# A prediction model to predict activity quality from activity monitors.

Author: Rainer-Anton Englisch

## Executive summary

The task of the course project can be summarized by a quote of the course projects summary: "Using devices such as Jawbone Up, Nike FuelBand, and Fitbit it is now possible to collect a large amount of data about personal activity relatively inexpensively. These type of devices are part of the quantified self movement - a group of enthusiasts who take measurements about themselves regularly to improve their health, to find patterns in their behavior, or because they are tech geeks. One thing that people regularly do is quantify how much of a particular activity they do, but they rarely quantify how well they do it. In this project, your goal will be to use data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants. They were asked to perform barbell lifts correctly and incorrectly in 5 different ways. More information is available from the website here: http://groupware.les.inf.puc-rio.br/har (see the section on the Weight Lifting Exercise Dataset)."

Simply put: We want to build a prediction model to predict activity quality from activity monitors.

## Predictor Analysis and Reduction

Before letting caret to create a prediction model we will try to reduce the number of predictors in order to speed up the creation of the prediction model. First we load the training data set and then remove some obvious predictors that could lead to overfitting: Namely the column "X"" and all the timestamp columns.

The activity quality of an observation is classified by the factor variable classe which we store in seperate variables for later use for training and prediction.

```r
library(caret)
set.seed(1312)
pml_training = read.csv("pml-training.csv")
pml_training = pml_training[,-1]
winColIndex <-  grep("window",colnames(pml_training))
pml_training = pml_training[,-winColIndex]
timestampColIndex <-  grep("timestamp",colnames(pml_training))
pml_training = pml_training[,-timestampColIndex]
dim(pml_training)
```

```
## [1] 19622   154
```

### Create training and test/validation data set

Next we split the original training in a new training and a test data. The test data will be used later for estimating the out of sample error.

The activity quality of an observation is classified by the factor variable **classe** which we store in seperate variables for later use for training and prediction.

```r
library(caret)
inTrain <- createDataPartition(pml_training$classe,p=0.8, list=FALSE)
training <- pml_training[inTrain,]
trainclasse <- training$classe
testing <- pml_training[-inTrain,]
testingclasse <- testing$classe
```

### Remove predictors with near zero variability

Next we throw out all predictors that have near zero variability because we assume that these predictors have minimal influence on the prediction.

```r
nZV <- nearZeroVar(training,saveMetrics=TRUE)
# extract the predictor names which have near zero variability
nzvcolnames <- rownames(nZV[nZV$nzv==TRUE,])
# compute the index of these predictor  names in the training data frame
nzvcolindex <- which(names(training) %in% nzvcolnames)
# remove these predictors
training <- training[,-nzvcolindex]
# print these predictors
nzvcolnames
```

```
##  [1] "kurtosis_roll_belt"      "kurtosis_picth_belt"
##  [3] "kurtosis_yaw_belt"       "skewness_roll_belt"
##  [5] "skewness_roll_belt.1"    "skewness_yaw_belt"
##  [7] "max_yaw_belt"            "min_yaw_belt"
##  [9] "amplitude_yaw_belt"      "avg_roll_arm"
## [11] "stddev_roll_arm"         "var_roll_arm"
## [13] "avg_pitch_arm"           "stddev_pitch_arm"
## [15] "var_pitch_arm"           "avg_yaw_arm"
## [17] "stddev_yaw_arm"          "var_yaw_arm"
## [19] "kurtosis_roll_arm"       "kurtosis_picth_arm"
## [21] "kurtosis_yaw_arm"        "skewness_roll_arm"
## [23] "skewness_pitch_arm"      "skewness_yaw_arm"
## [25] "max_roll_arm"            "min_roll_arm"
## [27] "amplitude_roll_arm"      "amplitude_pitch_arm"
## [29] "kurtosis_roll_dumbbell"  "kurtosis_picth_dumbbell"
## [31] "kurtosis_yaw_dumbbell"   "skewness_roll_dumbbell"
## [33] "skewness_pitch_dumbbell" "skewness_yaw_dumbbell"
## [35] "max_yaw_dumbbell"        "min_yaw_dumbbell"
## [37] "amplitude_yaw_dumbbell"  "kurtosis_roll_forearm"
## [39] "kurtosis_picth_forearm"  "kurtosis_yaw_forearm"
## [41] "skewness_roll_forearm"   "skewness_pitch_forearm"
## [43] "skewness_yaw_forearm"    "max_roll_forearm"
## [45] "max_yaw_forearm"         "min_roll_forearm"
## [47] "min_yaw_forearm"         "amplitude_roll_forearm"
## [49] "amplitude_yaw_forearm"   "avg_roll_forearm"
## [51] "stddev_roll_forearm"     "var_roll_forearm"
## [53] "avg_pitch_forearm"       "stddev_pitch_forearm"
## [55] "var_pitch_forearm"       "avg_yaw_forearm"
## [57] "stddev_yaw_forearm"      "var_yaw_forearm"
```

We removed 58 predictors in the training data frame.

### Remove predictors with high linear correlation

Next we want to throw out predictors that have a *high linear correlation*.

```
#As the correlation matix can only be computed for numeric variables we need to identify numeric variab
colsnumeric <- sapply(training, is.numeric)
#compute the correlation matrix
cortraining <- cor(training[,colsnumeric],use="pairwise.complete.obs")
```

Within the computed correlation matrix we select all predictors that have a high correlation. Let us define a high corelation as a value *equal or greater than 0.7*. Thus let's find these predictors and remove them from the training data set.

```
# retrieve variables that have a correlation greater or equal to 0.7
highlyCor <- findCorrelation(cortraining, 0.70,verbose=FALSE)
highlyCorcolnames <- colnames(training)[highlyCor]
# remove the highly correlated predictors from the training data frame
training <- training[,-highlyCor]
# print the removed predictors
highlyCorcolnames
```

```
##  [1] "max_roll_dumbbell"        "total_accel_belt"
##  [3] "var_pitch_belt"           "max_picth_belt"
##  [5] "pitch_belt"               "accel_belt_z"
##  [7] "var_pitch_dumbbell"       "var_roll_belt"
##  [9] "var_total_accel_belt"     "max_roll_belt"
## [11] "gyros_belt_z"             "min_roll_belt"
## [13] "user_name"                "accel_dumbbell_y"
## [15] "accel_belt_y"             "yaw_belt"
## [17] "pitch_dumbbell"           "accel_belt_x"
## [19] "yaw_dumbbell"             "amplitude_pitch_dumbbell"
## [21] "accel_dumbbell_x"         "gyros_dumbbell_z"
## [23] "amplitude_roll_dumbbell"  "min_pitch_dumbbell"
## [25] "avg_yaw_dumbbell"         "var_roll_dumbbell"
## [27] "avg_pitch_dumbbell"       "min_pitch_forearm"
## [29] "avg_roll_dumbbell"        "accel_dumbbell_z"
## [31] "magnet_dumbbell_x"        "gyros_arm_z"
## [33] "magnet_belt_x"            "var_accel_dumbbell"
## [35] "accel_arm_y"              "min_yaw_arm"
## [37] "avg_roll_belt"            "amplitude_roll_belt"
## [39] "amplitude_pitch_belt"     "magnet_arm_x"
## [41] "accel_forearm_x"          "avg_pitch_belt"
## [43] "var_yaw_dumbbell"         "gyros_arm_x"
## [45] "gyros_dumbbell_y"         "min_pitch_belt"
## [47] "avg_yaw_belt"             "gyros_forearm_x"
```

We removed 48 highly correlated predictors.

### Remove predictors which are unimportant for prediction

Now we want to quickly create a small prediction model in order to query the importance of the variables for the prediction model. Let's fit a model, print the important predictors and *keep these important predictors* in the training data set.

```r
modFit <- train(classe ~.,data=training,method="rpart")
```

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

```r
# print summary of the model fit
modFit
```

```
## CART
##
## 15699 samples
##    47 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
##
## Summary of sample sizes: 322, 322, 322, 322, 322, 322, ...
##
## Resampling results across tuning parameters:
##
##   cp          Accuracy   Kappa       Accuracy SD  Kappa SD
##   0.06329114  0.4787436  0.33855083  0.04631047   0.06483518
##   0.09282700  0.4613826  0.31722356  0.04390608   0.06444678
##   0.20675105  0.3225707  0.09253235  0.07027722   0.10073289
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was cp = 0.06329114.
```

```r
# retrieve the variable importance list
importance <- varImp(modFit, scale=FALSE)
importance <- importance[[1]]
# bind the rownames as columns
importance <- cbind(rownames(importance),importance )
# retrieve column names which have importance greater zero
impcolnames <- importance[importance$Overall>0.0,1]
# print important variables
impcolnames
```

```
##  [1] amplitude_yaw_arm     magnet_belt_y         magnet_dumbbell_y
##  [4] pitch_forearm         roll_belt             roll_dumbbell
##  [7] stddev_pitch_belt     stddev_pitch_dumbbell stddev_roll_belt
## [10] stddev_roll_dumbbell  stddev_yaw_dumbbell   var_accel_arm
## [13] var_yaw_belt
## 47 Levels: accel_arm_x accel_arm_z accel_forearm_y ... yaw_forearm
```

```r
# retrieve indexes of column names
impcolindex <- which(names(training) %in% impcolnames)
removedCols <- ncol(training) - length(impcolnames)
# keep important columns in training data frame
training <- training[,impcolindex]
```

In the last step we have removed 35 unimportant predictors.

### Train final prediction model

Now that we have reduced the predictors significantly from 160 to *13 predictors* we will fit a more complex machine learning algorithm based on *random forest.* Additionally we will use *cross validation* to minimize the out of sample error and preprocess the training data to remove NA values by knnImpute.

```
fitControl <- trainControl(method = "repeatedcv",number = 10, repeats = 3)
preObj <- preProcess(training,method=c("knnImpute"))
trainingImputed <- predict(preObj,newdata=training)
modFit <- train(trainclasse ~.,data=trainingImputed,
                method="rf"
               ,trControl = fitControl
               )
```

```
## Loading required package: randomForest
## randomForest 4.6-10
## Type rfNews() to see new features/changes/bug fixes.
```

```
# print summary of model
print(modFit)
```

```
## Random Forest
##
## 15699 samples
##    12 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
##
## Summary of sample sizes: 14128, 14131, 14130, 14131, 14128, 14129, ...
##
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa      Accuracy SD  Kappa SD
##    2    0.8708828  0.8367212  0.009769552  0.01238376
##    7    0.8977852  0.8707482  0.008102536  0.01025303
##   13    0.8980611  0.8711046  0.008348221  0.01056069
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 13.
```

### Compute the out of sample error based on a seperate training (or validation) set

Let's use our separate test (or validation) set to compute an out of sample error.

```
# subselect in the test set the predictors used for the training set
traincolnames <- colnames(training)
traincolindex <- which(names(testing) %in% traincolnames)
testing <- testing[,traincolindex]
# impute NAs like for the training set
preObj <- preProcess(testing,method=c("knnImpute"))
```

```r
testingImputed <- predict(preObj,newdata=testing)
predictions <- predict(modFit,newdata=testingImputed)
```

```
## Loading required package: randomForest
## randomForest 4.6-10
## Type rfNews() to see new features/changes/bug fixes.
```

```r
# summarize results
confusionMatrix(predictions, testingclasse)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   A   B   C   D   E
##          A 957 118  20  57   9
##          B  65 475 105  61  14
##          C  20  88 410  44   5
##          D  29  50 127 473  95
##          E  45  28  22   8 598
##
## Overall Statistics
##
##                Accuracy : 0.7425
##                  95% CI : (0.7286, 0.7562)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6741
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.8575   0.6258   0.5994   0.7356   0.8294
## Specificity            0.9273   0.9226   0.9515   0.9082   0.9678
## Pos Pred Value         0.8243   0.6597   0.7231   0.6111   0.8531
## Neg Pred Value         0.9424   0.9113   0.9184   0.9460   0.9618
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2439   0.1211   0.1045   0.1206   0.1524
## Detection Prevalence   0.2959   0.1835   0.1445   0.1973   0.1787
## Balanced Accuracy      0.8924   0.7742   0.7755   0.8219   0.8986
```

```r
# compute the out of sample error
outOfSampleAccuracy <- sum(predictions==testingclasse)/length(testingclasse)
outOfSampleAccuracy
```

```
## [1] 0.742544
```

```r
outOfSampleError <- 1-outOfSampleAccuracy
outOfSampleError
```

```
## [1] 0.257456
```

## Compare in sample error and out of bag sample error and out of sample error

Finally we want to compare the in sample error and the out of bag sample error and the out of sample error based on the seperate test set. The first two sample errors need to be derived from accuracy variables stored within the results variable within the prediction model.

```
# the out of bag sample accuracy of the prediction model
inSampleError <- 1-max(modFit$results$Accuracy)
inSampleError
```

```
## [1] 0.1019389
```

```
cvoutOfSampleError <- 1-max(modFit$results$Kappa)
cvoutOfSampleError
```

```
## [1] 0.1288954
```

The in *sample error* is **10.19%** whereas the *out of bag sample error* (which is the estimated out of sample error based on training with repeated cross validation) is **12.89%**. Additionally a real *out of sample error* based on a seperate test (or validation) set is **25.75%**. We observe that the real out of sample error is much higher than the out of bag sample error by about **199.74%**.