Prediction of Movement Quality

S. Wassenburg

November 21, 2018

Course Project Instruction - Practical Machine Learning

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://web.archive.org/web/20161224072740/http:/groupware.les.inf.puc-rio.br/har (see the section on the Weight Lifting Exercise Dataset).

The data for this project can be found here: Training data: https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv Test data: https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv

The data for this project come from this source: http://web.archive.org/web/20161224072740/http:/groupware.les.inf.puc-rio.br/har

Getting Data

Load data and packages:

```
TrainData <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
TestData <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"
train <- read.csv(TrainData)
test <- read.csv(TestData)
library(dplyr)
library(caret)
```

Because the goal is to predict the variable 'classe', we will first look at the number of cases per class in the training data. Every category has at least 3200 cases.

```
summarize(group_by(train, classe), Training=n())
```

```
## # A tibble: 5 x 2
##   classe Training
##   <fct>     <int>
## 1 A          5580
## 2 B          3797
## 3 C          3422
## 4 D          3216
## 5 E          3607
```

Cross Validation

For this report, we will use a simple type of cross validation technique, known as the Holdout Method. This means that the training will be subset into a training and test set. This way, we can train several machine learning models and try them on the test subset to see how they generalize to new data. Based on these

tests, we can choose the best fitting model, which is not overfitted to the training data (evidenced by a good fit to the sub test data).

We will now split the training dataset into a training subset (75%) and test subset (25%) so we can test multiple machine learning algorithms, before testing the best algorithm on the final test dataset.

```
set.seed(101)
inTrain <- createDataPartition(y=train$classe, p=0.75, list=FALSE)
subtrain <- train[inTrain, ]; subtest <- train[-inTrain, ]
dim(subtrain); dim(subtest)
```

```
## [1] 14718   160
```

```
## [1] 4904  160
```

Preprocessing

Data transformations will be performed based on the subtraining data and the exact same transformations will be performed on the sub test and final test data, without exploring these datasets.

First, as many machine learning algorithms do not support missing values, we will check for columns with missing values in the subtraining dataset.

```
na <-sapply(subtrain, function(y) sum(length(which(is.na(y)))))
na <- data.frame(na)
na <- tibble::rownames_to_column(na)
blank <- sapply(subtrain, function(y) sum(length(which(y==""))))
blank <- data.frame(blank)
blank <- tibble::rownames_to_column(blank)
na_blank <- full_join(na, blank, by=c("rowname","rowname"))
na_blank <- mutate(na_blank, NaBlank=na+blank, NaBlankPerc=(na+blank)/nrow(subtrain)*100)
filter(na_blank, NaBlank > 0)
```

```
##                      rowname    na blank NaBlank NaBlankPerc
## 1         kurtosis_roll_belt     0 14415   14415     97.9413
## 2        kurtosis_picth_belt     0 14415   14415     97.9413
## 3          kurtosis_yaw_belt     0 14415   14415     97.9413
## 4         skewness_roll_belt     0 14415   14415     97.9413
## 5       skewness_roll_belt.1     0 14415   14415     97.9413
## 6          skewness_yaw_belt     0 14415   14415     97.9413
## 7              max_roll_belt 14415     0   14415     97.9413
## 8             max_picth_belt 14415     0   14415     97.9413
## 9                max_yaw_belt     0 14415   14415     97.9413
## 10             min_roll_belt 14415     0   14415     97.9413
## 11            min_pitch_belt 14415     0   14415     97.9413
## 12               min_yaw_belt     0 14415   14415     97.9413
## 13       amplitude_roll_belt 14415     0   14415     97.9413
## 14      amplitude_pitch_belt 14415     0   14415     97.9413
## 15        amplitude_yaw_belt     0 14415   14415     97.9413
## 16      var_total_accel_belt 14415     0   14415     97.9413
## 17             avg_roll_belt 14415     0   14415     97.9413
## 18          stddev_roll_belt 14415     0   14415     97.9413
## 19             var_roll_belt 14415     0   14415     97.9413
## 20            avg_pitch_belt 14415     0   14415     97.9413
## 21         stddev_pitch_belt 14415     0   14415     97.9413
## 22            var_pitch_belt 14415     0   14415     97.9413
## 23              avg_yaw_belt 14415     0   14415     97.9413
```

```
## 24           stddev_yaw_belt 14415     0   14415     97.9413
## 25              var_yaw_belt 14415     0   14415     97.9413
## 26             var_accel_arm 14415     0   14415     97.9413
## 27              avg_roll_arm 14415     0   14415     97.9413
## 28           stddev_roll_arm 14415     0   14415     97.9413
## 29              var_roll_arm 14415     0   14415     97.9413
## 30             avg_pitch_arm 14415     0   14415     97.9413
## 31          stddev_pitch_arm 14415     0   14415     97.9413
## 32             var_pitch_arm 14415     0   14415     97.9413
## 33               avg_yaw_arm 14415     0   14415     97.9413
## 34            stddev_yaw_arm 14415     0   14415     97.9413
## 35               var_yaw_arm 14415     0   14415     97.9413
## 36         kurtosis_roll_arm     0 14415   14415     97.9413
## 37        kurtosis_picth_arm     0 14415   14415     97.9413
## 38          kurtosis_yaw_arm     0 14415   14415     97.9413
## 39         skewness_roll_arm     0 14415   14415     97.9413
## 40        skewness_pitch_arm     0 14415   14415     97.9413
## 41          skewness_yaw_arm     0 14415   14415     97.9413
## 42              max_roll_arm 14415     0   14415     97.9413
## 43             max_picth_arm 14415     0   14415     97.9413
## 44               max_yaw_arm 14415     0   14415     97.9413
## 45              min_roll_arm 14415     0   14415     97.9413
## 46             min_pitch_arm 14415     0   14415     97.9413
## 47               min_yaw_arm 14415     0   14415     97.9413
## 48        amplitude_roll_arm 14415     0   14415     97.9413
## 49       amplitude_pitch_arm 14415     0   14415     97.9413
## 50         amplitude_yaw_arm 14415     0   14415     97.9413
## 51    kurtosis_roll_dumbbell     0 14415   14415     97.9413
## 52   kurtosis_picth_dumbbell     0 14415   14415     97.9413
## 53     kurtosis_yaw_dumbbell     0 14415   14415     97.9413
## 54    skewness_roll_dumbbell     0 14415   14415     97.9413
## 55   skewness_pitch_dumbbell     0 14415   14415     97.9413
## 56     skewness_yaw_dumbbell     0 14415   14415     97.9413
## 57         max_roll_dumbbell 14415     0   14415     97.9413
## 58        max_picth_dumbbell 14415     0   14415     97.9413
## 59          max_yaw_dumbbell     0 14415   14415     97.9413
## 60         min_roll_dumbbell 14415     0   14415     97.9413
## 61        min_pitch_dumbbell 14415     0   14415     97.9413
## 62          min_yaw_dumbbell     0 14415   14415     97.9413
## 63   amplitude_roll_dumbbell 14415     0   14415     97.9413
## 64  amplitude_pitch_dumbbell 14415     0   14415     97.9413
## 65    amplitude_yaw_dumbbell     0 14415   14415     97.9413
## 66        var_accel_dumbbell 14415     0   14415     97.9413
## 67         avg_roll_dumbbell 14415     0   14415     97.9413
## 68      stddev_roll_dumbbell 14415     0   14415     97.9413
## 69         var_roll_dumbbell 14415     0   14415     97.9413
## 70        avg_pitch_dumbbell 14415     0   14415     97.9413
## 71     stddev_pitch_dumbbell 14415     0   14415     97.9413
## 72        var_pitch_dumbbell 14415     0   14415     97.9413
## 73          avg_yaw_dumbbell 14415     0   14415     97.9413
## 74       stddev_yaw_dumbbell 14415     0   14415     97.9413
## 75          var_yaw_dumbbell 14415     0   14415     97.9413
## 76     kurtosis_roll_forearm     0 14415   14415     97.9413
## 77    kurtosis_picth_forearm     0 14415   14415     97.9413
```

```
## 78      kurtosis_yaw_forearm     0 14415   14415     97.9413
## 79     skewness_roll_forearm     0 14415   14415     97.9413
## 80    skewness_pitch_forearm     0 14415   14415     97.9413
## 81      skewness_yaw_forearm     0 14415   14415     97.9413
## 82          max_roll_forearm 14415     0   14415     97.9413
## 83         max_picth_forearm 14415     0   14415     97.9413
## 84           max_yaw_forearm     0 14415   14415     97.9413
## 85          min_roll_forearm 14415     0   14415     97.9413
## 86         min_pitch_forearm 14415     0   14415     97.9413
## 87           min_yaw_forearm     0 14415   14415     97.9413
## 88    amplitude_roll_forearm 14415     0   14415     97.9413
## 89   amplitude_pitch_forearm 14415     0   14415     97.9413
## 90     amplitude_yaw_forearm     0 14415   14415     97.9413
## 91         var_accel_forearm 14415     0   14415     97.9413
## 92          avg_roll_forearm 14415     0   14415     97.9413
## 93       stddev_roll_forearm 14415     0   14415     97.9413
## 94          var_roll_forearm 14415     0   14415     97.9413
## 95         avg_pitch_forearm 14415     0   14415     97.9413
## 96      stddev_pitch_forearm 14415     0   14415     97.9413
## 97         var_pitch_forearm 14415     0   14415     97.9413
## 98           avg_yaw_forearm 14415     0   14415     97.9413
## 99        stddev_yaw_forearm 14415     0   14415     97.9413
## 100          var_yaw_forearm 14415     0   14415     97.9413
```

There are 100 variables with more than 97% missing values. I will remove these variables from the dataset.

```
notMissing <- filter(na_blank, NaBlank == 0)
NewCol <- notMissing$rowname
subtrain <- select(subtrain,one_of(NewCol))
```

Second, we will check for variables without variance, because these are not useful to use as a predictor in the model.

```
NZV <- nearZeroVar(subtrain, saveMetrics=TRUE)
```

Only one variable has no variance: new_window. We remove the window and timestamp variables and the ID column from the subtrain dataset.

```
subtrain = subtrain[-c(1,2,3,4,5,6,7)]
```

The remaining columns will be used in the machine learning models.

```
str(subtrain)
```

```
## 'data.frame':    14718 obs. of  53 variables:
##  $ roll_belt            : num  1.41 1.48 1.48 1.45 1.42 1.45 1.45 1.43 1.42 1.42 ...
##  $ pitch_belt           : num  8.07 8.05 8.07 8.06 8.13 8.17 8.18 8.18 8.2 8.21 ...
##  $ yaw_belt             : num  -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 ...
##  $ total_accel_belt     : int  3 3 3 3 3 3 3 3 3 3 ...
##  $ gyros_belt_x         : num  0.02 0.02 0.02 0.02 0.02 0.03 0.03 0.02 0.02 0.02 ...
##  $ gyros_belt_y         : num  0 0 0.02 0 0 0 0 0 0 0 ...
##  $ gyros_belt_z         : num  -0.02 -0.03 -0.02 -0.02 -0.02 0 -0.02 -0.02 0 -0.02 ...
##  $ accel_belt_x         : int  -22 -22 -21 -21 -22 -21 -21 -22 -22 -22 ...
##  $ accel_belt_y         : int  4 3 2 4 4 4 2 2 4 4 ...
##  $ accel_belt_z         : int  22 21 24 21 21 22 23 23 21 21 ...
##  $ magnet_belt_x        : int  -7 -6 -6 0 -2 -3 -5 -2 -3 -8 ...
##  $ magnet_belt_y        : int  608 604 600 603 603 609 596 602 606 598 ...
##  $ magnet_belt_z        : int  -311 -310 -302 -312 -313 -308 -317 -319 -309 -310 ...
```

```
##  $ roll_arm            : num  -128 -128 -128 -128 -128 -128 -128 -128 -128 -128 ...
##  $ pitch_arm           : num  22.5 22.1 22.1 22 21.8 21.6 21.5 21.5 21.4 21.4 ...
##  $ yaw_arm             : num  -161 -161 -161 -161 -161 -161 -161 -161 -161 -161 ...
##  $ total_accel_arm     : int  34 34 34 34 34 34 34 34 34 34 ...
##  $ gyros_arm_x         : num  0.02 0.02 0 0.02 0.02 0.02 0.02 0.02 0.02 0.02 ...
##  $ gyros_arm_y         : num  -0.02 -0.03 -0.03 -0.03 -0.02 -0.03 -0.03 -0.03 -0.02 0 ...
##  $ gyros_arm_z         : num  -0.02 0.02 0 0 0 -0.02 0 0 -0.02 -0.03 ...
##  $ accel_arm_x         : int  -290 -289 -289 -289 -289 -288 -290 -288 -287 -288 ...
##  $ accel_arm_y         : int  110 111 111 111 111 110 110 111 111 111 ...
##  $ accel_arm_z         : int  -125 -123 -123 -122 -124 -124 -123 -123 -124 -124 ...
##  $ magnet_arm_x        : int  -369 -372 -374 -369 -372 -376 -366 -363 -372 -371 ...
##  $ magnet_arm_y        : int  337 344 337 342 338 334 339 343 338 331 ...
##  $ magnet_arm_z        : int  513 512 506 513 510 516 509 520 509 523 ...
##  $ roll_dumbbell       : num  13.1 13.4 13.4 13.4 12.8 ...
##  $ pitch_dumbbell      : num  -70.6 -70.4 -70.4 -70.8 -70.3 ...
##  $ yaw_dumbbell        : num  -84.7 -84.9 -84.9 -84.5 -85.1 ...
##  $ total_accel_dumbbell: int  37 37 37 37 37 37 37 37 37 37 ...
##  $ gyros_dumbbell_x    : num  0 0 0 0 0 0 0 0 0.02 ...
##  $ gyros_dumbbell_y    : num  -0.02 -0.02 -0.02 -0.02 -0.02 -0.02 -0.02 -0.02 -0.02 -0.02 ...
##  $ gyros_dumbbell_z    : num  0 -0.02 0 0 0 0 0 0 -0.02 -0.02 ...
##  $ accel_dumbbell_x    : int  -233 -232 -233 -234 -234 -235 -233 -233 -234 -234 ...
##  $ accel_dumbbell_y    : int  47 48 48 48 46 48 47 47 48 48 ...
##  $ accel_dumbbell_z    : int  -269 -269 -270 -269 -272 -270 -269 -270 -269 -268 ...
##  $ magnet_dumbbell_x   : int  -555 -552 -554 -558 -555 -558 -564 -554 -552 -554 ...
##  $ magnet_dumbbell_y   : int  296 303 292 294 300 291 299 291 302 295 ...
##  $ magnet_dumbbell_z   : num  -64 -60 -68 -66 -74 -69 -64 -65 -69 -68 ...
##  $ roll_forearm        : num  28.3 28.1 28 27.9 27.8 27.7 27.6 27.5 27.2 27.2 ...
##  $ pitch_forearm       : num  -63.9 -63.9 -63.9 -63.9 -63.8 -63.8 -63.8 -63.8 -63.9 -63.9 ...
##  $ yaw_forearm         : num  -153 -152 -152 -152 -152 -152 -152 -152 -151 -151 ...
##  $ total_accel_forearm : int  36 36 36 36 36 36 36 36 36 36 ...
##  $ gyros_forearm_x     : num  0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0 0 ...
##  $ gyros_forearm_y     : num  0 -0.02 0 -0.02 -0.02 0 -0.02 0.02 0 -0.02 ...
##  $ gyros_forearm_z     : num  -0.02 0 -0.02 -0.03 0 -0.02 -0.02 -0.03 -0.03 -0.03 ...
##  $ accel_forearm_x     : int  192 189 189 193 193 190 193 191 193 193 ...
##  $ accel_forearm_y     : int  203 206 206 203 205 205 205 203 205 202 ...
##  $ accel_forearm_z     : int  -216 -214 -214 -215 -213 -215 -214 -215 -215 -214 ...
##  $ magnet_forearm_x    : int  -18 -16 -17 -9 -9 -22 -17 -11 -15 -14 ...
##  $ magnet_forearm_y    : num  661 658 655 660 660 656 657 657 655 659 ...
##  $ magnet_forearm_z    : num  473 469 473 478 474 473 465 478 472 478 ...
##  $ classe              : Factor w/ 5 levels "A","B","C","D",..: 1 1 1 1 1 1 1 1 1 1 ...
```

Before trying out machine learning algorithms, the exact same transformations have to be performed on the subtest and final test data. Also, we will remove the 'classe' column from the final test dataset, as we will predict this variable with our machine learning model.

```
transformsubtest <- colnames(subtrain)
transformtest <- colnames(subtrain[, -53])
subtest <- subtest[transformsubtest]
test <- test[transformtest]
```

Machine Learning Algorithms

Because we want to predict to which class an observation belongs, we will choose **classification algorithms**, such as Decision Trees and Random Forest. - Naive Bayes is not suitable because it assumes independence between predictors. - Logistics regression is not suitable because it requires a dichotomous outcome.

Decision Tree

We will start with a Decision (classification) Tree model using the rpart package and using all variables to predict 'classe'.

```
set.seed(345)
modTree <- train(classe~., data = subtrain, method="rpart")
modTree
```

```
## CART
##
## 14718 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 14718, 14718, 14718, 14718, 14718, 14718, ...
## Resampling results across tuning parameters:
##
##   cp          Accuracy   Kappa
##   0.03531757  0.5100897  0.36045149
##   0.06133105  0.4133340  0.20344659
##   0.11620621  0.3270011  0.06523629
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.03531757.
```

The accuracy of prediction is not very high (51%). We will test the model on the subtest data. The accuracy is only 49%.

```
predTree <- predict(modTree, subtest)
confusionMatrix(predTree, subtest$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1255  391  396  338  123
##          B   27  327   37  174  126
##          C  110  231  422  292  256
##          D    0    0    0    0    0
##          E    3    0    0    0  396
##
## Overall Statistics
##
##                Accuracy : 0.4894
##                  95% CI : (0.4753, 0.5035)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3334
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
```

```
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.8996  0.34457  0.49357   0.0000  0.43951
## Specificity            0.6443  0.90796  0.78044   1.0000  0.99925
## Pos Pred Value         0.5014  0.47323  0.32189      NaN  0.99248
## Neg Pred Value         0.9417  0.85236  0.87949   0.8361  0.88790
## Prevalence             0.2845  0.19352  0.17435   0.1639  0.18373
## Detection Rate         0.2559  0.06668  0.08605   0.0000  0.08075
## Detection Prevalence   0.5104  0.14091  0.26733   0.0000  0.08136
## Balanced Accuracy      0.7720  0.62627  0.63700   0.5000  0.71938
```

Random Forest

We will now train a Random Forest model on the subtraining data using all included variables to predict 'classe' (note that it takes a long time to run this on a regular computer).

```
set.seed(456)
modRF <- train(classe~., data = subtrain, method="rf")
modRF
```

```
## Random Forest
##
## 14718 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 14718, 14718, 14718, 14718, 14718, 14718, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9892402  0.9863885
##   27    0.9898080  0.9871070
##   52    0.9775501  0.9716013
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 27.
```

We find that this model fits the subtraining data really well, with 99% accuracy. We will test the model on the subtest data to get an indication of the in-sample error.

```
predRF <- predict(modRF, subtest)
confusionMatrix(predRF, subtest$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1391   10    0    0    0
##          B    2  935    1    1    0
##          C    1    4  850    7    1
##          D    0    0    4  795    3
##          E    1    0    0    1  897
##
## Overall Statistics
##
```

```
##                Accuracy : 0.9927
##                  95% CI : (0.9899, 0.9949)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9907
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9971   0.9852   0.9942   0.9888   0.9956
## Specificity            0.9972   0.9990   0.9968   0.9983   0.9995
## Pos Pred Value         0.9929   0.9957   0.9849   0.9913   0.9978
## Neg Pred Value         0.9989   0.9965   0.9988   0.9978   0.9990
## Prevalence             0.2845   0.1935   0.1743   0.1639   0.1837
## Detection Rate         0.2836   0.1907   0.1733   0.1621   0.1829
## Detection Prevalence   0.2857   0.1915   0.1760   0.1635   0.1833
## Balanced Accuracy      0.9971   0.9921   0.9955   0.9935   0.9975
```

Again, the model works really well on the subtest data and predict the 'classe' variable with 99% accuracy. Therefore, it makes sense to choose the Random Forest model over the Decision Tree model.

Prediction

Finally, we will predict the 20 cases in the test dataset for the quiz. Based on the good model fit on the subtraining data and the high accuracy on the subtest data, I expect a small out-of-sample error when testing the model on the final test set. I expect to at least predict 19 out of 20 cases correctly (20*0.99=19.8).

```
predict(modRF, 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
```

Thanks for reading!