Predicting Movement Patterns Using Wearable Sensors

January 27, 2016

Executive Summary

In this report I aim to build a model for predicting body movement patterns in performing dumbbell lifting exercises based on data collected from sensors mounted on subjects' body and dumbbell [1, 2]. The data were labelled into five classes: (A) correct Unilateral Dumbbell Biceps Curl movement, (B) mistake by throwing the elbows to the front, (C) mistake by lifting the dumbbell only halfway, (D) mistake by lowering the dumbbell only halfway, and (E) mistake by throwing the hips to the front. I focused on ensemble learning approach, specifically, random forest method. The resulting accuracy in testing dataset prediction was above 99.8%.

Data Exploration and Pre-processing

loading R libraries

```
library(data.table); library(caret); library(kernlab); library(randomForest)
```

loading data

```
d0 <- fread("pml-training.csv", sep=",", header=TRUE, na.strings="NA")
t0 <- fread("pml-testing.csv", sep=",", header=TRUE, na.strings="NA")
dim(d0);dim(t0)
```

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

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

exploring data

I inspected the data types and representative values of each column. There are many columns with large amount of NA. Here is a sampling view to show the first 25 columns.

```
str(d0, list.len=25)
```

```
## Classes 'data.table' and 'data.frame':   19622 obs. of  160 variables:
##  $ V1                      : chr  "1" "2" "3" "4" ...
##  $ user_name               : chr  "carlitos" "carlitos" "carlitos" "carlitos" ...
##  $ 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 484
##  $ cvtd_timestamp          : chr  "05/12/2011 11:23" "05/12/2011 11:23" "05/12/2011 11:23" "05/12/20
##  $ new_window              : chr  "no" "no" "no" "no" ...
##  $ 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     : chr  "" "" "" "" ...
##  $ kurtosis_picth_belt    : chr  "" "" "" "" ...
##  $ kurtosis_yaw_belt      : chr  "" "" "" "" ...
##  $ skewness_roll_belt     : chr  "" "" "" "" ...
##  $ skewness_roll_belt.1   : chr  "" "" "" "" ...
##  $ skewness_yaw_belt      : chr  "" "" "" "" ...
##  $ 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           : chr  "" "" "" "" ...
##  $ 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           : chr  "" "" "" "" ...
##  $ 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 ...
##   [list output truncated]
##  - attr(*, ".internal.selfref")=<externalptr>
```

The detailed descriptions of the about 150 measurements of the sensor data can be found in the paper [3] and will not be repeated here. The percentage of the 5 classes in the training data is shown below.

```
prop.table(table(d0$classe))
```

```
##
##         A         B         C         D         E
## 0.2843747 0.1935073 0.1743961 0.1638977 0.1838243
```

Pre-processing data

First, I removed features with no or very few unique values, using the nearZeroVar method in the caret package. This reduced 60 columns out of the 160.

```
idx_nearZeroCol <- nearZeroVar(d0, saveMetrics=FALSE)
d1 <- subset(d0, select=(-idx_nearZeroCol))
dim(d1)
```

```
## [1] 19622   100
```

Then, I removed features with majority ($> 60\%$) of the rows having NA value. This further reduced 41 columns out of 100.

```
col_lowNA <- apply(!is.na(d1), 2, sum)>nrow(d1)*.6
d2 <- subset(d1, select=(col_lowNA))
dim(d2)
```

```
## [1] 19622    59
```

Finally, I removed the first 5 columns, because they are for record keeping purposes and irrelevant to the sensor measurements. This resulted in final 54 columns, including the target label column "classe".

```
d3 <- subset(d2,select=6:59)
dim(d3)
```

```
## [1] 19622    54
```

I split the data by 75/25 on the "classe" values for training and validation test, respectively.

```
set.seed(37219)
inTrain <- createDataPartition(y=d3$classe, p=0.75, list=FALSE)
training <- d3[inTrain,]
testing <- d3[-inTrain,]
dim(training);dim(testing)
```

```
## [1] 14718    54
```

```
## [1] 4904   54
```

I also converted the data type of "classe" column from char to factor.

```
training$classe <- as.factor(training$classe)
testing$classe <- as.factor(testing$classe)
```

Feature Selection

Because the remaining feature count is only 53, which is not too big, I decided to let model building process automatically choose best features.

Model Selection

According to Kaggle, the machine learning competitions web site, the most successful predictive model for structured data is ensemble trees, including random forest method. Therefore, I decided to try random forest method first.

It is worth noting that in random forests, there is no need for cross-validation to get an unbiased estimate of the test set error. It is estimated internally, during the run. [4]

To avoid long running time, I first built a random forest with 100 trees, instead of the default 500 trees. As shown in the results below, the algorithm randomly selected 7 remaining features at each split to determine best feature for that split. The internal out-of-bag error estimate (similar to cross-validation) in the model building was very low, only 0.28%.

```
fitModel <- randomForest(classe~., data=training, importance=TRUE, ntree=100)
print(fitModel)
```

```
##
## Call:
##  randomForest(formula = classe ~ ., data = training, importance = TRUE,      ntree = 100)
##                Type of random forest: classification
##                      Number of trees: 100
## No. of variables tried at each split: 7
```

```
##
##         OOB estimate of  error rate: 0.28%
## Confusion matrix:
##      A    B    C    D    E  class.error
## A 4182    2    0    0    1 0.0007168459
## B    5 2840    3    0    0 0.0028089888
## C    0    8 2559    0    0 0.0031164784
## D    0    0   17 2393    2 0.0078772803
## E    0    0    0    3 2703 0.0011086475
```

The out of sample error can be estimated by using the model to classify the validation test set. As shown below, the overall accuracy was 0.9984; therefore, the out of sample error was 0.0016, or 0.16%.

```
predictions <- predict(fitModel, testing, type = "class")
confusionMatrix(predictions, testing$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1395    1    0    0    0
##          B    0  947    1    0    0
##          C    0    1  854    4    0
##          D    0    0    0  799    0
##          E    0    0    0    1  901
##
## Overall Statistics
##
##                Accuracy : 0.9984
##                  95% CI : (0.9968, 0.9993)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9979
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9979   0.9988   0.9938   1.0000
## Specificity            0.9997   0.9997   0.9988   1.0000   0.9998
## Pos Pred Value         0.9993   0.9989   0.9942   1.0000   0.9989
## Neg Pred Value         1.0000   0.9995   0.9998   0.9988   1.0000
## Prevalence             0.2845   0.1935   0.1743   0.1639   0.1837
## Detection Rate         0.2845   0.1931   0.1741   0.1629   0.1837
## Detection Prevalence   0.2847   0.1933   0.1752   0.1629   0.1839
## Balanced Accuracy      0.9999   0.9988   0.9988   0.9969   0.9999
```

Because the process above ran very fast, I decided to increase the number of trees to 500 and see how much change will occur. The results below show that the internal out-of-bag error was slightly improved to 0.25%, from 0.28%.

```r
fitModel2 <- randomForest(classe~., data=training)    # Note: ntree = 500 by default
print(fitModel2)
```

```
##
## Call:
##  randomForest(formula = classe ~ ., data = training)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 0.25%
## Confusion matrix:
##      A    B    C    D    E  class.error
## A 4183    2    0    0    0 0.0004778973
## B    4 2841    3    0    0 0.0024578652
## C    0   11 2556    0    0 0.0042851578
## D    0    0   15 2397    0 0.0062189055
## E    0    0    0    2 2704 0.0007390983
```

On the other hand, the overall accuracy in predicting validation test set was slightly reduced to 0.9982, from 0.9984; therefore, out of sample error was slightly increased to 0.18%, from 0.16%.

```r
predictions2 <- predict(fitModel2, testing, type = "class")
confusionMatrix(predictions2, testing$classe)
```

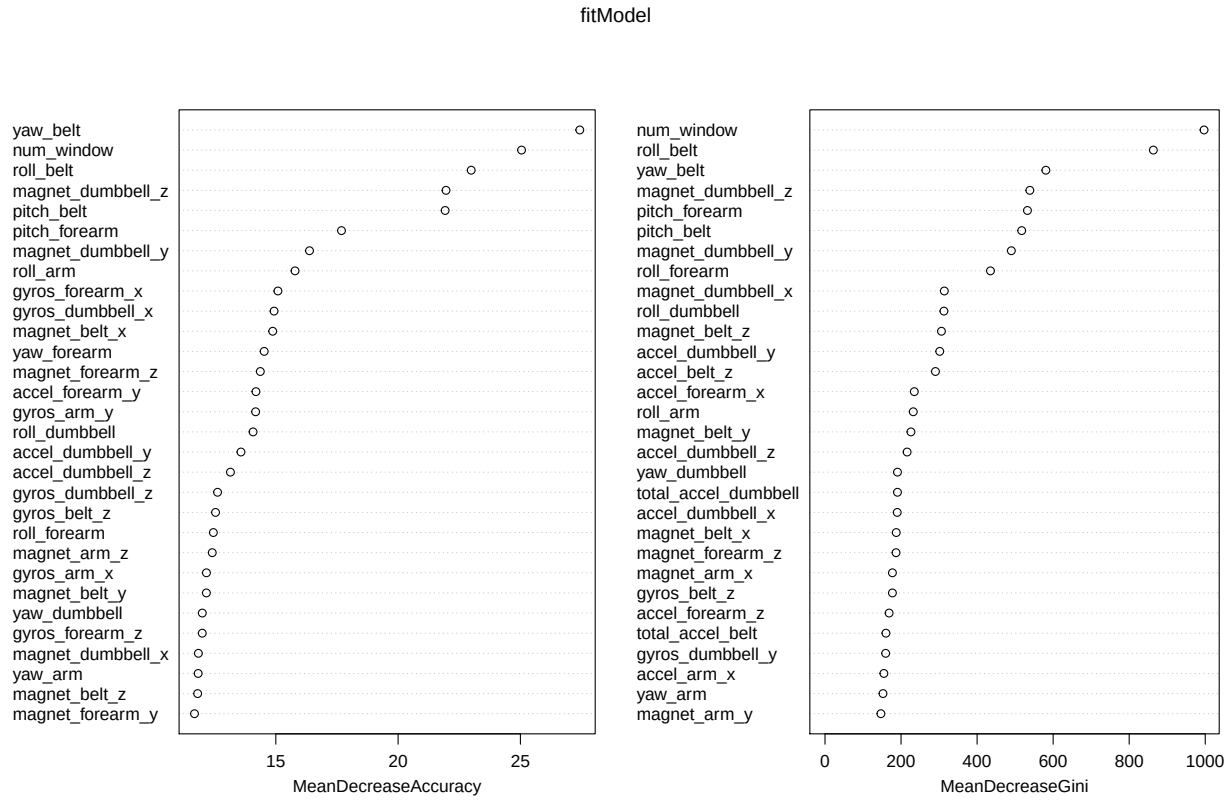
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1395    2    0    0    0
##          B    0  946    1    0    0
##          C    0    1  854    4    0
##          D    0    0    0  799    0
##          E    0    0    0    1  901
##
## Overall Statistics
##
##                Accuracy : 0.9982
##                  95% CI : (0.9965, 0.9992)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9977
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9968   0.9988   0.9938   1.0000
## Specificity            0.9994   0.9997   0.9988   1.0000   0.9998
## Pos Pred Value         0.9986   0.9989   0.9942   1.0000   0.9989
## Neg Pred Value         1.0000   0.9992   0.9998   0.9988   1.0000
```

```
## Prevalence             0.2845   0.1935   0.1743   0.1639   0.1837
## Detection Rate         0.2845   0.1929   0.1741   0.1629   0.1837
## Detection Prevalence   0.2849   0.1931   0.1752   0.1629   0.1839
## Balanced Accuracy      0.9997   0.9983   0.9988   0.9969   0.9999
```

It is clear that 100 trees were as good as 500 trees in the random forests method, given this training data set. Because the accuracy of the random forest models are extremely good (>99.8%), I decided not to try other models at this stage.

We can also use the varImpPlot method in the randomForest package to see the variable importance measures [4]. The order of importance may provide some insight to exercisers or physical trainers on what to look out for in performing dumbbell lifting exercises.

```
varImpPlot(fitModel)
```



Predicting the 20 test cases I used the first model above to predict the 20 test cases.

```
predictionsT <- predict(fitModel, newdata=t0)
predictionsT
```

```
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
##  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
```

References

1. Training Dataset https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv; Testing Dataset https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv

2. Weight Lifting Exercise Dataset at http://groupware.les.inf.puc-rio.br/har
3. Velloso, E.; Bulling, A.; Gellersen, H.; Ugulino, W.; Fuks, H. Qualitative Activity Recognition of Weight Lifting Exercises. Proceedings of 4th International Conference in Cooperation with SIGCHI (Augmented Human '13) . Stuttgart, Germany: ACM SIGCHI, 2013.
4. https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm