Practical Machine Learning - Course Project

Derek Dixon

4/1/2020

Background, Prompt, & Data Sets

This is the final course project for the Practical Machine Learning course on Coursera as part of the Data Science Specialization by Johns Hopkins University. Course Prompt:

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://web.archive.org/web/20161224072740/http:/groupware.les.inf.puc-rio.br/har (see the section on the Weight Lifting Exercise Dataset).

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

Test Data: https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv

Source: http://web.archive.org/web/20161224072740/http:/groupware.les.inf.puc-rio.br/har

Preperation

Setting seeds and loading datasets.

```r
Sys.info()
```

```
##          sysname          release          version         nodename
##        "Windows"         "10 x64"    "build 18363" "DESKTOP-DP7KPRO"
##          machine            login             user   effective_user
##         "x86-64"          "Derek"          "Derek"          "Derek"
```

```r
set.seed(314)

trainURL <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
testURL <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"

training <- read.csv(url(trainURL), na.strings = c("NA","#DIV/0!",""))
testing <- read.csv(url(testURL), na.strings = c("NA","#DIV/0!",""))
```

Installing packages and loading libraries.

```r
library(h2o)
library(ggplot2)
library(dplyr)
```

Exploring and Tidying the Data

```r
str(training)
```

```
## 'data.frame':    19622 obs. of  160 variables:
##  $ X                       : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ user_name               : Factor w/ 6 levels "adelmo","carlitos",..: 2 2 2 2 2 2 2 2 2 2 ...
##  $ 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 4844
##  $ cvtd_timestamp          : Factor w/ 20 levels "02/12/2011 13:32",..: 9 9 9 9 9 9 9 9 9 9 ...
##  $ new_window              : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 1 ...
##  $ 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      : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ kurtosis_picth_belt     : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ kurtosis_yaw_belt       : logi  NA NA NA NA NA NA ...
##  $ skewness_roll_belt      : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ skewness_roll_belt.1    : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ skewness_yaw_belt       : logi  NA NA NA NA NA NA ...
##  $ 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            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ 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            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ 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 ...
##  $ amplitude_yaw_belt      : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_total_accel_belt    : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_roll_belt           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_roll_belt        : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_roll_belt           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_pitch_belt          : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_pitch_belt       : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_pitch_belt          : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_yaw_belt            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_yaw_belt         : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_yaw_belt            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ gyros_belt_x            : num  0 0.02 0 0.02 0.02 0.02 0.02 0.02 0.02 0.03 ...
##  $ gyros_belt_y            : num  0 0 0 0 0.02 0 0 0 0 0 ...
##  $ gyros_belt_z            : num  -0.02 -0.02 -0.02 -0.03 -0.02 -0.02 -0.02 -0.02 -0.02 0 ...
##  $ accel_belt_x            : int  -21 -22 -20 -22 -21 -21 -22 -22 -20 -21 ...
##  $ accel_belt_y            : int  4 4 5 3 2 4 3 4 2 4 ...
##  $ accel_belt_z            : int  22 22 23 21 24 21 21 21 24 22 ...
```

```
##  $ magnet_belt_x           : int  -3 -7 -2 -6 -6 0 -4 -2 1 -3 ...
##  $ magnet_belt_y           : int  599 608 600 604 600 603 599 603 602 609 ...
##  $ magnet_belt_z           : int  -313 -311 -305 -310 -302 -312 -311 -313 -312 -308 ...
##  $ roll_arm                : num  -128 -128 -128 -128 -128 -128 -128 -128 -128 -128 ...
##  $ pitch_arm               : num  22.5 22.5 22.5 22.1 22.1 22 21.9 21.8 21.7 21.6 ...
##  $ yaw_arm                 : num  -161 -161 -161 -161 -161 -161 -161 -161 -161 -161 ...
##  $ total_accel_arm         : int  34 34 34 34 34 34 34 34 34 34 ...
##  $ var_accel_arm           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_roll_arm            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_roll_arm         : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_roll_arm            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_pitch_arm           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_pitch_arm        : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_pitch_arm           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_yaw_arm             : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_yaw_arm          : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_yaw_arm             : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ gyros_arm_x             : num  0 0.02 0.02 0.02 0 0.02 0 0.02 0.02 0.02 ...
##  $ gyros_arm_y             : num  0 -0.02 -0.02 -0.03 -0.03 -0.03 -0.03 -0.02 -0.03 -0.03 ...
##  $ gyros_arm_z             : num  -0.02 -0.02 -0.02 0.02 0 0 0 0 -0.02 -0.02 ...
##  $ accel_arm_x             : int  -288 -290 -289 -289 -289 -289 -289 -289 -288 -288 ...
##  $ accel_arm_y             : int  109 110 110 111 111 111 111 111 109 110 ...
##  $ accel_arm_z             : int  -123 -125 -126 -123 -123 -122 -125 -124 -122 -124 ...
##  $ magnet_arm_x            : int  -368 -369 -368 -372 -374 -369 -373 -372 -369 -376 ...
##  $ magnet_arm_y            : int  337 337 344 344 337 342 336 338 341 334 ...
##  $ magnet_arm_z            : int  516 513 513 512 506 513 509 510 518 516 ...
##  $ kurtosis_roll_arm       : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ kurtosis_picth_arm      : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ kurtosis_yaw_arm        : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ skewness_roll_arm       : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ skewness_pitch_arm      : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ skewness_yaw_arm        : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_roll_arm            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_picth_arm           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_yaw_arm             : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_roll_arm            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_pitch_arm           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_yaw_arm             : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_roll_arm      : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_pitch_arm     : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_yaw_arm       : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ roll_dumbbell           : num  13.1 13.1 12.9 13.4 13.4 ...
##  $ pitch_dumbbell          : num  -70.5 -70.6 -70.3 -70.4 -70.4 ...
##  $ yaw_dumbbell            : num  -84.9 -84.7 -85.1 -84.9 -84.9 ...
##  $ kurtosis_roll_dumbbell  : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ kurtosis_picth_dumbbell : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ kurtosis_yaw_dumbbell   : logi  NA NA NA NA NA NA ...
##  $ skewness_roll_dumbbell  : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ skewness_pitch_dumbbell : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ skewness_yaw_dumbbell   : logi  NA NA NA NA NA NA ...
##  $ max_roll_dumbbell       : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_picth_dumbbell      : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_yaw_dumbbell        : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_roll_dumbbell       : num  NA NA NA NA NA NA NA NA NA NA ...
```

```
##  $ min_pitch_dumbbell      : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_yaw_dumbbell        : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_roll_dumbbell : num  NA NA NA NA NA NA NA NA NA NA ...
##   [list output truncated]
```

```r
# Removing the first 7 columns because they aren't necessary for prediction
training_clean <- training[,8:length(colnames(training))]
testing_clean <- testing[,8:length(colnames(testing))]

# Removing columns that have 40% or more values as NA
cntlength <- sapply(training_clean, function(x){
  sum(!(is.na(x) | x == ""))
})

nullcol <- names(cntlength[cntlength < 0.6 * length(training_clean$classe)])

training_clean <- training_clean[, !names(training_clean) %in% nullcol]

# Repeating for the test data
cntlength_test <- sapply(testing_clean, function(x){
  sum(!(is.na(x) | x == ""))
})

nullcol_test <- names(cntlength[cntlength < 0.6 * length(testing_clean$classe)])

testing_clean <- testing_clean[, !names(testing_clean) %in% nullcol_test]
```

We can see from the structure of the data that the first 7 columns are not needed, so I remove them from the training and testing data, storing the new sets into "clean" objects. Additionally, I check for and drop all columns that are at least 40% missing values. The 40% is arbitrary.

Using the H2O package for model selection

I want to use the h2o package for training multiple models and identifying the best ones for prediction.

```r
# Initializes the H2O server
h2o.init()

# Converting our R data frames into H2O objects
train_h2o <- as.h2o(training_clean, "train_h2o")
test_h2o <- as.h2o(testing_clean, "test_h2o")

# Training the models using "classe" as the response and all other columns as predictors
aml <- h2o.automl(y = "classe",
                  training_frame = train_h2o,
                  max_models = 10,
                  seed = 1)
```

```r
# Shows the model leaderboard and statistics
lb <- h2o.get_leaderboard(aml, extra_columns = "ALL")
print(lb, n = nrow(lb))
```

```
##                                                model_id mean_per_class_error
## 1                          GBM_5_AutoML_20200402_115600          0.001954465
## 2  StackedEnsemble_BestOfFamily_AutoML_20200402_115600          0.002017401
## 3                          GBM_4_AutoML_20200402_115600          0.002256448
## 4     StackedEnsemble_AllModels_AutoML_20200402_115600          0.002272310
## 5                          GBM_3_AutoML_20200402_115600          0.002592355
## 6                          GBM_1_AutoML_20200402_115600          0.002715612
## 7                          GBM_2_AutoML_20200402_115600          0.002880545
## 8                          XRT_1_AutoML_20200402_115600          0.004431335
## 9                          DRF_1_AutoML_20200402_115600          0.004679122
## 10          GBM_grid__1_AutoML_20200402_115600_model_1          0.005317119
## 11                         GLM_1_AutoML_20200402_115600          0.271622475
## 12                DeepLearning_1_AutoML_20200402_115600          0.550349032
##        logloss       rmse         mse training_time_ms predict_time_per_row_ms
## 1  0.005125982 0.03569953 0.001274457            10234                0.154202
## 2  0.011459182 0.04002946 0.001602358             7312                0.277832
## 3  0.006137920 0.03952191 0.001561981             8095                0.109241
## 4  0.010809415 0.04209687 0.001772146            18150                0.780999
## 5  0.008294948 0.04258968 0.001813881             5541                0.087454
## 6  0.011475857 0.04779760 0.002284610             6283                0.104737
## 7  0.010289096 0.04599035 0.002115112             5446                0.088122
## 8  0.094907234 0.12551627 0.015754334             4175                0.061999
## 9  0.092852668 0.12382231 0.015331965             3577                0.060847
## 10 0.840562352 0.56728298 0.321809978            11307                0.117873
## 11 0.698016576 0.47796839 0.228453784             6996                0.000814
## 12 2.825974408 0.68031060 0.462822511              748                0.002033
##
## [12 rows x 7 columns]
```

```
aml@leader
```

```
## Model Details:
## ==============
##
## H2OMultinomialModel: gbm
## Model ID:  GBM_5_AutoML_20200402_115600
## Model Summary:
##   number_of_trees number_of_internal_trees model_size_in_bytes min_depth
## 1             127                      635              761518        10
##   max_depth mean_depth min_leaves max_leaves mean_leaves
## 1        15   14.97953         36        123    90.63779
##
##
## H2OMultinomialMetrics: gbm
## ** Reported on training data. **
##
## Training Set Metrics:
## =====================
##
## Extract training frame with `h2o.getFrame("automl_training_train_h2o")`
## MSE: (Extract with `h2o.mse`) 3.31836e-07
## RMSE: (Extract with `h2o.rmse`) 0.0005760521
## Logloss: (Extract with `h2o.logloss`) 0.0002109604
## Mean Per-Class Error: 0
```

```
## R^2: (Extract with `h2o.r2`) 0.9999998
## Confusion Matrix: Extract with `h2o.confusionMatrix(<model>,train = TRUE)`)
## =========================================================================
## Confusion Matrix: Row labels: Actual class; Column labels: Predicted class
##           A    B    C    D    E  Error          Rate
## A      5580    0    0    0    0 0.0000 =  0 / 5,580
## B         0 3797    0    0    0 0.0000 =  0 / 3,797
## C         0    0 3422    0    0 0.0000 =  0 / 3,422
## D         0    0    0 3216    0 0.0000 =  0 / 3,216
## E         0    0    0    0 3607 0.0000 =  0 / 3,607
## Totals 5580 3797 3422 3216 3607 0.0000 = 0 / 19,622
##
## Hit Ratio Table: Extract with `h2o.hit_ratio_table(<model>,train = TRUE)`
## =======================================================================
## Top-5 Hit Ratios:
##   k hit_ratio
## 1 1  1.000000
## 2 2  1.000000
## 3 3  1.000000
## 4 4  1.000000
## 5 5  1.000000
##
##
##
## H2OMultinomialMetrics: gbm
## ** Reported on cross-validation data. **
## ** 5-fold cross-validation on training data (Metrics computed for combined holdout predictions) **
##
## Cross-Validation Set Metrics:
## =====================
##
## Extract cross-validation frame with `h2o.getFrame("automl_training_train_h2o")`
## MSE: (Extract with `h2o.mse`) 0.001274457
## RMSE: (Extract with `h2o.rmse`) 0.03569953
## Logloss: (Extract with `h2o.logloss`) 0.005125982
## Mean Per-Class Error: 0.001954465
## R^2: (Extract with `h2o.r2`) 0.9994146
## Hit Ratio Table: Extract with `h2o.hit_ratio_table(<model>,xval = TRUE)`
## =======================================================================
## Top-5 Hit Ratios:
##   k hit_ratio
## 1 1  0.998267
## 2 2  0.999949
## 3 3  1.000000
## 4 4  1.000000
## 5 5  1.000000
##
##
## Cross-Validation Metrics Summary:
##                                 mean           sd   cv_1_valid   cv_2_valid
## accuracy                   0.9982673  4.901056E-4   0.99796176   0.99821657
## err                     0.0017327308  4.901056E-4 0.0020382165 0.0017834395
## err_count                        6.8    1.9235384          8.0          7.0
## logloss                 0.0051259706 5.4778316E-4   0.00531318 0.0051552816
```

```
## max_per_class_error      0.005688895 0.0021471763   0.00777605 0.0039473684
## mean_per_class_accuracy   0.99804574  4.845474E-4   0.99764144   0.99800366
## mean_per_class_error    0.0019542442  4.845474E-4 0.0023585341  0.001996353
## mse                     0.0012744488 1.8327362E-4  0.001466364 0.0012339567
## r2                        0.99941456 8.4176594E-5    0.9993265    0.9994333
## rmse                      0.03562202  0.002626896  0.038293134  0.035127718
##                           cv_3_valid   cv_4_valid   cv_5_valid
## accuracy                   0.9977064   0.99847096   0.99898064
## err                     0.0022935779  0.001529052  0.001019368
## err_count                        9.0          6.0          4.0
## logloss                 0.0056248163 0.0053425864  0.004193989
## max_per_class_error      0.005839416   0.00777605 0.0031055901
## mean_per_class_accuracy    0.9976218    0.9981528    0.9988091
## mean_per_class_error    0.0023782453 0.0018471808 0.0011909081
## mse                     0.0014085377 0.0012679781  9.954075E-4
## r2                         0.9993529    0.9994174    0.9995428
## rmse                      0.03753049   0.03560868   0.03155008
```

The code trains 10 models (the number of models specified), reports the peformance metrics for all on a 5-fold cross-validation of the training data. From the 10 models, it also constructs 2 Stacked Ensemble models, one a best-of-family, and for all models. We can see from the leaderboard that the Stacked Ensemble BestOfFamily model performs the best, having the lowest mean_per_class_error rate on the cross-validation.

We can now use H2O to predict on the test data.

Prediction & Summary

```r
pred <- h2o.predict(aml, test_h2o)
```

```
##   |                                                                              |
```

```r
print(pred, n = nrow(pred))
```

```
##    predict            A            B            C            D            E
## 1        B 1.510986e-04 9.991425e-01 6.777364e-04 1.770453e-05 1.096685e-05
## 2        A 9.998993e-01 9.782421e-05 2.605974e-06 1.618142e-07 1.434018e-07
## 3        B 2.418283e-04 9.989461e-01 5.615012e-04 2.784856e-05 2.227348e-04
## 4        A 9.999510e-01 1.910917e-06 3.225506e-05 1.470849e-05 9.254066e-08
## 5        A 9.999836e-01 1.834349e-06 1.356621e-05 1.081409e-07 9.156071e-07
## 6        E 7.187315e-08 7.556345e-05 1.427270e-04 4.345741e-06 9.997773e-01
## 7        D 3.694419e-06 8.242172e-06 2.021223e-04 9.997733e-01 1.265752e-05
## 8        B 2.682600e-05 9.998000e-01 8.557250e-05 7.408665e-05 1.355662e-05
## 9        A 9.999989e-01 4.103324e-07 1.926508e-07 4.526764e-07 8.259034e-08
## 10       A 9.999910e-01 6.663163e-06 1.133285e-06 8.101864e-07 4.178117e-07
## 11       B 4.023769e-05 9.994962e-01 2.369987e-04 1.605834e-04 6.599297e-05
## 12       C 2.443520e-05 8.825774e-04 9.990375e-01 1.426466e-05 4.121505e-05
## 13       B 5.948694e-06 9.999656e-01 1.158162e-05 5.874123e-06 1.103287e-05
## 14       A 9.999996e-01 1.167559e-07 2.135967e-07 2.574301e-08 5.094793e-08
## 15       E 1.225239e-05 1.355548e-04 2.978642e-06 1.358913e-05 9.998356e-01
## 16       E 3.460797e-05 2.382755e-04 4.223949e-06 1.206785e-05 9.997108e-01
## 17       A 9.999763e-01 3.328696e-06 4.505485e-06 4.024174e-06 1.181721e-05
```

```
## 18       B 1.240781e-05 9.999257e-01 4.758505e-06 3.708350e-05 2.000868e-05
## 19       B 1.185825e-04 9.998139e-01 1.654915e-05 4.753072e-05 3.405866e-06
## 20       B 6.049064e-07 9.999890e-01 2.468140e-06 1.140262e-06 6.776943e-06
##
## [20 rows x 6 columns]
```

This output shows that, for each observation, the probability of that observation belonging to a particular class. The model chooses the class for which the probability is greatest.

I expect the out-of-sample error rate to be around ~0.2% as that is about where the model predicts the mean_per_class_error rate to be for the cross-validations.

I chose to use the H2O package rather than, say, caret, because I wanted to experiment with it's autoML capabilities. It seems to have worked wonderfully.