# Bicep Curl Analysis

*Steve Bachmeier*

*September 4, 2018*

# Synopsis

A two-fold random forest model is efficient and provides an acceptably high 98.1% accuracy (1.9% out of sample error) when applied to the testing set (obtained from the raw training set). The model is fit to 17 covariates which all have variable-variable corrleations of less than 50%.

This model correctly predicted 19/20 of the downloaded test set observations.

# Background

This analysis attempts to quantify how well participants complete a dumbbell bicep curl. Six people (all male, 20-28 years old, and with little weight lifting experience) performed the exercise in six different ways (one correct (classe A) and five incorrect (classe B-F)). Accelerometers were placed on each participant's belt, forearm, bicep, and dumbbell.

# Data

The data for this project came from: http://groupware.les.inf.puc-rio.br/har (http://groupware.les.inf.puc-rio.br/har)

Training data set: https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv (https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv)

Test data set: https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv (https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv)

# Goal

To goal of this analysis is to predict the manner in which the participants completed the exercise (the "classe" variable in the training data set). Results should include the expected out of sample error.

# Analysis

This section outlines the analysis completed.

## Set up and data cleaning

We start by loading libraries and loading/cleaning the raw data. Cleaning the data consisted of finding all columns in the raw test set that included only NAs (since most machine learning algorithms cannot work with empty values imputation cannot work with all-NA variables) as well as any columns deemed irrelevant for the task at hand (specifically, participant names and columns related to time - the purpose of this project is not the effect of time on

workout quality but rather predicting quality based on the sensor data). These columns are removed from the working (training and validation) data set; there were no NA values left in the validation set and so no imputation is required. Note that the columns removed from the validation set were not removed from the test set.

We are left with 52 variables in the working data set, all of which have no NA observations and all of which are sensor-related. As a last step, we convert all variables to numeric (many were integers) to prevent problems later. This conversion was also completed on corresponding columns in the test set.

```
# ========================================
# LOAD LIBRARIES
# ========================================
library(caret)
library(parallel)
library(doParallel)
library(dplyr)
library(ggplot2)
library(GGally)

# ========================================
# LOAD/CLEAN DATA
# ========================================
testRaw <- read.csv("Data/pml-testing.csv")
trainingRaw <- read.csv("Data/pml-training.csv")

# Compare test and training sets
# Number of training set variables not in test set
sum(!(names(trainingRaw) %in% names(testRaw)))
```

```
## [1] 1
```

```
names(trainingRaw)[!(names(trainingRaw) %in% names(testRaw))] # Unique training set variable
```

```
## [1] "classe"
```

```
names(testRaw)[!(names(trainingRaw) %in% names(testRaw))] # Unique test set variable
```

```
## [1] "problem_id"
```

```
# Find all not-all-NA columns in test set
keepVars <- names(Filter(f=function(x) {!all(is.na(x))}, testRaw))

# Remove irrelevant columns
keepVars <- keepVars[-c(1, 2, 3, 4, 5, 6, 7, 60)]

# Create working df
working <- select(trainingRaw, keepVars, classe)

# Check for NAs in working set
sum(is.na(working)) # None
```

```
## [1] 0
```

```
# Convert integers to numerics
working <- mutate_all(working, function(x) {if(is.integer(x)) as.numeric(x) else x})
test <- mutate_all(testRaw, function(x) {if(is.integer(x)) as.numeric(x) else x})

# Check for NAs in test set keepVars
sum(is.na(select(test, keepVars)))
```

```
## [1] 0
```

# Variable reduction

Many of the variables left over after data cleaning are highly correlated (eg an acceleromater might provide acceleration values in three directions plus a total acceleration all at the same location). To prevent over-fitting, we omit highly-correlated variables. For this analysis, we consider the threshold correlation to be 50%.

We start by looking for near-zero variance variables, of which there are none. We then look for variables that are highly correlated with the independent variable of interest, classe; there were none of those, either.

```
# ========================================
# VARIABLE REDUCTION
# ========================================
# Check for near-zero variance variables
sum(nearZeroVar(working, saveMetrics=TRUE)[,4])
```

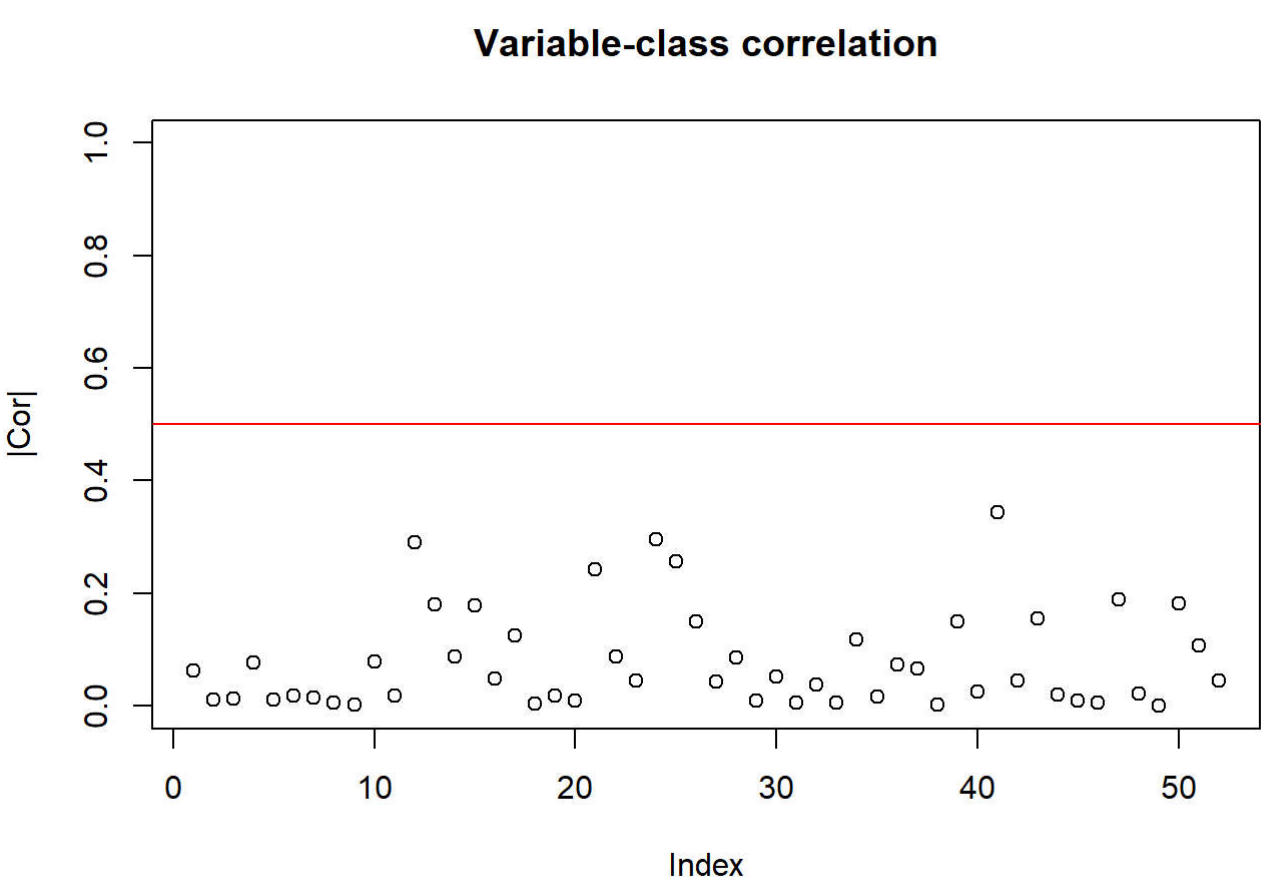
```
## [1] 0
```

```
# Check for high correlation variables to classe
workingNum <- working
workingNum$classe <- as.numeric(workingNum$classe)
max(abs(cor(workingNum)[1:ncol(workingNum) - 1,ncol(workingNum)])) # < 0.5
```

```
## [1] 0.3438258
```

The figure below shows that none of the correlations fall above the threshold line:

```
plot(abs(cor(workingNum)[1:(ncol(workingNum) - 1),ncol(workingNum)]), ylim=c(0,1),
     main="Variable-class correlation", ylab="|Cor|")
abline(h=0.5, col="red")
```



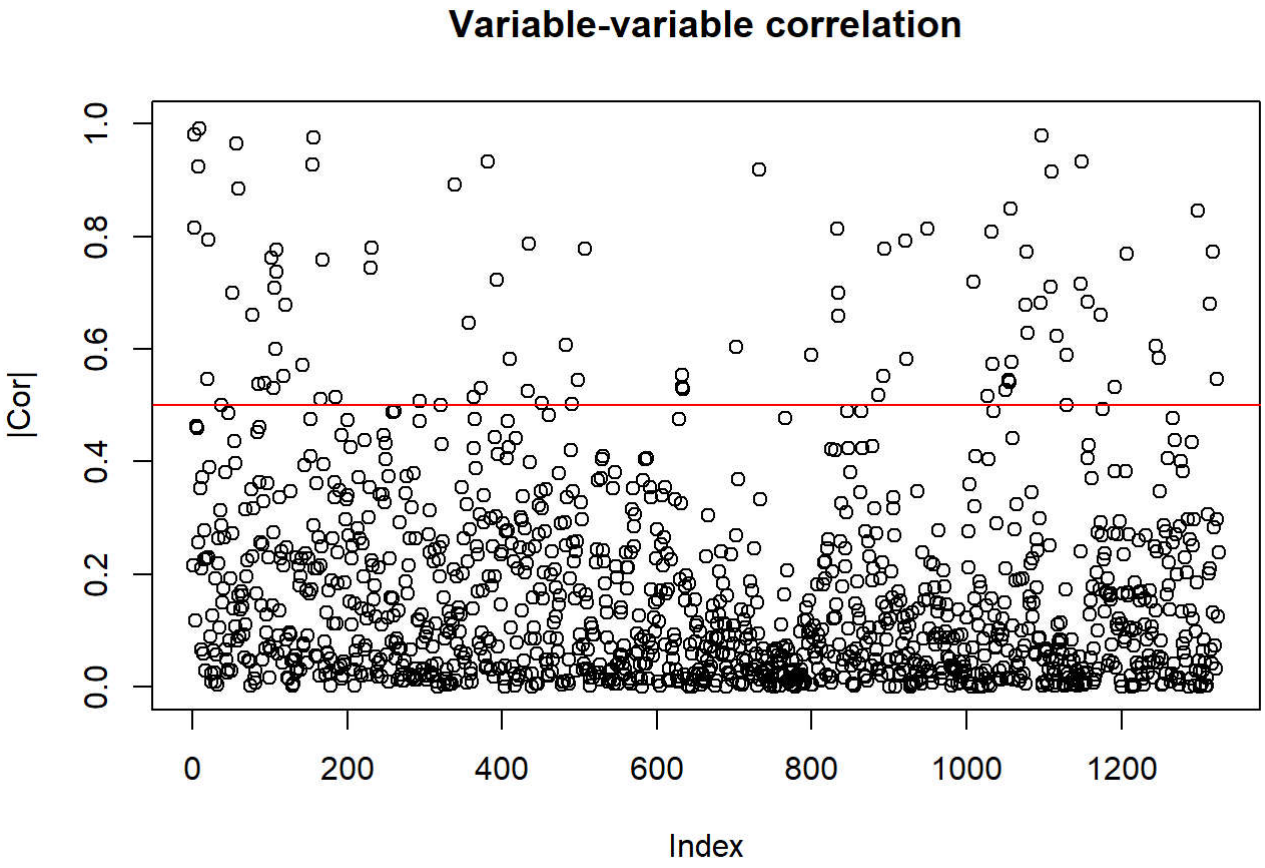

Next, we look for variables that are highly correlated with other variables, of which there were many. This is done by calculating the absolute value of the correlation matrix and setting the diagonal and upper triangle to zero.

```
# Check for high correlation between variables
varMat <- workingNum[, 1:(ncol(workingNum) - 1)]
corVarMat <- abs(cor(varMat))
corVarMat[!lower.tri(corVarMat)] <- 0
max(corVarMat) # > 0.5
```

```
## [1] 0.9920085
```

The plot below shows that many of the variables have correlations above the threshold:

```
plot(corVarMat[lower.tri(corVarMat)], ylim=c(0,1),
     main="Variable-variable correlation", ylab="|Cor|")
abline(h=0.5, col="red")
```



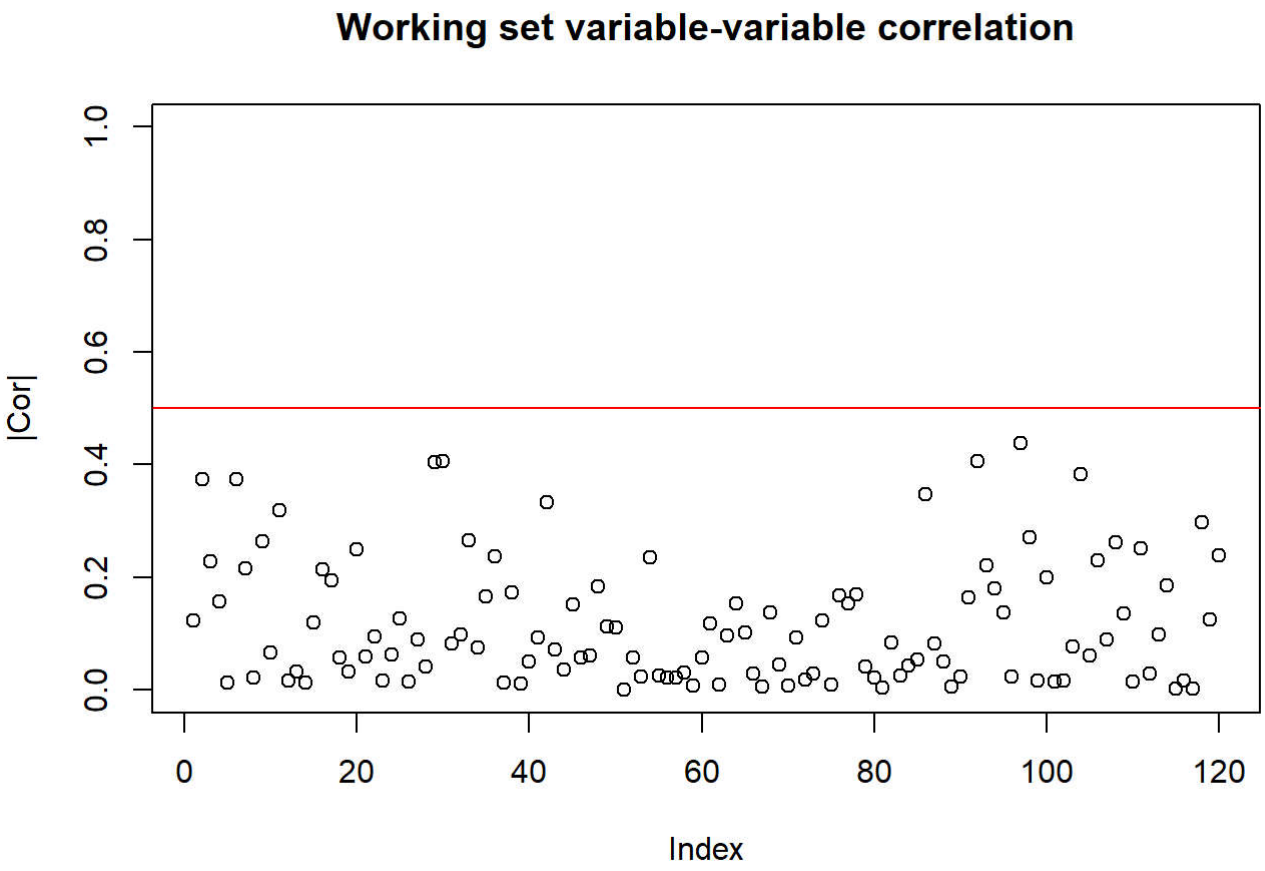

We proceed by subsetting the working data set to only include the variables with correlations less than the threshold to all other variables. We find these low correlation variables by grabbing only the column names of the lower triangle correlation matrix where every item is less than the threshold - if even one of the values is greater then that variable is not included since it is too highly correlated with something else.

```
# Find variables with low inter-varable correlation
corReqt <- 0.5
lowCorVars <- names(data.frame(corVarMat[, apply(corVarMat, 2, function(x) {all(x < corReq
t)})]))

# Subset working data set to only include lowCorVars
working <- select(working, lowCorVars, classe)
```

We are left with 16 variables. The plot below shows that none of these variables have correlations with the other variables greater than the threshold:

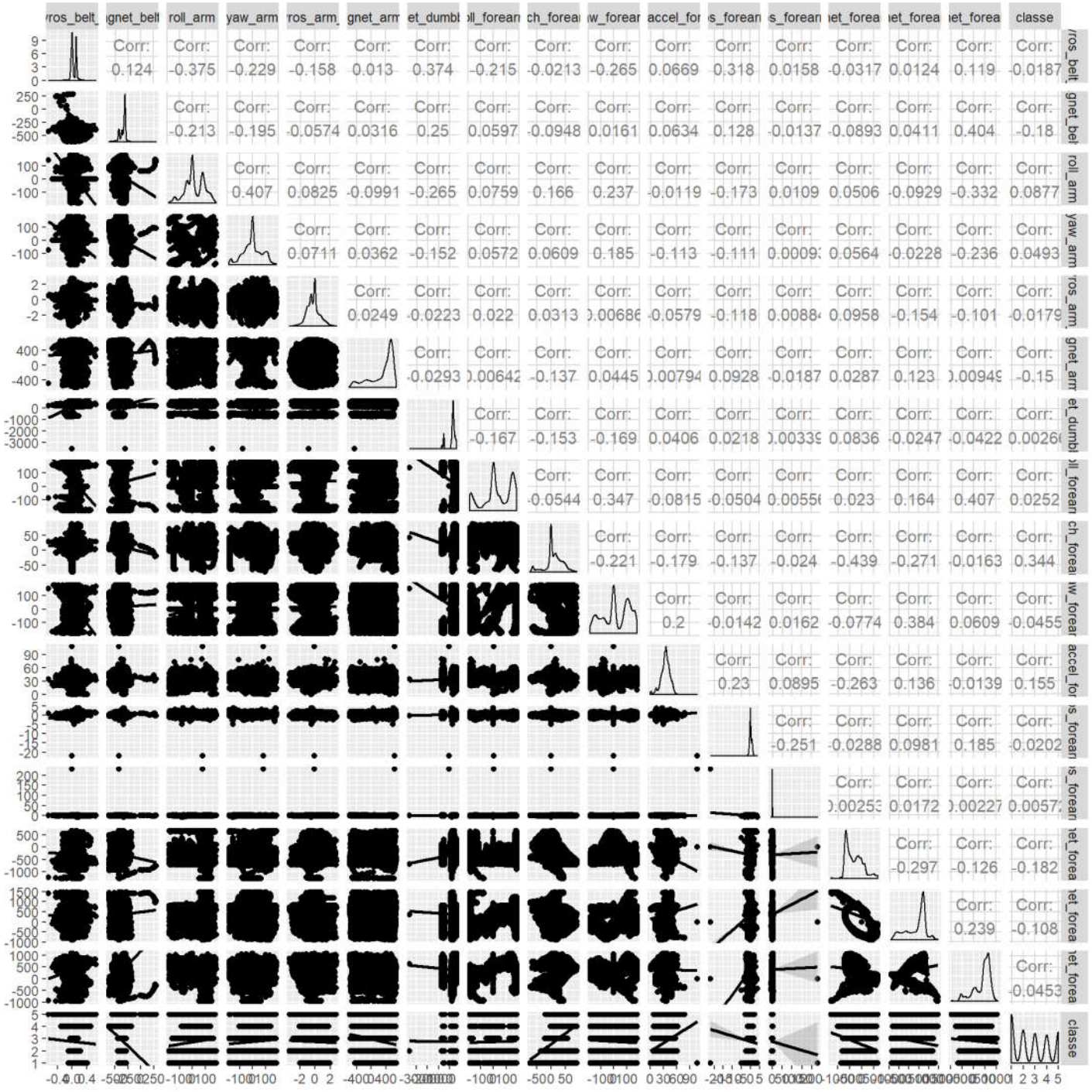
```
corVarWorkingMat <- abs(cor(working[, 1:(ncol(working) - 1)]))
corVarWorking <- corVarWorkingMat[lower.tri(corVarWorkingMat)]
plot(corVarWorking, ylim=c(0,1), main="Working set variable-variable correlation",
     ylab="|Cor|")
abline(h=0.5, col="red")
```




# Exploratory data analysis

With such a large set of potential coefficients, exploratory data analysis is a bit tricky to do. We can, however, create a pair plot showing the correlations, distributions, and densities of each variable compared with all of the other variables. It's a bit hard to read with so many variables, but do note that it confirms that all of the (absolute values of the) correlations are less than 0.50.

```
# ========================================
# EXPLORATORY DATA ANALYSIS
# ========================================
working2 <- working
working2$classe <- as.numeric(working2$classe)
ggpairs(data=working2, lower=list(continuous="smooth"))
```




# Train/test split

The cleaned and reduced working set can now be split into training and testing sets for cross-validation purposes - a random 75% of the original training set goes into the ‘training’ data frame and the remaining 25% goes into the ‘testing’ data frame.

```
# ======================================
# TRAIN/TEST SPLIT
# ======================================
set.seed(101)
inTrain <- createDataPartition(y=working$classe, p=0.75, list=F)
training <- working[inTrain,]
testing <- working[-inTrain,]
```

# Models

Refer to the appendix for the code used to fit the various models.

Several classification models were then fit to the training data set with 'classe' as the outcome and the remaining 16 variables as covariates:

- Decision tree
- Random forest (2-, 3, and 5-fold)
- Boosting (2- and 3-fold)
- LDA
- Naive-Bayes

The accuracies of each of the above models when applied to the testing data set are shown below:

```
##          tree  rf2  rf3  rf5 gbm2 gbm3  lda   nb
## Accuracy 46.9 98.1 98.1 98.1 85.2 85.8 42.7 63.8
```

A 2-fold random forest model provides excellent accuracy and is also efficient to run. To see if we can increase accuracy, we completed a principal component analyis on 2- and 3-fold random forests:

```
##          tree  rf2  rf3  rf5 gbm2 gbm3  lda   nb pcaRf2 pcaRf3
## Accuracy 46.9 98.1 98.1 98.1 85.2 85.8 42.7 63.8   95.6   95.7
```

Interestingly, both PCA analyses have lower accuracies than the random forest models; it appears that the 2-fold random forest model is the optimal solution. For due diligence, however, let's run a stacked model which stacks a 2-fold random forest model and a 2-fold boosted model:

```
##          tree  rf2  rf3  rf5 gbm2 gbm3  lda   nb pcaRf2 pcaRf3
## Accuracy 46.9 98.1 98.1 98.1 85.2 85.8 42.7 63.8   95.6   95.7
##          stackRf2Gbm2
## Accuracy         47.5
```

Although unclear why, the stacked model is far worse in accuracy than the two individual models that it stacked. It is decided that the best model to use is a 2-fold random forest model.

A table of accuracies and out-of-sample errors (1-accuracy) is shown below:

```
##           tree  rf2  rf3  rf5 gbm2 gbm3  lda   nb pcaRf2 pcaRf3
## Accuracy  47.0 98.0 98.0 98.0 85.0 86.0 43.0 64.0   96.0   96.0
## OOS Error 53.1  1.9  1.9  1.9 14.8 14.2 57.3 36.2    4.4    4.3
##           stackRf2Gbm2
## Accuracy          48.0
## OOS Error         52.5
```

# Predict on test set

Using a 2-fold random forest model, let's predict the classe of the 20 observations in the provided test set:

```
# ---- PREDICT ON TEST SET ----
# 2 fold
predRf2_test <- predict(fitRf2, newdata=test)
```

# Appendix

```
# ========================================
# ANALYSIS
# ========================================

# ---- MODEL FITTING ----

# ---- decision tree ----
set.seed(101)
fitRpart <- train(classe ~ ., data=training, method="rpart")
predRpart <- predict(fitRpart, newdata=testing)

# ---- random forest ----
# fitRpart <- train(classe ~ ., data=training, method="rf", prox=T)

# 2 fold
number <- 2
cluster <- makeCluster(detectCores() - 1)
registerDoParallel(cluster)
set.seed(101)
fitRf2 <- train(classe ~ ., data=training, method="rf",
    trControl=trainControl(method="cv", number=number, allowParallel=T))
stopCluster(cluster)
registerDoSEQ()
predRf2 <- predict(fitRf2, newdata=testing)

# 3 fold
number <- 3
cluster <- makeCluster(detectCores() - 1)
registerDoParallel(cluster)
set.seed(101)
fitRf3 <- train(classe ~ ., data=training, method="rf",
                trControl=trainControl(method="cv", number=number, allowParallel=T))
stopCluster(cluster)
registerDoSEQ()
predRf3 <- predict(fitRf3, newdata=testing)

# 5 fold
number <- 5
cluster <- makeCluster(detectCores() - 1)
registerDoParallel(cluster)
set.seed(101)
fitRf5 <- train(classe ~ ., data=training, method="rf",
                trControl=trainControl(method="cv", number=number, allowParallel=T))
stopCluster(cluster)
registerDoSEQ()
predRf5 <- predict(fitRf5, newdata=testing)

# ---- boosting ----

# 2 fold
number <- 2
cluster <- makeCluster(detectCores() - 1)
registerDoParallel(cluster)
```

```
set.seed(101)
fitGbm2 <- train(classe ~ ., data=training, method="gbm", verbose=F,
    trControl=trainControl(method="cv", number=number, allowParallel=T))
stopCluster(cluster)
registerDoSEQ()
predGbm2 <- predict(fitGbm2, newdata=testing)

# 3 fold
number <- 3
cluster <- makeCluster(detectCores() - 1)
registerDoParallel(cluster)
set.seed(101)
fitGbm3 <- train(classe ~ ., data=training, method="gbm", verbose=F,
                 trControl=trainControl(method="cv", number=number, allowParallel=T))
stopCluster(cluster)
registerDoSEQ()
predGbm3 <- predict(fitGbm3, newdata=testing)

# ---- LDA ----
set.seed(101)
fitLda <- train(classe ~ ., data=training, method="lda")
predLda <- predict(fitLda, testing)

# ---- Naive-Bayes ----
cluster <- makeCluster(detectCores() - 1)
registerDoParallel(cluster)
set.seed(101)
fitNb <- train(classe ~ ., data=training, method="nb")
stopCluster(cluster)
registerDoSEQ()
predNb <- predict(fitNb, testing)

# ---- Accuracies on test set ----
accDF <- data.frame(
    tree=confusionMatrix(predRpart, testing$classe)$overall["Accuracy"],
    rf2=confusionMatrix(predRf2, testing$classe)$overall["Accuracy"],
    rf3=confusionMatrix(predRf3, testing$classe)$overall["Accuracy"],
    rf5=confusionMatrix(predRf5, testing$classe)$overall["Accuracy"],
    gbm2=confusionMatrix(predGbm2, testing$classe)$overall["Accuracy"],
    gbm3=confusionMatrix(predGbm3, testing$classe)$overall["Accuracy"],
    lda=confusionMatrix(predLda, testing$classe)$overall["Accuracy"],
    nb=confusionMatrix(predNb, testing$classe)$overall["Accuracy"])

accDF

# ---- pca ----

# random forest, 2 fold
number <- 2
set.seed(101)
pca <- preProcess(select(training, -classe), method="pca")
trainPca <- predict(pca, select(training, -classe))
cluster <- makeCluster(detectCores() - 1)
registerDoParallel(cluster)
```

```
set.seed(101)
fitPcaRf2 <- train(x=trainPca, y=training$classe, method="rf",
    trControl=trainControl(method="cv", number=number, allowParallel=T))
stopCluster(cluster)
registerDoSEQ()
testPcaRf <- predict(pca, select(testing, -classe))
predPcaRf2 <- predict(fitPcaRf2, testPcaRf)

# random forest, 3 fold
number <- 3
set.seed(101)
pca <- preProcess(select(training, -classe), method="pca")
trainPca <- predict(pca, select(training, -classe))
cluster <- makeCluster(detectCores() - 1)
registerDoParallel(cluster)
set.seed(101)
fitPcaRf3 <- train(x=trainPca, y=training$classe, method="rf",
                   trControl=trainControl(method="cv", number=number, allowParallel=T))
stopCluster(cluster)
registerDoSEQ()
testPcaRf <- predict(pca, select(testing, -classe))
predPcaRf3 <- predict(fitPcaRf3, testPcaRf)

# ---- Update accuracy table ----
accDF$pcaRf2 <- confusionMatrix(predPcaRf2, testing$classe)$overall["Accuracy"]
accDF$pcaRf3 <- confusionMatrix(predPcaRf3, testing$classe)$overall["Accuracy"]
accDF

# ---- ENSEMBLE ----

# Train/test/validation split
set.seed(101)
inValidation <- createDataPartition(y=working$classe, p=0.20, list=F)
validation <- working[inValidation,]
workingV <- working[-inValidation,]
set.seed(101)
inTrainV <- createDataPartition(y=workingV$classe, p=0.75, list=F)
trainingV <- workingV[inTrainV,]
testingV <- workingV[-inTrainV,]

# rf2 + gbm2
# rf2
number <- 2
cluster <- makeCluster(detectCores() - 1)
registerDoParallel(cluster)
set.seed(101)
fitRf2_2 <- train(classe ~ ., data=trainingV, method="rf",
                  trControl=trainControl(method="cv", number=number, allowParallel=T))
stopCluster(cluster)
registerDoSEQ()
predRf2_2 <- predict(fitRf2_2, newdata=testingV)
predRf2_2V <- predict(fitRf2_2, newdata=validation)

# gbm2
```

```
number <- 2
cluster <- makeCluster(detectCores() - 1)
registerDoParallel(cluster)
set.seed(101)
fitGbm2_2 <- train(classe ~ ., data=trainingV, method="gbm", verbose=F,
                   trControl=trainControl(method="cv", number=number, allowParallel=T))
stopCluster(cluster)
registerDoSEQ()
predGbm2_2 <- predict(fitGbm2_2, newdata=testingV)
predGbm2_2V <- predict(fitGbm2_2, newdata=validation)

# Stack
predDFRf2Gbm2 <- data.frame(predRf2_2, predGbm2_2, classe=testingV$classe)
fitStackRf2Gbm2 <- train(classe ~ ., data=predDFRf2Gbm2, method="gam")
predDFRf2Gbm2V <- data.frame(predRf2_2=predRf2_2V, predGbm2_2=predGbm2_2V)
predRf2Gbm2V <- predict(fitStackRf2Gbm2, predDFRf2Gbm2V)

# ---- Update accuracy table ----
accDF$stackRf2Gbm2 <- confusionMatrix(predRf2Gbm2V, validation$classe)$overall["Accuracy"]
```