Course Project: Predicting Exercise Performance: Attempt 2

Allen (github: alc00)

11/27/2020

Introduction

We will be creating a predictive model based on the performance metrics of exercise devices from the data, specified by the course requirements, and classify them into exercise types. In short, we will be taking movement data and classify them into the type of exercise.

Loading the Data

```
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

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

Data Cleaning

Check for Irrelavant Columns

```
str(train)
length(names(train))
```

Remove Irrelevant Columns

Since we will be utilizing activity/movement data to classify the records into specific exercises. Certain data, such as the row number, timestamp, user, etc. seems irrelevant to the model. We will be removing them in order to reduce noise and hopefully improve the predictive capabilities of our model.

We removed the first 7 columns.

```
train2 <- train[,8:160]
str(train2)
```

Convert Character Columns to Numeric

We will be converting some numeric columns that were incorrect tagged as characters.

Column Checking

Let's check each character column first.

```
train2_char <- train2[,sapply(train2, class) == 'character']

str(train2_char)
train2_char_test1 <- sapply(train2_char, max)
train2_char_test2 <- sapply(train2_char, min)

train2_char_test1
train2_char_test2
```

Column Checking 2

Based on manually looking into each column without any numeric or blank value based on the max and min functions. Let's check if they actually contain any numbers so we can determine if they should be handled separately or removed entirely.

Columns that will be manually checked: kurtosis_picth_belt kurtosis_yaw_belt skewness_yaw_belt kurtosis_yaw_dumbbell skewness_yaw_dumbbell kurtosis_yaw_forearm skewness_yaw_forearm

```
check_list <- c("kurtosis_picth_belt",
"kurtosis_yaw_belt",
"skewness_yaw_belt",
"kurtosis_yaw_dumbbell",
"skewness_yaw_dumbbell",
"kurtosis_yaw_forearm",
"skewness_yaw_forearm")

unique_values_check <- lapply(train2[,check_list], unique)
unique_values_check
```

Based on these results, we will only be retaining the column "kurtosis_picth_belt" and removing the rest. We have confirmed that there is no predictor that is formatted as a character.

Filtering of Columns and Converting to Numeric

```
list_columns_removal <- c("kurtosis_yaw_belt",
"skewness_yaw_belt",
"kurtosis_yaw_dumbbell",
"skewness_yaw_dumbbell",
```

```r
"kurtosis_yaw_forearm",
"skewness_yaw_forearm")

train3 <- train2[, !(names(train2) %in% list_columns_removal)]
train4_predictor <- data.frame(lapply(train3[,which(!(names(train3) == "classe"))], as.numeric))
```

```
## Warning in lapply(train3[, which(!(names(train3) == "classe"))], as.numeric):
## NAs introduced by coercion

## Warning in lapply(train3[, which(!(names(train3) == "classe"))], as.numeric):
## NAs introduced by coercion

## Warning in lapply(train3[, which(!(names(train3) == "classe"))], as.numeric):
## NAs introduced by coercion

## Warning in lapply(train3[, which(!(names(train3) == "classe"))], as.numeric):
## NAs introduced by coercion

## Warning in lapply(train3[, which(!(names(train3) == "classe"))], as.numeric):
## NAs introduced by coercion

## Warning in lapply(train3[, which(!(names(train3) == "classe"))], as.numeric):
## NAs introduced by coercion

## Warning in lapply(train3[, which(!(names(train3) == "classe"))], as.numeric):
## NAs introduced by coercion

## Warning in lapply(train3[, which(!(names(train3) == "classe"))], as.numeric):
## NAs introduced by coercion

## Warning in lapply(train3[, which(!(names(train3) == "classe"))], as.numeric):
## NAs introduced by coercion

## Warning in lapply(train3[, which(!(names(train3) == "classe"))], as.numeric):
## NAs introduced by coercion

## Warning in lapply(train3[, which(!(names(train3) == "classe"))], as.numeric):
## NAs introduced by coercion

## Warning in lapply(train3[, which(!(names(train3) == "classe"))], as.numeric):
## NAs introduced by coercion

## Warning in lapply(train3[, which(!(names(train3) == "classe"))], as.numeric):
## NAs introduced by coercion

## Warning in lapply(train3[, which(!(names(train3) == "classe"))], as.numeric):
## NAs introduced by coercion

## Warning in lapply(train3[, which(!(names(train3) == "classe"))], as.numeric):
## NAs introduced by coercion

## Warning in lapply(train3[, which(!(names(train3) == "classe"))], as.numeric):
## NAs introduced by coercion
```

```
## Warning in lapply(train3[, which(!(names(train3) == "classe"))], as.numeric):
## NAs introduced by coercion

## Warning in lapply(train3[, which(!(names(train3) == "classe"))], as.numeric):
## NAs introduced by coercion

## Warning in lapply(train3[, which(!(names(train3) == "classe"))], as.numeric):
## NAs introduced by coercion

## Warning in lapply(train3[, which(!(names(train3) == "classe"))], as.numeric):
## NAs introduced by coercion

## Warning in lapply(train3[, which(!(names(train3) == "classe"))], as.numeric):
## NAs introduced by coercion

## Warning in lapply(train3[, which(!(names(train3) == "classe"))], as.numeric):
## NAs introduced by coercion

## Warning in lapply(train3[, which(!(names(train3) == "classe"))], as.numeric):
## NAs introduced by coercion

## Warning in lapply(train3[, which(!(names(train3) == "classe"))], as.numeric):
## NAs introduced by coercion

## Warning in lapply(train3[, which(!(names(train3) == "classe"))], as.numeric):
## NAs introduced by coercion

## Warning in lapply(train3[, which(!(names(train3) == "classe"))], as.numeric):
## NAs introduced by coercion

## Warning in lapply(train3[, which(!(names(train3) == "classe"))], as.numeric):
## NAs introduced by coercion
```

```r
classe <- train3[,"classe"]
train4 <- cbind(classe, train4_predictor)
```

Secondary Data Cleaning Checking

Let's check if there are any columns with any issues (i.e. column only has 1 unique value)

```r
unique_cnt_per_col <- sapply(lapply(train4, unique), length)
which(!unique_cnt_per_col[]>2)

unique(train4$amplitude_yaw_belt)
unique(train4$amplitude_yaw_dumbbell)
unique(train4$amplitude_yaw_forearm)
```

We can see that these 3 columns have only 2 values NA or 0. As this data is not useful, we will be removing these columns.

Secondary Cleaning - Removing Unneccessary Columns and Changing NA to 0

```
list_column_removal2 <- c("amplitude_yaw_belt","amplitude_yaw_dumbbell", "amplitude_yaw_forearm")
train5 <- train4[,which(!names(train4) %in% list_column_removal2)]
train5[is.na(train5)] <- 0
train5 <- data.frame(train5)
train5$classe <- factor(train5$classe)
```

Split the Training Data to Create a Validation Set

We will be splitting the training data into an actual "training" data and a testing/validation set, so we can test our model internally.

```
set.seed(20201127)
inTrain <- createDataPartition(y=train5$classe, p=0.7, list=FALSE)
v_train1 <- train5[inTrain,]
validation <- train5[-inTrain,]
```

Model Creation and Internal Evaluation

Linear Discriminate Analysis Model Creation and Evaluation

```
set.seed(20201127)
model_lda <- train(factor(classe)~., data=v_train1, method="lda")
```

```
## Warning in lda.default(x, grouping, ...): variables are collinear

## Warning in lda.default(x, grouping, ...): variables are collinear

## Warning in lda.default(x, grouping, ...): variables are collinear

## Warning in lda.default(x, grouping, ...): variables are collinear

## Warning in lda.default(x, grouping, ...): variables are collinear

## Warning in lda.default(x, grouping, ...): variables are collinear

## Warning in lda.default(x, grouping, ...): variables are collinear

## Warning in lda.default(x, grouping, ...): variables are collinear

## Warning in lda.default(x, grouping, ...): variables are collinear

## Warning in lda.default(x, grouping, ...): variables are collinear

## Warning in lda.default(x, grouping, ...): variables are collinear

## Warning in lda.default(x, grouping, ...): variables are collinear
```

```
## Warning in lda.default(x, grouping, ...): variables are collinear

## Warning in lda.default(x, grouping, ...): variables are collinear

## Warning in lda.default(x, grouping, ...): variables are collinear

## Warning in lda.default(x, grouping, ...): variables are collinear

## Warning in lda.default(x, grouping, ...): variables are collinear

## Warning in lda.default(x, grouping, ...): variables are collinear

## Warning in lda.default(x, grouping, ...): variables are collinear

## Warning in lda.default(x, grouping, ...): variables are collinear

## Warning in lda.default(x, grouping, ...): variables are collinear

## Warning in lda.default(x, grouping, ...): variables are collinear

## Warning in lda.default(x, grouping, ...): variables are collinear

## Warning in lda.default(x, grouping, ...): variables are collinear

## Warning in lda.default(x, grouping, ...): variables are collinear

## Warning in lda.default(x, grouping, ...): variables are collinear
```

```
print("Using Training Data")
```

```
## [1] "Using Training Data"
```

```
results <- table(v_train1$classe, predict(model_lda, v_train1))
sum(diag(results))/sum(results)
```

```
## [1] 0.7104171
```

```
print("Using Validation Data")
```

```
## [1] "Using Validation Data"
```

```
results <- table(validation$classe, predict(model_lda, validation))
sum(diag(results))/sum(results)
```

```
## [1] 0.7028037
```

Decision Tree Model Creation and Evaluation

```r
set.seed(20201127)
model_rpart <- train(factor(classe)~., data=v_train1, method="rpart")

print("Using Training Data")
```

```
## [1] "Using Training Data"
```

```r
results <- table(v_train1$classe, predict(model_rpart, v_train1))
sum(diag(results))/sum(results)
```

```
## [1] 0.5202737
```

```r
print("Using Validation Data")
```

```
## [1] "Using Validation Data"
```

```r
results <- table(validation$classe, predict(model_rpart, validation))
sum(diag(results))/sum(results)
```

```
## [1] 0.5058624
```

Gradient Boost - Decision Trees Model Creation and Evaluation

```r
set.seed(20201127)
train_control <- trainControl(method = "repeatedcv", number= 2, repeats = 2)
model_gbm <- train(factor(classe)~., data=v_train1, method="gbm", trControl = train_control, verbose = F

print("Using Training Data")
```

```
## [1] "Using Training Data"
```

```r
results <- table(v_train1$classe, predict(model_gbm, v_train1))
sum(diag(results))/sum(results)
```

```
## [1] 0.9745214
```

```r
print("Using Validation Data")
```

```
## [1] "Using Validation Data"
```

```r
results <- table(validation$classe, predict(model_gbm, validation))
sum(diag(results))/sum(results)
```

```
## [1] 0.9614274
```

Based on the results of the internal evaluation of the model, we will be selecting the GBM Model for our actual/test data.

Model Implementation

Testing Data Cleaning

Retain Only Relevant Columns

```
test2 <- test[,which(names(test) %in% names(v_train1))]
```

Convert Character Columns to Numeric

```
test3 <- data.frame(lapply(test2, as.numeric))
```

Convert NA to 0

```
test3[is.na(test3)] <- 0
test3 <- data.frame(test3)
```

Run the test data through the Model

```
results <- predict(model_gbm, test3)
print(results)
```

Evaluation

Based on the Quiz Results, our Model had an accuracy of 100%.

Benchmark (Accuracy) Random Guessing: 20% or 1 out of 5 chance PCA with Decision Tree Model (Attempt#1): 35% Gradient Boosting with Trees Model (Attempt#2): 100%