Evaluating Exercise with Machine Learning

by Mark Sucato

Executive Summary

The HAR *WLE* dataset contains on-body sensor information from six participants performing 10 repetitions of a unilateral dumbell biceps curl in five different but specific manners. A stacked ensemble of three different classification tree-based models trained on a training set of 13,737 observations predicted a validation set of 4,127 observations with 99.98% accuracy.

Project objective and data

Objective: predict the manner of exercise for twenty observations drawn from the Human Activity Recognition *Weight Lifting Exercises* dataset.

The HAR *WLE* dataset contains on-body sensor information from six participants performing 10 repetitions of a unilateral dumbell biceps curl in five different but specific manners (http://web.archive.org/web/20161224072740/http:/groupware.les.inf.puc-rio.br/har). Each observation is a time-stamped capture of sensor data; one complete repetition by a given participant includes multiple sequential observations. The 2013 *Qualitative Activity Recognition of Weight Lifting Exercises* HAR paper by Velloso et al, available at the same website, analyzed time slices of sequential data to assess repetitions [1] . Because the provided test set for this project only contains single-observation data excerpted from the greater dataset, sequential data analysis is not feasible for this project.

1. 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.

Exploratory Data Analysis

The provided training set includes 19,622 observations of 160 variables. 100 of these variables, all seemingly summary calculations from the raw sensor data, are almost entirely missing. Because this project will not rely on any sequential analysis, the time stamp observations, row labels, and movement window indicators are also unnecessary. A *skim* summary of the parsed data is provided below.

```
library(tidyverse)
library(caret)
library(skimr)
library(doParallel)

training <- read_csv("pml-training.csv")
testing <- read_csv("pml-testing.csv")

training <- training %>%
  select(where(~mean(is.na(.))< 0.9)) %>%
  select(-c(X1, raw_timestamp_part_1, raw_timestamp_part_2, cvtd_timestamp, new_window, num_window)) %>%
  mutate(classe=factor(classe), user_name = factor(user_name))
fix_windows_histograms()   # skimr package utility to fix histogram printing on Windows
skim(training)
```

Data summary

Name	training
Number of rows	19622
Number of columns	54
_____	
Column type frequency:	
factor	2
numeric	52
_____	
Group variables	None

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
user_name	0	1	FALSE	6	ade: 3892, cha: 3536, jer: 3402, car: 3112
classe	0	1	FALSE	5	A: 5580, B: 3797, E: 3607, C: 3422

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
roll_belt	0	1	64.41	62.75	-28.90	1.10	113.00	123.00	162.00	▇▁▁▆▇
pitch_belt	0	1	0.31	22.35	-55.80	1.76	5.28	14.90	60.30	▂▅▇▂▁
yaw_belt	0	1	-11.21	95.19	-180.00	-88.30	-13.00	12.90	179.00	▁▃▇▁▂
total_accel_belt	0	1	11.31	7.74	0.00	3.00	17.00	18.00	29.00	▇▁▂▆▁
gyros_belt_x	0	1	-0.01	0.21	-1.04	-0.03	0.03	0.11	2.22	▁▇▁▁▁
gyros_belt_y	0	1	0.04	0.08	-0.64	0.00	0.02	0.11	0.64	▁▁▇▁▁
gyros_belt_z	0	1	-0.13	0.24	-1.46	-0.20	-0.10	-0.02	1.62	▁▁▇▁▁
accel_belt_x	0	1	-5.59	29.64	-120.00	-21.00	-15.00	-5.00	85.00	▁▁▇▂▁
accel_belt_y	0	1	30.15	28.58	-69.00	3.00	35.00	61.00	164.00	▁▃▇▂▁
accel_belt_z	0	1	-72.59	100.45	-275.00	-162.00	-152.00	27.00	105.00	▇▂▁▂▅
magnet_belt_x	0	1	55.60	64.18	-52.00	9.00	35.00	59.00	485.00	▇▁▁▁▁
magnet_belt_y	0	1	593.68	35.68	354.00	581.00	601.00	610.00	673.00	▁▁▁▁▇
magnet_belt_z	0	1	-345.48	65.21	-623.00	-375.00	-320.00	-306.00	293.00	▁▁▁▇▁
roll_arm	0	1	17.83	72.74	-180.00	-31.78	0.00	77.30	180.00	▁▃▅▇▅
pitch_arm	0	1	-4.61	30.68	-88.80	-25.90	0.00	11.20	88.50	▁▃▇▂▁
yaw_arm	0	1	-0.62	71.36	-180.00	-43.10	0.00	45.88	180.00	▁▃▇▃▁
total_accel_arm	0	1	25.51	10.52	1.00	17.00	27.00	33.00	66.00	▂▅▇▂▁
gyros_arm_x	0	1	0.04	1.99	-6.37	-1.33	0.08	1.57	4.87	▁▂▅▇▁
gyros_arm_y	0	1	-0.26	0.85	-3.44	-0.80	-0.24	0.14	2.84	▁▃▇▂▁
gyros_arm_z	0	1	0.27	0.55	-2.33	-0.07	0.23	0.72	3.02	▁▂▇▃▁
accel_arm_x	0	1	-60.24	182.04	-404.00	-242.00	-44.00	84.00	437.00	▁▃▇▅▁
accel_arm_y	0	1	32.60	109.87	-318.00	-54.00	14.00	139.00	308.00	▁▃▇▃▁
accel_arm_z	0	1	-71.25	134.65	-636.00	-143.00	-47.00	23.00	292.00	▁▂▇▃▁
magnet_arm_x	0	1	191.72	443.64	-584.00	-300.00	289.00	637.00	782.00	▇▂▁▂▇
magnet_arm_y	0	1	156.61	201.91	-392.00	-9.00	202.00	323.00	583.00	▁▂▅▇▂
magnet_arm_z	0	1	306.49	326.62	-597.00	131.25	444.00	545.00	694.00	▁▁▂▃▇

roll_dumbbell	0	1	23.84	69.93	-153.71	-18.49	48.17	67.61	153.55
pitch_dumbbell	0	1	-10.78	36.99	-149.59	-40.89	-20.96	17.50	149.40
yaw_dumbbell	0	1	1.67	82.52	-150.87	-77.64	-3.32	79.64	154.95
total_accel_dumbbell	0	1	13.72	10.23	0.00	4.00	10.00	19.00	58.00
gyros_dumbbell_x	0	1	0.16	1.51	-204.00	-0.03	0.13	0.35	2.22
gyros_dumbbell_y	0	1	0.05	0.61	-2.10	-0.14	0.03	0.21	52.00
gyros_dumbbell_z	0	1	-0.13	2.29	-2.38	-0.31	-0.13	0.03	317.00
accel_dumbbell_x	0	1	-28.62	67.32	-419.00	-50.00	-8.00	11.00	235.00
accel_dumbbell_y	0	1	52.63	80.75	-189.00	-8.00	41.50	111.00	315.00
accel_dumbbell_z	0	1	-38.32	109.47	-334.00	-142.00	-1.00	38.00	318.00
magnet_dumbbell_x	0	1	-328.48	339.72	-643.00	-535.00	-479.00	-304.00	592.00
magnet_dumbbell_y	0	1	220.97	326.87	-3600.00	231.00	311.00	390.00	633.00
magnet_dumbbell_z	0	1	46.05	139.96	-262.00	-45.00	13.00	95.00	452.00
roll_forearm	0	1	33.83	108.04	-180.00	-0.74	21.70	140.00	180.00
pitch_forearm	0	1	10.71	28.15	-72.50	0.00	9.24	28.40	89.80
yaw_forearm	0	1	19.21	103.22	-180.00	-68.60	0.00	110.00	180.00
total_accel_forearm	0	1	34.72	10.06	0.00	29.00	36.00	41.00	108.00
gyros_forearm_x	0	1	0.16	0.65	-22.00	-0.22	0.05	0.56	3.97
gyros_forearm_y	0	1	0.08	3.10	-7.02	-1.46	0.03	1.62	311.00
gyros_forearm_z	0	1	0.15	1.75	-8.09	-0.18	0.08	0.49	231.00
accel_forearm_x	0	1	-61.65	180.59	-498.00	-178.00	-57.00	76.00	477.00
accel_forearm_y	0	1	163.66	200.13	-632.00	57.00	201.00	312.00	923.00
accel_forearm_z	0	1	-55.29	138.40	-446.00	-182.00	-39.00	26.00	291.00
magnet_forearm_x	0	1	-312.58	346.96	-1280.00	-616.00	-378.00	-73.00	672.00
magnet_forearm_y	0	1	380.12	509.37	-896.00	2.00	591.00	737.00	1480.00
magnet_forearm_z	0	1	393.61	369.27	-973.00	191.00	511.00	653.00	1090.00

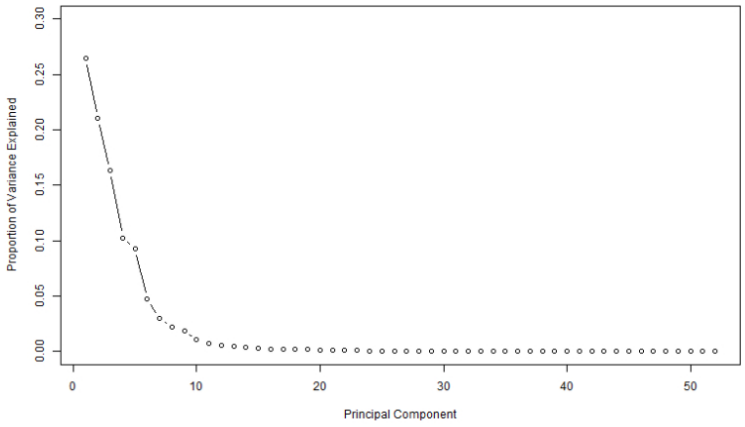
The resultant data set only includes complete cases. Examination of the numeric variable summary statistics and thumbnail histograms indicate many of the features display non-Gaussian indications. The analysis will use variable transformations to standardize the data. Due to the presence of negative and zero values, a Yeo-Johnson transformation is utilized.

For this analysis, the *training* data is partitioned into *training* and *validation* subsets. The *testing* data does not contain the *classe* variable and is dedicated to a prediction test of 20 observations.

A Principal Component Analysis variance-explained plot of the *training* set indicates PCA transformation could reduce the number of features and thus improve computational efficiency. The variance explained per principal component asymptotically reaches a limit around the 20th principal component.

```
inTrain <- createDataPartition(training$classe, p = 0.7, list = FALSE)
training <- training[inTrain, ]
validation <- training[-inTrain, ]

trainingPCA <- prcomp(x = training[-c(1,54)], scale = F, center = T)
pcaVar <- trainingPCA$sdev^2
varExp <- pcaVar / sum(pcaVar)
plot(varExp, xlab = "Principal Component", ylab = "Proportion of Variance Explained",
    ylim = c(0,.3), type = "b")
```



Modeling

This analysis uses an ensemble of three different machine learning algorithms stacked via a simple majority voting scheme. If all three models disagree, the model with the highest accuracy on the *validation* set provides the answer. As discussed above, *Yeo-Johnson* and *pca* transformations are used to standardize and scale the data. *Center* and *scale* transformations are prerequisites for *pca* transformation, and a *near-zero variance* transformation is used to check for isolated features that might bias the predictions.

The *caret* calculated PCA analysis indicates slightly more than the preliminary estimate of principle components are required to capture 95% of the variance. Additionally, the lack of additional variable removal during *caret* preprocessing indicates no features possess near-zero variance.

```
trainingPP <- preProcess(training[, -54], method = c("center", "scale", "YeoJohnson", "nzv", "pca"))
training1 <- predict(trainingPP, newdata = training)
validation1 <- predict(trainingPP, newdata = validation)
trainingPP  # No NSV features; passed 53, ignored user, transformed
```

```
## Created from 13737 samples and 53 variables
##
## Pre-processing:
##   - centered (52)
##   - ignored (1)
##   - principal component signal extraction (52)
##   - scaled (52)
##   - Yeo-Johnson transformation (51)
```

```
##   - Yeo-Johnson transformation (31)
##
## Lambda estimates for Yeo-Johnson transformation:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.2606  0.9079  0.9976  0.9744  1.0673  1.4373
##
## PCA needed 28 components to capture 95 percent of the variance
```

To improve accuracy, all three algorithms in the stacked ensemble utilize some form of boosting or bagging. All were chosen for their predictive abilities with classification problems. The three algorithms in the ensemble are:

- *Random Forest* via *ranger*
- *Stochastic Gradient Boosting* via *gbm*
- *Bagged Classification and Regression Tree* via *treebag*

To prevent overfitting, k-fold cross-validation using 10 folds and five repetitions is utilized. These numbers were chosen as a compromise between modeling desires and required computation time. For computational efficiency, parallel computation via a multi-core processor and the *doParallel* package is employed.

```
set.seed(12345)
cl <- makePSOCKcluster(3)  # doParallel package for parallel processing
registerDoParallel(cl)  # doParallel package for parallel processing

modControl <- trainControl(method = "repeatedcv", number = 10, repeats = 5)
gbmFit <- train(classe ~ ., data = training1, method = "gbm", trControl = modControl, verbose = FALSE)
rfFit <- train(classe ~ ., data = training1, method = "ranger", trControl = modControl, verbose = FALSE)
treefit <- train(classe ~ ., data = training1, method = "treebag", trControl = modControl, verbose = FALSE)

stopCluster(cl)  # doParallel package for parallel processing

gbmVote <- predict(gbmFit, newdata = validation1)
rfVote <- predict(rfFit, newdata = validation1)
treeVote <- predict(treefit, newdata = validation1)
voting <- function(a, b, c) {
  for(i in 1:length(a)) {
    if(b[i] == c[i]) a[i] = b[i]
  }
  return(a)
}
voteTally <- voting(rfVote, gbmVote, treeVote)
```

The random forest's out-of-bag prediction error estimate based on the training set is provided below.

```
rfFit$finalModel
```

```
## Ranger result
##
## Call:
##  ranger::ranger(dependent.variable.name = ".outcome", data = x,      mtry = min(param$mtry, ncol(x)), m
##
## Type:                             Classification
## Number of trees:                  500
## Sample size:                      13737
## Number of independent variables:  33
## Mtry:                             33
## Target node size:                 1
## Variable importance mode:         none
## Splitrule:                        extratrees
## Number of random splits:          1
## OOB prediction error:             1.89 %
```

The three model's accuracies on the validation set are provided below. The random forest model perfectly predicted the validation set, exceeding the ~98% OOB training set-derived estimate, and the bagged tree model only missed perfection by a few elements. The boosted gradient tree performed slightly worse at ~86% accuracy.

```
postResample(pred = rfVote, obs = validation1$classe)
```

```
## Accuracy    Kappa
##        1        1
```

```
postResample(pred = treeVote, obs = validation1$classe)
```

```
##  Accuracy     Kappa
## 0.9997577 0.9996933
```

```
postResample(pred = gbmVote, obs = validation1$classe)
```

```
##  Accuracy     Kappa
## 0.8589775 0.8212099
```

The final ensemble confusion matrix and accuracies on the validation set are provided below. In one case, the less accurate models outvoted the random forest model to the minute detriment of overall accuracy. Because of the tiny difference and the upside potential of the ensemble against future unknown data, this project retains the ensemble approach.

```
confusionMatrix(voteTally, validation1$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1186    0    0    0    0
##          B    0  798    0    0    0
##          C    0    0  709    0    1
##          D    0    0    0  679    0
##          E    0    0    0    0  754
##
## Overall Statistics
##
```

```
##               Accuracy : 0.9998
##                 95% CI : (0.9987, 1)
##    No Information Rate : 0.2874
##    P-Value [Acc > NIR] : < 2.2e-16
##
##                  Kappa : 0.9997
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                     Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   1.0000   1.0000   1.0000   0.9987
## Specificity            1.0000   1.0000   0.9997   1.0000   1.0000
## Pos Pred Value         1.0000   1.0000   0.9986   1.0000   1.0000
## Neg Pred Value         1.0000   1.0000   1.0000   1.0000   0.9997
## Prevalence             0.2874   0.1934   0.1718   0.1645   0.1829
## Detection Rate         0.2874   0.1934   0.1718   0.1645   0.1827
## Detection Prevalence   0.2874   0.1934   0.1720   0.1645   0.1827
## Balanced Accuracy      1.0000   1.0000   0.9999   1.0000   0.9993
```

Predictions

The predictions of the testing set variables are provided below.

```
testing <- testing %>%
  select(where(~mean(is.na(.))< 0.9)) %>%
  select(-c(X1, raw_timestamp_part_1, raw_timestamp_part_2,
            cvtd_timestamp, new_window, num_window)) %>%
  mutate(user_name = factor(user_name))
testing1 <- predict(trainingPP, newdata = testing)
v1 <- predict(rfFit, newdata = testing1)
v2 <- predict(gbmFit, newdata = testing1)
v3 <- predict(treefit, newdata = testing1)
voting(v1, v2, v3)
```

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

Conclusions

The simple majority stacked ensemble utilized predicts the validation set with 99.98% accuracy. The individual algorithms performance on the *validation* set indicate the ensemble approach is probably unnecessary in this case, but that might not be the case with other datasets.