Automatically Predicting the correctness of Weight-Lifting Techniques

d1m2r3

Sunday, December 21, 2014

Introduction

This document describes how a machine learning algorithm can be developed to automatically assess weight-lifting techniques. This is part of the project of the course “Practical Machine Learning” of Coursera.

Main technical reference

This work is based on the following technical paper, which is cited in the project description by the course:

E. Velloso et al, “Qualitative Activity Recognition of Weight Lifting Exercises”, Proceedings of the 4th International Conference in Cooperation with SIGCHI (Augmented Human ’13), Stuttgart, Germany; ACM SIGCHI 2013. Available at: http:// groupware.les.inf.pub-rio.br/work.jsf?p1=11201

Steps of the analysis

0. Preparation In RStudio, read the data files, and load the necessary machine learning libraries (such as, the caret package)
1. [Create tidy data set] Create a tidy training data set for machine learning, using hints from the technical reference
2. [Train the model] Select a learning method, and fit a model; assess its accuracy on training data.
3. [Predict on test data] Apply the model on training data and evaluate its accuracy.

Preparation

We read the necessary data files into data frames and load the necessary packages, such as ‘caret’.

```
library(caret)

#Files names of the training and test data sets
f_train <- "./pml-training.csv"

f_test <- "./pml-testing.csv"

#Read them in to data frames

df_train <- read.csv(f_train)
dim(df_train); length(names(df_train))
```

```
## [1] 19622   160
```

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

```
df_test <- read.csv(f_test)
dim(df_test); length(names(df_test))
```

```
## [1]  20 160
```

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

We see that the training data is of dimension 19622, 160. The test data is of dimension 20, 160.

Creating a tidy data set

We examine the data and tidy it up with these steps

1. Retain only numeric data columns

```
#Get only the numeric columns

tr_numeric <- sapply(df_train, is.numeric)

#Get the names of these columns

nmC <- names(which(tr_numeric))
```

There are 123 columns that have numeric data.

2. Select only those columns indicated by the technical reference

The paper by Velloso says which columns are relevant to the analysis.

Quoting from the paper, Sec. 5.1. "Feature extraction and selection: *The algorithm was con gured to use a "Best First" strategy based on backtracking. 17 features were selected: in the belt, were selected the mean and variance of the roll, maximum, range and variance of the accelerometer vector, variance of the gyro and variance of the magnetometer. In the arm, the variance of the accelerometer vector and the maximum and minimum of the magnetometer were selected. In the dumbbell, the selected features were the maximum of the acceleration, variance of the gyro and maximum and minimum of the magnetometer, while in the glove, the sum of the pitch and the maximum and minimum of the gyro were selected.*

So, subset the data set to the relevant columns. This is done by simple selection (using grep()) on the column names

1. Belt

In the belt, we selected the a) mean and variance of the roll, b) maximum, range and variance of the accelerometer vector, c) variance of the gyro and variance of the magnetometer.

```
inxBelt <- grep("belt", nmC, ignore.case = TRUE)
inxRoll <- grep("roll", nmC, ignore.case = TRUE)
inxAccel <- grep("accel", nmC, ignore.case = TRUE)
inxGyro  <- grep("gyro", nmC,ignore.case= TRUE)
inxMag <-  grep("magn", nmC,ignore.case= TRUE)
inx_1 <- union( intersect(inxBelt, inxRoll),
```

```
                intersect(inxBelt, inxAccel))
inx_1 <- union(inx_1, intersect(inxBelt, inxGyro))
inx_1 <- union(inx_1, intersect(inxBelt, inxMag))
beltNames <- nmC[inx_1]
```

The are 18 columns from the belt, which are

roll_belt, max_roll_belt, min_roll_belt, amplitude_roll_belt, avg_roll_belt, stddev_roll_belt, var_roll_belt, total_accel_belt, var_total_accel_belt, accel_belt_x, accel_belt_y, accel_belt_z, gyros_belt_x, gyros_belt_y, gyros_belt_z, magnet_belt_x, magnet_belt_y, magnet_belt_z.

2. ARM

In the arm, a) the variance of the accelerometer vector b) maximum and minimum of the magnetometer were selected.

```
inxArm <- grep("_arm", nmC,ignore.case= TRUE)
inxVar <- grep("var", nmC, ignore.case = TRUE)
inx_2 <- intersect(inxArm, inxAccel); inx_2 <- intersect(inx_2, inxVar)
inx_2 <- union(inx_2, intersect(inxArm, inxMag))
armNames <- nmC[inx_2]
```

In the arm, we selected 4, which were

var_accel_arm, magnet_arm_x, magnet_arm_y, magnet_arm_z.

3. Dumbbell

In the dumbbell, the selected features were the
a) maximum ofthe acceleration, b) variance of the gyro c) and maximum and minimum of the magnetometer.

```
inxDumbbell <- grep("dumbbell", nmC, ignore.case = TRUE)
inx_3 <- intersect(inxDumbbell, inxAccel);
inx_3 <- union(inx_3, intersect(intersect(inxDumbbell, inxGyro), inxVar))
inx_3 <- union(inx_3, intersect(inxDumbbell,inxMag))
dumbbellNames <- nmC[inx_3]
```

The were 8 columns from dumbbell, which were

total_accel_dumbbell, var_accel_dumbbell, accel_dumbbell_x, accel_dumbbell_y, accel_dumbbell_z, magnet_dumbbell_x, magnet_dumbbell_y, magnet_dumbbell_z.

4. Glove

In the glove, the sum of the pitch and the maximum and minimum of the gyro were selected. (glove = forearm)

```
inxGlove <- grep("forearm", nmC, ignore.case = TRUE)
inxPitch <- grep("pit", nmC, ignore.case = TRUE)
inx_4 <- intersect(inxGlove, inxPitch)
inx_4 <- union( inx_4, intersect(inxGlove, inxGyro))
gloveNames <- nmC[inx_4];
```

This gave 9 columns, which were

pitch_forearm, min_pitch_forearm, amplitude_pitch_forearm, avg_pitch_forearm, stddev_pitch_forearm, var_pitch_forearm, gyros_forearm_x, gyros_forearm_y, gyros_forearm_z.

5. Retain only columns that contain data

First, we take the union of all relevant columns, and subset the training data frame to this selected list.

```
#Get the union of all the relevant indexes

inxAll <- union(inx_1, union(inx_2, union(inx_3, inx_4)))
selNames <- nmC[inxAll]
df_cleanTrain <- df_train[,selNames]
```

We see that this is of size 19622, 39.

Now, we bind the “classe” column, and remove all empty columns.

```
df_cleanTrain <- cbind(df_cleanTrain, df_train$classe)
names(df_cleanTrain)[length(names(df_cleanTrain))] <- "classe"

#Remove all columns with NA, using row 1 as reference
r1 <- df_cleanTrain[1,]
naData  <- sapply(r1, is.na)

#Apply this to the data frame
df_train2 <- df_cleanTrain[,!naData]
```

We end up with a tidy training data set of dimension 19622, 26.

Training the model

Making a smaller training set

Since the training data has large number of rows (nearly 20,000), we reduce its size to about 4000 rows by creating a data partition

```
inx3 <- createDataPartition( y = df_train2$classe,
                             p = 0.2,
                            list = FALSE)
df_train3 <- df_train2[inx3,]
```

This has dimensions of 3927, 26.

Verify that this has no NA’s (all entries are valid).

```
table(is.na(df_train3))
```

```
##
##  FALSE
## 102102
```

Fitting the random forest model

The technical paper mentions that the selected model is random forest. So, we use the train() method from caret package, with appropriate preProcess(), to fit a prediction model.

```
fitControl <- trainControl(method = "repeatedcv",
                           number = 10,
                           repeats = 10)

pFit_rf <- train(classe ~ .,
                 data = df_train3,
                 method = "rf",
                 preprocess = c("center", "scale"),
                 trControl = fitControl
                 )
```

```
## Loading required package: randomForest
## randomForest 4.6-10
## Type rfNews() to see new features/changes/bug fixes.
```

```
print(pFit_rf)
```

```
## Random Forest
##
## 3927 samples
##   25 predictor
##    5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 10 times)
##
## Summary of sample sizes: 3535, 3534, 3535, 3535, 3532, 3534, ...
##
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa      Accuracy SD  Kappa SD
##    2    0.9378948  0.9213405  0.01155291   0.01463827
##   13    0.9451247  0.9305400  0.01091419   0.01382661
##   25    0.9376610  0.9210868  0.01138747   0.01442492
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 13.
```

Out of sample accuracy

Since the model chosen had mtry value of 13, with accuracy about 93%, and standard deviation of 13%, we expect the out of sample accuracy of at least 93 - 2*13 = 67%. (worst case of two standard deviations less than mean accuracy).

Predicting on the test data

We clean up the test data to the relevant columns, and run the predict() method. We subset the test data to relevant columns, and remove the last column with the “classe” data.

```
df_cleanTest <- df_test[,selNames]; dim(df_cleanTest)
```

```
## [1] 20 39
```

```
df_test2 <- df_cleanTest[,!naData[-length(naData)]]
dim(df_test2)
```

```
## [1] 20 25
```

```
pred_rf <- predict(pFit_rf, df_test2)

pred_rf
```

```
##  [1] B A A A A E D D A A B C B A E E A B B B
## Levels: A B C D E
```

This gave the predicted classes of

[1] B A B A A E D D A A B C B A E E A B B B On the test data, the accuracy of the model was 95%.