Practical Machine Learning - Course Project

Arne Schoch

9/9/2019

Aim of the project

Predict classe response based on movement predictors.

```r
training <- read.csv("pml-training.csv", header = TRUE, row.names = 1)
testing <- read.csv("pml-testing.csv", header = TRUE, row.names = 1)

str(training)
```

```
## 'data.frame':    19622 obs. of  159 variables:
##  $ user_name               : Factor w/ 6 levels "adelmo","carlitos",..: 2 2 2 2 2 2 2 2 2 2 ...
##  $ raw_timestamp_part_1    : int  1323084231 1323084231 1323084231 1323084232 1323084232 1323084232 1
##  $ raw_timestamp_part_2    : int  788290 808298 820366 120339 196328 304277 368296 440390 484323 4844
##  $ cvtd_timestamp          : Factor w/ 20 levels "02/12/2011 13:32",..: 9 9 9 9 9 9 9 9 9 9 ...
##  $ new_window              : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 1 ...
##  $ num_window              : int  11 11 11 12 12 12 12 12 12 12 ...
##  $ roll_belt               : num  1.41 1.41 1.42 1.48 1.48 1.45 1.42 1.42 1.43 1.45 ...
##  $ pitch_belt              : num  8.07 8.07 8.07 8.05 8.07 8.06 8.09 8.13 8.16 8.17 ...
##  $ 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 ...
##  $ kurtosis_roll_belt      : Factor w/ 397 levels "","-0.016850",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ kurtosis_picth_belt     : Factor w/ 317 levels "","-0.021887",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ kurtosis_yaw_belt       : Factor w/ 2 levels "","#DIV/0!": 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_roll_belt      : Factor w/ 395 levels "","-0.003095",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_roll_belt.1    : Factor w/ 338 levels "","-0.005928",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_yaw_belt       : Factor w/ 2 levels "","#DIV/0!": 1 1 1 1 1 1 1 1 1 1 ...
##  $ max_roll_belt           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_picth_belt          : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_yaw_belt            : Factor w/ 68 levels "","-0.1","-0.2",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ min_roll_belt           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_pitch_belt          : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_yaw_belt            : Factor w/ 68 levels "","-0.1","-0.2",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ amplitude_roll_belt     : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_pitch_belt    : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_yaw_belt      : Factor w/ 4 levels "","#DIV/0!","0.00",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ var_total_accel_belt    : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_roll_belt           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_roll_belt        : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_roll_belt           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_pitch_belt          : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_pitch_belt       : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_pitch_belt          : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_yaw_belt            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_yaw_belt         : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_yaw_belt            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ gyros_belt_x            : num  0 0.02 0 0.02 0.02 0.02 0.02 0.02 0.02 0.03 ...
##  $ gyros_belt_y            : num  0 0 0 0 0.02 0 0 0 0 0 ...
```

```
##  $ gyros_belt_z            : num  -0.02 -0.02 -0.02 -0.03 -0.02 -0.02 -0.02 -0.02 -0.02 0 ...
##  $ accel_belt_x            : int  -21 -22 -20 -22 -21 -21 -22 -22 -20 -21 ...
##  $ accel_belt_y            : int  4 4 5 3 2 4 3 4 2 4 ...
##  $ accel_belt_z            : int  22 22 23 21 24 21 21 21 24 22 ...
##  $ magnet_belt_x           : int  -3 -7 -2 -6 -6 0 -4 -2 1 -3 ...
##  $ magnet_belt_y           : int  599 608 600 604 600 603 599 603 602 609 ...
##  $ magnet_belt_z           : int  -313 -311 -305 -310 -302 -312 -311 -313 -312 -308 ...
##  $ roll_arm                : num  -128 -128 -128 -128 -128 -128 -128 -128 -128 -128 ...
##  $ pitch_arm               : num  22.5 22.5 22.5 22.1 22.1 22 21.9 21.8 21.7 21.6 ...
##  $ 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 ...
##  $ var_accel_arm           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_roll_arm            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_roll_arm         : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_roll_arm            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_pitch_arm           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_pitch_arm        : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_pitch_arm           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_yaw_arm             : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_yaw_arm          : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_yaw_arm             : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ gyros_arm_x             : num  0 0.02 0.02 0.02 0 0.02 0 0.02 0.02 0.02 ...
##  $ gyros_arm_y             : num  0 -0.02 -0.02 -0.03 -0.03 -0.03 -0.03 -0.02 -0.03 -0.03 ...
##  $ gyros_arm_z             : num  -0.02 -0.02 -0.02 0.02 0 0 0 0 -0.02 -0.02 ...
##  $ accel_arm_x             : int  -288 -290 -289 -289 -289 -289 -289 -289 -288 -288 ...
##  $ accel_arm_y             : int  109 110 110 111 111 111 111 111 109 110 ...
##  $ accel_arm_z             : int  -123 -125 -126 -123 -123 -122 -125 -124 -122 -124 ...
##  $ magnet_arm_x            : int  -368 -369 -368 -372 -374 -369 -373 -372 -369 -376 ...
##  $ magnet_arm_y            : int  337 337 344 344 337 342 336 338 341 334 ...
##  $ magnet_arm_z            : int  516 513 513 512 506 513 509 510 518 516 ...
##  $ kurtosis_roll_arm       : Factor w/ 330 levels "","-0.02438",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ kurtosis_picth_arm      : Factor w/ 328 levels "","-0.00484",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ kurtosis_yaw_arm        : Factor w/ 395 levels "","-0.01548",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_roll_arm       : Factor w/ 331 levels "","-0.00051",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_pitch_arm      : Factor w/ 328 levels "","-0.00184",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_yaw_arm        : Factor w/ 395 levels "","-0.00311",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ max_roll_arm            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_picth_arm           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_yaw_arm             : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_roll_arm            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_pitch_arm           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_yaw_arm             : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_roll_arm      : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_pitch_arm     : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_yaw_arm       : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ roll_dumbbell           : num  13.1 13.1 12.9 13.4 13.4 ...
##  $ pitch_dumbbell          : num  -70.5 -70.6 -70.3 -70.4 -70.4 ...
##  $ yaw_dumbbell            : num  -84.9 -84.7 -85.1 -84.9 -84.9 ...
##  $ kurtosis_roll_dumbbell  : Factor w/ 398 levels "","-0.0035","-0.0073",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ kurtosis_picth_dumbbell : Factor w/ 401 levels "","-0.0163","-0.0233",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ kurtosis_yaw_dumbbell   : Factor w/ 2 levels "","#DIV/0!": 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_roll_dumbbell  : Factor w/ 401 levels "","-0.0082","-0.0096",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_pitch_dumbbell : Factor w/ 402 levels "","-0.0053","-0.0084",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_yaw_dumbbell   : Factor w/ 2 levels "","#DIV/0!": 1 1 1 1 1 1 1 1 1 1 ...
```

```
##  $ max_roll_dumbbell       : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_picth_dumbbell      : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_yaw_dumbbell        : Factor w/ 73 levels "","-0.1","-0.2",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ min_roll_dumbbell       : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_pitch_dumbbell      : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_yaw_dumbbell        : Factor w/ 73 levels "","-0.1","-0.2",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ amplitude_roll_dumbbell : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_pitch_dumbbell: num  NA NA NA NA NA NA NA NA NA NA ...
##   [list output truncated]
```

Some of the predictors appear to exhibit a large number of NA-values. Which predictors have more than 50% NA-observations and/or blank values?

```
index_na <- vector()
counter <- 1

for(i in 1:ncol(training)){
  ratio_na <- as.data.frame(prop.table(table(is.na(training[,i]))))

  if(ratio_na$Freq[ratio_na$Var1 == FALSE] < 0.5 | sum(training[,i] == "") > 1){
    index_na[counter] <- i
    counter <- counter+1
  }
}
```

In a more elaborate walkthrough, you could try to impute the missing values for the "NA-rich" predictors. For now, I will discard respective predictors. Note that there are no more NA-values in the training dataset. Therefore, there is no need for imputation later on. If imputation was needed, I would probably use a proximity matrix to do so, since I will classify the data based on the random Forest method.

I will also remove the timestamp-columns.

```
# does the testing data have the same column order as the training data?
check_col <- colnames(training) == colnames(testing)
which(check_col == FALSE)
```

```
## [1] 159
```

```
# the last column holds different variables in training and testing data, the others are the same

# delete na-rich or blank-rich predictors
training <- training[,-index_na]
testing <- testing[,-index_na]

# delete time stamp columns
training <- training[,-(2:4)]
testing <- testing[,-(2:4)]
```

Let's try to classify the data using the randomForest method. First, we split the data to allow for cross-validation.

Random forest computation takes a long time without parallelization. Hence, I will use the parallel and doParallel package to reduce time expenses.

```
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```r
library(parallel)
library(doParallel)
```

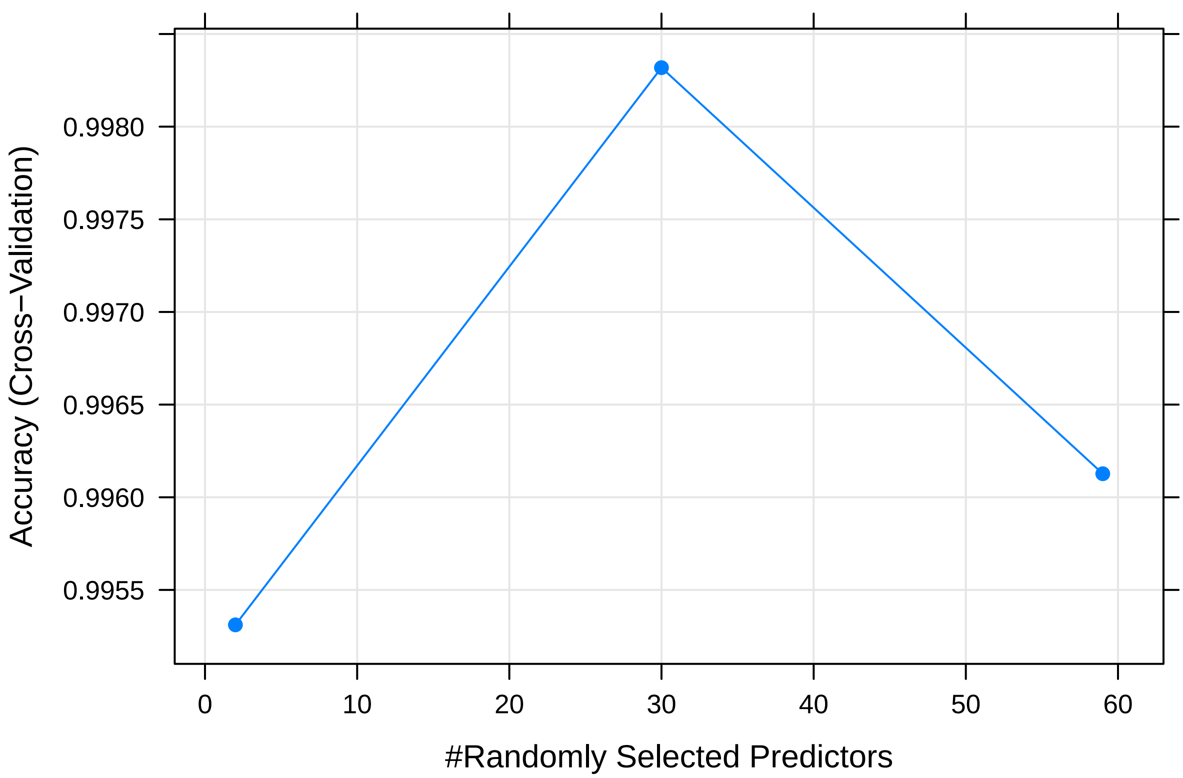
```
## Loading required package: foreach
```

```
## Loading required package: iterators
```

```r
library(ggplot2)
set.seed(42)

# define cluster
cluster <- makeCluster(detectCores() - 1)
registerDoParallel(cluster)

# define 5-fold cross-validation
trControl <- trainControl(method="cv", number=5, allowParallel = TRUE)

# create random forest model
model_rf <- train(classe~., data=training, method="rf", trControl=trControl, verbose=FALSE)

# shutting down cluster after successful computation
stopCluster(cluster)
registerDoSEQ()
```

The random forest has one major hyperparameter, mtry, i.e. the number of randomly selected predictors per node. Since I applied cross-validation, I can estimate an optimal mtry-value based on cross-validation accuracy.

```r
plot(model_rf, pch = 19)
```



```r
abline
```

```
## function (a = NULL, b = NULL, h = NULL, v = NULL, reg = NULL,
##     coef = NULL, untf = FALSE, ...)
## {
##     int_abline <- function(a, b, h, v, untf, col = par("col"),
##         lty = par("lty"), lwd = par("lwd"), ...) .External.graphics(C_abline,
##         a, b, h, v, untf, col, lty, lwd, ...)
##     if (!is.null(reg)) {
##         if (!is.null(a))
##             warning("'a' is overridden by 'reg'")
##         a <- reg
##     }
##     if (is.object(a) || is.list(a)) {
##         p <- length(coefa <- as.vector(coef(a)))
##         if (p > 2)
##             warning(gettextf("only using the first two of %d regression coefficients",
##                 p), domain = NA)
##         islm <- inherits(a, "lm")
##         noInt <- if (islm)
##             !as.logical(attr(stats::terms(a), "intercept"))
##         else p == 1
##         if (noInt) {
##             a <- 0
##             b <- coefa[1L]
##         }
##         else {
##             a <- coefa[1L]
##             b <- if (p >= 2)
##                 coefa[2L]
##             else 0
##         }
##     }
##     if (!is.null(coef)) {
##         if (!is.null(a))
##             warning("'a' and 'b' are overridden by 'coef'")
##         a <- coef[1L]
##         b <- coef[2L]
##     }
##     int_abline(a = a, b = b, h = h, v = v, untf = untf, ...)
##     invisible()
## }
## <bytecode: 0x7fdf9bd068b0>
## <environment: namespace:graphics>
```

If I had more computational power and more time, I would scan more than three mtry-values. For now, I will build my final model with mtry=30 since it provided the best accuracy.

Note that cross-validation can also provide a good estimate of the out of sample error.

```r
model_rf$resample
```

```
##    Accuracy     Kappa Resample
## 1 0.9974529 0.9967783    Fold1
## 2 0.9979613 0.9974213    Fold3
## 3 0.9992355 0.9990330    Fold2
```

```
## 4 0.9987258 0.9983882    Fold5
## 5 0.9982161 0.9977437    Fold4
```

The accuracy in each fold is very high. Less than 1% of the observations were incorrectly classified. I assume that the model will perform similarly well on similar data.

Let's predict classe for the 20 test observations.

```
predictions <- predict(model_rf, newdata = testing[,-56])
```

Finally, let's save the predictions in an appropriate format.

```
results <- data.frame(problem_id = testing$problem_id, predictions = predictions)
write.csv(results, "Predictions_ArneSchoch.csv")
```