Prediction excerise for dummbell use

Bert Schwenk

2 september 2018

Introduction

The purpose of this assignment is to predict the correct execution of a dumbell lifting excercise.

For this assignment data was used from Weight Lifting Exercises Dataset (http://groupware.les.inf.puc-rio.br/har). The dataset is licensed under the Creative Commons license (CC BY-SA). It contains the following information: “Six young health participants were asked to perform one set of 10 repetitions of the Unilateral Dumbbell Biceps Curl in five different fashions: 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).” (Velloso, et al., 2013) (http://groupware.les.inf.puc-rio.br/har)

Class A in the dataset corresponds to the correct excecution of the excercise. Classes B until E are incorrect performed excercise executions. This document contains the following sections: 1. Exporatory analysis 2. Data Cleaning 3. Model building 4. Model testing 5. Prediction for assignment 6. Conclusion

Exploratory Analysis

I first load the training/testingsets.

```
training<-read.csv("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv")
testing<-read.csv("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv")
```

I start the analysis by looking at the structure and exploring patterns in the data (str/summary/View/table commands). The dataset contains in total 160 variables, which contains various movement details (e.g. pitch, roll, yaw on belt, arm, dumbell and gloves of the weightlifter), time intervaldata and time groupingdata in num_window and new_window variables. The “classe” variable holds the category how well the excercise was performed. In the testingset this last variable is missing. It is therefore not a real testset, in the sense that it can be used for crossvalidation. It is only used for prediction. I will create a 2nd ‘real’ testset for crossvalidation as part of the trainingset. More in the model building section on this.

Next, I looked into how to use the source data. It contains multiple movements per lift as the data gathered when lifting. There is not a clear cut marker in the data, that divides one dumbbell lift from the other. One weightlifter lifts 10 times correctly. The new_window/num_window variables are just markers that divide the data into ‘x’ time passed. I choose to not take the window markers into account.

```
table(training$classe[training$new_window=="yes"],training$user_name[training$new_window=="yes"])
```

```
##
##     adelmo carlitos charles eurico jeremy pedro
##   A     24       12      21     18     19    15
##   B     15       16      16      9     14     9
##   C     14        8      16      7     16     9
##   D      8        8      14     12     16    11
##   E     22       12      14      8     12    11
```

Furthermore, there are many variables that have a lot of NA’s. This would make prediction of the class in the testset impossible. The counter variable and time variables are left out as these do not have any relationship when applying the model to a new situation. Before I do preprocessing, I first concatenate the training and testset to ensure similar preprocessing. After preprocessing these will be split again for modelling.

```
testingNew<-testing
testingNew$classe<-"A" #assign dummy classe (not used at all)
testingNew<-select(testingNew,-problem_id) #not select problem_id only in testset
nrow(training)
```

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

```
nrow(testingNew)
```

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

```
trainingFiltered <- rbind(training,testingNew)
trainingFiltered <- select(trainingFiltered, classe, user_name,8:159) #do not select timevars/counts
```

Data cleaning

1 skewness_roll_belt.1 variable is named wrong. This is corrected to skewness_pitch_belt.
2 All factor variables are in reality numeric. This is probably because the #DIV/X values. Convert toclass numeric.
3 check ‘summary(training)’ gave insights into which variables are almost not filled or have invalid features. This was explored further with a nearZeroVar analysis, to remove variables that have almost no variance and would have too little predictive power.

```
trainingPrep <- rename(trainingFiltered, skewness_pitch_belt = skewness_roll_belt.1)  # 1) change variable name

trainingPrep[sapply(trainingPrep, is.factor)][,-1]<-lapply(trainingPrep[sapply(trainingPrep, is.factor)][,-1], as.
character) #2)convert all factors (except classe) to numeric (DIV/0 will be NA). (first to character and then to f
actor)
trainingPrep[sapply(trainingPrep, is.character)][,-1]<-lapply(trainingPrep[sapply(trainingPrep, is.character)][,-1
], as.numeric)

trainingPrep <- trainingPrep[,nearZeroVar(trainingPrep[1:19622,],saveMetrics = TRUE)$nzv=="FALSE"] # 3) remove var
iables with low variance.
```

Now we look at the NA values. Analysis showed that there are many variables with NA values (see below). Because for below variables there is almost 100% missing, I will remove these variables for model building. These would not have much predictive power in model building / nor would it help with prediction of classes in the testset.

```
 variables_with_NA <- sapply(trainingPrep, function(x) sum(is.na(x)))[1:length(trainingPrep)][sapply(trainingPrep,
function(x) sum(is.na(x)))[1:length(trainingPrep)]>0]
head(variables_with_NA) #show firstones
```

```
##  kurtosis_roll_belt kurtosis_picth_belt  skewness_roll_belt
##               19246               19268               19245
## skewness_pitch_belt       max_roll_belt      max_picth_belt
##               19268               19236               19236
```

```
trainingPrep<-select(trainingPrep, -c(names(variables_with_NA[variables_with_NA>60]))) #remove variable where too
many NA's (more than 60 NAs)

#replace na's with median value.
na_variables <- names(trainingPrep[,sapply(trainingPrep, function(x) sum(is.na(x)))[1:length(trainingPrep)]>0]) #g
et NA variables
if(length(na_variables)>0){
  for(i in 1:length(na_variables))
  {
     na_matrix <- is.na(trainingPrep[,names(trainingPrep)==na_variables[i]]) #get na/non-na value matrix
     trainingPrep[na_matrix==TRUE,names(trainingPrep)==na_variables[i]] <-     median(trainingPrep[na_matrix==FALS
E,names(trainingPrep)==na_variables[i]]) #get median of values and impute NA values
  }
}
```

Split the testset again from the trainingset before model building. This way training and prediction on test data remain separate. The data in testset remains untouched/unchanged, except for the same preprocessing as trainingset.

```
testingPrep<-trainingPrep[(nrow(trainingPrep)-19):nrow(trainingPrep),-1] #split unchanged but preproc. testset
trainingPrep<-trainingPrep[1:(nrow(trainingPrep)-20),]
```

Model creation

The data is clean enough to do model building. Because the provided testset, doesn't contain any 'classe' variable to check if the results are correct, I split the trainingset again into two parts: realTraining (60%) and realTest (40%). Random subsample without replacement. This way checking of the out of sample errors after model building and cross validation is possible. The assignment 'test set' will only be used for prediction. After dividing the result shows the variance of classes remains the same (see below for the before/after-fractions).

```
set.seed(1234) #random split but with steady results
inTrain <- createDataPartition(y=trainingPrep$classe,p=0.60, list=FALSE)
realTrain <- trainingPrep[inTrain,]
realTest <- trainingPrep[-inTrain,]
prop.table(table(trainingPrep$classe)) #check of proportions of classes are good
```

```
##
##         A         B         C         D         E
## 0.2843747 0.1935073 0.1743961 0.1638977 0.1838243
```

```
prop.table(table(realTrain$classe))
```

```
##
##         A         B         C         D         E
## 0.2843071 0.1935292 0.1744226 0.1638927 0.1838485
```

I start with random forests, which gives in general good results and incorporates additional crossvalidation by folding the trainingdata and select the best model out of many tries. Because this is a classification problem into classes, this would also probably work better with a treemodel than a linear model. I would like to have used all the trainingdata, unfortunatly my pc cannot handle the whole dataset (too little memory). I split off a smaller set.

```
set.seed(123456)
inTrain <- createDataPartition(y=realTrain$classe,p=0.06, list=FALSE)
realTrain1 <- realTrain[inTrain,]
nrow(realTrain1)
```

```
## [1] 708
```

```
modelRF <- train(classe~.,data=realTrain1,method="rf", prox=TRUE)
modelRF$finalModel
```

```
##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry, proximity = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 29
##
##         OOB estimate of  error rate: 13.7%
## Confusion matrix:
##     A   B   C  D   E class.error
## A 190   3   1  6   1  0.05472637
## B  15 109   7  4   2  0.20437956
## C   5   5 109  5   0  0.12096774
## D   6   3  13 91   3  0.21551724
## E   2   5   7  4 112  0.13846154
```

Random forests delivered an error rate of 13.7%. This is ok, but can we improve? Some variables contribute little to the model and could be removed to get better overall performance.

```
importance<-varImp(modelRF, numTrees=5)
importance_Matrix<-importance[1]$importance$Overall<1 #only important enough variables
names(realTrain[,importance_Matrix])
```

```
## [1] "classe"     "user_name"  "pitch_belt" "yaw_belt"
```

The above test shows that username, pitch_belt and yaw_belt only have little predictive value. I remove these. Then I get a new random sample.

```
set.seed(1453)
inTrain <- createDataPartition(y=realTrain$classe,p=0.20, list=FALSE)
realTrain2 <- realTrain[inTrain,]
nrow(realTrain2)
```

```
## [1] 2356
```

```
realTrain2<-select(realTrain2,-c("user_name","pitch_belt","yaw_belt"))
```

I use much more data this time with another random sample from the trainingdata. The traincontrol is also adjusted, to get a balance between more data and memory issues on my pc. I chose 3 cycles + smaller trainingpart in favor of bigger testpart. This resulted in a excellent in sample error rate (5,73%) (see below).

```
train.control<-trainControl(method="cv",number=3,p=0.5, repeats=1)
```

```
## Warning: `repeats` has no meaning for this resampling method.
```

```
modelRF2 <- train(classe~.,data=realTrain2,method="rf", prox=TRUE, trainControl=train.control)
modelRF2$finalModel
```

```
##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry, proximity = TRUE,      trainControl = ..2)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 26
##
##         OOB estimate of  error rate: 5.73%
## Confusion matrix:
##     A   B   C   D   E class.error
## A 662   5   3   0   0  0.01194030
## B  22 405  24   5   0  0.11184211
## C   1  17 384   7   2  0.06569343
## D   2   3  17 364   0  0.05699482
## E   0  11   8   8 406  0.06235566
```

After random forests also boosting is tried, to compare the random forrest result (check is random forest is overfitting). I use the full set of variables here on new sample of trainingdata.

```
set.seed(123)
inTrain <- createDataPartition(y=realTrain$classe,p=0.12, list=FALSE)
realTrain3 <- realTrain[inTrain,]
nrow(realTrain3)
```

```
## [1] 1415
```

```
modelBoosting <- train(classe~.,data=realTrain3, method="gbm", verbose=FALSE)
modelBoosting$results
```

```
##   shrinkage interaction.depth n.minobsinnode n.trees  Accuracy     Kappa
## 1       0.1                 1             10      50 0.7000252 0.6192333
## 4       0.1                 2             10      50 0.7921744 0.7363376
## 7       0.1                 3             10      50 0.8309561 0.7854721
## 2       0.1                 1             10     100 0.7575394 0.6926266
## 5       0.1                 2             10     100 0.8412278 0.7985933
## 8       0.1                 3             10     100 0.8660495 0.8300942
## 3       0.1                 1             10     150 0.7905457 0.7345012
## 6       0.1                 2             10     150 0.8639995 0.8275511
## 9       0.1                 3             10     150 0.8771424 0.8441637
##   AccuracySD    KappaSD
## 1 0.01962665 0.02496190
## 4 0.01787443 0.02272825
## 7 0.01342856 0.01696333
## 2 0.01830309 0.02285909
## 5 0.01715055 0.02167367
## 8 0.01458799 0.01827159
## 3 0.01865570 0.02346600
## 6 0.01640541 0.02062536
## 9 0.01276183 0.01604041
```

The boosting model, got a reasonable result. 0,87% accuracy. This result is comparable with random forrest. As the in accuracy / 'in sample error rates' look quite good for both models, I will move on to model testing.

Model testing

Below the random forest and boosting models are used to predict the testset and check the out of sample errorrate.

```
realTestSmall<-select(realTest,-c("user_name","pitch_belt","yaw_belt")) #apply the same selection as on trainingse
t
predictionRF2 <- predict(modelRF2,realTestSmall)
confusionMatrix(predictionRF2,realTestSmall$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2214   78    0    6    0
##          B    7 1365   61    4   17
##          C    5   65 1282   58   32
##          D    2    9   25 1212   29
##          E    4    1    0    6 1364
##
## Overall Statistics
##
##                Accuracy : 0.9479
##                  95% CI : (0.9427, 0.9527)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.934
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9919   0.8992   0.9371   0.9425   0.9459
## Specificity            0.9850   0.9859   0.9753   0.9901   0.9983
## Pos Pred Value         0.9634   0.9388   0.8890   0.9491   0.9920
## Neg Pred Value         0.9968   0.9761   0.9866   0.9887   0.9879
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2822   0.1740   0.1634   0.1545   0.1738
## Detection Prevalence   0.2929   0.1853   0.1838   0.1628   0.1752
## Balanced Accuracy      0.9885   0.9426   0.9562   0.9663   0.9721
```

The accuracy of the random forrest model is very good, with 0,947. Also Sensitivity, Specificity and other measures look good.

I also checked the Boosting model for out of sample error and prediction value. This is also good with marks in the 90% accuracy. I choose however the random forrest model as the accuracy is a bit higher.

```
predictionBoosting <- predict(modelBoosting,realTest)
confusionMatrix(predictionBoosting,realTest$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2149   89    0    4    5
##          B   34 1338   67   17   70
##          C   20   66 1277   89   44
##          D   20   10   18 1117   24
##          E    9   15    6   59 1299
##
## Overall Statistics
##
##                Accuracy : 0.9151
##                  95% CI : (0.9087, 0.9212)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8926
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9628   0.8814   0.9335   0.8686   0.9008
## Specificity            0.9825   0.9703   0.9662   0.9890   0.9861
## Pos Pred Value         0.9564   0.8768   0.8536   0.9394   0.9359
## Neg Pred Value         0.9852   0.9715   0.9857   0.9746   0.9779
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2739   0.1705   0.1628   0.1424   0.1656
## Detection Prevalence   0.2864   0.1945   0.1907   0.1515   0.1769
## Balanced Accuracy      0.9727   0.9259   0.9498   0.9288   0.9435
```

Predict testset for assignment

For the Coursera assignment we use the provided testset to predict the class. First we do the same preprocessing steps to the testset.

```
testingPrepSmall<-select(testingPrep,-c("user_name","pitch_belt","yaw_belt")) #apply the same selection as on trai
ningset
predictionAssignment <- predict(modelRF2,testingPrepSmall) #I left the answers out, because I am not sure it shoul
d be included in the peerreview :-S
```

Conclusion

The type of dumbbell lift can be predicted with 95% accuracy based on position/movement-measurements with a random forest model.