# Practical Machine Learning Course Project

*D. Ulrich*

# Synopsis

The training and test data are taken from the following study: Velloso, E.; Bulling, A.; Gellersen, H.; Ugulino, W.; Fuks, H. Qualitative Activity Recognition of Weight Lifting Exercises. Proceedings of 4th International Conference in Cooperation with SIGCHI (Augmented Human '13) . Stuttgart, Germany: ACM SIGCHI, 2013.

The goal of this project is to predict the manner in which they did the exercise. The description should adress the following questions:

-how the model was built.

-how cross validation was used.

-what the expected out of sample error is.

-what choices were done.

At the beginning the packages needed to produce the results are loaded.

```
library(caret,quietly=TRUE)
```

```
## Warning: package 'caret' was built under R version 3.2.2
```

```
## Warning: package 'ggplot2' was built under R version 3.2.2
```

```
library(AppliedPredictiveModeling,quietly=TRUE)
```

```
## Warning: package 'AppliedPredictiveModeling' was built under R version
## 3.2.2
```

```
library(rpart.plot,quietly=TRUE)
```

```
## Warning: package 'rpart.plot' was built under R version 3.2.2
```

```
## Warning: package 'rpart' was built under R version 3.2.2
```

```
library(randomForest,quietly=TRUE)
```

```
## Warning: package 'randomForest' was built under R version 3.2.2
```

```
## randomForest 4.6-12
## Type rfNews() to see new features/changes/bug fixes.
```

# Question

In the study, six participants participated in a dumbell lifting exercise five different ways. The five ways, as described in the study, were “exactly according to the specification (Class A), throwing the elbows to the front (Class B), lifting the dumbbell only halfway (Class C), lowering the dumbbell only halfway (Class D) and throwing the hips to the front (Class E). Class A corresponds to the specified execution of the exercise, while the other 4 classes correspond to common mistakes.”

By processing data gathered from accelerometers on the belt, forearm, arm, and dumbell of the participants in a machine learning algorithm, the question is can the appropriate activity quality (class A-E) be predicted?

# Input data

In a next step the data are loaded:

```
file_train <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
file_test <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"
#download.file(url=file_train, destfile=pml-training.csv, method="curl")
#download.file(url=file_test, destfile=pml-testing.csv, method="curl")

train <- read.table('pml-training.csv', na.strings=c("NA",""), header = TRUE, sep =
',')
test <- read.table('pml-testing.csv', na.strings=c("NA",""), header = TRUE, sep = ',')

colnames_train <- colnames(train)
colnames_test <- colnames(test)
```

# Features

Next I decided to eliminate columns with NA and other nonnumerical columns that are useless for prediction.

```
# Count the number of non-NAs in each col.
nonNAs <- function(x) {
  as.vector(apply(x, 2, function(x) length(which(!is.na(x)))))
}
# Build vector of missing data or NA columns to drop.
colcnts <- nonNAs(train)
drops <- c()
for (cnt in 1:length(colcnts)) {
  if (colcnts[cnt] < nrow(train)) {
    drops <- c(drops, colnames_train[cnt])
  }
}
# Drop NA data and the first 7 columns as they're unnecessary for predicting.
train <- train[,!(names(train) %in% drops)]
train <- train[,8:length(colnames(train))]

test <- test[,!(names(test) %in% drops)]
test <- test[,8:length(colnames(test))]
```

In a further step I make sure there are no variables with near zero variabillity.

```
nsv <- nearZeroVar(train, saveMetrics=TRUE)
nsv
```

```
##                      freqRatio percentUnique zeroVar   nzv
## roll_belt             1.101904     6.7781062   FALSE FALSE
## pitch_belt            1.036082     9.3772296   FALSE FALSE
## yaw_belt              1.058480     9.9734991   FALSE FALSE
## total_accel_belt      1.063160     0.1477933   FALSE FALSE
## gyros_belt_x          1.058651     0.7134849   FALSE FALSE
## gyros_belt_y          1.144000     0.3516461   FALSE FALSE
## gyros_belt_z          1.066214     0.8612782   FALSE FALSE
## accel_belt_x          1.055412     0.8357966   FALSE FALSE
## accel_belt_y          1.113725     0.7287738   FALSE FALSE
## accel_belt_z          1.078767     1.5237998   FALSE FALSE
## magnet_belt_x         1.090141     1.6664968   FALSE FALSE
## magnet_belt_y         1.099688     1.5187035   FALSE FALSE
## magnet_belt_z         1.006369     2.3290184   FALSE FALSE
## roll_arm             52.338462    13.5256345   FALSE FALSE
## pitch_arm            87.256410    15.7323412   FALSE FALSE
## yaw_arm              33.029126    14.6570176   FALSE FALSE
## total_accel_arm       1.024526     0.3363572   FALSE FALSE
## gyros_arm_x           1.015504     3.2769341   FALSE FALSE
## gyros_arm_y           1.454369     1.9162165   FALSE FALSE
## gyros_arm_z           1.110687     1.2638875   FALSE FALSE
## accel_arm_x           1.017341     3.9598410   FALSE FALSE
## accel_arm_y           1.140187     2.7367241   FALSE FALSE
## accel_arm_z           1.128000     4.0362858   FALSE FALSE
## magnet_arm_x          1.000000     6.8239731   FALSE FALSE
## magnet_arm_y          1.056818     4.4439914   FALSE FALSE
## magnet_arm_z          1.036364     6.4468454   FALSE FALSE
## roll_dumbbell         1.022388    84.2065029   FALSE FALSE
## pitch_dumbbell        2.277372    81.7449801   FALSE FALSE
## yaw_dumbbell          1.132231    83.4828254   FALSE FALSE
## total_accel_dumbbell  1.072634     0.2191418   FALSE FALSE
## gyros_dumbbell_x      1.003268     1.2282132   FALSE FALSE
## gyros_dumbbell_y      1.264957     1.4167771   FALSE FALSE
## gyros_dumbbell_z      1.060100     1.0498420   FALSE FALSE
## accel_dumbbell_x      1.018018     2.1659362   FALSE FALSE
## accel_dumbbell_y      1.053061     2.3748853   FALSE FALSE
## accel_dumbbell_z      1.133333     2.0894914   FALSE FALSE
## magnet_dumbbell_x     1.098266     5.7486495   FALSE FALSE
## magnet_dumbbell_y     1.197740     4.3012945   FALSE FALSE
## magnet_dumbbell_z     1.020833     3.4451126   FALSE FALSE
## roll_forearm         11.589286    11.0895933   FALSE FALSE
## pitch_forearm        65.983051    14.8557741   FALSE FALSE
## yaw_forearm          15.322835    10.1467740   FALSE FALSE
## total_accel_forearm   1.128928     0.3567424   FALSE FALSE
## gyros_forearm_x       1.059273     1.5187035   FALSE FALSE
## gyros_forearm_y       1.036554     3.7763735   FALSE FALSE
## gyros_forearm_z       1.122917     1.5645704   FALSE FALSE
## accel_forearm_x       1.126437     4.0464784   FALSE FALSE
## accel_forearm_y       1.059406     5.1116094   FALSE FALSE
## accel_forearm_z       1.006250     2.9558659   FALSE FALSE
## magnet_forearm_x      1.012346     7.7667924   FALSE FALSE
## magnet_forearm_y      1.246914     9.5403119   FALSE FALSE
## magnet_forearm_z      1.000000     8.5771073   FALSE FALSE
```

```
## classe                  1.469581     0.0254816   FALSE FALSE
```

We can see that all of the near zero variance variables (nsv) are FALSE, so there is no need to eliminate any covariates due to lack of variablility.

# Algorithm

For the analysis I spilted the data in a training (60%) and a test set (40%).

```
inTrain <- createDataPartition(y=train$classe, p=0.6, list=FALSE)
training <- train[inTrain,]
testing <- train[-inTrain,]
```

Based on the assumption that we have an nonlinar relationship between the variables and on the concensus in the coursera discussion forums, I chose two different algorithms from the caret package: classification trees (method = rpart) and random forests (method = rf).

# Evaluation

I tried the classification tree first. Together with crossvalidation and preprocessing.

```
set.seed(888)
modFit <- train(training$classe ~ .,  preProcess=c("center", "scale"), trControl=train
Control(method = "cv", number = 4), data = training, method="rpart")
print(modFit, digits=3)
```

```
## CART
##
## 11776 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## Pre-processing: centered (52), scaled (52)
## Resampling: Cross-Validated (4 fold)
## Summary of sample sizes: 8832, 8834, 8830, 8832
## Resampling results across tuning parameters:
##
##   cp      Accuracy  Kappa   Accuracy SD  Kappa SD
##   0.0331  0.516     0.3688  0.00562      0.00763
##   0.0605  0.400     0.1803  0.06557      0.10924
##   0.1162  0.325     0.0621  0.04707      0.07172
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was cp = 0.0331.
```

Run the model with the test set.

```
predictions <- predict(modFit, newdata=testing)
print(confusionMatrix(predictions, testing$classe), digits=4)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    A    B    C    D    E
##          A 2032  643  628  588  219
##          B   37  490   39  214  209
##          C  159  385  701  484  372
##          D    0    0    0    0    0
##          E    4    0    0    0  642
## 
## Overall Statistics
##                                           
##                Accuracy : 0.4926          
##                  95% CI : (0.4815, 0.5037)
##     No Information Rate : 0.2845          
##     P-Value [Acc > NIR] : < 2.2e-16       
##                                           
##                   Kappa : 0.3365          
##  Mcnemar's Test P-Value : NA              
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9104  0.32279  0.51243   0.0000  0.44521
## Specificity            0.6299  0.92114  0.78388   1.0000  0.99938
## Pos Pred Value         0.4944  0.49545  0.33365      NaN  0.99381
## Neg Pred Value         0.9465  0.85008  0.88390   0.8361  0.88889
## Prevalence             0.2845  0.19347  0.17436   0.1639  0.18379
## Detection Rate         0.2590  0.06245  0.08934   0.0000  0.08183
## Detection Prevalence   0.5238  0.12605  0.26778   0.0000  0.08233
## Balanced Accuracy      0.7701  0.62197  0.64816   0.5000  0.72230
```

The accuracy rate is with 0.49 to low for a serious prediction. So I tried the random forests also with crossvalidation and preprocessing.

```
set.seed(888)
modFit <- train(training$classe ~ .,  preProcess=c("center", "scale"), trControl=train
Control(method = "cv", number = 4), data = training, method="rf")
print(modFit, digits=3)
```

```
## Random Forest
##
## 11776 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## Pre-processing: centered (52), scaled (52)
## Resampling: Cross-Validated (4 fold)
## Summary of sample sizes: 8832, 8834, 8830, 8832
## Resampling results across tuning parameters:
##
##   mtry  Accuracy  Kappa  Accuracy SD  Kappa SD
##    2    0.990     0.987  0.000275     0.000348
##   27    0.990     0.987  0.001010     0.001279
##   52    0.977     0.971  0.003070     0.003886
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 2.
```

Run the new model with the test set.

```
predictions <- predict(modFit, newdata=testing)
print(confusionMatrix(predictions, testing$classe), digits=4)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    A    B    C    D    E
##          A 2230   23    0    0    0
##          B    1 1493   18    0    0
##          C    1    2 1349   23    0
##          D    0    0    1 1263    3
##          E    0    0    0    0 1439
## 
## Overall Statistics
##                                           
##                Accuracy : 0.9908          
##                  95% CI : (0.9885, 0.9928)
##     No Information Rate : 0.2845          
##     P-Value [Acc > NIR] : < 2.2e-16       
##                                           
##                   Kappa : 0.9884          
##  Mcnemar's Test P-Value : NA              
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9991   0.9835   0.9861   0.9821   0.9979
## Specificity            0.9959   0.9970   0.9960   0.9994   1.0000
## Pos Pred Value         0.9898   0.9874   0.9811   0.9968   1.0000
## Neg Pred Value         0.9996   0.9961   0.9971   0.9965   0.9995
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2842   0.1903   0.1719   0.1610   0.1834
## Detection Prevalence   0.2872   0.1927   0.1752   0.1615   0.1834
## Balanced Accuracy      0.9975   0.9903   0.9910   0.9908   0.9990
```

With this model we get a accuracy rate of 0.9908. So the out of sample error is: 1-0.9908=0.0092.

# Conclusion

With an accuracy rate of 0.9908 the random forrests model with preprocessing an crossvalidation seems accurate for calculating the values of the 20 testing set, which are asked in the second part of the course project.

```
print(predict(modFit, newdata=test))
```

```
##  [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
```