Human Activity Recognition Class Prediction

Shenay

July 9, 2019

Synopsis

In this report, we will focus on predicting the "classe" variable provided in the human activity recognition research training data set. Each of the five classes in the "classe" variable correspond to the specified execution of the exercise (*Class A*) or common mistakes that were made (*Classes B, C, D, and E*). The data was generated from accelerometers placed on the belt, forearm, arm, and dumbell of 6 male participants.

The source of the data is available here. (http://web.archive.org/web/20161224072740/http:/groupware.les.inf.puc-rio.br/har). See the section on the weight lifting exercise data set.

The training (https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv) and testing (https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv) data sets were retrieved from the given links.

Load and Process the Data:

```
library(caret)
library(dplyr)
library(tidyr)
library(tibble)

setwd("C:/Users/sheng/Coursera")
if (!file.exists("./data")) {dir.create("./data")}
trainURL <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
testURL <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"
download.file(trainURL, destfile = "./data/pml-training.csv", mode = "wb")
download.file(testURL, destfile = "./data/pml-testing.csv", mode="wb")

train <- read.csv("pml-training.csv", header=TRUE)
test <- read.csv("pml-testing.csv", header=TRUE)

dim(train); dim(test)
```

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

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

```
#str(train); str(test)
```

Since there are a number of columns with missing values, we need to filter out those that may impact our prediction models. We will choose a benchmark of 50% and remove all variables that contain more than 50% NAs.

```
train <- train[, -which(colMeans(is.na(train)) > 0.5)]
test <- test[, -which(colMeans(is.na(test)) > 0.5)]
```

Next, let's remove unnecessary identifier and timestamp variables. This includes the first seven columns.

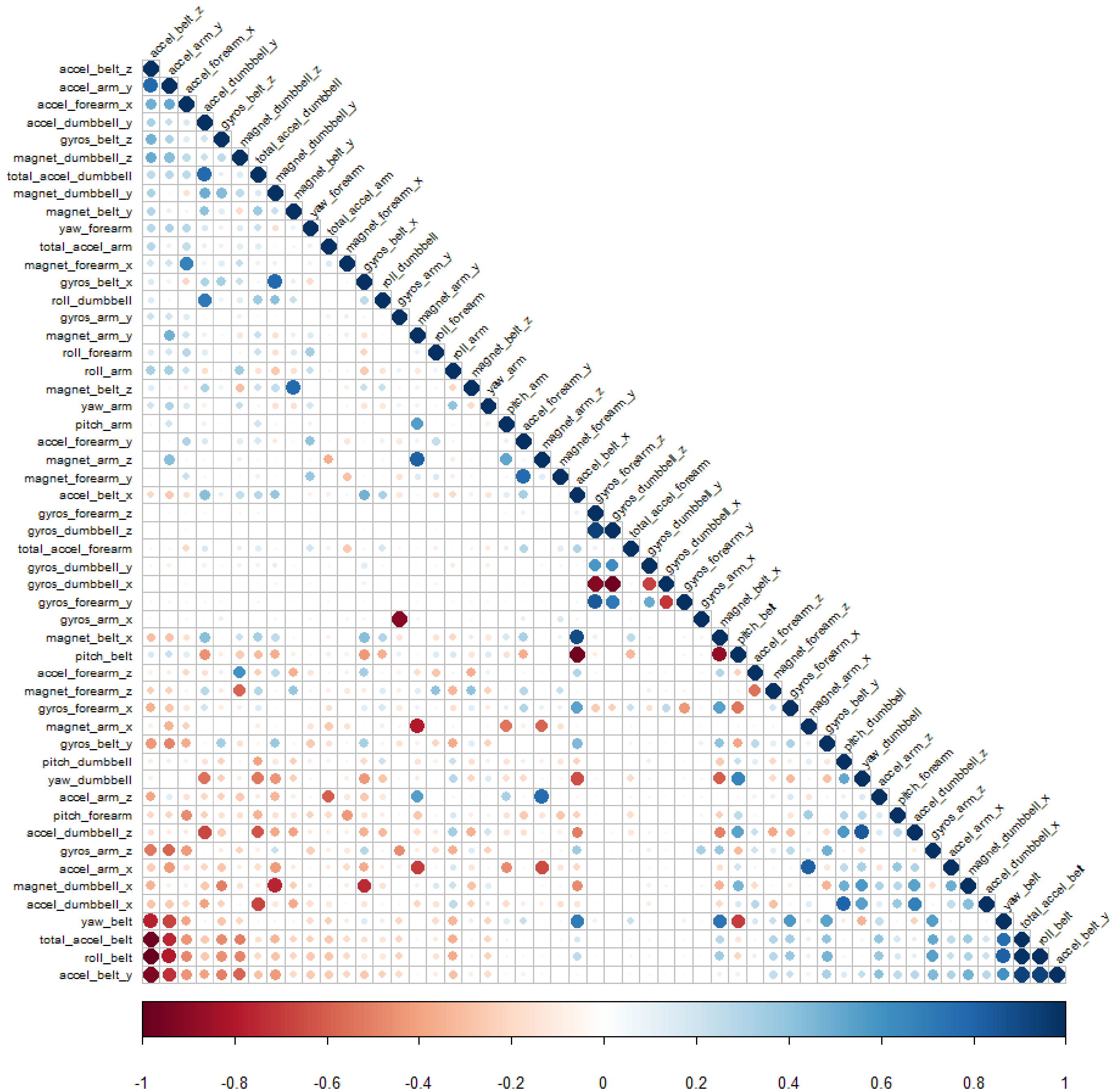
```
train <- train[, -c(1:7)]
test <- test[, -c(1:7)]
```

In order to further filter our variables, we choose to remove columns with values that are mostly identical to each other (variance of zero).

```
library(caret)
zvar <- nearZeroVar(train)
train <- train[, -zvar]
```

To check for correlations among variables, we will compute the correlation matrix.

```
corMatrix <- cor(train[, -53])
corrplot::corrplot(corMatrix, method="circle", order="FPC", type="lower",
                   tl.col="black", tl.cex=0.6, tl.srt=45)
```



Based on the correlation matrix above, it seems that accel_belt_z & accel_arm_y is highly associated with yaw_belt, total_accel_belt, roll_belt, and accel_belt_y with correlation values close to 1. However, these variables will not be omitted from our training data set during this analysis since 52 explanatory variables are sufficient.

Partition Training Data:

We will further partition the training data set into 70/30 training_data vs. testing_data for cross validation purposes.

```
set.seed(1234)
indexes <- createDataPartition(train$classe, times = 1,
                               p = 0.7, list = FALSE)

train_data <- train[indexes, ]
test_data <- train[-indexes, ]

dim(train_data); dim(test_data)
```

```
## [1] 13737    53
```

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

Random Forest Prediction:

```
modFit_rf <- train(classe ~ ., method="rf", data=train_data)
print(modFit_rf$finalModel)
```

```
##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 27
##
##         OOB estimate of  error rate: 0.61%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 3901    3    1    0    1 0.001280082
## B   23 2630    5    0    0 0.010534236
## C    0   10 2380    6    0 0.006677796
## D    0    0   22 2228    2 0.010657194
## E    0    0    3    8 2514 0.004356436
```

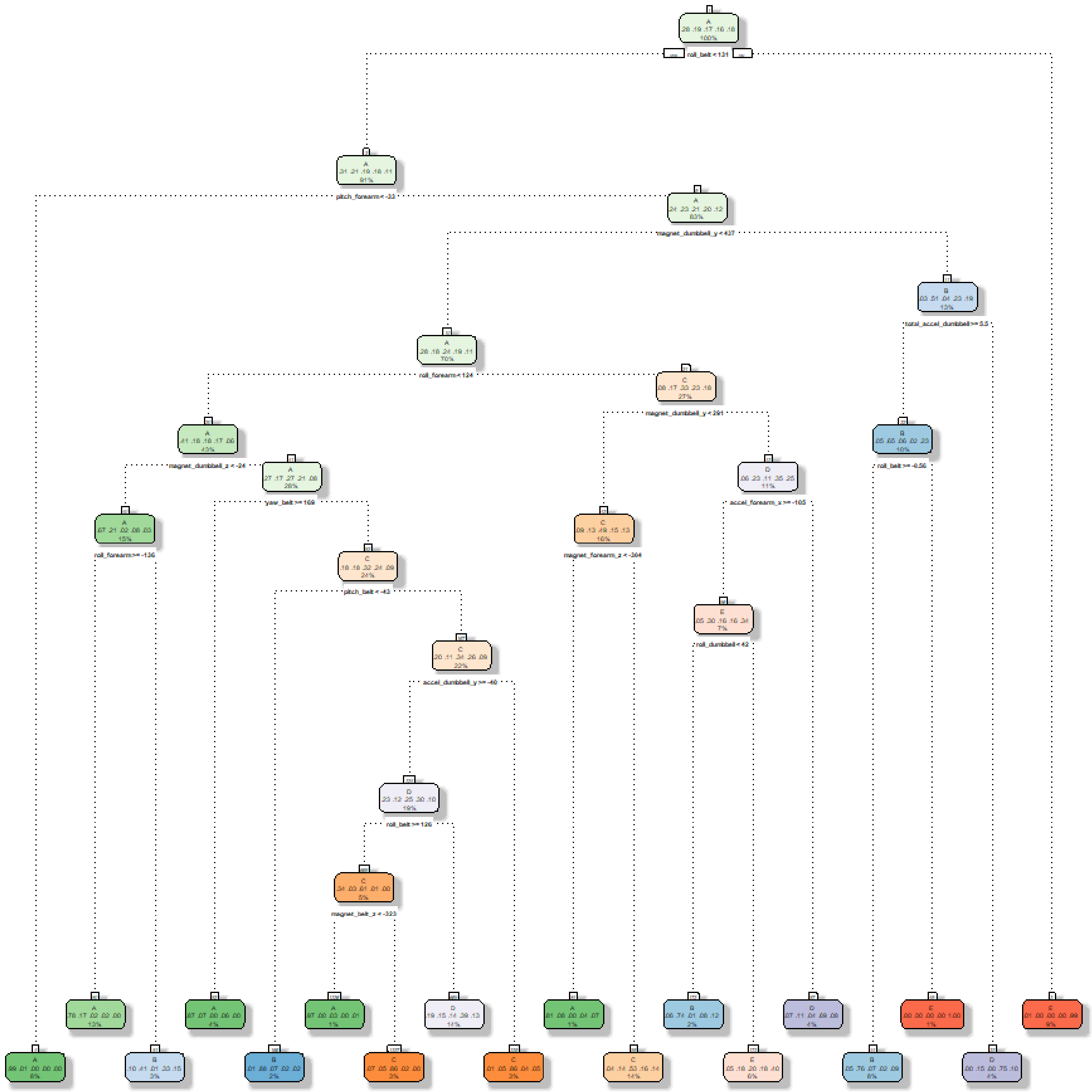
```
predict_rf <- predict(modFit_rf, newdata=test_data)
confMatrix_rf <- confusionMatrix(predict_rf, test_data$classe)
confMatrix_rf
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1674   11    0    0    0
##          B    0 1127    5    2    1
##          C    0    1 1017    6    2
##          D    0    0    4  955    3
##          E    0    0    0    1 1076
##
## Overall Statistics
##
##                Accuracy : 0.9939
##                  95% CI : (0.9915, 0.9957)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9923
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9895   0.9912   0.9907   0.9945
## Specificity            0.9974   0.9983   0.9981   0.9986   0.9998
## Pos Pred Value         0.9935   0.9930   0.9912   0.9927   0.9991
## Neg Pred Value         1.0000   0.9975   0.9981   0.9982   0.9988
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2845   0.1915   0.1728   0.1623   0.1828
## Detection Prevalence   0.2863   0.1929   0.1743   0.1635   0.1830
## Balanced Accuracy      0.9987   0.9939   0.9947   0.9946   0.9971
```

According to the random forest prediction model, the estimated accuracy rate is around 99.39%.

Decision Tree Prediction:

```
modFit_dt <- rpart::rpart(classe ~ ., method="class", data=train_data)
rattle::fancyRpartPlot(modFit_dt)
```



Rattle 2019-Jul-09 22:38:59 sheng

```
predict_dt <- predict(modFit_dt, type="class", newdata=test_data)
confMatrix_dt <- confusionMatrix(predict_dt, test_data$classe)
confMatrix_dt
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    A    B    C    D    E
##          A 1364  169   24   48   16
##          B   60  581   46   79   74
##          C   52  137  765  129  145
##          D  183  194  125  650  159
##          E   15   58   66   58  688
## 
## Overall Statistics
##                                           
##                Accuracy : 0.6879          
##                  95% CI : (0.6758, 0.6997)
##     No Information Rate : 0.2845          
##     P-Value [Acc > NIR] : < 2.2e-16       
##                                           
##                   Kappa : 0.6066          
##                                           
##  Mcnemar's Test P-Value : < 2.2e-16       
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.8148  0.51010   0.7456   0.6743   0.6359
## Specificity            0.9390  0.94543   0.9047   0.8657   0.9590
## Pos Pred Value         0.8415  0.69167   0.6230   0.4958   0.7774
## Neg Pred Value         0.9273  0.88940   0.9440   0.9314   0.9212
## Prevalence             0.2845  0.19354   0.1743   0.1638   0.1839
## Detection Rate         0.2318  0.09873   0.1300   0.1105   0.1169
## Detection Prevalence   0.2754  0.14274   0.2087   0.2228   0.1504
## Balanced Accuracy      0.8769  0.72776   0.8252   0.7700   0.7974
```

As shown from the confusion matrix, the approximate accuracy rate of the decision tree prediction model is 68.79%.

Bagging Prediction:

```
modFit_gb <- train(classe ~ ., method="gbm", data=train_data, verbose=FALSE)
modFit_gb
```

```
## Stochastic Gradient Boosting
##
## 13737 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 13737, 13737, 13737, 13737, 13737, 13737, ...
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy   Kappa
##   1                   50      0.7463167  0.6785074
##   1                  100      0.8150096  0.7658811
##   1                  150      0.8500299  0.8102060
##   2                   50      0.8498390  0.8097185
##   2                  100      0.9026085  0.8767027
##   2                  150      0.9287997  0.9098699
##   3                   50      0.8922966  0.8635996
##   3                  100      0.9383750  0.9219822
##   3                  150      0.9572389  0.9458756
##
## 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.
```

```
predict_gb <- predict(modFit_gb, newdata=test_data)
confMatrix_gb <- confusionMatrix(predict_gb, test_data$classe)
confMatrix_gb
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1655   44    0    0    1
##          B   13 1070   33    4   11
##          C    1   24  979   20   13
##          D    4    0   11  932   16
##          E    1    1    3    8 1041
##
## Overall Statistics
##
##                Accuracy : 0.9647
##                  95% CI : (0.9596, 0.9692)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9553
##
##  Mcnemar's Test P-Value : 8.891e-07
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9886   0.9394   0.9542   0.9668   0.9621
## Specificity            0.9893   0.9871   0.9881   0.9937   0.9973
## Pos Pred Value         0.9735   0.9461   0.9441   0.9678   0.9877
## Neg Pred Value         0.9955   0.9855   0.9903   0.9935   0.9915
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2812   0.1818   0.1664   0.1584   0.1769
## Detection Prevalence   0.2889   0.1922   0.1762   0.1636   0.1791
## Balanced Accuracy      0.9890   0.9633   0.9711   0.9803   0.9797
```

Results from applying a gradient boosting model to our training data subset indicate that the estimated accuracy rate is about 95.72%.

Predict with Testing Data:

Because the random forest model has the highest cross-validation accuracy rate, we will use its fitted model to predict our final testing data set of 20 individuals.

```
rf_prediction <- predict(modFit_rf, test)
rf_prediction
```

```
##  [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
```