Practical Machine Learning - Course Project

Evan Falcone

14 February 2017

Contents

Introduction

Background

Using devices such as Jawbone Up, Nike FuelBand, and Fitbit it is now possible to collect a large amount of data about personal activity relatively inexpensively. These type of devices are part of the quantified self movement - a group of enthusiasts who take measurements about themselves regularly to improve their health, to find patterns in their behavior, or because they are tech geeks. One thing that people regularly do is quantify how much of a particular activity they do, but they rarely quantify how well they do it. In this project, your goal will be to use data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants. They were asked to perform barbell lifts correctly and incorrectly in 5 different ways. More information is available from the website here (see the section on the Weight Lifting Exercise Dataset).

Data

The training data for this project are available here.

The test data are available here.

The data for this project come from this source. If you use the document you create for this class for any purpose please cite them as they have been very generous in allowing their data to be used for this kind of assignment.

Goal

The goal of your project is to predict the manner in which they did the exercise. This is the "classe" variable in the training set. You may use any of the other variables to predict with. You should create a report describing how you built your model, how you used cross validation, what you think the expected out of sample error is, and why you made the choices you did. You will also use your prediction model to predict 20 different test cases.

Getting and Loading the Data

```r
library(caret)
```

```
## Loading required package: lattice

## Loading required package: ggplot2
```

```r
library(rpart)
library(rpart.plot)
library(RColorBrewer)
library(rattle)
```

```
## R session is headless; GTK+ not initialized.

## Rattle: A free graphical interface for data science with R.
## Version 5.1.0 Copyright (c) 2006-2017 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

```r
library(randomForest)
```

```
## randomForest 4.6-12

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:rattle':
##
##     importance

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```r
library(knitr)

# set.seed(12345) # test seed, compare to bookmark
set.seed(12345)

trainUrl <- "http://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
testUrl <- "http://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"

training <- read.csv(url(trainUrl), na.strings=c("NA","#DIV/0!",""))
testing <- read.csv(url(testUrl), na.strings=c("NA","#DIV/0!",""))
```

Partition the test and training sets using the caret package createDataPartition function:

```r
inTrain <- createDataPartition(training$classe, p=0.6, list=FALSE)
myTraining <- training[inTrain, ]
myTesting <- training[-inTrain, ]
# print the dimensions of myTraining and myTesting for exploratory purposes:
dim(myTraining); dim(myTesting)
```

```
## [1] 11776   160

## [1] 7846  160
```

Data Cleaning/Prep

Remove Near-Zero variance variables:

```r
nzv <- nearZeroVar(myTraining, saveMetrics=TRUE)
myTraining <- myTraining[,nzv$nzv==FALSE]

nzv<- nearZeroVar(myTesting,saveMetrics=TRUE)
myTesting <- myTesting[,nzv$nzv==FALSE]

rm(nzv)
```

Remove columns with more than 80% entries as NAs:

```r
# Here we get the indexes of the columns having at least 80% of NA or blank values on the training data
indColToRemove <- which(colSums(is.na(myTraining) | myTraining == "") > 0.8 * dim(myTraining)[1])
# Remove the columns with 80% NAs
trainNa <- myTraining[,-indColToRemove]
# Remove the first seven columns (timestampes, unused in our analysis)
trainNa <- trainNa[,-c(1:7)]
dim(trainNa)
```

```
## [1] 11776    52
```

Perform the same cleaning on the test sets as on the training data:

```r
# Here we get the indexes of the columns having at least 80% of NA or blank values on the testing datas
indColToRemove <- which(colSums(is.na(myTesting) | myTesting == "") > 0.8 * dim(myTesting)[1])
# Remove the columns with 80% NAs
testNa <- myTesting[,-indColToRemove]
# Remove the first column from test (unused in our analysis)
testNa <- testNa[,-1]
dim(testNa)
```

```
## [1] 7846   58
```

```r
inTrainNew <- createDataPartition(trainNa$classe, p=0.75, list=FALSE)
trainNew <- trainNa[inTrainNew,]
testNew <- trainNa[-inTrainNew,]
dim(trainNew)
```

```
## [1] 8834   52
```

```r
dim(testNew)
```

```
## [1] 2942   52
```

In the following sections, we will test 3 different models:

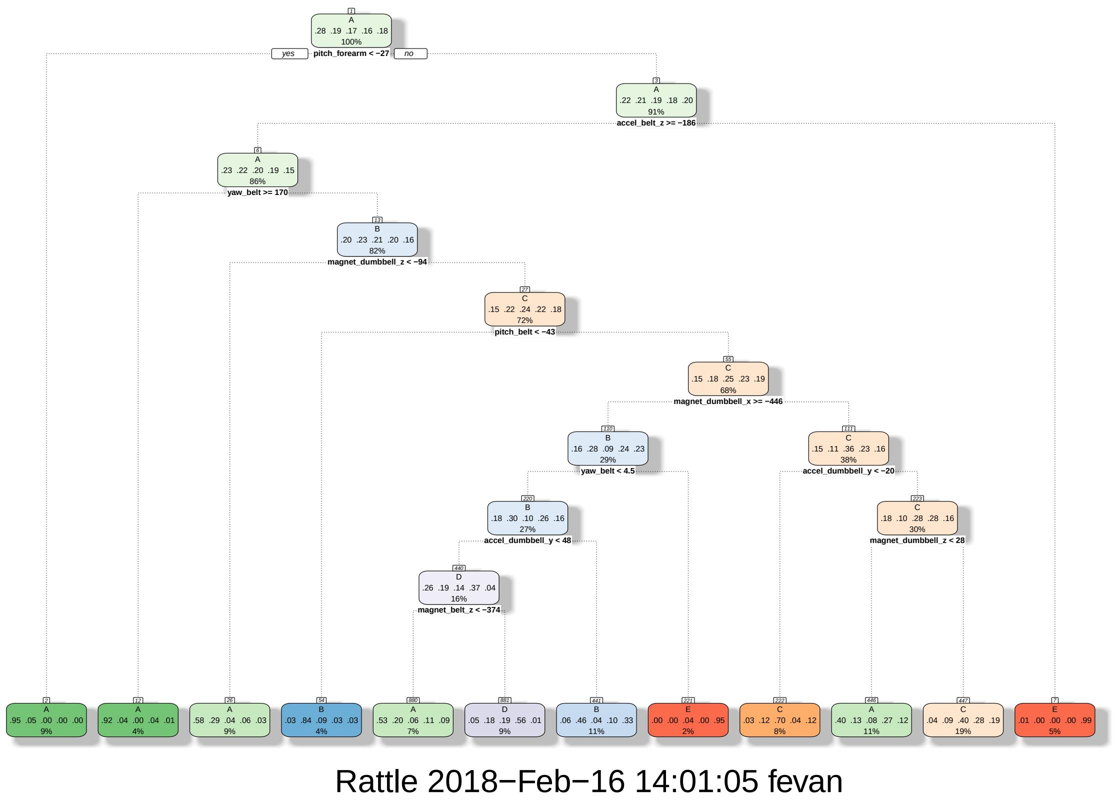
1. Classification Tree
2. Random Forest
3. Gradient Boosted Method

In order to limit the effects of overfitting, and improve the efficicency of the models, we will use cross-validation: we will use 5 folds (usually, 5 or 10 can be used, but 10 folds gives higher run times with no significant increase of the accuracy).

Classification Trees model

Train with classification tree

```
trControl <- trainControl(method="cv", number=5)
model_CT <- train(classe~., data=trainNew, method="rpart", trControl=trControl)

# Print your model_CT:
fancyRpartPlot(model_CT$finalModel)
```



```
# Predict using the model from trainNew on testNew & create the Confusion Matrix:
trainPred <- predict(model_CT,newdata=testNew)
confMatCT <- confusionMatrix(testNew$classe,trainPred)

# Display confusion matrix:
confMatCT$table
```

```
##           Reference
## Prediction   A   B   C   D   E
##          A 770  10  37  16   4
##          B 175 273  75  46   0
##          C  51  24 372  58   8
##          D 136  36 159 151   0
##          E  72 111 121   2 235
```

```
# Display model accuracy:
confMatCT$overall[1]
```

```
##  Accuracy
## 0.6121686
```

The accuracy of the Classification Tree model is low (approximately ~59%). This implies that the outcome will be poorly predicted by the other predictors.

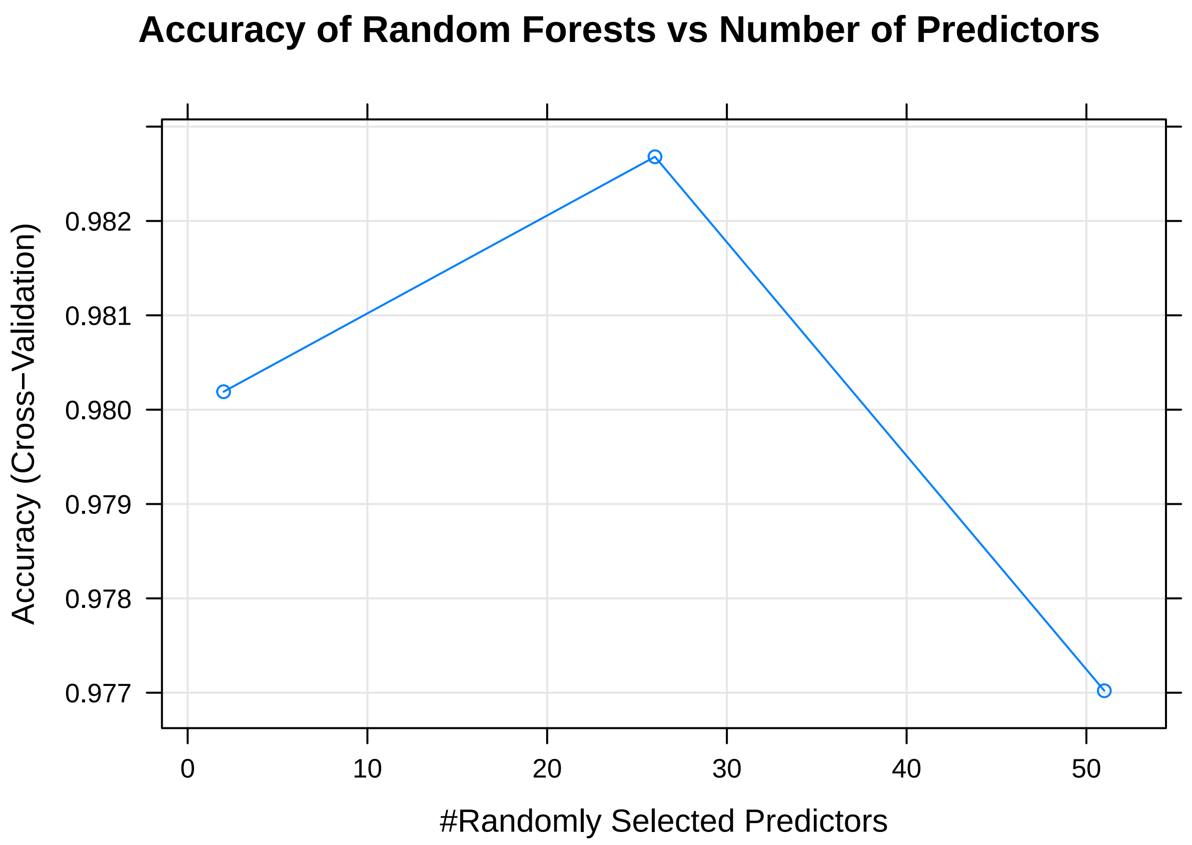
Random Forests model

Train with Random Forests

```
model_RF <- train(classe~., data=trainNew, method="rf", trControl=trControl, verbose=FALSE)

# Print your model_RF:
print(model_RF)
```

```
## Random Forest
##
## 8834 samples
##   51 predictors
##    5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 7068, 7067, 7067, 7066, 7068
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9801904  0.9749366
##   26    0.9826806  0.9780888
##   51    0.9770211  0.9709276
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 26.
```

```
# Plot your model_RF:
plot(model_RF, main = "Accuracy of Random Forests vs Number of Predictors")
```



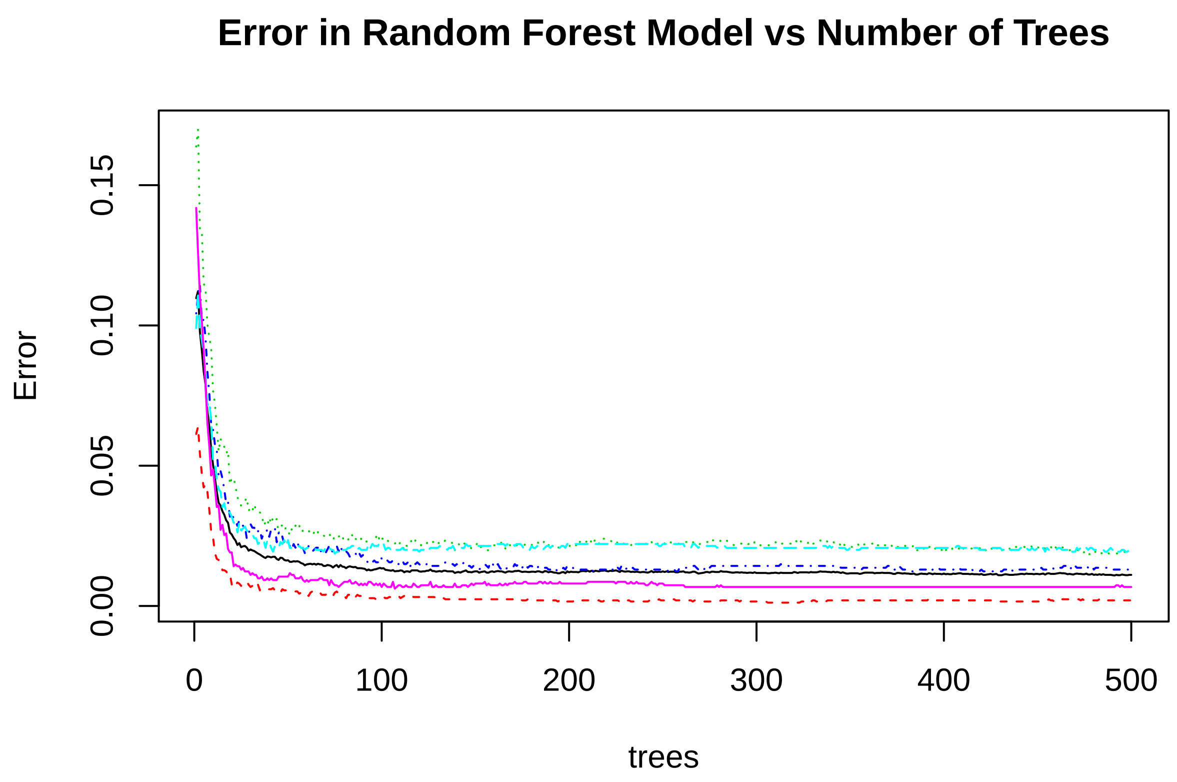
```r
# Predict using the model from trainNew on testNew & create the Confusion Matrix:
trainPred <- predict(model_RF,newdata=testNew)
confMatRF <- confusionMatrix(testNew$classe,trainPred)

# Display confusion matrix:
confMatRF$table
```

```
##             Reference
## Prediction    A   B   C   D   E
##            A 837   0   0   0   0
##            B   5 562   2   0   0
##            C   0   0 511   2   0
##            D   0   0   9 473   0
##            E   0   0   2   2 537
```

```r
# Display model accuracy:
confMatRF$overall[1]
```

```
##  Accuracy
## 0.9925221
```

```r
# Plot the Random Forest
plot(model_RF$finalModel, main = "Error in Random Forest Model vs Number of Trees")
```



```r
# Display variable importance:
(MostImpVars <- varImp(model_RF))
```

```
## rf variable importance
##
##   only 20 most important variables shown (out of 51)
##
##                        Overall
## yaw_belt                100.00
## pitch_forearm            87.77
## pitch_belt               69.29
## magnet_dumbbell_z        65.55
## magnet_dumbbell_y        51.89
## roll_forearm             46.77
## accel_belt_z             43.27
## magnet_belt_z            29.81
## magnet_belt_y            29.14
## magnet_dumbbell_x        28.90
## roll_dumbbell            27.97
## accel_dumbbell_y         27.09
## gyros_belt_z             26.68
## accel_forearm_x          23.87
## accel_dumbbell_z         19.94
## magnet_belt_x            17.51
## magnet_forearm_z         16.84
## total_accel_dumbbell     16.08
## yaw_arm                  14.16
## roll_arm                 13.84
```

```
# MostImpVars
```

The accuracy of the Random Forests model is very high (~99%) using 5-fold cross-validation. This implies that the outcome will be very well predicted by the predictors.

We also see that the optimal number of predictors - say, the number of predictors giving the highest accuracy - is mtry = **26**. The accuracy not being significantly worse with all the available predictors suggests that there may be some dependencies between predictors. Using 27+ trees does not reduce the error significantly (as seen by the downword slope in Accuracy vs # of Randomly Selected Predictors).

Gradient Boosting model

Train with Gradient Boosting

```
model_GBM <- train(classe~., data=trainNew, method="gbm", trControl=trControl, verbose=FALSE)
```

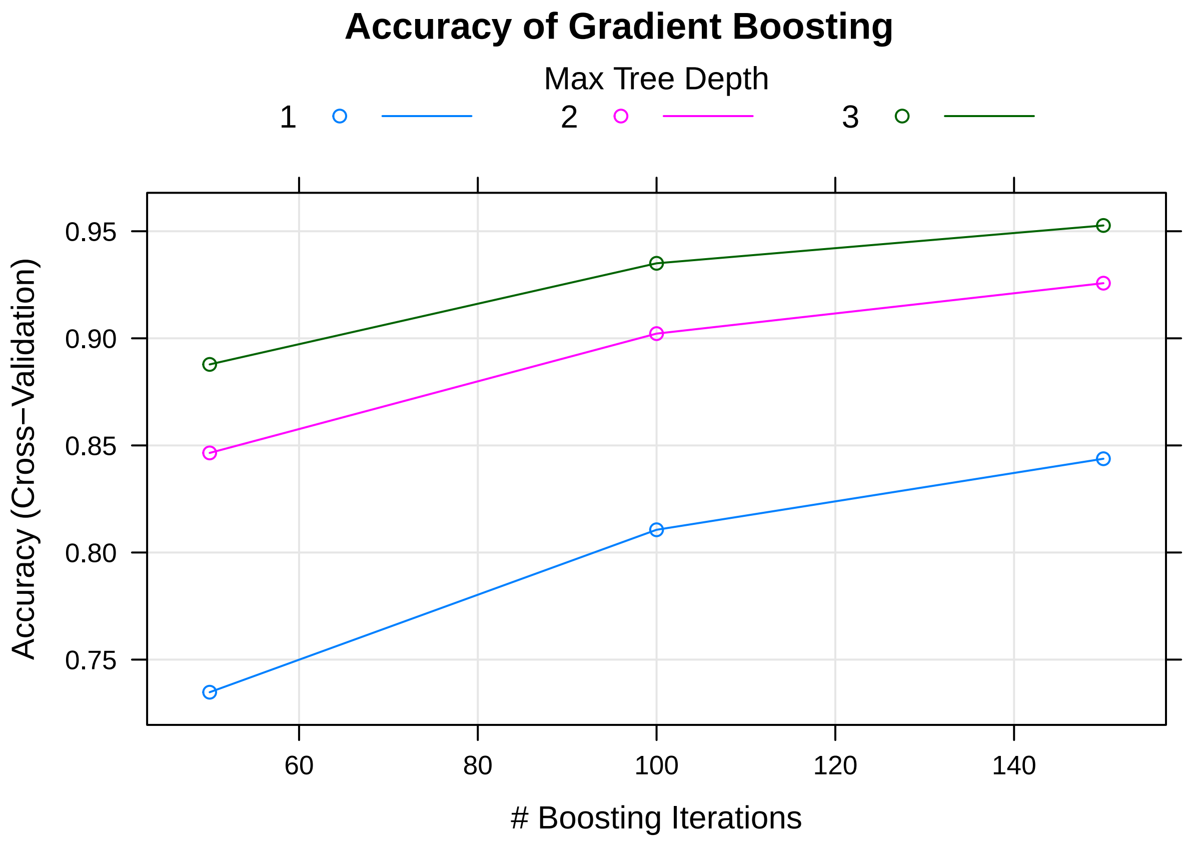
```
## Loading required package: survival

##
## Attaching package: 'survival'

## The following object is masked from 'package:caret':
##
##     cluster

## Loading required package: splines

## Loading required package: parallel

## Loaded gbm 2.1.3
```

```
# Print your model_RF:
print(model_GBM)
```

```
## Stochastic Gradient Boosting
##
## 8834 samples
##   51 predictors
##    5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 7067, 7067, 7068, 7066, 7068
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy   Kappa
##   1                   50      0.7347725  0.6639314
##   1                  100      0.8106174  0.7602934
##   1                  150      0.8437849  0.8022652
##   2                   50      0.8465007  0.8054594
##   2                  100      0.9021970  0.8761847
##   2                  150      0.9257429  0.9060167
##   3                   50      0.8878214  0.8579249
##   3                  100      0.9350236  0.9177757
##   3                  150      0.9526832  0.9401384
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
```

```
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## Accuracy was used to select the optimal model using  the largest value.
## The final values used for the model were n.trees = 150,
##  interaction.depth = 3, shrinkage = 0.1 and n.minobsinnode = 10.
```

```
# Plot your model_RF:
plot(model_GBM, main = "Accuracy of Gradient Boosting")
```



```
# Predict using the model from trainNew on testNew & create the Confusion Matrix:
trainPred <- predict(model_GBM,newdata=testNew)
confMatGBM <- confusionMatrix(testNew$classe,trainPred)

# Display confusion matrix:
confMatGBM$table
```

```
##             Reference
## Prediction   A   B   C   D   E
##          A 822   9   4   1   1
##          B  16 528  18   2   5
##          C   0  13 496   4   0
##          D   2   1  16 456   7
##          E   1   7   9   9 515
```

```
# Display model accuracy:
confMatGBM$overall[1]
```

```
##  Accuracy
## 0.9575119
```

The accuracy of the Gradient Boosting model is also high (~95%) using 5-fold cross-validation This implies that the outcome will be very well predicted by the predictors, but is still shy of the accuracy of the Random Forests model.

Conclusion

Of the three models evaluated, notably Classification Trees, Random Forest and Gradient Boosting, the Random Forest model's performance leads to the lowest error in prediction (and highest predictive ability with an accuracy of ~99%) using 5-fold cross-validation to prevent over-fitting. The Gradient Boosting model has the next lowest error with an accuracy of ~95%. Lastly, the Classification Trees has the weakest accuracy of three.