Evaluate Exercise Performance using Machine Learning

1. Introduction

Nowadays, we use smart wearables to collect data while doing exercise. It is possible to use these data to evaluate our exercise performance and help us improve exercise efficiency and correct the wrong doings. Therefore, here we use machine learning to build models to predict one's exercise performance.

2. Results

Load libraries

```
library(caret)
library(corrplot)
library(rpart)
library(rpart.plot)
library(rattle)
library(corrplot)
library(rpart)
```

2.1. Get the data

We first download and load the data.

```
# Download files
# download.file(url = "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv", d
estfile = "pml-training.csv")
#
# download.file(url = "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv", de
stfile = "pml-training.csv")

# load the training data sets
data_set <- read.csv("./pml-training.csv")
test_set <- read.csv("./pml-testing.csv")
```

2.2. Dataset check and clean-up

After loading the data, we go through the data structure using str(data_set), results omitted due to page limit. We notice there are unnecessary variables, numbers kept as character, and variables with too many NAs. So, we do the following data clean-up.

```
str(data_set)
```

2.2.1. Remove unnecessary variables

First of all, we removed identities and timestamps variables as we see no need to include in the model building.

```
data_set1 <- data_set[,c(-1:-5)]
```

2.2.2. Convert character to numeric

Next, we checked the data type. We noticed some features are actually numeric but kept as character, so, we convert thoes to numeric first.

```
x <- sapply(data_set1,mode)
chr_id <- x=="character"
data_set2 <- data_set1

for (i in 2:c(length(chr_id)-1)){
  if (chr_id[i] == TRUE){
    data_set2[,i] = as.numeric(data_set1[,i])
  }
}
```

2.2.3. Missing values

As we covert the character to numeric, those blanks are coerced to NAs. On the other hand, many logi variables are basically NAs, so we remove variables with NAs more than 60%.

```
na_count <- apply(data_set2, 2, function(x)sum(is.na(x)))
na_id <- na_count < nrow(data_set2)*0.6
data_set3 <- data_set2[, na_id]
```

Now, we are ready to use the data set (data_set3) to build models.

2.3. Build models

2.3.1. Training and testing sets

As usual, we'll first split the data_set3 to train and test sets.

```
inTrain  <- createDataPartition(data_set3$classe, p=0.7, list=FALSE)
TrainSet <- data_set3[inTrain, ]
TestSet  <- data_set3[-inTrain, ]
dim(TrainSet)
```

```
## [1] 13737    55
```

2.3.2. Check data (co-)variations

Since we still have 54 variables to go, we wonder are there any variables not vary much so that would not contribute to model differentiation but only add-up calculation load, so we first check the data variation.

```
nv_id <- nearZeroVar(TrainSet)
TrainSet1 <- TrainSet[, -nv_id]
TestSet1 <- TestSet[, -nv_id]
dim(TrainSet1)
```

```
## [1] 13737    54
```

So, we see that "new_window" does vary too much, as 97% of the data is "no", using the following code to check.

```
x <- TrainSet$new_window
table(x)
```

Next, we checked the covariance of these variables to see if we need to use PCA to build models on principle components.

```
corMatrix <- cor(TrainSet1[, -54])
corrplot(corMatrix, order = "FPC", method = "color", type = "lower", tl.cex = 0.6, tl.col = 'black')
```

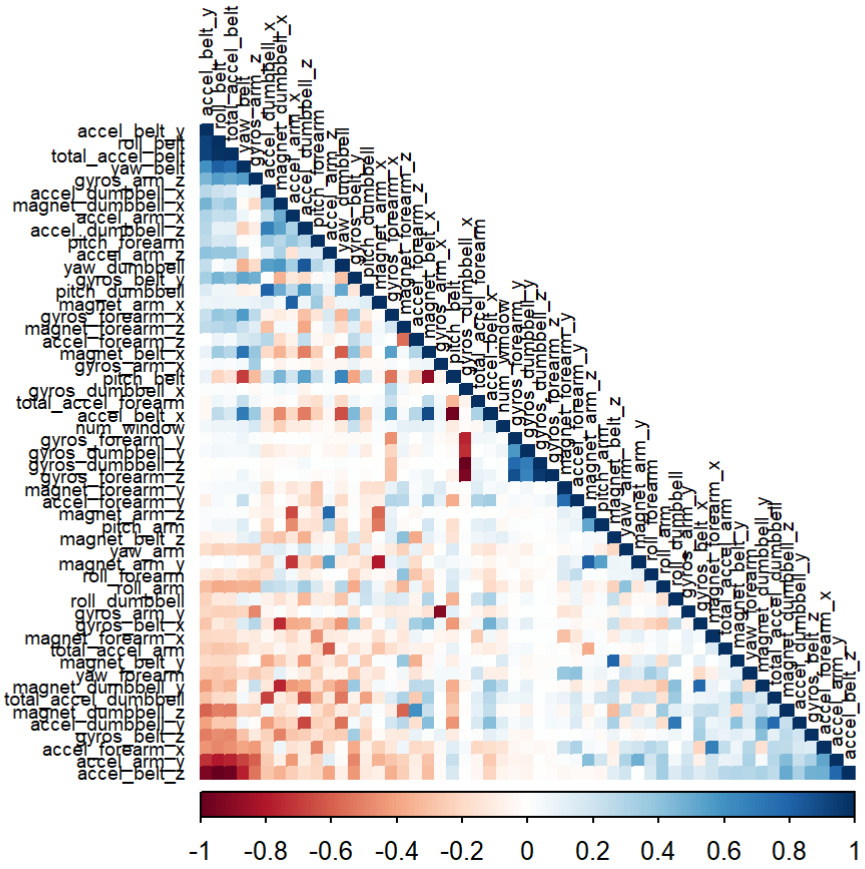


Fig.1 Covariance of the dataset. Heatmap is correlations, blue represent positive correlations and red negative.

As shown in Fig. 1, no significant positive correlations between variables are observed, as indicated by not many very blue blocks. So, we can now proceed to build our model using all 53 variables.

2.3.3. Build models

From what we have learned from this class, we decide to build models using random forest and decision tree, and use the accuracy from testset to compare the two. If both are weak model, we will consider combining the two to build a better one.

2.3.3.1 Random Forest

```
# set.seed(25331)
# build the random forest model
rf_control <- trainControl(method="boot", number=3, verboseIter=FALSE)
fit_rf <- train(classe ~ ., data=TrainSet1, method="rf", trControl=rf_control)
fit_rf $finalModel
```

```
##
## Call:
##  randomForest(x = x, y = y, mtry = min(param$mtry, ncol(x)))
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 27
##
##         OOB estimate of  error rate: 0.25%
## Confusion matrix:
##      A    B    C    D    E  class.error
## A 3905    0    0    0    1 0.0002560164
## B    5 2650    3    0    0 0.0030097818
## C    0    5 2391    0    0 0.0020868114
## D    0    0   11 2240    1 0.0053285968
## E    0    2    0    6 2517 0.0031683168
```

```
# get the accuracy of the model
predict_rf <- predict(fit_rf, newdata=TestSet1)
conf_rf <- confusionMatrix(predict_rf, as.factor(TestSet1$classe))
conf_rf
```

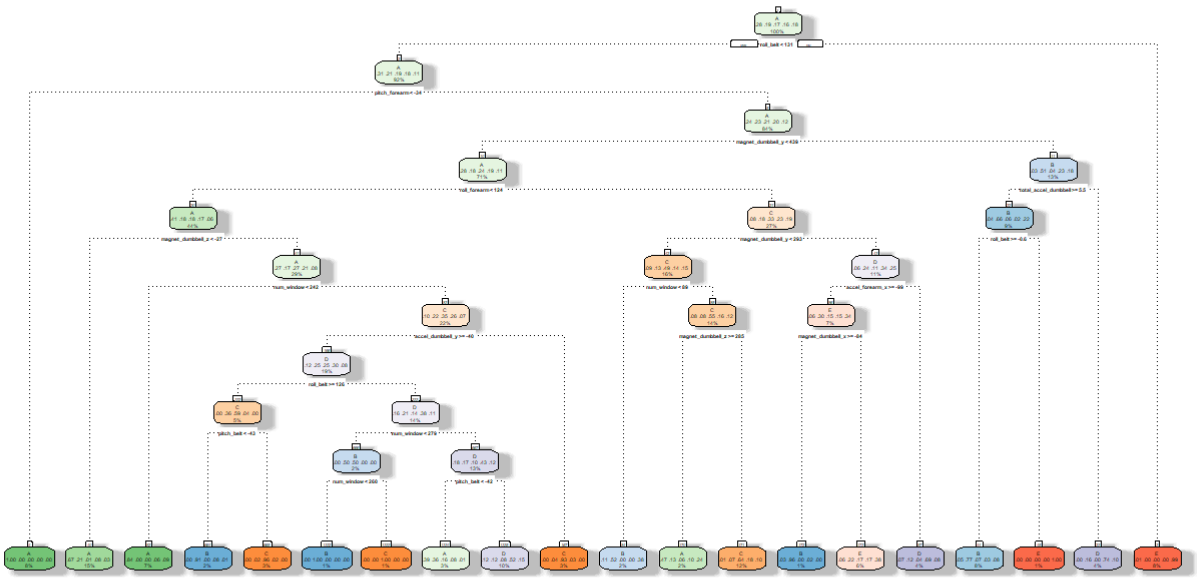
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1674    1    0    0    0
##          B    0 1137    1    0    0
##          C    0    1 1025    0    0
##          D    0    0    0  964    1
##          E    0    0    0    0 1081
##
## Overall Statistics
##
##                Accuracy : 0.9993
##                  95% CI : (0.9983, 0.9998)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9991
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9982   0.9990   1.0000   0.9991
## Specificity            0.9998   0.9998   0.9998   0.9998   1.0000
## Pos Pred Value         0.9994   0.9991   0.9990   0.9990   1.0000
## Neg Pred Value         1.0000   0.9996   0.9998   1.0000   0.9998
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2845   0.1932   0.1742   0.1638   0.1837
## Detection Prevalence   0.2846   0.1934   0.1743   0.1640   0.1837
## Balanced Accuracy      0.9999   0.9990   0.9994   0.9999   0.9995
```

So, the accuracy of this random forest is0.9993. Pretty good.

2.3.3.2 Decision Tree

```
# build the decision tree model
# set.seed(25331)
fit_dt <- rpart(classe ~ ., data=TrainSet1, method="class")
fancyRpartPlot(fit_dt, main = "Decision Tree", sub = "")
```

Decision Tree



```
# get the accuracy of the model
predict_dt <- predict(fit_dt, newdata=TestSet1,type="class")
conf_dt <- confusionMatrix(predict_dt, as.factor(TestSet1$classe))
conf_dt
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1515  265   50  113   99
##          B   39  578   30   15   78
##          C   12   63  826  142   66
##          D   91  142   50  646  121
##          E   17   91   70   48  718
##
## Overall Statistics
##
##                Accuracy : 0.7278
##                  95% CI : (0.7162, 0.7391)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6534
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9050  0.50746   0.8051   0.6701   0.6636
## Specificity            0.8749  0.96587   0.9418   0.9179   0.9529
## Pos Pred Value         0.7419  0.78108   0.7448   0.6152   0.7606
## Neg Pred Value         0.9586  0.89096   0.9581   0.9342   0.9263
## Prevalence             0.2845  0.19354   0.1743   0.1638   0.1839
## Detection Rate         0.2574  0.09822   0.1404   0.1098   0.1220
## Detection Prevalence   0.3470  0.12574   0.1884   0.1784   0.1604
## Balanced Accuracy      0.8899  0.73666   0.8734   0.7940   0.8083
```

So, the accuracy of this decision tree is 0.7278.

Compare the two models, though random forest takes fair amount of time, its accuracy is very satisfied. Therefore, we decide to use the random forest (fit_rf) to predict test dataset.

2.4. Apply the chosen model for tests

```
predict_test <- predict(fit_rf, newdata=test_set)
predict_test
```

```
##  [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
```

The above is the answer to the course project quiz.

3. Conclusions

In conclusion, we have built a model using random forest method to predict exercise performance class.