Effectivness of an Exercise using Predictive Analytics

Dhruva Karanth

2024-12-11

Introduction

Using devices such as Jawbone Up, Nike FuelBand, and Fitbit it is now possible to collect a large amount of data about personal activity relatively inexpensively. These type of devices are part of the quantified self movement – a group of enthusiasts who take measurements about themselves regularly to improve their health, to find patterns in their behavior, or because they are tech geeks. One thing that people regularly do is quantify how much of a particular activity they do, but they rarely quantify how well they do it. In this project, your goal will be to use data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants. They were asked to perform barbell lifts correctly and incorrectly in 5 different ways. The following steps would be followed:

1. Data Pre-processing and Cleaning
2. Predictive Model Selection
3. Validating Model and creating plots
4. Testing Model
5. Predicting Results

1. Data Pre-processing and Cleaning

We load the training and testing datasets from the online sources. Next, we split the training dataset further into separate subsets for training, validation, and testing purposes.

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.3.3
```

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

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

```
library(readxl)
```

```
## Warning: package 'readxl' was built under R version 4.3.3
```

```
acc_data_train <- read.csv("C:\\Users\\dhruv\\Downloads\\pml-training.csv")
acc_data_test <- read.csv("C:\\Users\\dhruv\\Downloads\\pml-testing.csv")
```

The data consists of many columns with near Zero variances , hence to improve our model we will get rid of these variables

```r
NZ <- nearZeroVar(acc_data_train)
NZ1 <- nearZeroVar(acc_data_test)
acc_data_train <- acc_data_train[,-NZ]
acc_data_test <- acc_data_test[,-NZ]
```

Removing Variables with NA's

```r
label <- apply(acc_data_train, 2, function(x) mean(is.na(x))) > 0.95
label1 <- apply(acc_data_test, 2, function(x) mean(is.na(x))) > 0.95
acc_data_train <- acc_data_train[, -which(label, label == FALSE)]
acc_data_test <- acc_data_test[, -which(label1, label == FALSE)]
```

2. Predictive Model Selection : Random Forest

Random Forest is an ensemble learning method used for classification, regression, and other tasks. It works by constructing multiple decision trees during training and combining their outputs (e.g., majority voting for classification or averaging for regression) to improve accuracy and reduce overfitting. The algorithm introduces randomness by selecting a random subset of features for splitting at each tree node, making it robust to noise and capable of handling large datasets with high dimensionality.

```r
modFit <- train(classe ~ roll_belt, data = acc_data_train , method = "rf")
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid mtry:
## reset to within valid range
```

```
modFit$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: 1
##
##         OOB estimate of  error rate: 49.72%
## Confusion matrix:
##      A    B   C    D    E class.error
## A 3776  505 445  759   95   0.3232975
## B 1706 1303 414  349   25   0.6568343
## C 1789  276 949  394   14   0.7226768
## D 1287  101 248 1396  184   0.5659204
## E  659  149 128  230 2441   0.3232603
```

3. Validating Model and creating plots

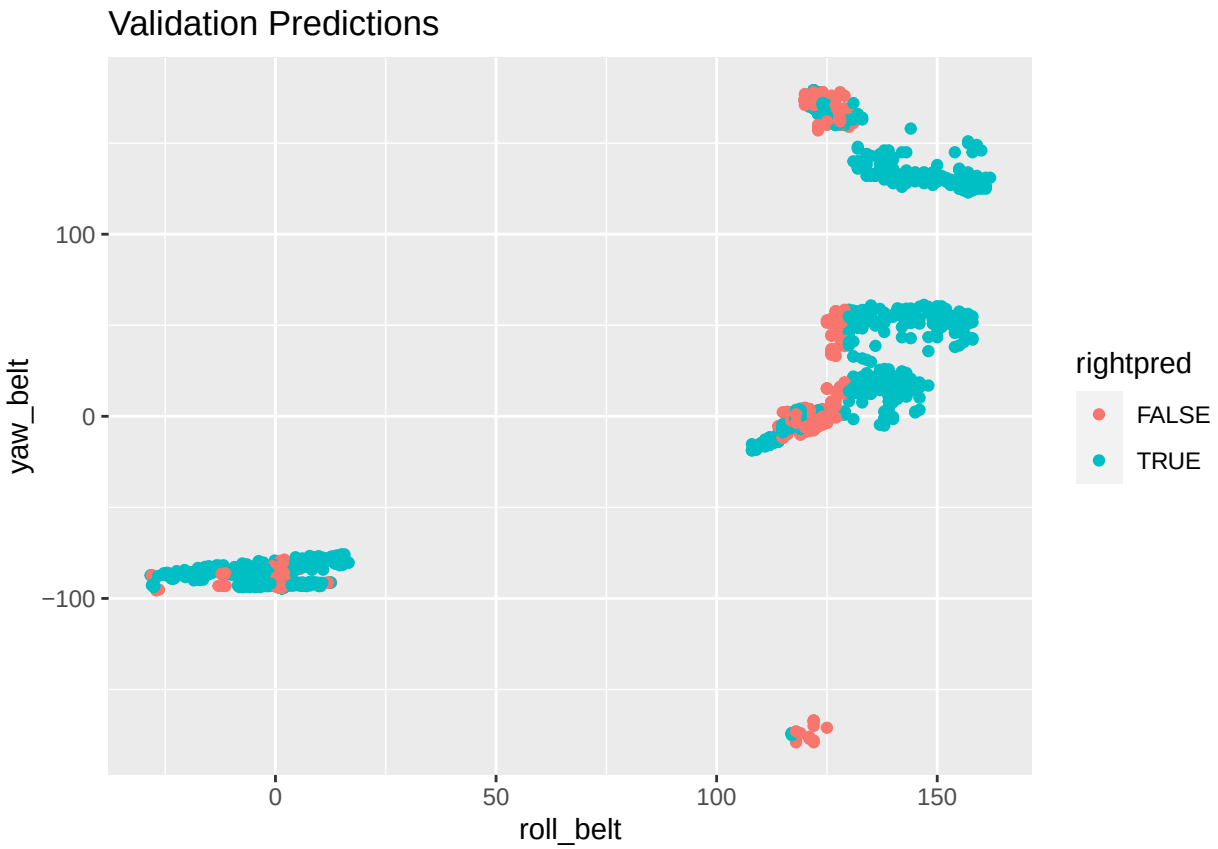
Creating a Validation Set within the training set to validate results from the predictive model built

```
inTrain <- createDataPartition(y= acc_data_train$classe ,p=0.7, list=FALSE)
validate <- acc_data_train[-inTrain,]

pred <- predict(modFit,validate)
validate$rightpred <- pred == validate$classe

##plots
qplot(roll_belt,yaw_belt,colour=rightpred,data=validate,main="Validation Predictions")
```

```
## Warning: ‘qplot()‘ was deprecated in ggplot2 3.4.0.
## This warning is displayed once every 8 hours.
## Call ‘lifecycle::last_lifecycle_warnings()‘ to see where this warning was
## generated.
```



4. Testing Model

Testing the model built on the given test set

```
pred_test <- predict(modFit,acc_data_test)
```

5. Predicting Results

We get the results as :

```
pred_test
```

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