Prediction Assignment Writeup

Bradley Burquest

5/21/2021

Introduction

For this Prediction Assignment, we need to predict “how well” a exercise activity was done. The exercise activity to be determined is the sample bicep curl. The data was collect from several people wearing sensors arm, hip, dumbbell, and forearm. There are 5 classes (in the classe variable/outcome) that describe how well an activity was performed.

Implementation

The Approach

The approach is to use the training data to train 3 separate models that will be made using different methods. The chosen model methods are ‘rf’ (random forest), ‘gbm’ (gradient boosting), and ‘lda’(linear discriminant analysis). These models will be trained and tested. The model with the highest accuracy and low RMSE (root mean square error) will be used to predict the finaltest HAR classes.

Load the Data

Load the data from the source on the internet.

```
library(caret)
library(gbm)
library(dplyr)

set.seed(2112)
# Load the data from the web
training <- read.csv("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv")
finaltest <- read.csv("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv")

dim(training)
```

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

```
dim(finaltest)
```

```
## [1]  20 160
```

There are 160 variables and 19622 rows in the training data. The finaltest set has 160 variables and 20 rows. The finaltest data is to be used to test the accuracy of the model class predictions for this data. The predicted classes are to be used to answer the assignment quiz questions (not shown in this document).

A quick look at the data shows there are many columns that contain NA values or have extremely incomplete data. The data needs be cleaned-up before fitting the models. The same changes made to the training data must also be made to the finaltest data or test data.

Here the NA columns are removed and the first 7 columns of the resulting dataset are also removed as they are not significant.

```
# Get rid of columns that contain NA in the testing set because they are not useful
val_na_cols <-  finaltest %>% select_if(~any(is.na(.)))
# remove the NA columns from the final test data and training data
finalData <- finaltest %>% select(-colnames(val_na_cols))
# Remove the same NA testing columns from the training set
trainData <- training %>% select(-colnames(val_na_cols))
# Remove the first seven columns as they not significant
trainData <- subset(trainData, select=-c(1:7))
finalData <- subset(finalData, select=-c(1:7))
# convert the classe variable to a factor
trainData$classe <- as.factor(trainData$classe)

print(str(trainData))
```

```
## 'data.frame':    19622 obs. of  53 variables:
##  $ roll_belt           : num  1.41 1.41 1.42 1.48 1.48 1.45 1.42 1.42 1.43 1.45 ...
##  $ pitch_belt          : num  8.07 8.07 8.07 8.05 8.07 8.06 8.09 8.13 8.16 8.17 ...
##  $ yaw_belt            : num  -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 ...
##  $ total_accel_belt    : int  3 3 3 3 3 3 3 3 3 ...
##  $ gyros_belt_x        : num  0 0.02 0 0.02 0.02 0.02 0.02 0.02 0.02 0.03 ...
##  $ gyros_belt_y        : num  0 0 0 0 0.02 0 0 0 0 0 ...
##  $ gyros_belt_z        : num  -0.02 -0.02 -0.02 -0.03 -0.02 -0.02 -0.02 -0.02 -0.02 0 ...
##  $ accel_belt_x        : int  -21 -22 -20 -22 -21 -21 -22 -22 -20 -21 ...
##  $ accel_belt_y        : int  4 4 5 3 2 4 3 4 2 4 ...
##  $ accel_belt_z        : int  22 22 23 21 24 21 21 21 24 22 ...
##  $ magnet_belt_x       : int  -3 -7 -2 -6 -6 0 -4 -2 1 -3 ...
##  $ magnet_belt_y       : int  599 608 600 604 600 603 599 603 602 609 ...
##  $ magnet_belt_z       : int  -313 -311 -305 -310 -302 -312 -311 -313 -312 -308 ...
##  $ roll_arm            : num  -128 -128 -128 -128 -128 -128 -128 -128 -128 -128 ...
##  $ pitch_arm           : num  22.5 22.5 22.5 22.1 22.1 22 21.9 21.8 21.7 21.6 ...
##  $ yaw_arm             : num  -161 -161 -161 -161 -161 -161 -161 -161 -161 -161 ...
##  $ total_accel_arm     : int  34 34 34 34 34 34 34 34 34 34 ...
##  $ gyros_arm_x         : num  0 0.02 0.02 0.02 0 0.02 0 0.02 0.02 0.02 ...
##  $ gyros_arm_y         : num  0 -0.02 -0.02 -0.03 -0.03 -0.03 -0.03 -0.02 -0.03 -0.03 ...
##  $ gyros_arm_z         : num  -0.02 -0.02 -0.02 0.02 0 0 0 0 -0.02 -0.02 ...
##  $ accel_arm_x         : int  -288 -290 -289 -289 -289 -289 -289 -289 -288 -288 ...
##  $ accel_arm_y         : int  109 110 110 111 111 111 111 111 109 110 ...
##  $ accel_arm_z         : int  -123 -125 -126 -123 -123 -122 -125 -124 -122 -124 ...
##  $ magnet_arm_x        : int  -368 -369 -368 -372 -374 -369 -373 -372 -369 -376 ...
##  $ magnet_arm_y        : int  337 337 344 344 337 342 336 338 341 334 ...
##  $ magnet_arm_z        : int  516 513 513 512 506 513 509 510 518 516 ...
##  $ roll_dumbbell       : num  13.1 13.1 12.9 13.4 13.4 ...
##  $ pitch_dumbbell      : num  -70.5 -70.6 -70.3 -70.4 -70.4 ...
##  $ yaw_dumbbell        : num  -84.9 -84.7 -85.1 -84.9 -84.9 ...
##  $ total_accel_dumbbell: int  37 37 37 37 37 37 37 37 37 37 ...
##  $ gyros_dumbbell_x    : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ gyros_dumbbell_y    : num  -0.02 -0.02 -0.02 -0.02 -0.02 -0.02 -0.02 -0.02 -0.02 -0.02 ...
##  $ gyros_dumbbell_z    : num  0 0 0 -0.02 0 0 0 0 0 0 ...
##  $ accel_dumbbell_x    : int  -234 -233 -232 -232 -233 -234 -232 -234 -232 -235 ...
##  $ accel_dumbbell_y    : int  47 47 46 48 48 48 47 46 47 48 ...
##  $ accel_dumbbell_z    : int  -271 -269 -270 -269 -270 -269 -270 -272 -269 -270 ...
##  $ magnet_dumbbell_x   : int  -559 -555 -561 -552 -554 -558 -551 -555 -549 -558 ...
##  $ magnet_dumbbell_y   : int  293 296 298 303 292 294 295 300 292 291 ...
##  $ magnet_dumbbell_z   : num  -65 -64 -63 -60 -68 -66 -70 -74 -65 -69 ...
##  $ roll_forearm        : num  28.4 28.3 28.3 28.1 28 27.9 27.9 27.8 27.7 27.7 ...
##  $ pitch_forearm       : num  -63.9 -63.9 -63.9 -63.9 -63.9 -63.9 -63.9 -63.8 -63.8 -63.8 ...
##  $ yaw_forearm         : num  -153 -153 -152 -152 -152 -152 -152 -152 -152 -152 ...
##  $ total_accel_forearm : int  36 36 36 36 36 36 36 36 36 36 ...
##  $ gyros_forearm_x     : num  0.03 0.02 0.03 0.02 0.02 0.02 0.02 0.02 0.03 0.02 ...
##  $ gyros_forearm_y     : num  0 0 -0.02 -0.02 0 -0.02 0 -0.02 0 0 ...
##  $ gyros_forearm_z     : num  -0.02 -0.02 0 0 -0.02 -0.03 -0.02 0 -0.02 -0.02 ...
##  $ accel_forearm_x     : int  192 192 196 189 189 193 195 193 193 190 ...
##  $ accel_forearm_y     : int  203 203 204 206 206 203 205 205 204 205 ...
##  $ accel_forearm_z     : int  -215 -216 -213 -214 -214 -215 -215 -213 -214 -215 ...
##  $ magnet_forearm_x    : int  -17 -18 -18 -16 -17 -9 -18 -9 -16 -22 ...
##  $ magnet_forearm_y    : num  654 661 658 658 655 660 659 660 653 656 ...
##  $ magnet_forearm_z    : num  476 473 469 469 473 478 470 474 476 473 ...
```

```
##  $ classe              : Factor w/ 5 levels "A","B","C","D",..: 1 1 1 1 1 1 1 1 1 1 ...
## NULL
```

The resulting dataset has been simplified down to 53 variables of significant sensor data and is ready to be split into training and test sets for model training and testing purposes.

```
# Create partition indexes
inTrain <- createDataPartition(y=training$classe, p=0.75, list=FALSE)

# Create a training data set and testing data set
testData <- trainData[-inTrain,]
trainData <- trainData[inTrain,]

# resulting data sets
dim(trainData)
```

```
## [1] 14718    53
```

```
dim(testData)
```

```
## [1] 4904   53
```

The test data is broken into two parts, trainData (75%) and testData (25%).

Train the models

The approach is to train several models (random forest, gbm, lda) using popular methods and pick the best on based on accuracy. The first model we are going to use is the random forest ‘rf’ method. We are using the trainControl for the models to perform K-fold of 10 and repeats this 1 time during training.

Random Forest

The first model to build is the random forest method. This training method can take a long time to complete.

```
set.seed(1859)

# Do 10 folds and repeat 1 time
control <- trainControl(method='cv',
                        number=10)
#Number randomly variable selected is mtry
mtry <- sqrt(ncol(testData)-1)
tunegrid <- expand.grid(.mtry=mtry)

start <- Sys.time()
modRF <- train(classe~.,
                    data=trainData,
                    method='rf',
                    metric='Accuracy',
                    tuneGrid=tunegrid,
                    trControl=control)
predRF <- predict( modRF, newdata=testData)
cm1 <- confusionMatrix(predRF, testData$classe )
print( paste( "RF accuracy: ", cm1$overall[1]))
```

```
## [1] "RF accuracy:  0.996329526916803"
```

```
rmse_rf <- RMSE(as.numeric(predRF), as.numeric(testData$classe))
print(rmse_rf)
```

```
## [1] 0.08203171
```

This model has excellent accuracy of 0.9963295 and a low RMSE of 0.0820317. This model is a good candidate for the final predictions.

GBM (Gradient Boosting Machine)

The GBM method will be used to generate a model from the trainData.

```
modGBM <- train(classe~.,
                data=trainData,
                method='gbm',
                metric='Accuracy',
                trControl=control,
                verbose=FALSE)
predGBM <- predict( modGBM, newdata=testData)
cm2 <- confusionMatrix(predGBM, testData$classe )
print( paste( "GMB accuracy: ", cm2$overall[1]))
```

```
## [1] "GMB accuracy:  0.963703099510604"
```

```
rmse_gbm <- RMSE(as.numeric(predGBM), as.numeric(testData$classe))
print( rmse_gbm)
```

```
## [1] 0.2633081
```

This model also has great accuracy of 0.9637031 and RMSE of 0.2633081 could be better. This model is another good candidate for the final predictions.

LDA (Linear Discriminent Analysis)

The final model to used for the thrid method is the LDA model.

```
modLDA <-train(classe~.,
               data=trainData,
               method='lda',
               metric='Accuracy',
               trControl=control)
predLDA <- predict( modLDA, newdata=testData)
cm3 <- confusionMatrix(predLDA, testData$classe )
print( paste( "LBA accuracy:", cm3$overall[1]))
```

```
## [1] "LBA accuracy: 0.706362153344209"
```

```
rmse_lda <- RMSE(as.numeric(predLDA), as.numeric(testData$classe))
print(rmse_lda)
```

```
## [1] 1.095278
```

This model has an accuracy is 0.7063622 which is much worse then the other two models. This model will not be used in the finaltest predictions.

Results

Out-of-sample error and Accuracy

The out-of-sample error will give some measure of the effectiveness of the model on data not seen before.

```
# create a comparison data frame
oose <- data.frame(c("RF","GBM","LDA"),c(rmse_rf, rmse_gbm, rmse_lda),c(cm1$overall[1], cm2$over
all[1], cm3$overall[1]))
colnames(oose) <- c("Method","RMSE","Accuracy")

print("Model Comparison")
```

```
## [1] "Model Comparison"
```

```
print(oose)
```

```
##   Method       RMSE  Accuracy
## 1     RF 0.08203171 0.9963295
## 2    GBM 0.26330811 0.9637031
## 3    LDA 1.09527757 0.7063622
```

The above table shows the out-of-sample error lowest for the RF or random forest model. The RMSE error for RF model is 0.08 which is better than the 0.26 RMSE for the GBM method

Cross Validation

The random forest (rf) method doesn't require separate cross validation because in the k-fold process, cross validation is inherent in the method.

Model Selection

The GBM and Random Forest models are very accurate and either can be used in the finaltest predictions but the Random Forest method is better based on RMSE alone. They both correctly produce the same set of predictions (which will not be shown here) from the finalData test set.

The random forest model has the highest accuracy and the lowest RMSE error rate of the three models created to make the predictions.

The model with the highest accuracy, the Random Forest (rf), produces the correct prediction result. The gbm model also produces the correct prediction result. Comparing the two results from rf and gbm methods produces the sames predictions, as is shown below, but the RF model has better RMSE on unseen data.

```
rffinal <- predict(modRF, newdata=finalData)
gbmfinal <- predict(modGBM, newdata=finalData)

## See that these have equal predictions
print("The models produce the same results on the final test set")
```

```
## [1] "The models produce the same results on the final test set"
```

```
print(rffinal==gbmfinal)
```

```
##  [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [16] TRUE TRUE TRUE TRUE TRUE
```