PracticalMachineLearning

Reb Paralleon

9/6/2022

Overview

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, the goal is 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.

Loading the Dataset

The initial step to this project is to load the Weight Lifting Exercise datasets. The datasets are already partitioned into train and test split where we use train set for tuning our models while we use the test set to generate our predictions.

```r
train_data <- read.csv("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv")
test_data <- read.csv("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv")
```

Exploring the Data

As described in our overview, the model is expected to predict 5 different exercise class (`classe` variable) from our 6 participants. Let us verify if this is aligned to our data set.

```r
# Participants
table(train_data$user_name)
```

```
##
##   adelmo carlitos  charles   eurico   jeremy    pedro
##     3892     3112     3536     3070     3402     2610
```

```r
# Exercise Classifications
table(train_data$classe)
```

```
##
##    A    B    C    D    E
## 5580 3797 3422 3216 3607
```

Upon checking the dataset, we have 6 participants namely Adelmo, Carlitos, Charles, Eurico, Jeremy, and Pedro. On the other hand, we have also verified that we have 5 different class exercise (Class A to E). We have confirmed that we have confirmed that the variable to be predicted is in our dataset. Let us check the other variables we can potentially use when we fit our model as our predictors.

```
dim(train_data)
```

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

Our dataset currently has 160 columns. We can explore the option to minimize the number of variables to be used when we clean our data for better model explainability.

Cleaning the Data

We can start by removing columns that has too many NULL or blank values

```
# NULL Value Count
table(colSums(is.na(train_data) | train_data == ""))
```

```
##
##     0 19216
##    60   100
```

Out of 160 columns, 100 of which have 98% null values (19,216/19,622). Removing them will result to 60 columns remaining. Let us check what other variables we can exclude that may not be relevant as predictors to our exercise classifications.

```
train_data = train_data[ , colSums(is.na(train_data)) == 0]
train_data = train_data[ , colSums(train_data == "" ) == 0]
str(train_data[,c(1:10)]) # Just to preview first 10 columns.
```

```
## 'data.frame':    19622 obs. of  10 variables:
##  $ X                   : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ user_name           : chr  "carlitos" "carlitos" "carlitos" "carlitos" ...
##  $ raw_timestamp_part_1: int  1323084231 1323084231 1323084231 1323084232 1323084232 1323084232 1323
##  $ raw_timestamp_part_2: int  788290 808298 820366 120339 196328 304277 368296 440390 484323 484434
##  $ cvtd_timestamp      : chr  "05/12/2011 11:23" "05/12/2011 11:23" "05/12/2011 11:23" "05/12/2011 1
##  $ new_window          : chr  "no" "no" "no" "no" ...
##  $ num_window          : int  11 11 11 12 12 12 12 12 12 12 ...
##  $ 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 ...
```

From our data structure, we can remove the index, timestamps and the window columns (total of 7) since they do not pertain to any kind of measurement that directly describe the participant's weightlifting. This leaves us to a total of 53 variables (52 if excluding the classification column).

```
train_dim <- train_data[,c(1:7)]
train_data <- train_data[,-c(1:7)]
```

With our final variables on hand, we can now partition our data to train and validation split. We use our train set for fitting the model and our validation set to measure the model's performance. The validation set will serve as our out-of-sample evaluation set.

```r
inTrain <- createDataPartition(train_data$classe, p=0.7, list=FALSE)
train_set <- train_data[inTrain,]
validation_set <- train_data[-inTrain,]
```

Bulding the Model

Since the nature of our project is to predict classifications and we still have 52 variables to work on which is still relatively high, using **Random Forest** may be a good option. Random Forest is great with high dimensionality since it works with subsets of data.

In tuning the model, different hyperparameters were tested to find the optimal out-of-sample accuracy and we arrived at 250 trees. For our control, we use cross validation (5) which balances the training time and bias.

```r
#Cross-validation
rfControl <- trainControl(method="cv", 5)

#Train the model
rfModel <- train(classe~., data=train_set, method="rf", trControl = rfControl, tuneLength = 5, ntree=25
rfModel
```

```
## Random Forest
##
## 13737 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 10990, 10990, 10991, 10988, 10989
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9908275  0.9883959
##   14    0.9932300  0.9914362
##   27    0.9906090  0.9881200
##   39    0.9879882  0.9848045
##   52    0.9841299  0.9799236
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 14.
```

```r
rfPredictions <- predict(rfModel, validation_set)
confusionMatrix(factor(validation_set$classe), rfPredictions)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1674    0    0    0    0
##          B    3 1136    0    0    0
##          C    0    7 1019    0    0
##          D    0    0   11  951    2
##          E    0    0    2    1 1079
##
## Overall Statistics
##
##                Accuracy : 0.9956
##                  95% CI : (0.9935, 0.9971)
##     No Information Rate : 0.285
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9944
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9982   0.9939   0.9874   0.9989   0.9981
## Specificity            1.0000   0.9994   0.9986   0.9974   0.9994
## Pos Pred Value         1.0000   0.9974   0.9932   0.9865   0.9972
## Neg Pred Value         0.9993   0.9985   0.9973   0.9998   0.9996
## Prevalence             0.2850   0.1942   0.1754   0.1618   0.1837
## Detection Rate         0.2845   0.1930   0.1732   0.1616   0.1833
## Detection Prevalence   0.2845   0.1935   0.1743   0.1638   0.1839
## Balanced Accuracy      0.9991   0.9966   0.9930   0.9982   0.9988
```

Upon evaluating our model's performance on our out-of-sample data set, we got an **`overall accuracy of 99%`**.

Predicting the Test Set

The final step to this project is to generate our predictions to our test set (total of 20 observations) using our fitted model.

```r
train_cols <- colnames(train_data)
test_cols <- train_cols[train_cols != "classe"]
test_set <- test_data[, test_cols]

test_pred <- predict(rfModel, test_set)
test_pred
```

```
##  [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
```