Practical Machine Learning - Course Project: Quantification of Exercise Quality

Ajay Ghanti

February 8, 2017

Synopsis

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, our goal is 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 (See References section below for more details about the original research).

The goal of our project is to predict the manner in which they did the exercise. This is the "classe" variable in the training set. At the end, we will use the other variables to build a model that can be used to predict 20 different test cases.

Getting and Cleaning the Data

```
# download the training data
download.file(url = "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv",
              destfile = "./pml-training.csv", method = "curl")

# download the testing data
download.file(url = "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv",
              destfile = "./pml-testing.csv", method = "curl")

# read the data
dTrain <- read.csv("./pml-training.csv", na.strings=c("NA", ""))
dTest <- read.csv("./pml-testing.csv", na.strings=c("NA", ""))

# take a cursory look at the data
# commented out for sake of brevity of the report
#str(data.train)
#summary(data.train)
#sapply(data.train, class)
```

A quick look at the data shows that a lot of the variables are read in as factors, because of an NA string "#DIV/0!". Let us replace this string with NAs.

```
# replace missing/invalid values with NA
dTrain[dTrain == "#DIV/0!"] = NA
dTest[dTest == "#DIV/0!"] = NA

# force convert all the columns with measurement values to numeric
# columns 8 through the last but one
```

```
for (c in (8:ncol(dTrain)-1)) {
    dTrain[, c] = as.numeric(dTrain[, c])
}
for (c in (8:ncol(dTest)-1)) {
    dTest[, c] = as.numeric(dTest[, c])
}
```

The test data set has a lot of columns with NAs. We have only 20 observations in the test data set, so we need to ensure that we are considering only those features which have no NAs.

```
# ignore the features with NAs
featuresToUse <- names(dTest[, colSums(is.na(dTest)) == 0])
# ignore first 7 columns as they are metadata
# and the last column is either classe/problem_id
featuresToUse <- featuresToUse[8:length(featuresToUse)-1]
```

The features that we have decided to use as predictors are as follows:

```
featuresToUse
```

```
##  [1] "num_window"           "roll_belt"            "pitch_belt"
##  [4] "yaw_belt"             "total_accel_belt"     "gyros_belt_x"
##  [7] "gyros_belt_y"         "gyros_belt_z"         "accel_belt_x"
## [10] "accel_belt_y"         "accel_belt_z"         "magnet_belt_x"
## [13] "magnet_belt_y"        "magnet_belt_z"        "roll_arm"
## [16] "pitch_arm"            "yaw_arm"              "total_accel_arm"
## [19] "gyros_arm_x"          "gyros_arm_y"          "gyros_arm_z"
## [22] "accel_arm_x"          "accel_arm_y"          "accel_arm_z"
## [25] "magnet_arm_x"         "magnet_arm_y"         "magnet_arm_z"
## [28] "roll_dumbbell"        "pitch_dumbbell"       "yaw_dumbbell"
## [31] "total_accel_dumbbell" "gyros_dumbbell_x"     "gyros_dumbbell_y"
## [34] "gyros_dumbbell_z"     "accel_dumbbell_x"     "accel_dumbbell_y"
## [37] "accel_dumbbell_z"     "magnet_dumbbell_x"    "magnet_dumbbell_y"
## [40] "magnet_dumbbell_z"    "roll_forearm"         "pitch_forearm"
## [43] "yaw_forearm"          "total_accel_forearm"  "gyros_forearm_x"
## [46] "gyros_forearm_y"      "gyros_forearm_z"      "accel_forearm_x"
## [49] "accel_forearm_y"      "accel_forearm_z"      "magnet_forearm_x"
## [52] "magnet_forearm_y"     "magnet_forearm_z"
```

We can confirm that we have the right list, as none of these features have near zero variance in the training set.

```
nearZeroVar(dTrain[featuresToUse])
```

```
## integer(0)
```

Next, subset the training and test sets with the set of features we are considering, and partition the training set into model training and test sets (split 70:30).

```
set.seed(7777777)
dTrain <- dTrain[c(featuresToUse,"classe")]
dTest <- dTest[c(featuresToUse,"problem_id")]

# partition the training data into model training and test sets
inTrain <- createDataPartition(dTrain$classe, p=0.7, list=FALSE)
modTrain <- dTrain[inTrain,]
modTest <- dTrain[-inTrain,]
```

Evaluating Machine Learning Models

Model Building

Let us start with spot checking some machine learning algorithms provided in the caret package. Cross validation with 5 or 10 folds usually gives a decent trade off of speed vs. generalized error estimate. We shall use *Repeated Cross Validation* with 10 folds and 3 repeats to get a more robust estimate. To improve performance of the machine learning algortihms, we will also enable parallel processing that the caret package supports. We have picked a diverse set of models to evaluate - linear (LDA), trees (CART), and some ensemble methods (Bagged CART, Random Forest, Stochastic Gradient Boosting).

```
cluster <- makeCluster(detectCores() - 1) # convention to leave 1 core for OS
registerDoParallel(cluster)

fit.control <- trainControl(method="repeatedcv", number=10, repeats=3, allowParallel=TRUE)

# Linear Discriminant Analysis
fit.lda <- train(classe ~ ., data=modTrain, method="lda", metric="Accuracy", trControl=fit.control)

# CART
fit.cart <- train(classe ~ ., data=modTrain, method="rpart", metric="Accuracy", trControl=fit.control)

# Bagged CART
fit.treebag <- train(classe ~ ., data=modTrain, method="treebag", metric="Accuracy", trControl=fit.cont

# Random Forest
fit.rf <- train(classe ~ ., data=modTrain, method="rf", metric="Accuracy", trControl=fit.control)

# Stochastic Gradient Boosting (Generalized Boosted Modeling)
fit.gbm <- train(classe ~ ., data=modTrain, method="gbm", metric="Accuracy", trControl=fit.control)
```

```
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.2347
##      2        1.4585             nan     0.1000    0.1702
##      3        1.3519             nan     0.1000    0.1297
##      4        1.2692             nan     0.1000    0.1023
##      5        1.2028             nan     0.1000    0.0995
##      6        1.1412             nan     0.1000    0.0774
##      7        1.0912             nan     0.1000    0.0785
##      8        1.0425             nan     0.1000    0.0676
##      9        1.0007             nan     0.1000    0.0623
##     10        0.9630             nan     0.1000    0.0509
##     20        0.7040             nan     0.1000    0.0335
##     40        0.4518             nan     0.1000    0.0093
##     60        0.3307             nan     0.1000    0.0072
##     80        0.2510             nan     0.1000    0.0028
##    100        0.1915             nan     0.1000    0.0044
##    120        0.1517             nan     0.1000    0.0029
##    140        0.1241             nan     0.1000    0.0023
##    150        0.1119             nan     0.1000    0.0014
```

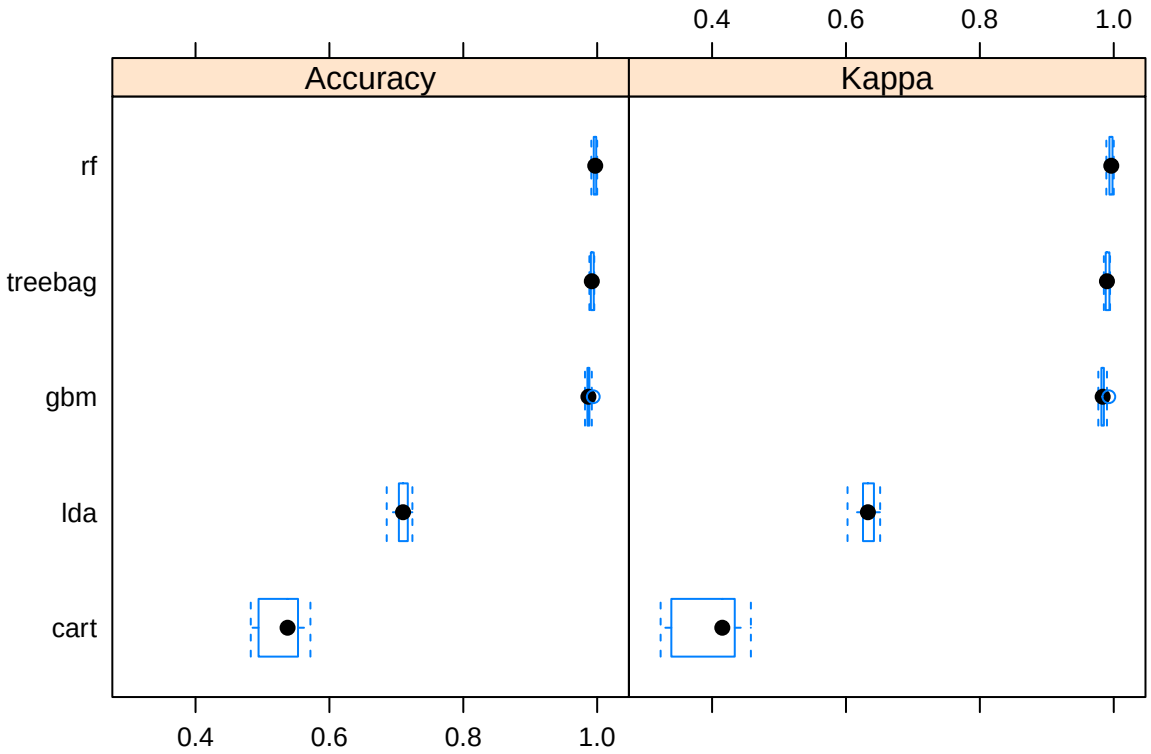
Model Selection

Now that we have trained the above models, let us compare and evaluate them.

```r
results <- resamples(list(lda=fit.lda, cart=fit.cart, treebag=fit.treebag, rf=fit.rf, gbm=fit.gbm))
# table comparison
summary(results)
```

```
##
## Call:
## summary.resamples(object = results)
##
## Models: lda, cart, treebag, rf, gbm
## Number of resamples: 30
##
## Accuracy
##           Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
## lda     0.6856  0.7039 0.7100 0.7102  0.7167 0.7240    0
## cart    0.4825  0.4953 0.5375 0.5267  0.5530 0.5716    0
## treebag 0.9884  0.9905 0.9920 0.9925  0.9947 0.9956    0
## rf      0.9913  0.9951 0.9971 0.9968  0.9985 1.0000    0
## gbm     0.9818  0.9854 0.9869 0.9868  0.9884 0.9942    0
##
## Kappa
##           Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
## lda     0.6023  0.6256 0.6328 0.6334  0.6415 0.6510    0
## cart    0.3231  0.3410 0.4153 0.3913  0.4338 0.4579    0
## treebag 0.9853  0.9880 0.9899 0.9905  0.9933 0.9945    0
## rf      0.9889  0.9938 0.9963 0.9960  0.9982 1.0000    0
## gbm     0.9770  0.9816 0.9834 0.9834  0.9853 0.9926    0
```

```r
# plot the results
bwplot(results)
```



From the above results, as well as the plot, it is evident that the ensemble methods perform way better than

the linear and tree models. The random forest model has the best accuracy (Mean: 0.9976), although with the narrowest range [0.9956 - 1.0]. Bagged CART is the second best (Mean: 0.9958). hence we shall select the random forest model for our further analysis.

Error Estimates

Let us compare the in sample errors for both these models.

```
# get the values predicted by Bagged CART
predModTrainTb <- predict(fit.treebag, newdata=modTrain[featuresToUse])
# get the values predicted by Random Forest
predModTrainRf <- predict(fit.rf, newdata=modTrain[featuresToUse])

# overall statistics: in sample error for Bagged CART
confusionMatrix(predModTrainTb, modTrain$classe)$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.9997088      0.9996317      0.9992546      0.9999207      0.2843416
## AccuracyPValue  McnemarPValue
##      0.0000000            NaN
```

```
# overall statistics: in sample error for Random Forest
confusionMatrix(predModTrainRf, modTrain$classe)$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      1.0000000      1.0000000      0.9997315      1.0000000      0.2843416
## AccuracyPValue  McnemarPValue
##      0.0000000            NaN
```

It can be seen that the in sample errors for both models are similar and unrealistically high. Now let us see the out sample errors, computing them on the test set, that we partitioned earlier from the larger training set.

```
# get the values predicted by Bagged CART
predModTestTb <- predict(fit.treebag, newdata=modTest[featuresToUse])
# get the values predicted by Random Forest
predModTestRf <- predict(fit.rf, newdata=modTest[featuresToUse])

# overall statistics: in sample error for Bagged CART
confusionMatrix(predModTestTb, modTest$classe)$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.9896347      0.9868883      0.9867050      0.9920623      0.2844520
## AccuracyPValue  McnemarPValue
##      0.0000000            NaN
```

```
# overall statistics: in sample error for Random Forest
confusionMatrix(predModTestRf, modTest$classe)$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.9969414      0.9961307      0.9951703      0.9981863      0.2844520
## AccuracyPValue  McnemarPValue
##      0.0000000            NaN
```

Here, the random forest model performs marginally better, with an accuracy of 99.81%. Let us print out the confusion matrix for this model.

```
confusionMatrix(predModTestRf, modTest$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1674    7    0    0    0
##          B    0 1132    1    0    0
##          C    0    0 1025    8    0
##          D    0    0    0  956    2
##          E    0    0    0    0 1080
##
## Overall Statistics
##
##                Accuracy : 0.9969
##                  95% CI : (0.9952, 0.9982)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9961
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9939   0.9990   0.9917   0.9982
## Specificity            0.9983   0.9998   0.9984   0.9996   1.0000
## Pos Pred Value         0.9958   0.9991   0.9923   0.9979   1.0000
## Neg Pred Value         1.0000   0.9985   0.9998   0.9984   0.9996
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2845   0.1924   0.1742   0.1624   0.1835
## Detection Prevalence   0.2856   0.1925   0.1755   0.1628   0.1835
## Balanced Accuracy      0.9992   0.9968   0.9987   0.9956   0.9991
```

Summary

Based on our analysis, we selected the **Random Forest** model, as it has the highest accuracy (99.81%), as well as very high per-class sensitivity and specifivity values. Hence, using this model, we will predict the *classe* variable for the actual test set of 20 observations.

```
# get the values predicted by Random Forest
predictions <- predict(fit.rf, newdata=dTest[featuresToUse])
predictions
```

```
##  [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
```

References

Practical Machine Learning: Required Model Accuracy for Course project

Improving Performance of Random Forest in caret::train()

HAR: Human Activity Recognition

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.