The Quantified Self:
Proper Lifting

Justin Dallmann

8/18/2017

Contents

1 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%. (Other prediction models explored, and their accuracy estimates, can be seen in the last section.)

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.

2 Loading the data

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

3 Cleaning the data

In building the final model I remove 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(1980)
inTrain <- createDataPartition(y=data$classe, p=0.80, list = FALSE)
train <- data[inTrain,]
test <- data[-inTrain,]
```

4 Modeling

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

```
## train random forest model
## [NB: the CARET package already takes care of
## n-fold cross-validation for random forests.]
# set.seed(1981)
# forMod <- train(classe~., data=train, method="rf",
#              trControl=trainControl(method="cv",number=10),
#              prox=TRUE, allowParallel=TRUE)
#
# save(forMod,file="forMod.RData")

load("forMod.RData")
```

4.1 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 27 variables (`mtry=27`). The accuracy 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: 14129, 14128, 14130, 14128, 14129, 14129, ...
| Resampling results across tuning parameters:
|
|   mtry  Accuracy   Kappa
|    2    0.9936938  0.9920225
|   27    0.9936938  0.9920229
|   52    0.9856677  0.9818683
|
| Accuracy was used to select the optimal model using  the largest value.
| The final value used for the model was mtry = 27.
```

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

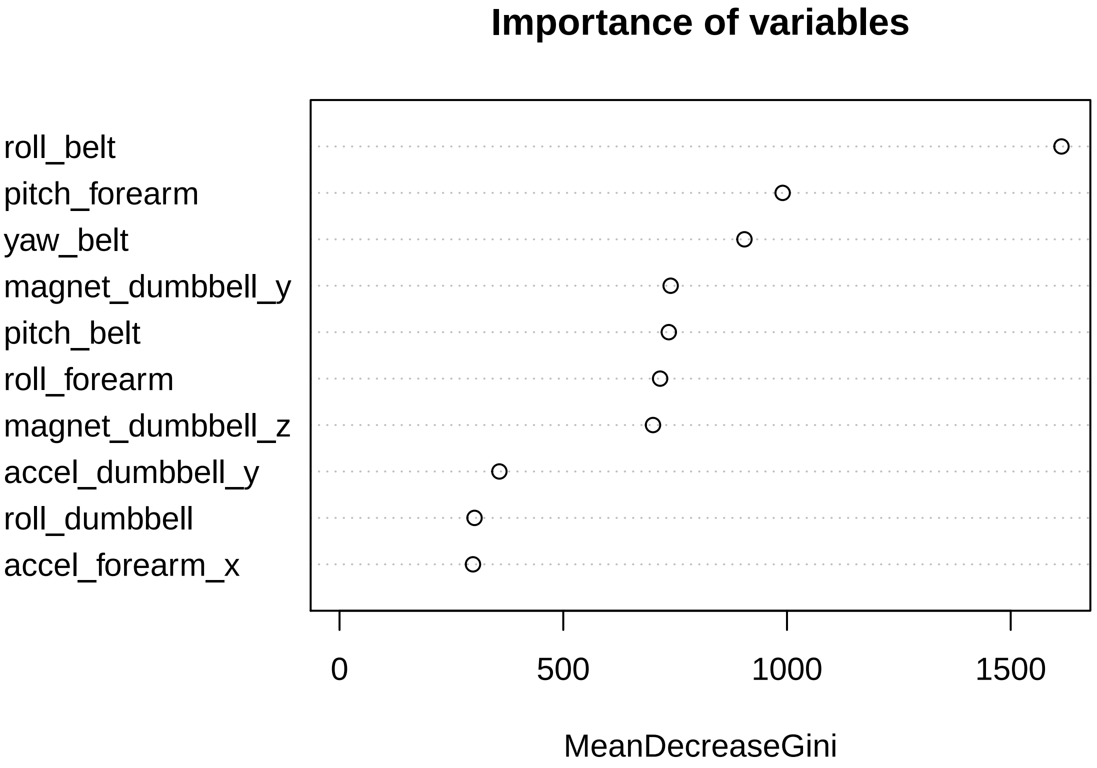
```
|
| Call:
|  randomForest(x = x, y = y, mtry = param$mtry, proximity = TRUE,      allowParallel = TRUE)
|                Type of random forest: classification
|                      Number of trees: 500
| No. of variables tried at each split: 27
|
|         OOB estimate of  error rate: 0.62%
| Confusion matrix:
|      A    B    C    D    E class.error
| A 4457    5    1    0    1 0.001568100
| B   22 3009    6    1    0 0.009545754
| C    0   13 2716    9    0 0.008035062
| D    0    0   23 2547    3 0.010104936
| E    0    2    4    8 2872 0.004851005
```

```
# 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 1113    2    0    0    0
|          B    3  756    1    0    0
|          C    0    1  681    4    2
|          D    0    0    2  639    1
|          E    0    0    0    0  718
|
| 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            0.9973   0.9960   0.9956   0.9938   0.9958
| Specificity            0.9993   0.9987   0.9978   0.9991   1.0000
| Pos Pred Value         0.9982   0.9947   0.9898   0.9953   1.0000
| Neg Pred Value         0.9989   0.9991   0.9991   0.9988   0.9991
| Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
| Detection Rate         0.2837   0.1927   0.1736   0.1629   0.1830
| Detection Prevalence   0.2842   0.1937   0.1754   0.1637   0.1830
| Balanced Accuracy      0.9983   0.9974   0.9967   0.9964   0.9979
```

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.



5 Other models explored and suggestions for further tuning

5.1 Recusive partitioning/CART

In exploration, I also fit a recursive partitioning CART model (Breiman et. at., 1984). With a best model out of sample accuracy rate ~88% (tuned for complexity values of cp in [.0005, .05] and default 10 cross-validations).

```
## train CART model
#
# set.seed(1982)
# treeMod <- train(classe ~ ., data=train,
#                method="rpart",
#                control = rpart.control(minsplit = 100),
#                tuneGrid = data.frame(cp = 0.001))
#
# save(treeMod,file="treeMod.RData")

load("treeMod.RData")
```

```
# print test set accuracy
treeModPreds <- predict(treeMod$finalModel,
                        test[,-53], type="class")
confusionMatrix(treeModPreds, test$classe)
```

```
| Confusion Matrix and Statistics
|
|           Reference
| Prediction    A    B    C    D    E
|          A 1061   60   13   16    4
|          B   25  608   56   28   18
|          C    9   43  580   28   25
|          D   12   18   25  543   26
|          E    9   30   10   28  648
|
| Overall Statistics
|
|                Accuracy : 0.8769
|                  95% CI : (0.8662, 0.887)
|     No Information Rate : 0.2845
|     P-Value [Acc > NIR] : < 2.2e-16
|
|                   Kappa : 0.8441
|  Mcnemar's Test P-Value : 0.0005464
|
| Statistics by Class:
|
|                      Class: A Class: B Class: C Class: D Class: E
| Sensitivity            0.9507   0.8011   0.8480   0.8445   0.8988
| Specificity            0.9669   0.9599   0.9676   0.9753   0.9760
| Pos Pred Value         0.9194   0.8272   0.8467   0.8702   0.8938
| Neg Pred Value         0.9801   0.9526   0.9679   0.9697   0.9772
| Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
| Detection Rate         0.2705   0.1550   0.1478   0.1384   0.1652
| Detection Prevalence   0.2942   0.1874   0.1746   0.1591   0.1848
| Balanced Accuracy      0.9588   0.8805   0.9078   0.9099   0.9374
```

5.2 Gradient boosting

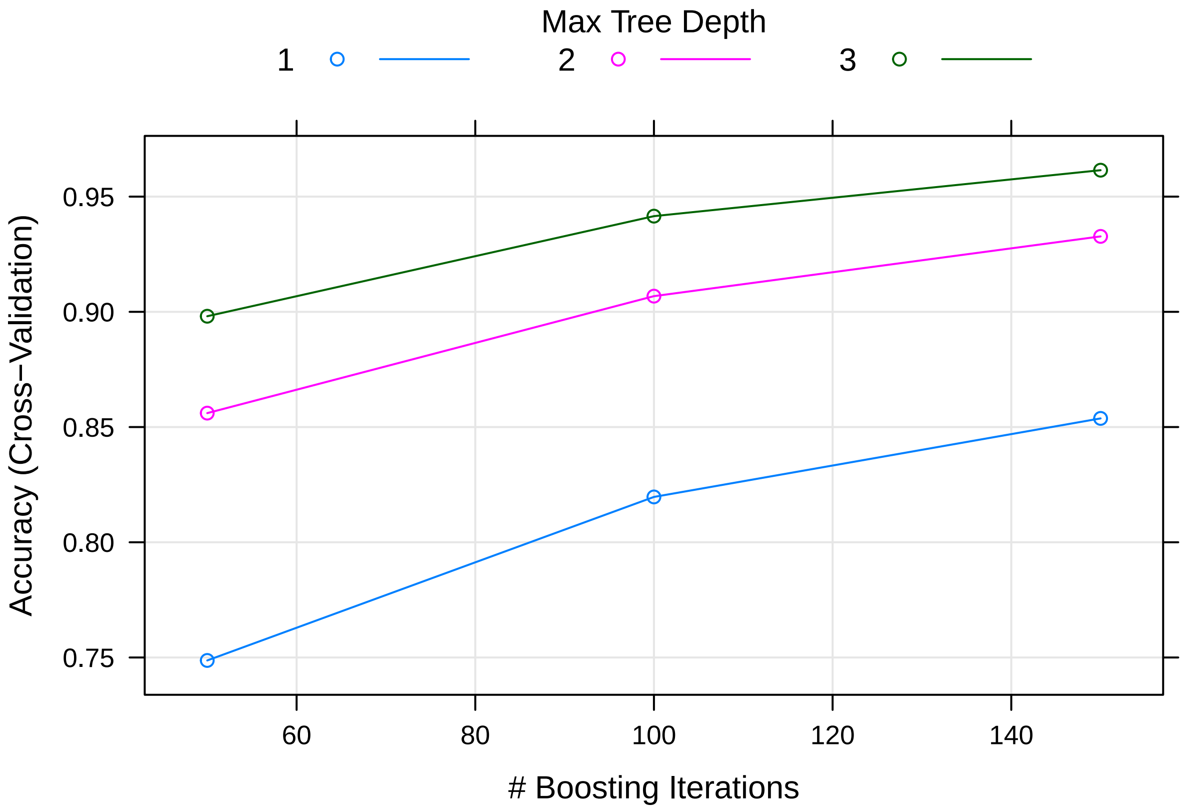
Boosting with trees (gradient boosting machine) has a best model out of sample accuracy rate ~96%. The final values used for the model were n.trees = 150, interaction.depth = 3, shrinkage = 0.1 and n.minobsinnode = 10. No further tuning of shrinkage for the model was performed.

```
## train gbm
#
# set.seed(1983)
# gbmMod <- train(classe ~ ., data=train,
#                 method="gbm", verbose = FALSE,
#                 trControl=
#                     trainControl(method="cv",number=10))
# gbmMod
# save(gbmMod,file="gbmMod.RData")

load("gbmMod.RData")
```

```
# print test set accuracy
gbmModPreds <- predict(gbmMod, test[,-53])
confusionMatrix(gbmModPreds, test$classe)
```

```
| Confusion Matrix and Statistics
|
|           Reference
| Prediction    A    B    C    D    E
|          A 1098   22    0    2    0
|          B   12  710   19    4    6
|          C    5   26  657   17    5
|          D    0    1    7  614    5
|          E    1    0    1    6  705
|
| Overall Statistics
|
|                Accuracy : 0.9646
|                  95% CI : (0.9583, 0.9701)
|     No Information Rate : 0.2845
|     P-Value [Acc > NIR] : < 2.2e-16
|
|                   Kappa : 0.9552
|  Mcnemar's Test P-Value : 0.002848
|
| Statistics by Class:
|
|                      Class: A Class: B Class: C Class: D Class: E
| Sensitivity            0.9839   0.9354   0.9605   0.9549   0.9778
| Specificity            0.9914   0.9870   0.9836   0.9960   0.9975
| Pos Pred Value         0.9786   0.9454   0.9254   0.9793   0.9888
| Neg Pred Value         0.9936   0.9846   0.9916   0.9912   0.9950
| Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
| Detection Rate         0.2799   0.1810   0.1675   0.1565   0.1797
| Detection Prevalence   0.2860   0.1914   0.1810   0.1598   0.1817
| Balanced Accuracy      0.9877   0.9612   0.9721   0.9755   0.9877
```



5.3 Further tuning

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-length of activity information. Costs of this approach include increasing computational time, and potentially adding spurious correlations across measurements. Following this approach it would also be important to measure time **from the begining of each exercise window**—since if the full timestamp were used, knowledge of which time slice the action is being performed in would provide illicit near perfect information regarding the class of exercise being undertaken.
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.