Machine Learning Project

S. Purvis

February 12, 2018

Summary

Research on exercise-related activity recognition has traditionally been focused on discriminating between activities and not on the the quality of execution of a specific activity. Using a simple bicep curl, Velloso et. al.[1] collected sensor data on 6 participants performing the exercise to specification and purposely making 4 specific errors. The authors then used a correlation-based method to select 17 features for Random Forest approach.

With ~58 features available for use in the native data set, using correlation-based methods to reduce this number down to 37. After using a simple tree prediction model, we were able to identify 4 features that we used to develop a working model.

- Belt sensor - roll_belt
- Dumbbell sensor - magnet_dumbbell_y & magnet_dumbbell_z
- Forearm sensor - pitch_forearm

Using these 4 variables and a Random Forest approach, we were able to generate a predictive model that was 99% accurate.

Introduction

Proper technique in weightlifting is important for effective training and has a positive impact on cardio-resipiratory fitness. in 2013, Velloso et. al. used machine learning and pattern recognition techniques to quantify *how well* certain weight lifting activities were being performed. Study participants completed simple curling exercise using sensors mounted in the users' glove, armband, lumbar belt and dumbbell. Under the supervision of an experienced, 6 participants performed 10 sets of bicep curls according to specifications and with 4 specific mistakes that correspond to poor exercise execution.

Fifty eight different measurements where captured for each participant.The investigators then chose 17 features in resulting data set (using created covariates, i.e. mean,variance, etc.) and random forest approach to see if they could build a model that accurately predicts. In this analysis, we attempt to generate

```
library(dplyr)
library(tidyr)
library(ggplot2)
library(lubridate)
library(caret)
library(rattle)
library(cowplot)
```

Data

Data was imported directly from the provide web address

[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.

```
# data files were from site
download.file("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv",
              destfile="training.csv",method="libcurl")
download.file("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv",
              destfile="testing.csv",method="libcurl")

#Create training dataframe
train.dl <- read.csv("training.csv",
                     header = TRUE,
                     na.strings = c("","#DIV/0!","NA"))


#Create test dataframe
test.dl <- read.csv("testing.csv",
                    header = TRUE,
                    na.strings = c("","#DIV/0!","NA"))
```

Cleaning/Exploration

```
dim(train.dl)
```

```
## [1] 19622   160
```

Using the dim command, we see a data set with 160 variables. Review of the dataframe summary (not shown), reveals a large number of variables that are comprised of NAs. These will be removed.

```
train.removeNAs<-train.dl %>% select(which(colMeans(is.na(.)) < 0.5))%>% select(-X)

dim(train.removeNAs)
```

```
## [1] 19622    59
```

This removes 101 variables.

Initial view is that all variables are assigned an appropriate class, so no need to make any changes(data not shown)

In absence of a data dictionary, it is impossible to draw any conclusions on value variables raw_timestamp_part_1:num_window, therefore, I chose to remove these variables.

```
train.remove.vars1 <- train.removeNAs %>% select(-c(raw_timestamp_part_1:num_window))
dim(train.remove.vars1)
```

```
## [1] 19622    54
```

This leaves 54 variables

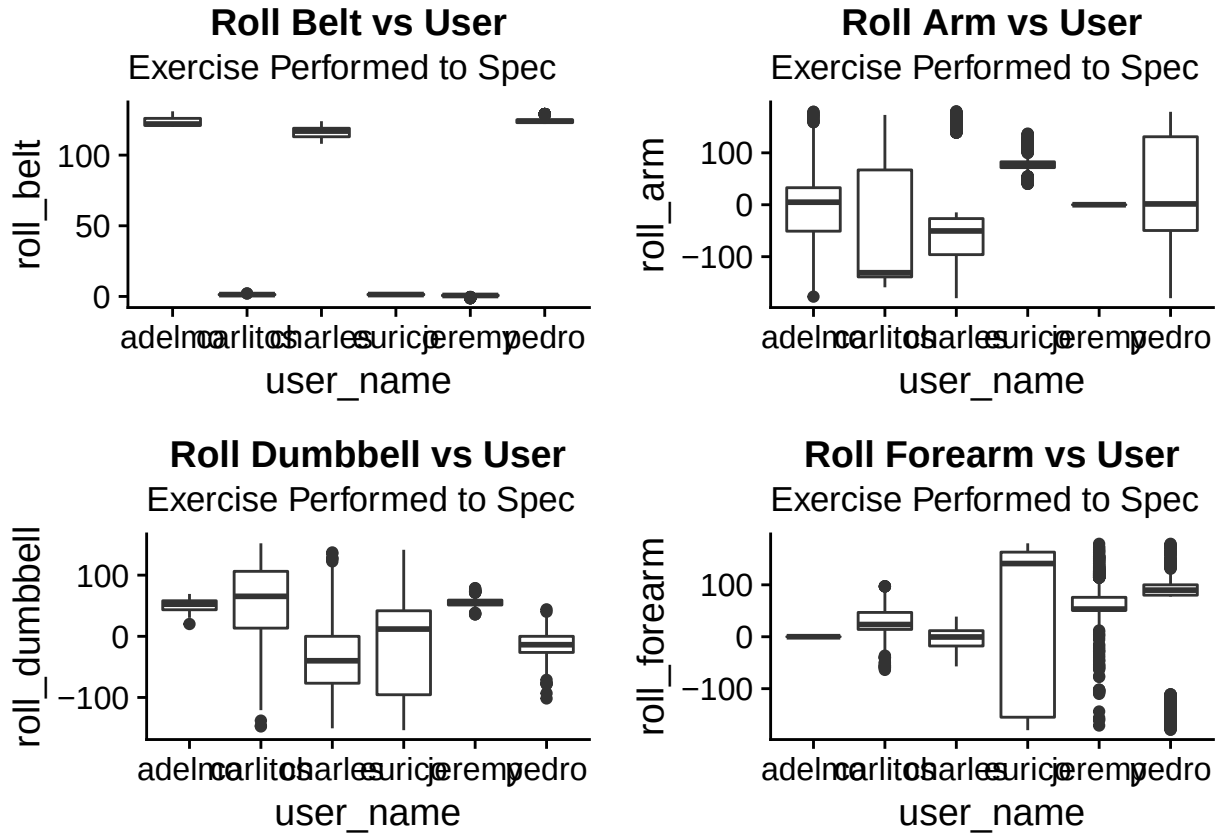
Repeat these cleaning steps on the test data set.

```
test.removeNAs<-test.dl %>% select(which(colMeans(is.na(.)) < 0.5))%>% select(-X)
test.remove.vars1 <- test.removeNAs %>% select(-c(raw_timestamp_part_1:num_window))
```

Creation of Dummy Variables

During initial analysis, it looked as if there was some differences in the sensor data between users. To determine if there was, we looked at data from all user when they did the exercise to specification (classe == A) and then looked at a few variables to see if there was any difference between users.

```
train.A <- train.remove.vars1 %>% filter(classe == "A")

# lets choose a single variable from each sensor that measures the same movement (roll)
belt<- ggplot(train.A, aes(x=user_name, y=roll_belt))+
  geom_boxplot()+
  ggtitle("Roll Belt vs User", subtitle = "Exercise Performed to Spec")

arm<-ggplot(train.A, aes(x=user_name, y=roll_arm))+
  geom_boxplot()+
  ggtitle("Roll Arm vs User", subtitle = "Exercise Performed to Spec")

dbell<- ggplot(train.A, aes(x=user_name, y=roll_dumbbell))+
  geom_boxplot()+
  ggtitle("Roll Dumbbell vs User", subtitle = "Exercise Performed to Spec")

fore<- ggplot(train.A, aes(x=user_name, y=roll_forearm))+
  geom_boxplot()+
  ggtitle("Roll Forearm vs User", subtitle = "Exercise Performed to Spec")

plot_grid(belt, arm, dbell, fore, label_size = 6)
```



So, even when the exercise is done to specification, there is difference between users. I chose to convert these users to dummy variables.

```
##Convert user_name to dummy variables in training----------------------------------------------
dumMod <- dummyVars(~user_name, data=train.remove.vars1, fullRank=T)
```

```
dumtrain <- data.frame(predict(dumMod, newdata = train.remove.vars1))
train.final <- cbind(dumtrain, train.remove.vars1)
train.final <- train.final %>% select(-user_name)


# Repeat on the test data set-------------------------------------------------------
dumModTest <- dummyVars(~user_name, data=test.remove.vars1, fullRank=T)
dumtest <- data.frame(predict(dumModTest, newdata = test.remove.vars1))
test.final <- cbind(dumtest, test.remove.vars1)
test.final <- test.final %>% select(-user_name)
```

Before we move on to feature selection, let's partition our training data into training (used to build the model) and validation (used to determine accuracy as we build our model) data sets.

```
trainparts <- createDataPartition(y=train.final$classe,
                                  p = 0.75,
                                  list = FALSE)
train.for.modeling <- train.final[trainparts,]
validation <- train.final[-trainparts,]
```

train-for.modeling dataframe - used for modeling building, assessment, and cross-validation validation dataframe - independent dataframe to determine model performance *test dataframe - used to make final prediction

Feature Selection

Colinear Features

The first step in feature selection is to remove the highly correlated variables as these will increase bias.

```
# calculate correlation matrix
set.seed(12345)
correlationMatrix <- cor(train.for.modeling[,6:57])
# find attributes that are highly corrected (ideally >0.75)
highlyCorrelated <- findCorrelation(correlationMatrix, cutoff=0.75)
# print indexes of highly correlated attributes that should be removed from the train dataframe
print(highlyCorrelated)
```

```
##  [1] 10  1  9  4 36  8  2 37 35 38 34 21 23 25 13 48 45 31 33 18
```

```
# remove them
train.final.cor <- train.final %>% select(-highlyCorrelated)

dim(train.final.cor)
```

```
## [1] 19622    38
```

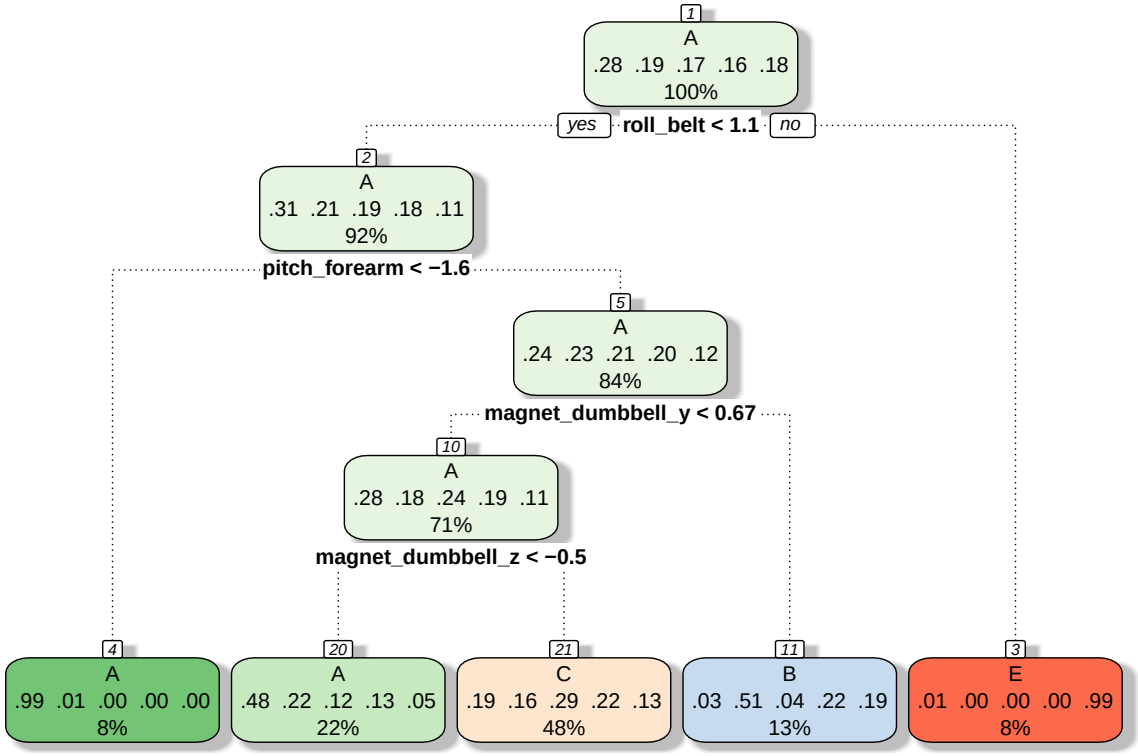
This removes another 15-20 variables.

Initial Modelling - Predicting with Trees

The goal of this step is to wade through the remaining 35 or so variables, to see if we can find the foundation of essential variables to use in model building. We will discuss preprocessing and cross-validation in the modelling section.

```
# prepare training scheme
set.seed(12345)
```

```
#simple train control
control <- trainControl(method="cv", number=5)
# train the initial model
model.rpart <- train(classe~., data=train.final.cor, method="rpart", preProcess=c("center","scale"), tr
#explor the tree
fancyRpartPlot(model.rpart$finalModel)
```



Rattle 2018−Feb−13 16:45:32 PurvisPC

Let's see what the accuracy is with this model.

```
pred.rpart <- predict(model.rpart, validation)
confusionMatrix(pred.rpart,validation$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   A   B   C   D   E
##          A 933 242 128 147  62
##          B  24 333  24 149 124
##          C 436 374 703 508 294
##          D   0   0   0   0   0
##          E   2   0   0   0 421
##
## Overall Statistics
##
##                Accuracy : 0.4874
##                  95% CI : (0.4733, 0.5015)
##     No Information Rate : 0.2845
```

```
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3497
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.6688   0.3509   0.8222   0.0000  0.46726
## Specificity            0.8350   0.9188   0.6019   1.0000  0.99950
## Pos Pred Value         0.6171   0.5092   0.3037      NaN  0.99527
## Neg Pred Value         0.8638   0.8551   0.9413   0.8361  0.89288
## Prevalence             0.2845   0.1935   0.1743   0.1639  0.18373
## Detection Rate         0.1903   0.0679   0.1434   0.0000  0.08585
## Detection Prevalence   0.3083   0.1334   0.4721   0.0000  0.08626
## Balanced Accuracy      0.7519   0.6349   0.7120   0.5000  0.73338
```

This simple prediction model uses only 4 variables but is only 50% accurate. Let's use these 4 variables moving forward to see how they perform.

Modeling

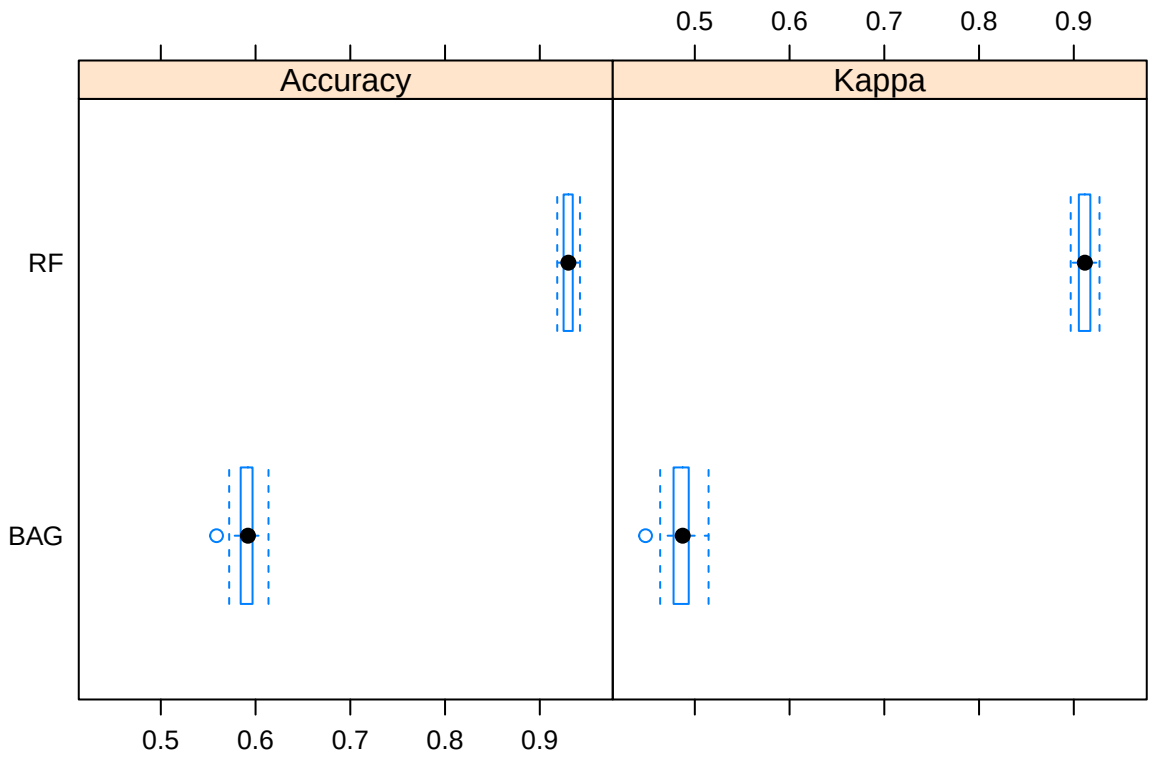
Preprocessing and Cross-Validation

During data exploration, it became clear that many of the variables were highly skewed. To prepare these data for modeling, I used the preProcess argument of the train function to normalize the data (center and scale).

Accuracy measures on the training set are optimistic. To minimize this overestimation of accuracy, we employed cross-validation techniques during modeling. Cross-validation is a powerful preventative measure against overfitting. To balance computation time and performance, I chose to use the "repeatedcv" method, done 10x and repeated 3x.

Model Selection

```
# this code creates a formula object that we can use to pick out the best algorithm
ini.formula <- as.formula(classe~roll_belt+pitch_forearm+magnet_dumbbell_y+magnet_dumbbell_z)

set.seed(1234)
control <- trainControl(method="repeatedcv", number=10, repeats = 3)
#train random forest model
train.model.rf <- train(ini.formula, data=train.final.cor, method="rf", preProcess=c("center","scale"),
#train bagging model
train.model.bag <- train(ini.formula, data=train.final.cor, method="bagFDA", preProcess=c("center","scal
# collect resamples
results <- resamples(list(RF=train.model.rf, BAG=train.model.bag))
# boxplots of results
bwplot(results)
```



The Random Forest Algorithm provides the better performance when compared to Bagging method.

Determining Accuracy using Validation data

```
rf.pred <- predict(train.model.rf, validation)
confusionMatrix(rf.pred, validation$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1391    1    2    3    0
##          B    0  934    2    0    1
##          C    2   14  850    1    0
##          D    2    0    1  800    0
##          E    0    0    0    0  900
##
## Overall Statistics
##
##                Accuracy : 0.9941
##                  95% CI : (0.9915, 0.996)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9925
##  Mcnemar's Test P-Value : NA
##
```

```
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9971   0.9842   0.9942   0.9950   0.9989
## Specificity            0.9983   0.9992   0.9958   0.9993   1.0000
## Pos Pred Value         0.9957   0.9968   0.9804   0.9963   1.0000
## Neg Pred Value         0.9989   0.9962   0.9988   0.9990   0.9998
## Prevalence             0.2845   0.1935   0.1743   0.1639   0.1837
## Detection Rate         0.2836   0.1905   0.1733   0.1631   0.1835
## Detection Prevalence   0.2849   0.1911   0.1768   0.1637   0.1835
## Balanced Accuracy      0.9977   0.9917   0.9950   0.9971   0.9994
```

We are able to use 4 variables to build a model with 99% accuracy.

Prediction on Test Data

```
testpred <- predict(train.model.rf, test.final)
testpred
```

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