Course Project: Predicting correct barbell lifts based on activity tracker data

Andreas Jahnen

November 18, 2015

Abstract

Activity trackers are an inexpensive method to track the movement of people. The data is used in application fields like fitness, health and track behavoir. In this course project, the activity data of 6 individuals is used to classify barbell lifts into several categories of how the exercise was carried out. Those categories are: “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)” [1]. The used WLE dataset [2] is available online [1] under the Creative Commons license (CC BY-SA) license and free to use. We present in this work a random forest Machine Learning algorithm that is able to predict new data into the classes A-E, including its cross valiation error. You can see that the proposed machine learning algrithm is a excellent method to predict those execution classes.

1. Data loading and preprocessing

As the initial step in our analysis, we load the training data and

```
# 1. load the data:
setwd("/home/jahnen/Dropbox/Learning/Data Science Degree/Practical Machine Lear
ning/week 3/assignment/")
data <- read.csv("pml-training.csv")

# 2. Remove factor variables and variables with a lot of NAs:
drops <- NULL # which columns to remove?
for(i in names(data)){
    if(is.factor(data[, i])){
        drops <- append(drops, i)
    }
}
# do not remove the outcome variable
drops <- drops[drops != "classe"]

# remove all variables with more than 15000 NA values:
drops <- append(drops, colnames(data)[colSums(is.na(data)) > 15000])

# now remove columns in "drops"
preprocessedData <- data[,!(names(data) %in% drops)]
```

2. Model building and validation

After the preprocessing we are ready to build a ramdom forest machine learning algorithm. I was experimenting with the data a bit (not shown) and a linear model and a tree model was not able to solve this problem in a good way. I therefore decided to use a random forest machine learning algorithm. First, I I used the “ramdomForest” package, as it runs much faster as the caret “rf” method, but resulted in a overfitted model (accurancy = 100 %). I therefore used the caret train function with a PCA preprocessing to avoid overfitting.

I then made the cross validation based on the validaiton data set, that I created in the beginning:

```
library(caret)

set.seed(12345) # for reprodutive results

# 3. Build the model with a reduced training set (and keep data for validatio
n):
inTrain <- createDataPartition(preprocessedData$classe, p = .7, list = FALSE)
trainData <- preprocessedData[inTrain,]
validationData <- preprocessedData[-inTrain,]

# make a random forest model with pca preprocessing:
barbellModel <- train(classe ~ ., data = trainData, method = "rf", preProcess =
"pca")
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
```

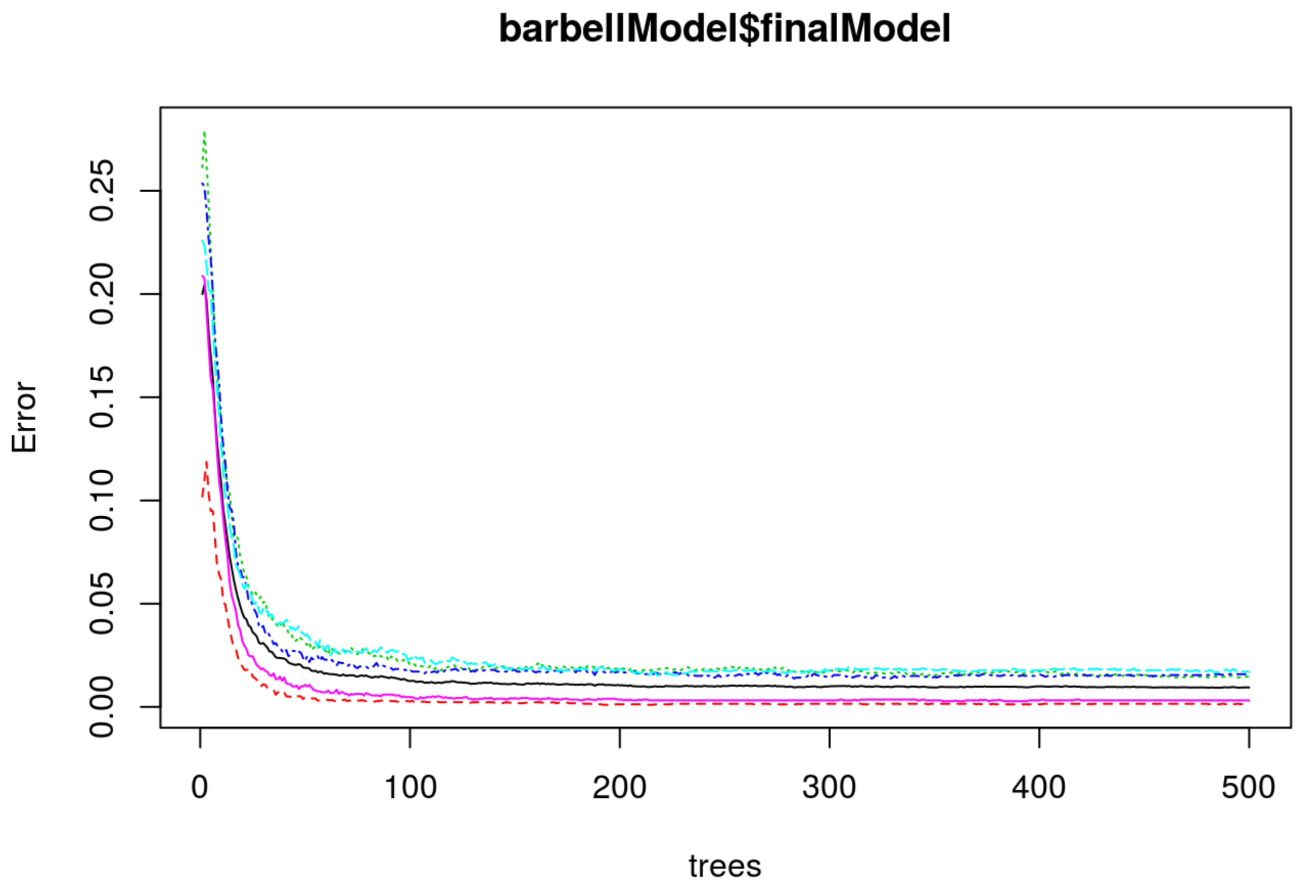
```
# 4.1 check the model based on the validation data (part of training set):
predValidation <- predict(barbellModel, newdata = validationData)
confusionMatrix(validationData$classe, predValidation)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1673    1    0    0    0
##          B    9 1118   12    0    0
##          C    0   12 1004   10    0
##          D    0    0   25  937    2
##          E    0    0    0    5 1077
##
## Overall Statistics
##
##                Accuracy : 0.9871
##                  95% CI : (0.9839, 0.9898)
##     No Information Rate : 0.2858
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9837
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9946   0.9885   0.9645   0.9842   0.9981
## Specificity            0.9998   0.9956   0.9955   0.9945   0.9990
## Pos Pred Value         0.9994   0.9816   0.9786   0.9720   0.9954
## Neg Pred Value         0.9979   0.9973   0.9924   0.9970   0.9996
## Prevalence             0.2858   0.1922   0.1769   0.1618   0.1833
## Detection Rate         0.2843   0.1900   0.1706   0.1592   0.1830
## Detection Prevalence   0.2845   0.1935   0.1743   0.1638   0.1839
## Balanced Accuracy      0.9972   0.9920   0.9800   0.9894   0.9986
```

```
# How many errors do we have?
table(predValidation == validationData$classe)
```

```
##
## FALSE  TRUE
##    76  5809
```

```
# plot the model:
plot(barbellModel$finalModel)
```



4. Conclusions

The developed ramdom forest can predict the validation data quite accurancy. We assume that new data will as well be well predicted.

References:

[1] WLE Dataset Homepage: http://groupware.les.inf.puc-rio.br/har (http://groupware.les.inf.puc-rio.br/har), Accessed online: 22.11.2015

[2] 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.