Practical ML

Shubham Patil

31/07/2020

Reading data

The data links are given below:

- Training Set
- Testing Set
- Documentation

Please, download the datasets on your working directory before proceeding further.

```
train <- read.csv("pml-training.csv")
test <- read.csv("pml-testing.csv")
```

Cleaning data

Checking the characteristics of the data.

```
dim(train)
```

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

```
str(train)
```

```
## 'data.frame':    19622 obs. of  160 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 1
##  $ raw_timestamp_part_2    : int  788290 808298 820366 120339 196328 304277 368296 440390 484323 484
##  $ cvtd_timestamp          : chr  "05/12/2011 11:23" "05/12/2011 11:23" "05/12/2011 11:23" "05/12/20
##  $ 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 ...
##  $ total_accel_belt        : int  3 3 3 3 3 3 3 3 3 3 ...
##  $ kurtosis_roll_belt      : chr  "" "" "" "" ...
##  $ kurtosis_picth_belt     : chr  "" "" "" "" ...
##  $ kurtosis_yaw_belt       : chr  "" "" "" "" ...
```

```
##  $ skewness_roll_belt      : chr  "" "" "" "" ...
##  $ skewness_roll_belt.1    : chr  "" "" "" "" ...
##  $ skewness_yaw_belt       : chr  "" "" "" "" ...
##  $ 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            : chr  "" "" "" "" ...
##  $ 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            : chr  "" "" "" "" ...
##  $ 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      : chr  "" "" "" "" ...
##  $ 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       : chr  "" "" "" "" ...
##  $ kurtosis_picth_arm      : chr  "" "" "" "" ...
##  $ kurtosis_yaw_arm        : chr  "" "" "" "" ...
##  $ skewness_roll_arm       : chr  "" "" "" "" ...
##  $ skewness_pitch_arm      : chr  "" "" "" "" ...
##  $ skewness_yaw_arm        : chr  "" "" "" "" ...
##  $ 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  : chr  "" "" "" "" ...
##  $ kurtosis_picth_dumbbell : chr  "" "" "" "" ...
##  $ kurtosis_yaw_dumbbell   : chr  "" "" "" "" ...
##  $ skewness_roll_dumbbell  : chr  "" "" "" "" ...
##  $ skewness_pitch_dumbbell : chr  "" "" "" "" ...
##  $ skewness_yaw_dumbbell   : chr  "" "" "" "" ...
##  $ 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        : chr  "" "" "" "" ...
##  $ 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        : chr  "" "" "" "" ...
##  $ amplitude_roll_dumbbell : num  NA NA NA NA NA NA NA NA NA NA ...
##   [list output truncated]
```

There are 19622 observations on 160 variables in the data.

There are two instant takeaways from the structure of the data: 1) The first seven columns of the data can be eliminated for the prediction purposes as they may introduce some noise in our model. 2) There are lot of missing values (NAs) in the data. We can impute these values by means of the respective columns in case of numerical columns. For columns with character values, we can leave them as they are to avoid loss of important data.

```r
train1 <- train[, - c(1:7)]
for (col in names(train1)) {
    if (is.numeric(train1[, col]) | is.integer(train1[, col])) {
        mu <- mean(train1[, col], na.rm = TRUE)
        na_indices <- which(is.na(train1[, col]))
        for (x in na_indices) {
            train1[x, col] <- mu
        }
    }
    else {
        next
    }
}
dim(train1)
```

```
## [1] 19622   153
```

Feature Selection

We can eliminate the features which have near zero variance (too few unique values).

```
nzv <- nearZeroVar(train1)
train2 <- train1[, - nzv]
dim(train2)
```

```
## [1] 19622    53
```

We have got rid of too many garbage variables in the data. We will now remove the variables which are highly correlated with each other. We will specify threshold of correlation to eliminate the variable to 80%.

```
correlations <- cor(train2[, - 53])
highlyCorDescr <- findCorrelation(correlations, cutoff = .80)
train3 <- train2[, - highlyCorDescr]
dim(train3)
```

```
## [1] 19622    40
```

Now, we seem to have tidier data than before. We can now create partitions of the data. We will assign it to `data` object.

```
data <- train3
```

Creating data partitions

We will split the data in 70% training and 30% testing sets. Our predictor variable is `classe`.

```
set.seed(041)
split_indices <- createDataPartition(data$classe, p = 0.7, list = FALSE)
training <- data[split_indices, ]
testing <- data[- split_indices, ]
X <- training[, - 40]
y <- training$classe
dim(training)
```

```
## [1] 13737    40
```

```
dim(testing)
```

```
## [1] 5885   40
```

Building & selecting model

We will build 3 models for the data: 1) Support Vector Machine Classifier 2) Decision Tree Classifier 3) Random Forest Classifier

We will also use 5 cross validation sets to train our models.

Model 1: Linear SVM

```
set.seed(041)
start <- proc.time()
controlparams <- trainControl(method = "cv", 5)
model_1 <- train(classe ~ ., data = training, method = "svmLinear2", trControl = controlparams, type =
predictions_1 <- predict(model_1, testing)
confusionMatrix(factor(testing$classe), predictions_1)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1461   85   73   44   11
##          B  211  701   76   58   93
##          C  124   96  727   59   20
##          D   85   88  129  630   32
##          E   84  217  108  114  559
##
## Overall Statistics
##
##                Accuracy : 0.6929
##                  95% CI : (0.681, 0.7047)
##     No Information Rate : 0.3339
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6091
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.7435   0.5906   0.6532   0.6961  0.78182
## Specificity            0.9457   0.9068   0.9373   0.9329  0.89884
## Pos Pred Value         0.8728   0.6155   0.7086   0.6535  0.51664
## Neg Pred Value         0.8803   0.8976   0.9206   0.9441  0.96752
## Prevalence             0.3339   0.2017   0.1891   0.1538  0.12150
## Detection Rate         0.2483   0.1191   0.1235   0.1071  0.09499
## Detection Prevalence   0.2845   0.1935   0.1743   0.1638  0.18386
## Balanced Accuracy      0.8446   0.7487   0.7953   0.8145  0.84033
```

```
proc.time() - start
```

```
##    user  system elapsed
##  428.02    2.21  432.28
```

Model 2: Decision Tree

```
set.seed(041)
start <- proc.time()
controlparams <- trainControl(method = "cv", 5)
model_2 <- train(classe ~ ., data = training, method = "rpart2", trControl = controlparams)
predictions_2 <- predict(model_2, testing)
confusionMatrix(factor(testing$classe), predictions_2)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1278   46  325   24    1
##          B  160  400  505   63   11
##          C    9   25  967   19    6
##          D   31   68  428  437    0
##          E   20  112  452   65  433
##
## Overall Statistics
##
##                Accuracy : 0.5973
##                  95% CI : (0.5846, 0.6098)
##     No Information Rate : 0.4549
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.494
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.8531  0.61444   0.3612  0.71875  0.96009
## Specificity            0.9097  0.85881   0.9816  0.90013  0.88057
## Pos Pred Value         0.7634  0.35119   0.9425  0.45332  0.40018
## Neg Pred Value         0.9478  0.94711   0.6481  0.96525  0.99625
## Prevalence             0.2545  0.11062   0.4549  0.10331  0.07664
## Detection Rate         0.2172  0.06797   0.1643  0.07426  0.07358
## Detection Prevalence   0.2845  0.19354   0.1743  0.16381  0.18386
## Balanced Accuracy      0.8814  0.73662   0.6714  0.80944  0.92033
```

```
proc.time() - start
```

```
##    user  system elapsed
##    3.76    0.00    3.81
```

Model 3: Random Forest

```
set.seed(041)
start <- proc.time()
controlparams <- trainControl(method = "cv", 5)
model_3 <- train(classe ~ ., data = training, method = "rf", trControl = controlparams)
predictions_3 <- predict(model_3, testing)
confusionMatrix(factor(testing$classe), predictions_3)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1672    2    0    0    0
##          B    7 1126    5    0    1
##          C    0    5 1007   14    0
##          D    5    1   14  944    0
##          E    0    0    1    7 1074
##
## Overall Statistics
##
##                Accuracy : 0.9895
##                  95% CI : (0.9865, 0.9919)
##     No Information Rate : 0.2862
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9867
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9929   0.9929   0.9805   0.9782   0.9991
## Specificity            0.9995   0.9973   0.9961   0.9959   0.9983
## Pos Pred Value         0.9988   0.9886   0.9815   0.9793   0.9926
## Neg Pred Value         0.9972   0.9983   0.9959   0.9957   0.9998
## Prevalence             0.2862   0.1927   0.1745   0.1640   0.1827
## Detection Rate         0.2841   0.1913   0.1711   0.1604   0.1825
## Detection Prevalence   0.2845   0.1935   0.1743   0.1638   0.1839
## Balanced Accuracy      0.9962   0.9951   0.9883   0.9871   0.9987
```

```
proc.time() - start
```

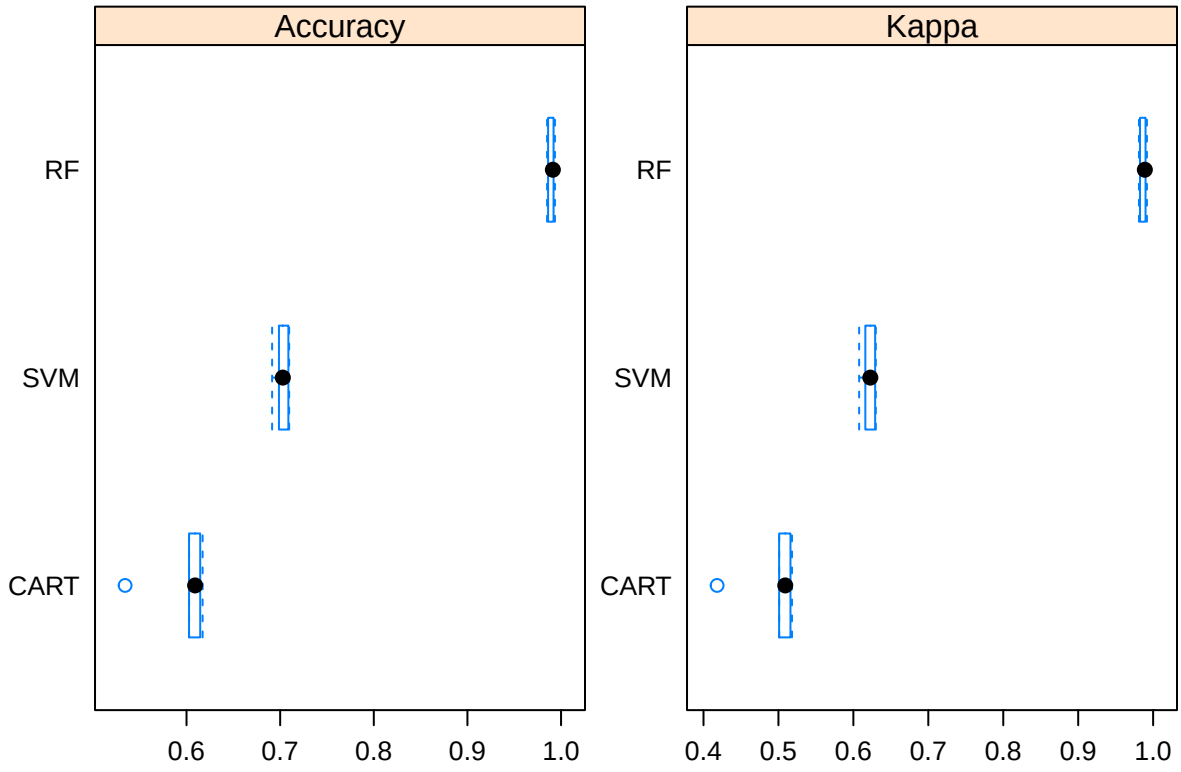
```
##    user  system elapsed
##  848.97    9.62 1260.72
```

Of all the 3 models, Random forest performs exceptionally well with 99% accuracy followed by Linear SVM (69%) and then Decision Tree (59%).

Comparing Models

We will compare the models using resampler function which takes list of the models to compare.

```
models_compare <- resamples(list(SVM = model_1, CART = model_2, RF = model_3))
scales <- list(x=list(relation="free"), y=list(relation="free"))
bwplot(models_compare, scales=scales)
```



As the plot suggests, Random Forest is clearly the winner among the 3 models considered. It has out of sample error rate of `1 - 0.9895`.

Making Predictions

```
results <- predict(model_3, newdata = test)
results
```

```
##  [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
```