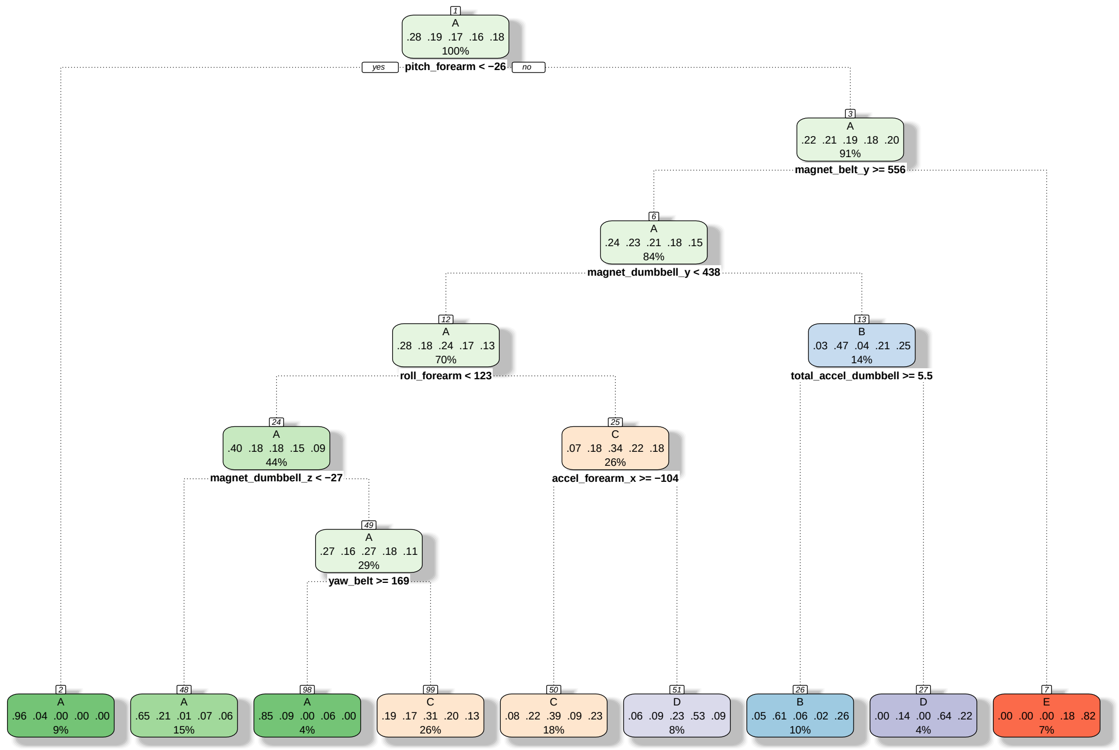
### Setup for 3-fold Cross-Validation

```
CV.k3 <- trainControl(method = "cv", number = 3, verboseIter = FALSE)
```

### 4.2 Analysis Part 1

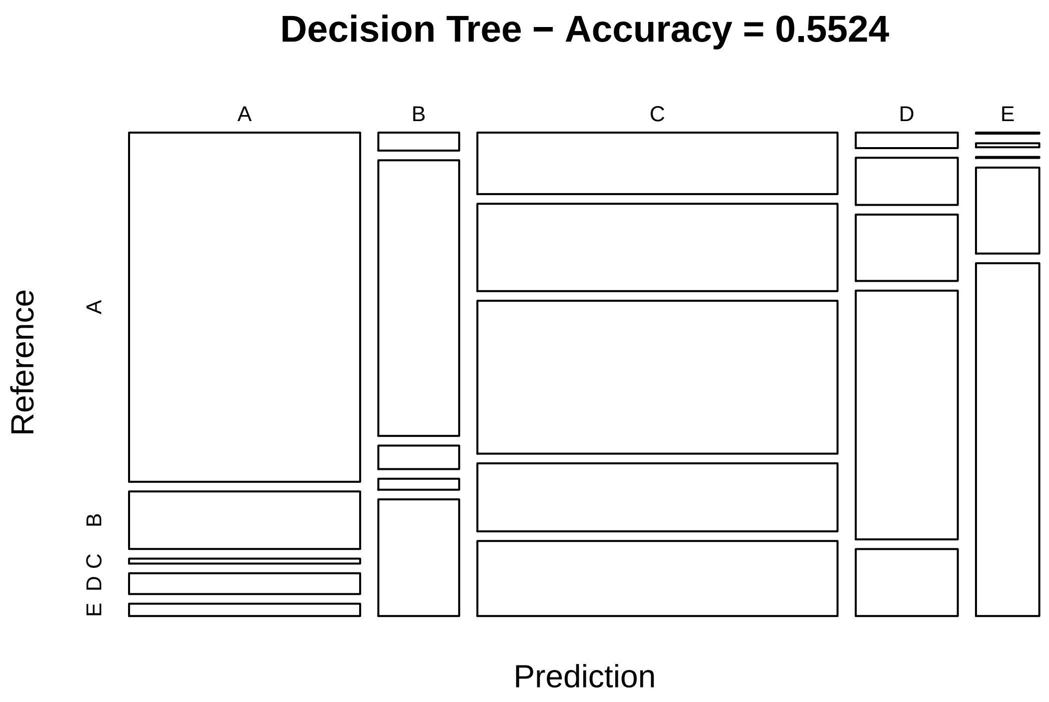
#### Model 1 - Decision Tree for Classification

```
set.seed(42)
treeFit.k3 <- train(classe ~ ., data = trainData, method = "rpart",
                    trControl = CV.k3, tuneLength = 5)
pred.tr.k3 <- predict(treeFit.k3, newdata = testData)
p1.tr.k3   <- fancyRpartPlot(model = treeFit.k3$finalModel,
                             sub = "Decision Tree plot")
```



Decision Tree plot

```
cm.tr.k3   <- confusionMatrix(pred.tr.k3, factor(testData$classe))
p2.tr.k3   <- plot(cm.tr.k3$table, col = cm.tr.k3$byClass,
                main = paste("Decision Tree - Accuracy =",
                             round(cm.tr.k3$overall["Accuracy"], 4)))
```



```
# Low Accuracy: .55 ; NoInfoRate: .28 ; Kappa: .44
```

### Model 2 - Random Forest

```
registerDoMC(cores = 4)
set.seed(42)
rf.Fit <- train(classe ~ ., data = trainData, method = "rf", trControl = CV.k3,
                tuneLength = 5)
pred.rf.Fit <- predict(rf.Fit, newdata = testData)
cm.rf.Fit   <- confusionMatrix(pred.rf.Fit, factor(testData$classe))
# Accuracy .996 ; Kappa .995
# Good enough (maybe too much? overfitting?)
# Will do rf on PCA-processed data as well
```

### Predictions on hold-out data (answers to the quiz)

```
answers.rf <- predict(rf.Fit, quizData.nohc)
answers.rf
```

```
##  [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
```

### 4.3 Analysis - Part 2

#### Using PCA to reduce redundant/collinear information

```
# Start back from 'training' data
trainingPCA.base  <- prcomp(training[, -53], scale = TRUE)
std.PCA <- trainingPCA.base$sdev
var.PCA <- std.PCA^2
propVar.PCA <- var.PCA / sum(var.PCA)
trainingPCA.caret <- preProcess(training[, -53], method = "pca", thresh = 0.9,
                             verbose = FALSE)
```

#### Split PCA data for training/testing

```
training.PCA <- data.frame(classe = training$classe, trainingPCA.base$x)
# From the cumulative variance explained plot we see that the first 30
# Principal Components explain more than 97.5% of the variance in the data
# (way less than 46 or 53 variables!): sum(propVar.PCA[1:30])
training.PCA <- training.PCA[, 1:30] # Select first 30 Principal Components only
set.seed(42)
inTrainPCA <- createDataPartition(training.PCA$classe, p = 0.7, list = FALSE)
trainData.PCA <- training.PCA[ inTrainPCA, ] # 13,737 x 30
testData.PCA  <- training.PCA[-inTrainPCA, ] # 5,885 x 30
```

#### Model 3 - Random Forest on PCA data

```
registerDoMC(cores = 4)
set.seed(42)
rf.PCA <- train(classe ~ ., data = trainData.PCA, method = "rf",
                trControl = CV.k3, tuneLength = 5)
pred.rf.PCA <- predict(rf.PCA, newdata = testData.PCA)
cm.rf.PCA   <- confusionMatrix(pred.rf.PCA, factor(testData.PCA$classe))
```

## 4.4 Confusion Matrices

### Confusion matrix Model 1

```
cm.tr.k3
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1275  210   18   76   45
##          B   23  352   30   14  149
##          C  350  497  870  387  427
##          D   25   76  107  401  108
##          E    1    4    1   86  353
##
## Overall Statistics
##
##                Accuracy : 0.5524
##                  95% CI : (0.5396, 0.5652)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4363
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.7616  0.30904   0.8480  0.41598  0.32625
## Specificity            0.9171  0.95449   0.6582  0.93579  0.98085
## Pos Pred Value         0.7851  0.61972   0.3437  0.55927  0.79326
## Neg Pred Value         0.9064  0.85198   0.9535  0.89106  0.86599
## Prevalence             0.2845  0.19354   0.1743  0.16381  0.18386
## Detection Rate         0.2167  0.05981   0.1478  0.06814  0.05998
## Detection Prevalence   0.2760  0.09652   0.4301  0.12184  0.07562
## Balanced Accuracy      0.8394  0.63177   0.7531  0.67588  0.65355
```

**Confusion matrix Model 2**

```
cm.rf.Fit
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1671    0    0    0    0
##          B    2 1137    3    0    0
##          C    1    2 1023    9    0
##          D    0    0    0  954    4
##          E    0    0    0    1 1078
##
## Overall Statistics
##
##                Accuracy : 0.9963
##                  95% CI : (0.9943, 0.9977)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9953
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9982   0.9982   0.9971   0.9896   0.9963
## Specificity            1.0000   0.9989   0.9975   0.9992   0.9998
## Pos Pred Value         1.0000   0.9956   0.9884   0.9958   0.9991
## Neg Pred Value         0.9993   0.9996   0.9994   0.9980   0.9992
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2839   0.1932   0.1738   0.1621   0.1832
## Detection Prevalence   0.2839   0.1941   0.1759   0.1628   0.1833
## Balanced Accuracy      0.9991   0.9986   0.9973   0.9944   0.9980
```

### Confusion matrix Model 3

```
cm.rf.PCA
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1667   18    2    2    2
##          B    3 1102   10    0    3
##          C    4   12 1010   35    6
##          D    0    0    3  924    8
##          E    0    7    1    3 1063
##
## Overall Statistics
##
##                Accuracy : 0.9798
##                  95% CI : (0.9759, 0.9832)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9744
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9958   0.9675   0.9844   0.9585   0.9824
## Specificity            0.9943   0.9966   0.9883   0.9978   0.9977
## Pos Pred Value         0.9858   0.9857   0.9466   0.9882   0.9898
## Neg Pred Value         0.9983   0.9922   0.9967   0.9919   0.9961
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2833   0.1873   0.1716   0.1570   0.1806
## Detection Prevalence   0.2873   0.1900   0.1813   0.1589   0.1825
## Balanced Accuracy      0.9951   0.9821   0.9863   0.9781   0.9901
```

## 5. Figures

### Figure 1 - Correlation Plot

```
corr.plot <- corrplot(corr.mat, order = "FPC", method = "color",  type = "upper",
                      col = brewer.pal(n = 11, name = "RdYlGn"), tl.cex = 0.8,
                      tl.col = "black") # variables ordered by FPC (PCA)
```

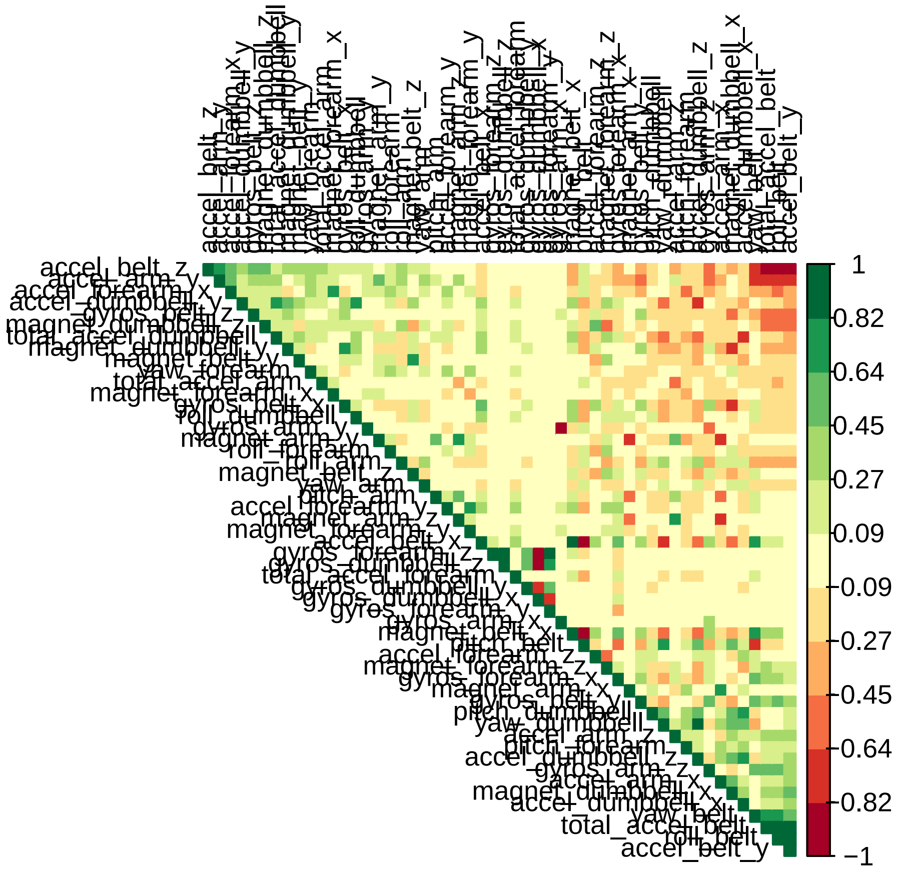


**Figure 2 - Cumulative Variance Explained by First Components**

```r
# Again, trying to be colorblindness-friendly, we select these colors
# (someone should write a proper manual & package for this!)
plot.PCA <- plot(cumsum(propVar.PCA),
                 xlab = "Principal Component",
                 ylab = "Cumulative Proportion of Variance Explained",
                 pch = 19, col = "#56B4E9" , type = "b")
plot.PCA
```

```
## NULL
```

```r
abline(h = 0.975, v = 30, col = "#E69F00")
```

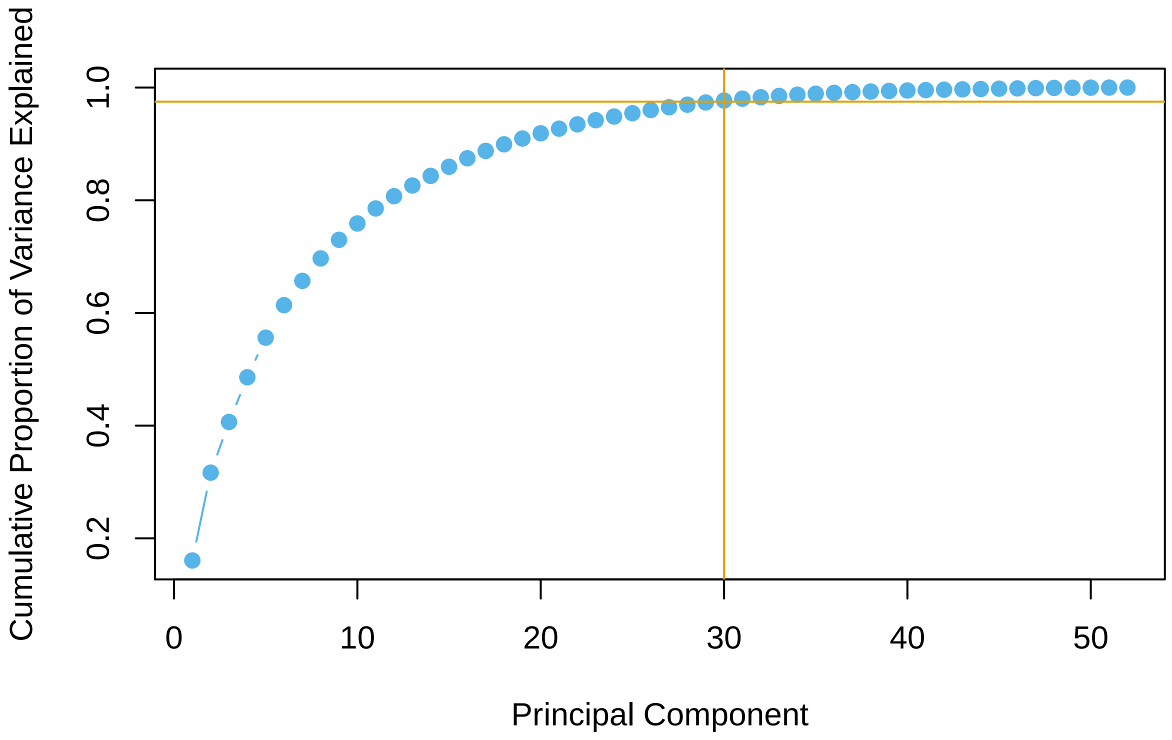


**Figure 3 - Accuracy of Model 1 (Decision Tree) as function of Complexity**

```
plot(treeFit.k3)
```

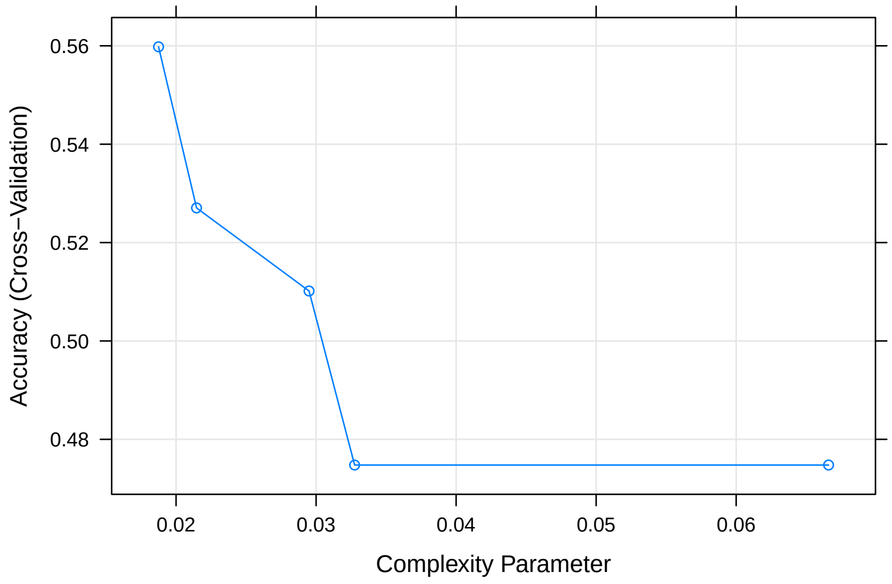


**Figure 4 - Accuracy of Model 2 (Random Forest) as function of n° of features**

```
plot(rf.Fit)
```

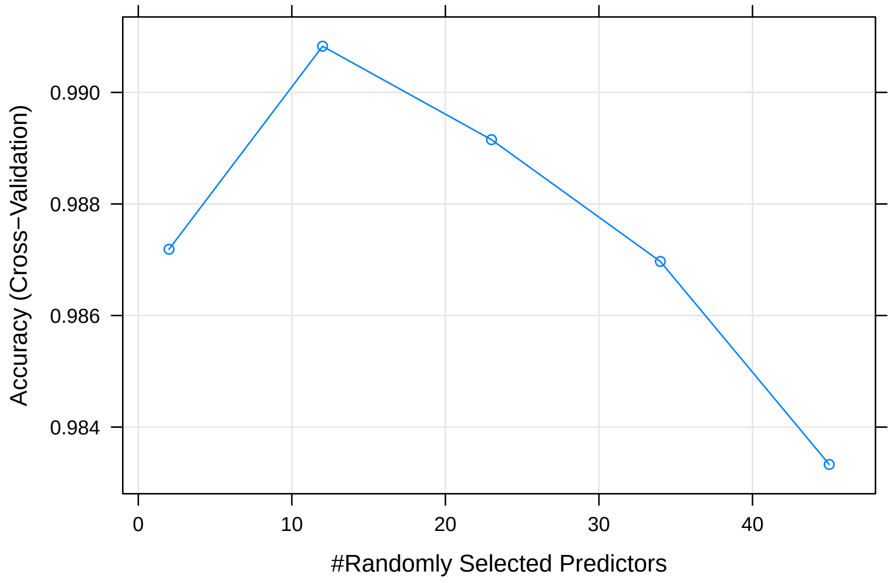
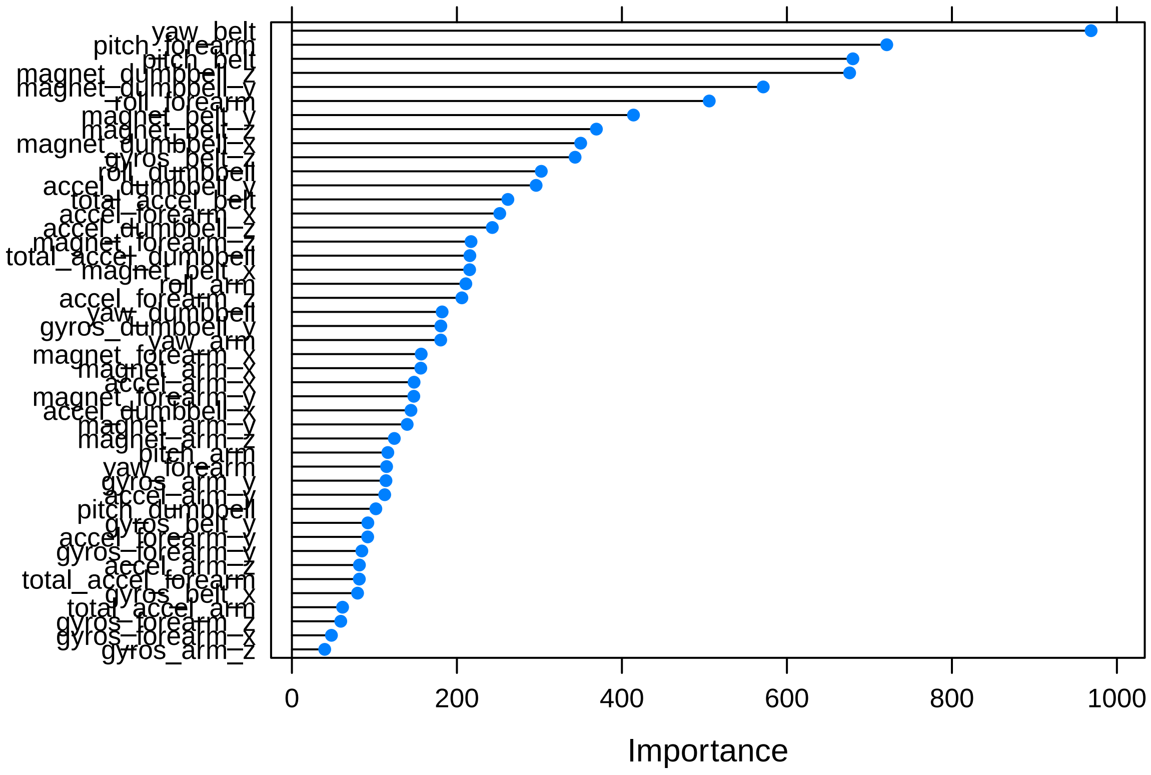


**Figure 5 - Variable Importance in the trained Random Forest model**

```
imp.rf <- varImp(rf.Fit, scale = FALSE)
plot(imp.rf)
```


yaw_belt
pitch_forearm
pitch_belt
magnet_dumbbell_z
magnet_dumbbell_y
roll_forearm
magnet_belt_y
magnet_belt_z
magnet_dumbbell_x
gyros_belt_z
roll_dumbbell
accel_dumbbell_y
total_accel_belt
accel_forearm_x
accel_dumbbell_z
magnet_forearm_z
total_accel_dumbbell
magnet_belt_x
roll_arm
accel_forearm_z
yaw_dumbbell
gyros_dumbbell_y
yaw_arm
magnet_forearm_x
magnet_arm_x
accel_arm_x
magnet_forearm_y
accel_dumbbell_x
magnet_arm_y
magnet_arm_z
pitch_arm
yaw_forearm
gyros_arm_y
accel_arm_y
pitch_dumbbell
gyros_belt_y
accel_forearm_y
gyros_forearm_y
accel_arm_z
total_accel_forearm
gyros_belt_x
total_accel_arm
gyros_forearm_z
gyros_forearm_x
gyros_arm_z
Importance