

The Quantified Self: Proper Lifting

Justin Dallmann

8/18/2017

Executive summary

Measurements from belt, arm, dumbbell, and forearm monitors are used to predict whether dumbbell bicep curls have been performed correctly. In what follows, I develop a prediction model for whether or a curl has been performed correctly using random forests with 10 fold cross-validation (via the `caret` package). The training set accuracy of the final model is 99.3%, while the out of sample error estimated on a large hold out set is 99.6%.

The data used to build the prediction model comes from Groupware@LES research group. The data set contains 19,622 observations of 159 variables in which 6 participants from 20 to 28 years of age were asked to perform one set of 10 repetitions of the Unilateral Dumbbell Biceps Curl in five ways: (i) exactly according to the specification, (ii) throwing the elbows to the front, (iii) lifting the dumbbell only halfway, (iv) lowering the dumbbell only halfway, and (v) throwing the hips to the front (Velloso et. al. 2013, p. 3). As they note, though much work has been done in the domain of automated activity recognition, automated *quality* of activity recognition has received considerably less attention. For further details, see:

- 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.

Loading the data

```
# Load data
data <- read.csv("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv",
  na.strings=c("NA", "", "#DIV/0!"))
```

Cleaning the data

In order to build the model I remove time and date information, and summary statistic variables for each measurement window (to ensure that the window identifier information is not used in prediction). The data is then randomly separated into a training set (80% of data) upon which to perform 10-fold cross-validation model selection and a hold out set (20% of data) to estimate out of sample error.

```
# Get rid of date-time info, window information, and subject identifiers
data <- data[,8:160]

# Get rid of NA value columns
data <- data[, colSums(is.na(data)) == 0]

# Set training set and test set
set.seed(1985)
```

```
inTrain <- createDataPartition(y=data$classe, p=0.80, list = FALSE)
train <- data[inTrain,]
test <- data[-inTrain,]
```

Modeling

To create the final prediction model, I used a random forest method with 10-fold cross-validation on all remaining variables.

```
# Train the model
# [NB: the CARET package already takes care of
# n-fold cross-validation for random forests.]
forMod <- train(classe~., data=train, method="rf",
               trControl=trainControl(method="cv", number=10),
               prox=TRUE, allowParallel=TRUE)
```

Summary of results

Of the models explored, the best model had an in training set 10-fold cross-validation error rate of 0.7% (accuracy of 99.3%) using random samples of 2 variables (`mtry=2`). The error in the hold-out set is (surprisingly) higher still at 99.6% and offers a good prediction of out of sample accuracy.

```
# Print details from the model training
print(forMod)
```

```
## Random Forest
##
## 15699 samples
##    52 predictor
##    5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 14130, 14130, 14128, 14128, 14130, 14130, ...
## Resampling results across tuning parameters:
##
##  mtry  Accuracy  Kappa
##    2    0.9929932 0.9911357
##   27    0.9926114 0.9906528
##   52    0.9857322 0.9819487
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

```
# Print training set accuracy
print(forMod$finalModel)
```

```
##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry, proximity = TRUE,      allowParallel = TRUE)
##                Type of random forest: classification
##                Number of trees: 500
## No. of variables tried at each split: 2
```

```
##
##          OOB estimate of  error rate: 0.65%
## Confusion matrix:
##      A    B    C    D    E  class.error
## A 4463    0    0    0    1 0.0002240143
## B   15 3019    4    0    0 0.0062541145
## C    0   27 2710    1    0 0.0102264427
## D    0    0   46 2526    1 0.0182666148
## E    0    0    2    5 2879 0.0024255024
```

```
# Print test set accuracy
testPreds <- predict(forMod, test[, -53])
confusionMatrix(testPreds, test$classe)
```

```
## Confusion Matrix and Statistics
```

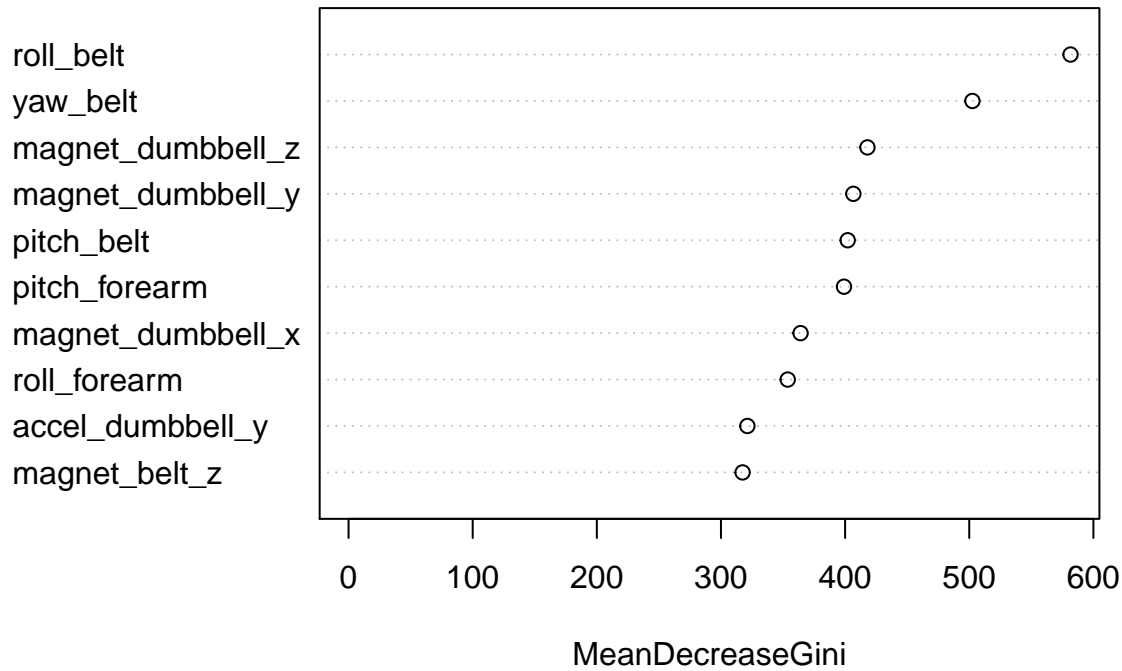
```
##
##          Reference
## Prediction    A    B    C    D    E
##          A 1116    1    0    0    0
##          B    0   758    4    0    0
##          C    0    0   677    5    0
##          D    0    0    3   637    2
##          E    0    0    0    1   719
```

```
##
## Overall Statistics
##
##          Accuracy : 0.9959
##          95% CI : (0.9934, 0.9977)
##    No Information Rate : 0.2845
##    P-Value [Acc > NIR] : < 2.2e-16
##
##          Kappa : 0.9948
##    McNemar's Test P-Value : NA
```

```
##
## Statistics by Class:
##
##          Class: A Class: B Class: C Class: D Class: E
## Sensitivity      1.0000   0.9987   0.9898   0.9907   0.9972
## Specificity      0.9996   0.9987   0.9985   0.9985   0.9997
## Pos Pred Value    0.9991   0.9948   0.9927   0.9922   0.9986
## Neg Pred Value    1.0000   0.9997   0.9978   0.9982   0.9994
## Prevalence        0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate    0.2845   0.1932   0.1726   0.1624   0.1833
## Detection Prevalence 0.2847   0.1942   0.1738   0.1637   0.1835
## Balanced Accuracy 0.9998   0.9987   0.9941   0.9946   0.9985
```

The most important variables (by Gini importance) were roll_belt, pitch_forearm, yaw_belt, magnet_dumbell_z, magnet_dumbell_y, pitch_belt, and roll_forearm, with importance trailing off substantially afterwards for the others.

forMod\$finalModel



Suggestions for further exploration

If greater accuracy were required, there are a couple of further approaches that might be worth trying. For example:

1. Tune the random forest model more systematically for other numbers of variables sampled (other values of `mtry`).
2. Generating further features, including the summary statistics that were originally removed from the data set for the analysis. Costs of this approach include increasing computational time, increasing the chances of over-fitting.
3. Including time-series information. Costs of this approach include increasing computational time, and potentially adding spurious correlations across measurements.
4. Adding within-subject estimation of quality of action. Costs of this approach include increasing computational time and reducing availability (since measurements for each new individual would be needed to calibrate the model).
5. Trying different methods like logistic regression, neural network predictors, or some predictor that aggregates the above approaches. This option would most increase analysis time and computational time, though would probably secure the greatest gains especially if combined with the above suggestions.