Prediction Model for Barbell Activity

asandmeyer

25 1 2021

Overview

Barbell activity can be predicted by measuring the acceleration and movement of certain body parts. Here, six young heath patients were asked to perform one set of ten repetitions of the Unilateral Dumbbell Biceps Curl in five different fashions.

- Class A: exactly according to the specification
- Class B: throwing the elbows to the front
- Class C: lifting the dumbbell only halfway
- Class D: lowering the dumbbell only halfway
- Class E: throwing the hips to the front

For this project we use the data to first build a prediction model and than apply it to testing data for answering the quiz questions.

Required libraries

```r
library(lattice)
library(ggplot2)
library(caret)
library(corrplot)
```

```
## corrplot 0.84 loaded
```

Preparing data

Raw data

The raw data is downloaded from the provided data source URL and saved as .csv files.

```r
training <- read.csv("./pml-training.csv")
testing <- read.csv("./pml-testing.csv")
```

The provided raw data is already split into two data set: 70% in `training` and 30% in `testing`. The `training` data will be used to train our model and to analyze our accuracy. The `testing` data will be applied to answer the quiz questions.

Cross-validation and cleaning data

The `training` data set needs to be split into two parts to build our prediction model.

```r
set.seed(123)
inTrain <- createDataPartition(training$classe, p=0.7, list=FALSE)
TrainSet <- training[inTrain, ]
TestSet <- training[-inTrain, ]
```

In `TrainSet` we put 70% of `training` which will be used to train our model. The rest (30%) was assigned to `TestSet` which will be used to validate our model.

As the next step, we need to clean the data sets since we have 160 variables and there are many `NA` values or blank values.

```
#removing NA
near_NA <- nearZeroVar(TrainSet)
TrainSet_clean <- TrainSet[ ,-near_NA]
TestSet_clean <- TestSet[ ,-near_NA]

#removing values close to zero, choose threshold of 95% is chosen
close_to_zero <- sapply(TrainSet_clean, function(x) mean(is.na(x))) > 0.95
TrainSet_clean <- TrainSet_clean[ ,close_to_zero==F]
TestSet_clean <- TestSet_clean[ ,close_to_zero==F]
```

We now have 59 variables. Since the first five columns are identification variables and the next two are timestamps, we are also going to remove them.

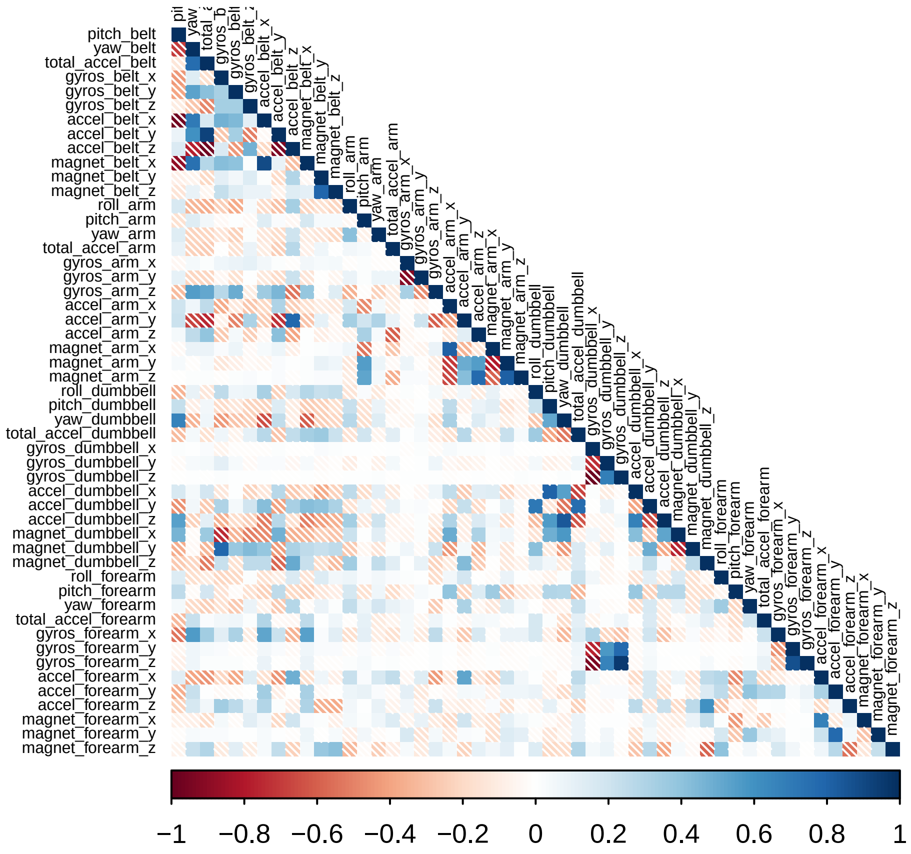
```
TrainSet_clean <- TrainSet_clean[, -(1:7)]
TestSet_clean <- TestSet_clean[, -(1:7)]
```

Finally, we only have 52 variables left.

Correlation Analysis

Since we might not need every predictor because some of them are highly correlated, we correlate the predictors and plot the result.

```
corMat <- cor(TrainSet_clean[,-52])
corrplot(corMat, method="shade", type="lower", tl.col="black", tl.cex=0.5)
```



In case two variables are highly correlated they either have a bold dark blue shade or a dark red shade with white strips. However, the dark blue shades at the diagonal are due to the fact, that these are variables (of course) highly correlated with themselves. The other spots with indicate correlation are just a few, and therefore the decision is made to keep the data sets as they are. Otherwise, *principal component analysis* could be done as a next step. Here, only variables are taken into account which distribute highly too explain a certain amount of the variance of the data.

Prediction Model

According to the Coursera R course *Practical Machine Learning* and literature, **Random Forest Model** is one of the most accurate prediction model. Therefore, we decide to take that model.

Building Model and validation

```
#Building model, with cache=TRUE to save time
cv_RF <- trainControl(method="cv", number=3, verboseIter=F)
modFit_RF <- train(classe ~ ., data=TrainSet_clean, method="rf", trControl=cv_RF, verbose=F)
modFit_RF$finalModel
```

```
##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry, verbose = ..1)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 26
##
##         OOB estimate of  error rate: 0.71%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 3899    5    1    0    1 0.001792115
## B   19 2630    7    0    2 0.010534236
## C    0   14 2372   10    0 0.010016694
## D    0    0   28 2222    2 0.013321492
## E    0    0    3    5 2517 0.003168317
```

```
#validation on test data generated from training data set
prediction_RF <- predict(modFit_RF, newdata=TestSet_clean)
conMat_RF <- confusionMatrix(table(TestSet_clean$classe, prediction_RF))
conMat_RF
```

```
## Confusion Matrix and Statistics
##
##    prediction_RF
##        A    B    C    D    E
##   A 1673    1    0    0    0
##   B    5 1128    6    0    0
##   C    0    5 1020    1    0
##   D    0    0    6  957    1
##   E    0    0    4    4 1074
##
## Overall Statistics
##
##                Accuracy : 0.9944
##                  95% CI : (0.9921, 0.9961)
##     No Information Rate : 0.2851
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9929
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9970   0.9947   0.9846   0.9948   0.9991
## Specificity            0.9998   0.9977   0.9988   0.9986   0.9983
## Pos Pred Value         0.9994   0.9903   0.9942   0.9927   0.9926
## Neg Pred Value         0.9988   0.9987   0.9967   0.9990   0.9998
## Prevalence             0.2851   0.1927   0.1760   0.1635   0.1827
## Detection Rate         0.2843   0.1917   0.1733   0.1626   0.1825
## Detection Prevalence   0.2845   0.1935   0.1743   0.1638   0.1839
## Balanced Accuracy      0.9984   0.9962   0.9917   0.9967   0.9987
```

According to our validation on the `TestSet_clean` we get an accuracy of 0.9943925 with our Random Forest Model `modFit_RF`.

Applying Model to testing data

Now, as a final step, we want to apply our model `modFit_RF` to the `testing` data set to answer the questions from the quiz.

```
quiz_solution <- predict(modFit_RF, newdata=testing)
quiz_solution
```

```
##  [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
```