Predicting the manner of exercise using HAR dataset

Nikhil Parimi

2024-12-15

Summary

Using a dataset provided by HAR http://web.archive.org/web/20161224072740/http:/groupware.les.inf.puc-rio.br/har we aim to build a predictive model(s) to accurately predict the “classe” variable.

This will be done as follows: - Processing the data - Data cleaning - Exploratory data analysis - Model selection - Make predictions

```
library(caret)

## Loading required package: ggplot2

## Loading required package: lattice

library(dplyr)

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

library(corrplot)

## corrplot 0.95 loaded
```

Processing data

Load the data from the data directory

```
training_raw <- read.csv("data/pml-training.csv")
testing_raw <- read.csv("data/pml-testing.csv")
dim(training_raw)

## [1] 19622   160
```

```r
dim(testing_raw)

## [1]  20 160
```

Here we can see our training data has 160 columns and 19622 rows, by exploring the data let's see how we can clean the data further.

```r
#head(training_raw, n = 2)
#summary(training_raw)
```

Data Cleaning

First we shall check which columns have NA values and see how many NA values each contain.

```r
na_cols <- colSums(is.na(training_raw)) != 0
only_na_cols <- colSums(is.na(training_raw[,na_cols]))
summary(only_na_cols)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   19216   19216   19216   19216   19216   19216
```

Here we can see that of the columns that contain NA values, all of them contain **19216** NA values out of **19622** total observations. That means **97.9%** of these columns are filled with NA values, this would be redundant to include in the model so these columns will be removed from both the training and testing set.

```r
training_1 <- training_raw[,!na_cols]
testing_1 <- testing_raw[,!na_cols]
```

Of the remaining columns we shall check for empty values.

```r
empty_cols <- colSums(training_1=="") != 0
only_empty_cols <- colSums(training_1[,empty_cols]=="")
summary(only_empty_cols)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   19216   19216   19216   19216   19216   19216
```

Here we measure a similar percentage of values (**97.9%**) of observations in a column are missing, so we can safely remove these columns as well.

```r
training_2 <- training_1[,!empty_cols]
testing_2 <- testing_1[,!empty_cols]
```

After viewing the data in a bit more detail we notice that the first 7 columns are metadata that is irrelevant to the outcome and so will be removed.

```r
training_3<- training_2[,-c(1:7)]
testing_3<- testing_2[,-c(1:7)]
```

Finally we will check for any columns with near zero value variances.

```r
nzv <- nearZeroVar(training_3, saveMetrics = TRUE)
nzv$nzv

##  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE
## [13] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE
## [25] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE
## [37] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE
## [49] FALSE FALSE FALSE FALSE FALSE
```

Here we can see that none of the columns have near zero / zero variance, so we shall use training_3 and testing 3 as our cleaned data after converting the "classe" variable to a factor.

```r
training_3$classe <- as.factor(training_3$classe)
training_cleaned <- training_3
testing_cleaned <- testing_3
dim(training_cleaned)

## [1] 19622    53

dim(testing_cleaned)

## [1] 20 53
```

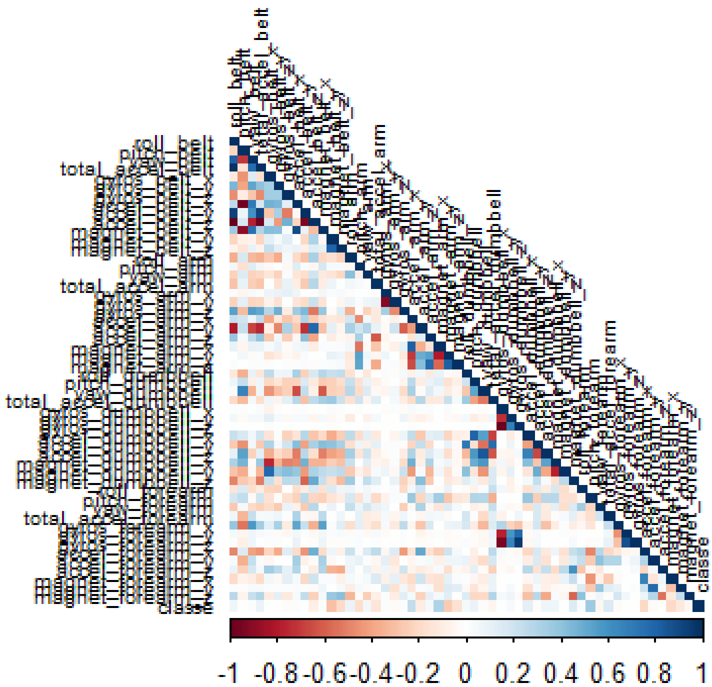
Exploratory data analysis

We shall now split the data into a training set and a validation set due to the testing set being the ultimate goal of what we would like to predict.

```r
set.seed(50)

inTrain <- createDataPartition(training_cleaned$classe, p=0.7, list=FALSE)
training_set <- training_cleaned[inTrain,]
validation_set <- training_cleaned[-inTrain,]
classe_index <- length(names(training_set))
```

Testing and plotting the correlation between variables including the predictor variable "classe"

```r
temp_set <- training_set
temp_set$classe <- as.numeric(temp_set$classe)
correlations <- cor(temp_set)

corrplot(correlations, method = "color", type = "lower", tl.cex = 0.6,
tl.col="black")
```



In the bottom row we can see that none of the variables have a **strong** correlation with the predictor “classe” variable (as evidenced by the softer colors), whilst there appears to be stronger correlations between some variables. We can see this in code below:

```
# Correlations contains the information in the matrix above, it suffices to
simply look at the final row of the column to analyse the correlations
between parameters.
dim(correlations)

## [1] 53 53

# Taking the 53rd row and excluding the last column
corr_with_classe <- correlations[53,]
corr_with_classe <- corr_with_classe[-53]
ordered_indices <- order(abs(corr_with_classe), decreasing = TRUE)
ordered_corr <- corr_with_classe[ordered_indices]
head(ordered_corr, n = 5)

## pitch_forearm  magnet_arm_x magnet_belt_y  magnet_arm_y   accel_arm_x
##     0.3562543     0.3048029    -0.2904076    -0.2596358     0.2481750
```

The variable with the strongest correlation to “classe” appears to be “pitch_forearm” which only has a magnitude of 0.356, which is quite poor relationship between the two variables.

Model Selection

The 3 models I will choose to train are Random Forest, Gradient Boosted Trees and Support Vector Machines. Between these 3 models, I will train each of the models using k-fold cross validation, where k = 3 and also perform pre-processing of PCA, with a threshold of 0.99 (Keep 99% of the variance). We will also set tuneLength = 3 or 5 for the models, which will also tune the parameters for each model between an evenly spaced range of values.

```
# Set up cross-validation control with k=3 folds and PCA pre-processing
control <- trainControl(
  method = "cv",
  number = 3,
  preProcOptions = list(thresh = 0.99)
)
```

Model 1: Random Forest

```
fit_rf <- train(classe ~ .,
                data = training_set,
                method = "rf",
                trControl = control,
                preProcess="pca",
                tuneLength = 3)
# Predictions on validation set and training set
pred_rf <- predict(fit_rf, validation_set)
pred_train_rf <- predict(fit_rf, training_set)

#Confusion matrices for validation set and training set
cm_rf <- confusionMatrix(pred_rf, validation_set$classe)
cm_train_rf <- confusionMatrix(pred_train_rf, training_set$classe)
cm_rf

## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1671   14    1    0    0
##          B    2 1115   11    0    1
##          C    0    7 1008   30    9
##          D    1    1    6  929    1
##          E    0    2    0    5 1071
##
## Overall Statistics
##
##                Accuracy : 0.9845
##                  95% CI : (0.981, 0.9875)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
```

```
##                   Kappa : 0.9804
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9982   0.9789   0.9825   0.9637   0.9898
## Specificity            0.9964   0.9971   0.9905   0.9982   0.9985
## Pos Pred Value         0.9911   0.9876   0.9564   0.9904   0.9935
## Neg Pred Value         0.9993   0.9950   0.9963   0.9929   0.9977
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2839   0.1895   0.1713   0.1579   0.1820
## Detection Prevalence   0.2865   0.1918   0.1791   0.1594   0.1832
## Balanced Accuracy      0.9973   0.9880   0.9865   0.9809   0.9942
```

Model 2: Gradient Boosted Trees

```
fit_gbm <- train(classe ~ .,
                data = training_set,
                method = "gbm",
                trControl = control,
                preProcess="pca",
                tuneLength = 5,
                verbose = FALSE)

# Predictions on validation set and training set
pred_gbm <- predict(fit_gbm, validation_set)
pred_train_gbm <- predict(fit_gbm, training_set)

#Confusion matrices for validation set and training set
cm_gbm <- confusionMatrix(pred_gbm, validation_set$classe)
cm_train_gbm <- confusionMatrix(pred_train_gbm, training_set$classe)
cm_gbm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1630   54   11   14    8
##          B   21 1022   34   10   19
##          C    7   42  955   56   30
##          D   15    9   20  876   16
##          E    1   12    6    8 1009
##
## Overall Statistics
##
##                Accuracy : 0.9332
##                  95% CI : (0.9265, 0.9395)
##     No Information Rate : 0.2845
```

```
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9155
##
##  Mcnemar's Test P-Value : 5.403e-09
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9737   0.8973   0.9308   0.9087   0.9325
## Specificity            0.9793   0.9823   0.9722   0.9878   0.9944
## Pos Pred Value         0.9493   0.9241   0.8761   0.9359   0.9739
## Neg Pred Value         0.9894   0.9755   0.9852   0.9822   0.9849
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2770   0.1737   0.1623   0.1489   0.1715
## Detection Prevalence   0.2918   0.1879   0.1852   0.1590   0.1760
## Balanced Accuracy      0.9765   0.9398   0.9515   0.9483   0.9635
```

Model 3: Support Vector Machines

```
fit_svm <- train(classe ~ .,
                data = training_set,
                method = "svmLinear",
                trControl = control,
                preProcess="pca",
                tuneLength = 5,
                verbose = FALSE)
# Predictions on validation set and training set
pred_svm <- predict(fit_svm, validation_set)
pred_train_svm <- predict(fit_svm, training_set)

#Confusion matrices for validation set and training set
cm_svm <- confusionMatrix(pred_svm, validation_set$classe)
cm_train_svm <- confusionMatrix(pred_train_svm, training_set$classe)
cm_svm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1338  193  171  103  112
##          B  103  736   85   87  149
##          C  117   69  703  116   93
##          D   95   47   39  597  127
##          E   21   94   28   61  601
##
## Overall Statistics
##
##                Accuracy : 0.6754
##                  95% CI : (0.6633, 0.6874)
```

```
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5873
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.7993   0.6462   0.6852   0.6193   0.5555
## Specificity            0.8625   0.9107   0.9187   0.9374   0.9575
## Pos Pred Value         0.6980   0.6345   0.6403   0.6597   0.7466
## Neg Pred Value         0.9153   0.9147   0.9325   0.9263   0.9053
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2274   0.1251   0.1195   0.1014   0.1021
## Detection Prevalence   0.3257   0.1971   0.1866   0.1538   0.1368
## Balanced Accuracy      0.8309   0.7784   0.8019   0.7784   0.7565
```

Comparing the models:

Printed below is a summary of the accuracy of the 3 models on the training set and the validation (test) set as well as the out of sample error.

```
models <- c("RF", "GBM", "SVM")
train_acc <- round(c(cm_train_rf$overall[1], cm_train_gbm$overall[1],
cm_train_svm$overall[1]),3)
test_acc <- round(c(cm_rf$overall[1], cm_gbm$overall[1],
cm_svm$overall[1]),3)
oos_error <- 1 - test_acc

data.frame(train_acc = train_acc, test_acc = test_acc, oos_error = oos_error,
row.names = models)
```

```
##     train_acc test_acc oos_error
## RF      1.000    0.985     0.015
## GBM     0.981    0.933     0.067
## SVM     0.688    0.675     0.325
```

From the results above the best model is the Random Forest model with a test accuracy of 0.985 and an out of sample error rate of 0.015. Although the training accuracy of 1.0 may be cause for concern as it may imply over fitting, the model generalises well to data it hasn’t seen before, as evidenced by the validation (test) set accuracy rate.

Predictions

In the code block below, we will run our Random Forest model on the test set (the set with unknown “classe” variable) to see the predictions we get.

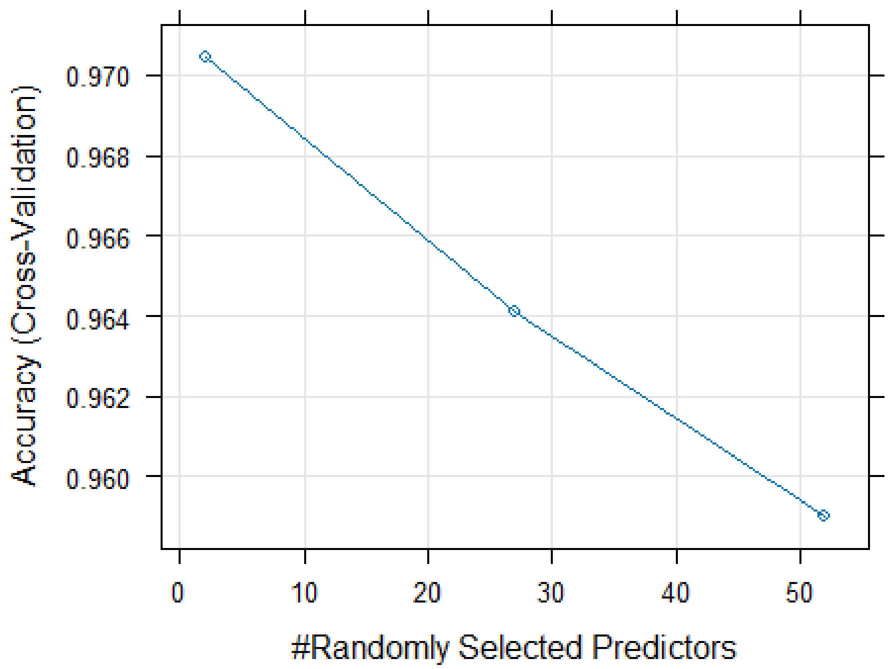
```r
# Exclude the final column (problem_id) from the testing_set
pred_final <- predict(fit_rf, testing_cleaned[,-
length(names(testing_cleaned))])
pred_final

##  [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
```

Appendix

Plotting the models selected using cross validation (k=3) and tuneLength (3 / 5).

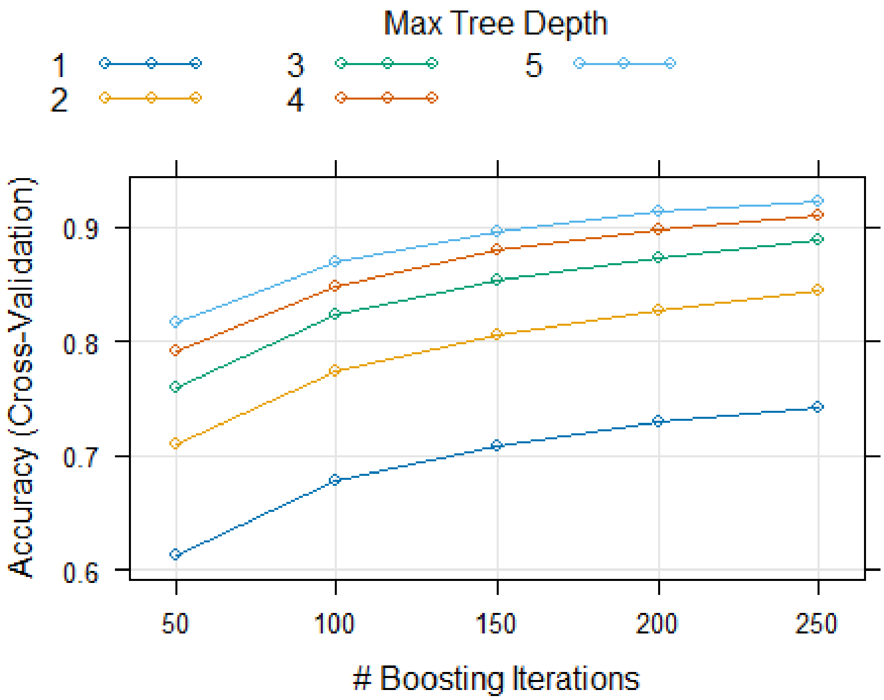
1.Random Forest tuning

```r
plot(fit_rf)
```



2.Gradient Boosted Trees tuning

```r
plot(fit_gbm)
```



Max Tree Depth
1
2
3
4
5
Accuracy (Cross-Validation)
0.6
0.7
0.8
0.9
50
100
150
200
250
Boosting Iterations