Practical Machine Learning Course Project

Lena Horsley

Background

The dataset used in this project was collected during an experiment measuring exercise activity via sensors placed in and on weight-lifting equipment. There were five different activities performed by the study subjects:

- Class A: perform a single biceps curl as directed
- Class B: move the elbow forward while performing a single curl
- Class C: single partial curl – starting with the arm fully extended, the arm is raised 90 degrees
- Class D: single partial curl – starting with the arm flexed, the arm is lowed 90 degrees
- Class E: leaning back (hips forward)

This project looks at several prediction models in addition to using the provided datasets (training and testing) to determine the manner of exercise.

For more information, please refer to the Weight Lifting Exercises Dataset section of the Human Activity Recognition website. You can also download the assocaited publication Qualitative Activity Recognition of Weight Lifting Exercises.

Prepare the data

Step 1. Load the libraries and read in the data. Get the dimensions of the raw training and testing sets.

```
suppressPackageStartupMessages(library(caret))
suppressPackageStartupMessages(library(gbm))
suppressPackageStartupMessages(library(ggplot2))
suppressPackageStartupMessages(library(randomForest))
suppressPackageStartupMessages(library(rattle))
suppressPackageStartupMessages(library(corrplot))
suppressPackageStartupMessages(library(kernlab))
suppressPackageStartupMessages(library(rpart))
suppressPackageStartupMessages(library(rpart.plot))
suppressPackageStartupMessages(library(knitr))

trainingRaw <- read.csv("./data/pml-training.csv")
testingRaw <- read.csv("./data/pml-testing.csv")
dim(trainingRaw)

## [1] 19622   160

dim(testingRaw)

## [1]  20 160
```

Step 2. Determine the number of NA values, replace "#DIV/0! with NA and remove them. Check the number of NA values again. There should be no NA values in the training and testing sets

```
sum(is.na(trainingRaw))

## [1] 1287472

sum(is.na(testingRaw))

## [1] 2000

trainingRaw <- replace(trainingRaw, trainingRaw == "#DIV/0!", NA)
testingRaw <- replace(testingRaw, testingRaw == "#DIV/0!", NA)

trainingRaw <- trainingRaw[, colSums(is.na(trainingRaw)) == 0]
testingRaw <- testingRaw[, colSums(is.na(testingRaw)) == 0]

sum(is.na(trainingRaw))

## [1] 0

sum(is.na(testingRaw))

## [1] 0
```

Step 3. Remove the first seven columns from the training and testing sets (not needed to create the model).Check the dimensions (should be very different than the original data sets)

```
cleanedTrainingData <- trainingRaw[,-c(1:7)]
cleanedTestingData <- testingRaw[,-c(1:7)]

dim(cleanedTrainingData)

## [1] 19622    53

dim(cleanedTestingData)

## [1] 20 53
```

Step 4. Prepare the training data. Partition the training data into training (70%) and validation sets (30%).

```
set.seed(7779311)
inTrain <- createDataPartition(cleanedTrainingData$classe, p=0.7, list=FALSE)
myTrainingData <- cleanedTrainingData[inTrain,]
myValidationData <- cleanedTrainingData[-inTrain,]
dim(myTrainingData)

## [1] 13737    53

dim(myValidationData)
```

```
## [1] 5885   53
```

Finding the appropriate model

We'll look at three models:

- GBM
- Random Forest
- Decision Tree

The GBM model

```
myControl <- trainControl(method="cv", number=5)
myGbmModel <- train(classe~.,
                    data=myTrainingData,
                    method="gbm",
                    trControl=myControl,
                    verbose=FALSE)
myGbmModel
```

```
## Stochastic Gradient Boosting
##
## 13737 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 10989, 10991, 10989, 10990, 10989
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy   Kappa
##   1                   50      0.7499447  0.6827869
##   1                  100      0.8217947  0.7744249
##   1                  150      0.8536801  0.8148527
##   2                   50      0.8547717  0.8160140
##   2                  100      0.9069674  0.8822479
##   2                  150      0.9316446  0.9134850
##   3                   50      0.8938636  0.8656064
##   3                  100      0.9391427  0.9229934
##   3                  150      0.9608354  0.9504551
##
## 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.
```

Using the validation set, determine the performance of the GBM model.

```r
myGbmModelPrediction <- predict(myGbmModel,myValidationData)
gbmCm <- confusionMatrix(myValidationData$classe, myGbmModelPrediction)
gbmCm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1654   14    4    1    1
##          B   36 1075   25    2    1
##          C    0   31  985    7    3
##          D    1    6   39  911    7
##          E    1    8    7   16 1050
##
## Overall Statistics
##
##                Accuracy : 0.9643
##                  95% CI : (0.9593, 0.9689)
##     No Information Rate : 0.2875
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9548
##
##  Mcnemar's Test P-Value : 3.824e-07
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9775   0.9480   0.9292   0.9723   0.9887
## Specificity            0.9952   0.9865   0.9915   0.9893   0.9934
## Pos Pred Value         0.9881   0.9438   0.9600   0.9450   0.9704
## Neg Pred Value         0.9910   0.9876   0.9846   0.9947   0.9975
## Prevalence             0.2875   0.1927   0.1801   0.1592   0.1805
## Detection Rate         0.2811   0.1827   0.1674   0.1548   0.1784
## Detection Prevalence   0.2845   0.1935   0.1743   0.1638   0.1839
## Balanced Accuracy      0.9864   0.9673   0.9604   0.9808   0.9910
```

Random Forest model

```r
myRfModel <- train(classe~.,
                   data=myTrainingData,
                   method="rf",
                   trControl=myControl,
                   verbose=FALSE)
myRfModel
```

```
## Random Forest
##
## 13737 samples
```

```
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 10989, 10989, 10990, 10990, 10990
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9908277  0.9883964
##   27    0.9911916  0.9888569
##   52    0.9858049  0.9820423
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 27.
```

Using the validation set, determine the performance of the Random Forest model.

```
myRfModelPrediction <- predict(myRfModel,myValidationData)
rfCm <- confusionMatrix(myValidationData$classe, myRfModelPrediction)
rfCm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1674    0    0    0    0
##          B   12 1123    4    0    0
##          C    0    4 1019    3    0
##          D    0    0   19  944    1
##          E    0    0    2    1 1079
##
## Overall Statistics
##
##                Accuracy : 0.9922
##                  95% CI : (0.9896, 0.9943)
##     No Information Rate : 0.2865
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9901
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9929   0.9965   0.9761   0.9958   0.9991
## Specificity            1.0000   0.9966   0.9986   0.9959   0.9994
## Pos Pred Value         1.0000   0.9860   0.9932   0.9793   0.9972
## Neg Pred Value         0.9972   0.9992   0.9949   0.9992   0.9998
## Prevalence             0.2865   0.1915   0.1774   0.1611   0.1835
```

```
## Detection Rate         0.2845   0.1908   0.1732   0.1604   0.1833
## Detection Prevalence   0.2845   0.1935   0.1743   0.1638   0.1839
## Balanced Accuracy      0.9964   0.9965   0.9873   0.9959   0.9992
```

Decision Tree model using rpart

```
myRpartTreeModel <- rpart(classe~.,
                    data=myTrainingData,
                    method="class")
myRpartTreeModelPrediction <- predict(myRpartTreeModel,myValidationData, type
= "class")
rpartTreeCm <- confusionMatrix(myValidationData$classe,
myRpartTreeModelPrediction)
rpartTreeCm
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction    A    B    C    D    E
##          A 1515   39   45   62   13
##          B  170  599  189   67  114
##          C   22   57  877   63    7
##          D   47   57  130  667   63
##          E   10   61  141   78  792
##
## Overall Statistics
##
##                Accuracy : 0.7562
##                  95% CI : (0.745, 0.7671)
##     No Information Rate : 0.2997
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6914
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.8588   0.7368   0.6346   0.7118   0.8008
## Specificity            0.9614   0.8935   0.9669   0.9400   0.9408
## Pos Pred Value         0.9050   0.5259   0.8548   0.6919   0.7320
## Neg Pred Value         0.9409   0.9549   0.8961   0.9451   0.9590
## Prevalence             0.2997   0.1381   0.2348   0.1592   0.1681
## Detection Rate         0.2574   0.1018   0.1490   0.1133   0.1346
## Detection Prevalence   0.2845   0.1935   0.1743   0.1638   0.1839
## Balanced Accuracy      0.9101   0.8152   0.8007   0.8259   0.8708
```

Discussion

The low accuracy and high out of sample error were expected for the decision tree model, which is better suited for exploratory analysis. As seen in the table below, the random forest model had the highest accuracy and lowest out of sample error.

```
errorTable <- data.frame(
 c(
   as.numeric(gbmCm$overall['Accuracy']),
   as.numeric(gbmCm$overall['Kappa']),
   ((1 - as.numeric(gbmCm$overall['Accuracy']))*100)
   ),
  c(
    as.numeric(rfCm$overall['Accuracy']),
    as.numeric(rfCm$overall['Kappa']),
    ((1 - as.numeric(rfCm$overall['Accuracy']))*100)
    ),
  c(
    as.numeric(rpartTreeCm$overall['Accuracy']),
    as.numeric(rpartTreeCm$overall['Kappa']),
    ((1 - as.numeric(rpartTreeCm$overall['Accuracy']))*100)
    )
  )
colnames(errorTable) <- c(
  "GBM",
  "Random Forest",
  "Tree (rpart)"
  )
row.names(errorTable) <- c(
  "Accuracy",
  "Kappa",
  "Out of Sample Error"
  )
kable(errorTable)
```

	GBM	Random Forest	Tree (rpart)
Accuracy	0.9643161	0.9921835	0.7561597
Kappa	0.9548454	0.9901106	0.6913723
Out of Sample Error	3.5683942	0.7816483	24.3840272

Accuracy and out of sample error for the GBM model were close to the random forest model. The slightly lower accuracy and higher error may have been due to the sensor data noise (see section 5.2 Recognition Performance in the paper). Performance degradation due to noise is a known drawback of the GBM model.

As a test I decided to use both models to predict the answers to the quiz. The results (which are the same for both models) are below:

Random forest

```
quizRfAnswer <- predict(myRfModel, cleanedTestingData)
quizRfAnswer
```

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

GBM

```r
quizGbmAnswer <- predict(myGbmModel, cleanedTestingData)
quizGbmAnswer
```

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

Appendix

Decision tree (rpart)

```r
par(mfrow=c(1,2))
fancyRpartPlot(myRpartTreeModel)
```

```
## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```

```r
prp(myRpartTreeModel)
```