Practical Machine Learning: Weight-Training Technique Evaluation

1. Background

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

Literature

E. Velloso, A. Bulling, H. Gellersen, W. Ugulino, H. Fuks. 2013. Qualitative activity recognition of weight lifting exercises. In **Proceedings of the 4th Augmented Human International Conference (AH '13)**. ACM, New York, NY, USA, 116-123.

2. Data

Training data: https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv

Assignment test data: https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv

The data for this project was downloaded from: http://groupware.les.inf.puc-rio.br/har

Data Transformations

From the training_set data and the testing_set data:

- remove first eight columns, containing experimental details; specifically: X, user_name, raw_timestamp_part_1, raw_timestamp_part_2, cvtd_timestamp, new_window, num_window
- remove variables that consist of a high number of missing values

From the assignment testing case data:

- remove first eight columns, containing experimental details; specifically: X, user_name, raw_timestamp_part_1, raw_timestamp_part_2, cvtd_timestamp, new_window, num_window
- remove the variable problem_id

Features

We used the following sensor information for our learning algorithm. We deliberately avoidede aggregate data, such as variance, maximum/minimum, range, etc. to provide raw 3-dimensional movement and velocity information.

Belt sensors:

- roll_belt, pitch_belt, yaw_belt
- gyros_belt_x, gyros_belt_y, gyros_belt_z
- accel_belt_x, accel_belt_y, accel_belt_z, total_accel_belt
- magnet_belt_x, magnet_belt_y, magnet_belt_z

Arm sensors:

- roll_arm, pitch_arm, yaw_arm
- gyros_arm_x, gyros_arm_y, gyros_arm_z
- accel_arm_x, accel_arm_y, accel_arm_z, total_accel_arm
- magnet_arm_x, "magnet_arm_y", "magnet_arm_z

Dumbbell sensors:

- roll_dumbbell, pitch_dumbbell, yaw_dumbbell
- gyros_dumbbell_x, gyros_dumbbell_y, gyros_dumbbell_z
- accel_dumbbell_x, accel_dumbbell_y, accel_dumbbell_z, total_accel_dumbbell
- magnet_dumbbell_x, magnet_dumbbell_y, magnet_dumbbell_z

Forearm sensors:

- roll_forearm, pitch_forearm, yaw_forearm
- gyros_forearm_x, gyros_forearm_y, gyros_forearm_z
- accel_forearm_x, accel_forearm_y, accel_forearm_z, total_accel_forearm
- magnet_forearm_x, magnet_forearm_y, magnet_forearm_z

3. Setting Up the Environment

```
library( caret )
```

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

```
library( randomForest )
```

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

Ensure reproducibility by setting the seed for the randon number generator, and the path to the input and output files

```
set.seed( 1234 )
setwd( "~/Desktop/Courses/Practical Machine Learning/Project" )
```

4. Input Training/Test Data and Preprocessing/Cleaning

```
training_df <- read.csv( "pml-training.csv", header = TRUE, sep = ",", na.strings=c( "NA", "#DIV/0!", "
testing_df <- read.csv( "pml-testing.csv", header = TRUE, sep = ",", na.strings=c( "NA", "#DIV/0!", "" ]
# Discard user, time/date stamp information, etc.
drops <- c( "X", "user_name", "raw_timestamp_part_1", "raw_timestamp_part_2", "cvtd_timestamp", "new_wi
training <- training_df[ , !( names( training_df ) %in% drops ) ]
# Perform the same transformation to the official test case date set:
testing <- testing_df[ , !( names( testing_df ) %in% drops ) ]
```

Training and Testing Sets

Generate training and test data sets, splitting 60% and 40% respectively:

```
train_vector <- createDataPartition( y=training$classe, p=0.6, list=FALSE )
training_set <- training[ train_vector, ]
testing_set <- training[ -train_vector, ]
```

Feature Selection

We reduce the data sets to exactly the significant variables. Our strategy will be to use only the raw sensor data that reflects movement and direction, and so eliminate all variables that make use of aggregations of information, such as the variance, maximum/minimum, range, etc.

```
features_list <- names( training_set ) %in% c(
        "roll_belt", "pitch_belt", "yaw_belt",
        "gyros_belt_x", "gyros_belt_y", "gyros_belt_z",
        "accel_belt_x", "accel_belt_y", "accel_belt_z", "total_accel_belt",
        "magnet_belt_x", "magnet_belt_y", "magnet_belt_z",

        "roll_arm", "pitch_arm", "yaw_arm",
        "gyros_arm_x", "gyros_arm_y", "gyros_arm_z",
        "accel_arm_x", "accel_arm_y", "accel_arm_z",  "total_accel_arm",
        "magnet_arm_x", "magnet_arm_y", "magnet_arm_z",

        "roll_dumbbell", "pitch_dumbbell", "yaw_dumbbell",
        "gyros_dumbbell_x", "gyros_dumbbell_y", "gyros_dumbbell_z",
        "accel_dumbbell_x", "accel_dumbbell_y", "accel_dumbbell_z", "total_accel_dumbbell",
        "magnet_dumbbell_x", "magnet_dumbbell_y", "magnet_dumbbell_z",

        "roll_forearm", "pitch_forearm", "yaw_forearm",
        "gyros_forearm_x", "gyros_forearm_y", "gyros_forearm_z",
        "accel_forearm_x", "accel_forearm_y", "accel_forearm_z", "total_accel_forearm",
        "magnet_forearm_x", "magnet_forearm_y", "magnet_forearm_z",

        "classe" )
```

```
training_set <- training_set[ features_list ]
testing_set <- testing_set[ features_list ]
```

Perform a similar transformation on the final testing data as well:

```
# Note: the value for the last variable in the Boolean-vector is true, coinciding with the problem_id v
testing <- testing[ features_list ] # Test case data set
testing <- testing[ -length( testing ) ] # Trim of the problem_id
```

We remove any overly-sparse variables, , i.e in this case missing more than 10% of their values:

```
# Simple function to calculate the percentage of NAs observations in a row
sQuotient <- function( r ){ return( ( length( r ) - sum( is.na( r ) ) )/(length( r )) ) }
# Identify the variables that are sparse: creating a sparseness vector and map against dataset row valu
sparseness_quotient <- apply( training_set, 2, sQuotient )
# Remove variables that contribute little to the calculation
training_set <- training_set[ , sparseness_quotient > 0.90 ]
```

5. Model Building and Validation

We build the model based on the random forest learning method, using our selected features, and verify their importance in the model. Random forests are a way of averaging multiple deep decision trees, trained on different parts of the same training set, with the goal of reducing the variance. This comes at the expense of a small increase in the bias and some loss of interpretability, but generally greatly boosts the performance of the final model.

```
model <- randomForest( classe ~ ., data=training_set )
varImp( model, scale = FALSE )
```

```
##                          Overall
## roll_belt              741.65355
## pitch_belt             409.15848
## yaw_belt               552.66632
## total_accel_belt       107.74605
## gyros_belt_x            59.83696
## gyros_belt_y            63.32958
## gyros_belt_z           166.46126
## accel_belt_x            75.23579
## accel_belt_y            78.09609
## accel_belt_z           249.21136
## magnet_belt_x          152.15220
## magnet_belt_y          257.65160
## magnet_belt_z          246.54210
## roll_arm               197.53005
## pitch_arm              104.05554
## yaw_arm                143.59417
## total_accel_arm         62.33294
## gyros_arm_x             77.74852
## gyros_arm_y             79.26595
## gyros_arm_z             36.27296
## accel_arm_x            132.22233
## accel_arm_y           93.93047
## accel_arm_z           81.66408
## magnet_arm_x         153.31611
## magnet_arm_y         139.60997
## magnet_arm_z         111.81588
## roll_dumbbell        249.87949
## pitch_dumbbell       111.81256
## yaw_dumbbell         155.19546
## total_accel_dumbbell 159.93131
## gyros_dumbbell_x      79.09687
## gyros_dumbbell_y     158.12326
## gyros_dumbbell_z      55.94049
## accel_dumbbell_x     142.46626
## accel_dumbbell_y     256.16858
## accel_dumbbell_z     198.38738
## magnet_dumbbell_x    303.42982
## magnet_dumbbell_y    413.51985
## magnet_dumbbell_z    438.74107
## roll_forearm         354.85998
## pitch_forearm        472.62284
## yaw_forearm          101.49043
## total_accel_forearm   62.75164
## gyros_forearm_x       48.63577
## gyros_forearm_y       76.00116
## gyros_forearm_z       51.67170
## accel_forearm_x      190.69544
## accel_forearm_y       82.75738
## accel_forearm_z      143.44333
## magnet_forearm_x     123.18468
## magnet_forearm_y     131.24132
## magnet_forearm_z     174.46241
```

Prediction Results and Expectation For Out-of-sample Error

The results show a very high accuracy on the testing set:

```r
predictions <- predict( model, testing_set, type = "class")
confusionMatrix( predictions, testing_set$classe )
```

```
## Loading required namespace: e1071
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2232   10    0    0    0
##          B    0 1503    9    0    0
##          C    0    5 1356   19    2
##          D    0    0    3 1265    3
##          E    0    0    0    2 1437
##
## Overall Statistics
##
```

```
##                Accuracy : 0.9932
##                  95% CI : (0.9912, 0.9949)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9915
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9901   0.9912   0.9837   0.9965
## Specificity            0.9982   0.9986   0.9960   0.9991   0.9997
## Pos Pred Value         0.9955   0.9940   0.9812   0.9953   0.9986
## Neg Pred Value         1.0000   0.9976   0.9981   0.9968   0.9992
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2845   0.1916   0.1728   0.1612   0.1832
## Detection Prevalence   0.2858   0.1927   0.1761   0.1620   0.1834
## Balanced Accuracy      0.9991   0.9943   0.9936   0.9914   0.9981
```

The accuracy is 0.9932 with a 95% confidence interval of (0.9912, 0.9949). The Kappa statistic is a measure of concordance for categorical data that measures agreement relative to what would be expected by chance. Values of 1 indicate perfect agreement, while a value of zero would indicate a lack of agreement. We see a value very close to one in this case, indicating high accuracy.

6. Perform Evaluation on Assignment Test Case Data

Using the random tree model from the previous section, we now run the assignment cases through our machine learning model, and write the output to individual files for submission:

```
testing_predictions <- predict( model, testing, type = "class" )
for( i in 1:length( testing_predictions ) ){ write.table( testing_predictions[ i ], file=paste0( "probl
```