Practical-Machine-Learning-Project

Prafful Agrawal

July 23, 2020

Introduction

Devices such as *Jawbone Up*, *Nike FuelBand*, and *Fitbit* enable humans to collect a large amount of data about their personal activity at a relatively inexpensive price. Using these devices, 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, we use data from the accelerometers on the belt, forearm, arm, and dumbell of **6 participants** who were asked to perform barbell lifts first *correctly* and then *incorrectly* in **5 different ways**. Our goal is to predict the manner in which they did the exercise by using the above data.

The data for this project come from this source: http://groupware.les.inf.puc-rio.br/har (see the section on the Weight Lifting Exercise Dataset).

Data Preprocessing and Exploratory Analysis

Import the packages.

```
library(caret)
library(dplyr)
```

The *training data* for this project are available here:

https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv

The *testing data* are available here:

https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv

We download the data in the `data` folder.

```
# Check to see if the directory exists
if(!file.exists("./data")) {dir.create("data")}

# Download Training data
if(!file.exists("./data/pml-training.csv")) {
  url <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
  download.file(url, destfile = "./data/pml-training.csv")
}

# Download Testing data
if(!file.exists("./data/pml-testing.csv")) {
  url <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"
  download.file(url, destfile = "./data/pml-testing.csv")
}
```

And, then read the data into a *dataframe*.

```
# Read Training data
training_csv <- read.csv("./data/pml-training.csv")

# Read Testing data
testing_csv <- read.csv("./data/pml-testing.csv")
```

Look at the dimensions of training data.

```
dim(training_csv)
```

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

Look at the dimensions of testing data.

```
dim(testing_csv)
```

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

Look at the structure of training data.

```
str(training_csv)
```

```
## 'data.frame':    19622 obs. of  160 variables:
##  $ X                       : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ user_name               : Factor w/ 6 levels "adelmo","carlitos",..: 2 2 2 2 2 2 2 2 2 2 ...
##  $ raw_timestamp_part_1    : int  1323084231 1323084231 1323084231 1323084232 1323084232 1323084232 13230
84232 1323084232 1323084232 1323084232 ...
##  $ raw_timestamp_part_2    : int  788290 808298 820366 120339 196328 304277 368296 440390 484323 484434
...
##  $ cvtd_timestamp          : Factor w/ 20 levels "02/12/2011 13:32",..: 9 9 9 9 9 9 9 9 9 9 ...
##  $ new_window              : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 1 ...
##  $ 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 ...
##  $ total_accel_belt        : int  3 3 3 3 3 3 3 3 3 3 ...
##  $ kurtosis_roll_belt      : Factor w/ 397 levels "","-0.016850",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ kurtosis_picth_belt     : Factor w/ 317 levels "","-0.021887",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ kurtosis_yaw_belt       : Factor w/ 2 levels "","#DIV/0!": 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_roll_belt      : Factor w/ 395 levels "","-0.003095",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_roll_belt.1    : Factor w/ 338 levels "","-0.005928",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_yaw_belt       : Factor w/ 2 levels "","#DIV/0!": 1 1 1 1 1 1 1 1 1 1 ...
##  $ max_roll_belt           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_picth_belt          : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_yaw_belt            : Factor w/ 68 levels "","-0.1","-0.2",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ min_roll_belt           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_pitch_belt          : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_yaw_belt            : Factor w/ 68 levels "","-0.1","-0.2",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ amplitude_roll_belt     : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_pitch_belt    : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_yaw_belt      : Factor w/ 4 levels "","#DIV/0!","0.00",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ var_total_accel_belt    : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_roll_belt           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_roll_belt        : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_roll_belt           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_pitch_belt          : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_pitch_belt       : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_pitch_belt          : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_yaw_belt            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_yaw_belt         : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_yaw_belt            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ gyros_belt_x            : num  0 0.02 0 0.02 0.02 0.02 0.02 0.02 0.02 0.03 ...
##  $ gyros_belt_y            : num  0 0 0 0 0.02 0 0 0 0 0 ...
##  $ gyros_belt_z            : num  -0.02 -0.02 -0.02 -0.03 -0.02 -0.02 -0.02 -0.02 -0.02 0 ...
##  $ accel_belt_x            : int  -21 -22 -20 -22 -21 -21 -22 -22 -20 -21 ...
##  $ accel_belt_y            : int  4 4 5 3 2 4 3 4 2 4 ...
##  $ accel_belt_z            : int  22 22 23 21 24 21 21 21 24 22 ...
##  $ magnet_belt_x           : int  -3 -7 -2 -6 -6 0 -4 -2 1 -3 ...
##  $ magnet_belt_y           : int  599 608 600 604 600 603 599 603 602 609 ...
##  $ magnet_belt_z           : int  -313 -311 -305 -310 -302 -312 -311 -313 -312 -308 ...
##  $ roll_arm                : num  -128 -128 -128 -128 -128 -128 -128 -128 -128 -128 ...
##  $ pitch_arm               : num  22.5 22.5 22.5 22.1 22.1 22 21.9 21.8 21.7 21.6 ...
##  $ yaw_arm                 : num  -161 -161 -161 -161 -161 -161 -161 -161 -161 -161 ...
##  $ total_accel_arm         : int  34 34 34 34 34 34 34 34 34 34 ...
##  $ var_accel_arm           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_roll_arm            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_roll_arm         : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_roll_arm            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_pitch_arm           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_pitch_arm        : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_pitch_arm           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_yaw_arm             : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_yaw_arm          : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_yaw_arm             : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ gyros_arm_x             : num  0 0.02 0.02 0.02 0 0.02 0 0.02 0.02 0.02 ...
##  $ gyros_arm_y             : num  0 -0.02 -0.02 -0.03 -0.03 -0.03 -0.03 -0.02 -0.03 -0.03 ...
##  $ gyros_arm_z             : num  -0.02 -0.02 -0.02 0.02 0 0 0 0 -0.02 -0.02 ...
```

```
##  $ accel_arm_x            : int  -288 -290 -289 -289 -289 -289 -289 -289 -288 -288 ...
##  $ accel_arm_y            : int  109 110 110 111 111 111 111 111 109 110 ...
##  $ accel_arm_z            : int  -123 -125 -126 -123 -123 -122 -125 -124 -122 -124 ...
##  $ magnet_arm_x           : int  -368 -369 -368 -372 -374 -369 -373 -372 -369 -376 ...
##  $ magnet_arm_y           : int  337 337 344 344 337 342 336 338 341 334 ...
##  $ magnet_arm_z           : int  516 513 513 512 506 513 509 510 518 516 ...
##  $ kurtosis_roll_arm      : Factor w/ 330 levels "","-0.02438",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ kurtosis_picth_arm     : Factor w/ 328 levels "","-0.00484",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ kurtosis_yaw_arm       : Factor w/ 395 levels "","-0.01548",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_roll_arm      : Factor w/ 331 levels "","-0.00051",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_pitch_arm     : Factor w/ 328 levels "","-0.00184",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_yaw_arm       : Factor w/ 395 levels "","-0.00311",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ max_roll_arm           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_picth_arm          : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_yaw_arm            : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_roll_arm           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_pitch_arm          : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_yaw_arm            : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_roll_arm     : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_pitch_arm    : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_yaw_arm      : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ roll_dumbbell          : num  13.1 13.1 12.9 13.4 13.4 ...
##  $ pitch_dumbbell         : num  -70.5 -70.6 -70.3 -70.4 -70.4 ...
##  $ yaw_dumbbell           : num  -84.9 -84.7 -85.1 -84.9 -84.9 ...
##  $ kurtosis_roll_dumbbell : Factor w/ 398 levels "","-0.0035","-0.0073",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ kurtosis_picth_dumbbell: Factor w/ 401 levels "","-0.0163","-0.0233",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ kurtosis_yaw_dumbbell  : Factor w/ 2 levels "","#DIV/0!": 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_roll_dumbbell : Factor w/ 401 levels "","-0.0082","-0.0096",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_pitch_dumbbell: Factor w/ 402 levels "","-0.0053","-0.0084",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_yaw_dumbbell  : Factor w/ 2 levels "","#DIV/0!": 1 1 1 1 1 1 1 1 1 1 ...
##  $ max_roll_dumbbell      : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_picth_dumbbell     : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_yaw_dumbbell       : Factor w/ 73 levels "","-0.1","-0.2",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ min_roll_dumbbell      : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_pitch_dumbbell     : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_yaw_dumbbell       : Factor w/ 73 levels "","-0.1","-0.2",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ amplitude_roll_dumbbell : num  NA NA NA NA NA NA NA NA NA NA ...
##   [list output truncated]
```

It appears that there are a significant number of columns (*variables*) having mostly missing data in them. Let us remove the columns containing more than *50 %* of their values as **NA** values.

```r
# Modified Training dataset
training_csv_mod <- training_csv[, -which(colMeans(is.na(training_csv)
                                                   | training_csv == "") > 0.50)]

# Modified Testing dataset
testing_csv_mod <- testing_csv[, -which(colMeans(is.na(training_csv)
                                                   | training_csv == "") > 0.50)]
```

Let us format the *three* `timestamp` columns.

```r
# Format date in Training data
training_csv_mod$cvtd_timestamp <- as.POSIXct(training_csv_mod$cvtd_timestamp,
                                               format = "%d/%m/%Y %H:%M")

# Format date in Testing data
testing_csv_mod$cvtd_timestamp <- as.POSIXct(testing_csv_mod$cvtd_timestamp,
                                               format = "%d/%m/%Y %H:%M")
```

```r
# Format time in Training data
training_csv_mod$raw_timestamp_part_1 <- as.POSIXlt(training_csv_mod$raw_timestamp_part_1,
                                                    origin = "1970-01-01",
                                                    tz = "UTC")
training_csv_mod$raw_timestamp_part_2 <- as.POSIXlt(training_csv_mod$raw_timestamp_part_2*1e-6,
                                                    origin = training_csv_mod$raw_timestamp_part_1,
                                                    tz = "UTC")

# Format time in Testing data
testing_csv_mod$raw_timestamp_part_1 <- as.POSIXlt(testing_csv_mod$raw_timestamp_part_1,
                                                   origin = "1970-01-01",
                                                   tz = "UTC")
testing_csv_mod$raw_timestamp_part_2 <- as.POSIXlt(testing_csv_mod$raw_timestamp_part_2*1e-6,
                                                   origin = testing_csv_mod$raw_timestamp_part_1,
                                                   tz = "UTC")
```

It is observed from above that the *three* `timestamp` columns represents the same piece of information split up into three parts and hence, are redundant. Let us format these columns into *one* variable. First, drop the redundant columns.

```r
# Drop the redundant columns
training_csv_mod <- training_csv_mod[, -which(names(training_csv_mod)
                                              %in% c("raw_timestamp_part_1", "cvtd_timestamp"))]
testing_csv_mod <- testing_csv_mod[, -which(names(testing_csv_mod)
                                            %in% c("raw_timestamp_part_1", "cvtd_timestamp"))]
```

Next, rename the `timestamp` variable.

```r
## Rename the column to 'time_stamp'
training_csv_mod <- rename(training_csv_mod, timestamp = raw_timestamp_part_2)
testing_csv_mod <- rename(testing_csv_mod, timestamp = raw_timestamp_part_2)
```

Finally, we have to convert the *time* format back to *numeric* format for modelling.

```r
## Convert back to numeric format for modelling
training_csv_mod$timestamp <- as.numeric(training_csv_mod$timestamp)
testing_csv_mod$timestamp <- as.numeric(testing_csv_mod$timestamp)
```

Now, check whether any column has *near-zero* variance.

```r
# Check for near-zero variance predictors
names(training_csv_mod)[nearZeroVar(training_csv_mod, saveMetrics = T)$nzv]
```

```
## [1] "new_window"
```

```r
names(testing_csv_mod)[nearZeroVar(testing_csv_mod, saveMetrics = T)$nzv]
```

```
## [1] "new_window"
```

This column will not be helpful during the modelling so let us drop it.

```r
# Drop near-zero variance predictor
training_csv_mod <- training_csv_mod[, !nearZeroVar(training_csv_mod, saveMetrics = T)$nzv]
testing_csv_mod <- testing_csv_mod[, !nearZeroVar(testing_csv_mod, saveMetrics = T)$nzv]
```

Lastly, the first column `X` is just the index variable, so drop it.

```r
# Drop the index variable
training_csv_mod <- training_csv_mod[, -which(names(training_csv_mod) == "X")]
testing_csv_mod <- testing_csv_mod[, -which(names(testing_csv_mod) == "X")]
```

Check the final dimensions of training and testing data.

```r
# Final dimensions
dim(training_csv_mod)
```

```
## [1] 19622    56
```

```
dim(testing_csv_mod)
```

```
## [1] 20 56
```

Modelling

Set the seed for reproducibility.

```
set.seed(123)
```

Let us split the training data into *three* subsets for modelling, namely `training_set` (about *50 %*) for model fitting, `testing_set` (about *20 %*) for model tuning and `validation_set` (about *30 %*) for model evaluation.

```
# Split training data into build data and validation set
inBuild <- createDataPartition(y = training_csv_mod$classe,
                               p = 0.70,
                               list = FALSE)

validation_set <- training_csv_mod[-inBuild, ]
buildData <- training_csv_mod[inBuild, ]

# Split build data into training set and testing set
inTrain <- createDataPartition(y = buildData$classe,
                               p = 0.70,
                               list = FALSE)

training_set <- buildData[inTrain, ]
testing_set <- buildData[-inTrain, ]
```

Let us fit a number of different models and compare them to find the most suitable model for our problem.

During modelling, we will use `center` and `scale` preprocessing on the data. We will also use *repeated K-fold cross validation* with `k=3` number of folds and `n=5` number of repetitions.

1. Decision tree

```
fit_rpart <- train(form = classe~.,
                   data = training_set,
                   method = "rpart",
                   preProcess = c("center", "scale"),
                   tuneLength = 3,
                   trControl = trainControl(method = "repeatedcv", number = 3, repeats = 5))
```

```
# For testing set
pred_t_rpart <- predict(fit_rpart, testing_set)

# For validation set
pred_v_rpart <- predict(fit_rpart, validation_set)
```

2. K-nearest neighbours

```
fit_knn <- train(form = classe~.,
                 data = training_set,
                 method = "knn",
                 preProcess = c("center", "scale"),
                 tuneLength = 3,
                 trControl = trainControl(method = "repeatedcv", number = 3, repeats = 5))
```

```
# For testing set
pred_t_knn <- predict(fit_knn, testing_set)

# For validation set
pred_v_knn <- predict(fit_knn, validation_set)
```

3. Support vector machine

```
fit_svmLinear <- train(form = classe~.,
                       data = training_set,
                       method = "svmLinear",
                       preProcess = c("center", "scale"),
                       tuneLength = 3,
                       trControl = trainControl(method = "repeatedcv", number = 3, repeats = 5))
```

```
# For testing set
pred_t_svmLinear <- predict(fit_svmLinear, testing_set)

# For validation set
pred_v_svmLinear <- predict(fit_svmLinear, validation_set)
```

Let us look at the accuracy of the above models on the `testing_set`.

```
# Decision tree
confusionMatrix(pred_t_rpart, testing_set$classe)$overall["Accuracy"]
```

```
##  Accuracy
## 0.4910151
```

```
# K-nearest neighbours
confusionMatrix(pred_t_knn, testing_set$classe)$overall["Accuracy"]
```

```
##  Accuracy
## 0.9538611
```

```
# Support vector machine
confusionMatrix(pred_t_svmLinear, testing_set$classe)$overall["Accuracy"]
```

```
##  Accuracy
## 0.8023312
```

From above, we see can see that the accuracy from the **K-nearest Neighbours** model is the highest among the given models (about *95 %*).

We can use model stacking/ensemble technique to stack the above three models and improve the accuracy further.

Ensemble

For model stacking, we will again try different algorithms for the *top* layer and find the best model with respect to accuracy.

We will have to prepare a *stacked* dataframe consisting of the predictions from the above three models and the actual `classe` variable from the `testing_set`. We will use this for training.

```
stacked_df_t <- data.frame(pred_rpart = pred_t_rpart,
                           pred_knn = pred_t_knn,
                           pred_svmLinear = pred_t_svmLinear,
                           classe = testing_set$classe)
```

We will use *bootstraping* with `n=10` number of repetitions.

1. Bagged decision tree

```
stacked_treebag <- train(form = classe~.,
                         data = stacked_df_t,
                         method = "treebag",
                         tuneLength = 4,
                         trControl = trainControl(method = "boot", number = 10))
```

```
pred_st_treebag <- predict(stacked_treebag, stacked_df_t)
```

2. Gradient boosting machine

```
stacked_gbm <- train(form = classe~.,
                     data = stacked_df_t,
                     method = "gbm",
                     verbose = FALSE,
                     tuneLength = 4,
                     trControl = trainControl(method = "boot", number = 10))
```

```
pred_st_gbm <- predict(stacked_gbm, stacked_df_t)
```

3. Random forest

```
stacked_rf <- train(form = classe~.,
                    data = stacked_df_t,
                    method = "rf",
                    tuneLength = 4,
                    trControl = trainControl(method = "boot", number = 10))
```

```
pred_st_rf <- predict(stacked_rf, stacked_df_t)
```

Let us look at the accuracy of the stacked models on the `testing_set`.

```
# Bagged decision tree
confusionMatrix(pred_st_treebag, testing_set$classe)$overall["Accuracy"]
```

```
##  Accuracy
## 0.9565323
```

```
# Gradient boosting machine
confusionMatrix(pred_st_gbm, testing_set$classe)$overall["Accuracy"]
```

```
##  Accuracy
## 0.9538611
```

```
# Random forest
confusionMatrix(pred_st_rf, testing_set$classe)$overall["Accuracy"]
```

```
##  Accuracy
## 0.9558038
```

From above, we can see that the *stacked* model using **Bagged Decision Tree** model as the *top* layer is the most accurate. It is also observed that the increase in accuracy is **only incremental**.

We will use this model to calculate the *Out-of-sample error*.

Out-of-sample Error

We will prepare a *stacked* dataframe using `validation_set` similar to the previous step. We will use this for validation.

```
stacked_df_v <- data.frame(pred_rpart = pred_v_rpart,
                           pred_knn = pred_v_knn,
                           pred_svmLinear = pred_v_svmLinear)
```

Let us check the *out-of-sample* error.

```
pred_sv_treebag <- predict(stacked_treebag, stacked_df_v)
```

```
oos_error <- round(((1 - confusionMatrix(pred_sv_treebag, validation_set$classe)$overall["Accuracy"]) * 100)
, 2)
print(paste0("The Out-of-sample error is ", oos_error, " %"))
```

```
## [1] "The Out-of-sample error is 5.1 %"
```

Prediction

Finally, we will use the above model to predict the `classe` variable for `testing_csv_mod` dataset.

First, we will predict the *base* layer.

```
# Decision tree
prediction_rpart <- predict(fit_rpart, testing_csv_mod)

# K-nearest neighbours
prediction_knn <- predict(fit_knn, testing_csv_mod)

# Support vector machine
prediction_svmLinear <- predict(fit_svmLinear, testing_csv_mod)
```

Prepare the *stacked* dataframe.

```
stacked_prediction <- data.frame(pred_rpart = prediction_rpart,
                                 pred_knn = prediction_knn,
                                 pred_svmLinear = prediction_svmLinear)
```

```
prediction <- predict(stacked_treebag, stacked_prediction)
print(prediction)
```

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

The above predictions were submitted as part of the assignment.