Human Activity Recognition

Julio Morales

29-11-2020

Introduction,

Ugulino, Cardador, Vega, Velloso, Milidiu and Fuks in their study entitled: “Wearable Computing: Accelerometers’ Data Classification of Body Postures and Movements”[1], they measured and qualified weight lifting exercise by classifying into 5 classes from correct execution (Class A) to 4 common mistake executions organized from class B to E. “The exercises were performed by six male participants aged between 20-28 years, with little weight lifting experience. (They) made sure that all participants could easily simulate the mistakes in a safe and controlled manner by using a relatively light dumbbell (1.25kg).”[2]

The main idea is to create a prediction model that could recognize excellent execution from wrong ones and deliver a feedback to athletes by orientating which movements they need to fix.

In this report, its goal is to predict the way athletes did the exercise by using data set provided by Ugulino and partners in their study.

Getting and Cleaning Data

Data sets are divided into training and test sets to build a predict model, predict its results and evaluate its accuracy. Information can be get from the following Urls:

Training data set: https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv (https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv)

Test data set: https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv (https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv)

These data sets have 160 variables and 19622 and 20 observations respectively. In a first look of these data sets, 100 variables have more than 95% of its observations with NA, blank or #DIV/0. These variables were removed from each data sets.

On the other hand, Test data set is a new set of data, because it does not have result or Classes. This new data set can be used to forecast its results.

There are 7 non predictor variables that can be removed. They are: “X”, “user_name”, “raw_timestamp_part_1”, “raw_timestamp_part_2”, “cvtd_timestamp”, “new_window” and “num_window”. These variables are related to index, names and time of the observations.

In addition, a non zero variance analysis can be done to find out which variables have unique or closer to unique values by using caret function nearZeroVar: “… It not only removes predictors that have one unique value across samples (zero variance predictors), but also removes predictors that have both 1) few unique values relative to the number of samples and 2) large ratio of the frequency of the most common value to the frequency of the second most common value (near-zero variance predictors).”[3]

None of predictors have been found as non zero variance:

```
rownames(nsvList)
```

```
##  [1] "roll_belt"            "pitch_belt"           "yaw_belt"
##  [4] "total_accel_belt"     "gyros_belt_x"         "gyros_belt_y"
##  [7] "gyros_belt_z"         "accel_belt_x"         "accel_belt_y"
## [10] "accel_belt_z"         "magnet_belt_x"        "magnet_belt_y"
## [13] "magnet_belt_z"        "roll_arm"             "pitch_arm"
## [16] "yaw_arm"              "total_accel_arm"      "gyros_arm_x"
## [19] "gyros_arm_y"          "gyros_arm_z"          "accel_arm_x"
## [22] "accel_arm_y"          "accel_arm_z"          "magnet_arm_x"
## [25] "magnet_arm_y"         "magnet_arm_z"         "roll_dumbbell"
## [28] "pitch_dumbbell"       "yaw_dumbbell"         "total_accel_dumbbell"
## [31] "gyros_dumbbell_x"     "gyros_dumbbell_y"     "gyros_dumbbell_z"
## [34] "accel_dumbbell_x"     "accel_dumbbell_y"     "accel_dumbbell_z"
## [37] "magnet_dumbbell_x"    "magnet_dumbbell_y"    "magnet_dumbbell_z"
## [40] "roll_forearm"         "pitch_forearm"        "yaw_forearm"
## [43] "total_accel_forearm"  "gyros_forearm_x"      "gyros_forearm_y"
## [46] "gyros_forearm_z"      "accel_forearm_x"      "accel_forearm_y"
## [49] "accel_forearm_z"      "magnet_forearm_x"     "magnet_forearm_y"
## [52] "magnet_forearm_z"
```

Finally, at the training data set it is found the following distribution of exercise observations from a total of 19622, about 28% of class A (right execution) and about 17% of the rest of the classes (mistake execution).

```
table(dfTraining$classe)
```

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

Building a Prediction Model.

After exploring and adjusting data sets, for predicting model can be used a random forest to “…build(s) multiple decision trees and merges them together to get a more accurate and stable prediction” [4], in this case, a classification prediction of “Classe” output.

```
set.seed(1234)
rf <- randomForest(classe~.,data=dfTraining)
print(rf)
```

```
## 
## Call:
##  randomForest(formula = classe ~ ., data = dfTraining) 
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 7
## 
##         OOB estimate of  error rate: 0.27%
## Confusion matrix:
##      A    B    C    D    E  class.error
## A 5578    1    0    0    1 0.0003584229
## B   10 3785    2    0    0 0.0031603898
## C    0    9 3411    2    0 0.0032144944
## D    0    0   20 3194    2 0.0068407960
## E    0    0    1    4 3602 0.0013861935
```

This is a first run of random forest prediction model with 7 variables tried at each split, this is a default value corresponding to square root of total variables 52.

As a result, Out of Bag estimate error is 0.27%, that is more than 99% of accuracy. Out of bag if a set of data taken from dfTraining and it is not used to build the model, if not then to test it.

At the Confusion Matrix, its observable error is minimum for each prediction, less than 0.7% for each class with a few misclassification, less than 23, that represents a good model.

On the other hand, after running the random forest model with dfTraining, the accuracy is measured by using a confusion matrix and it can be observed that its perfect: 1 or 100% and zero mistake or misclassification in its prediction as can be visualized at the matrix bellow.

```
p1 <- predict(rf,dfTraining)
confusionMatrix(p1,dfTraining$classe)
```

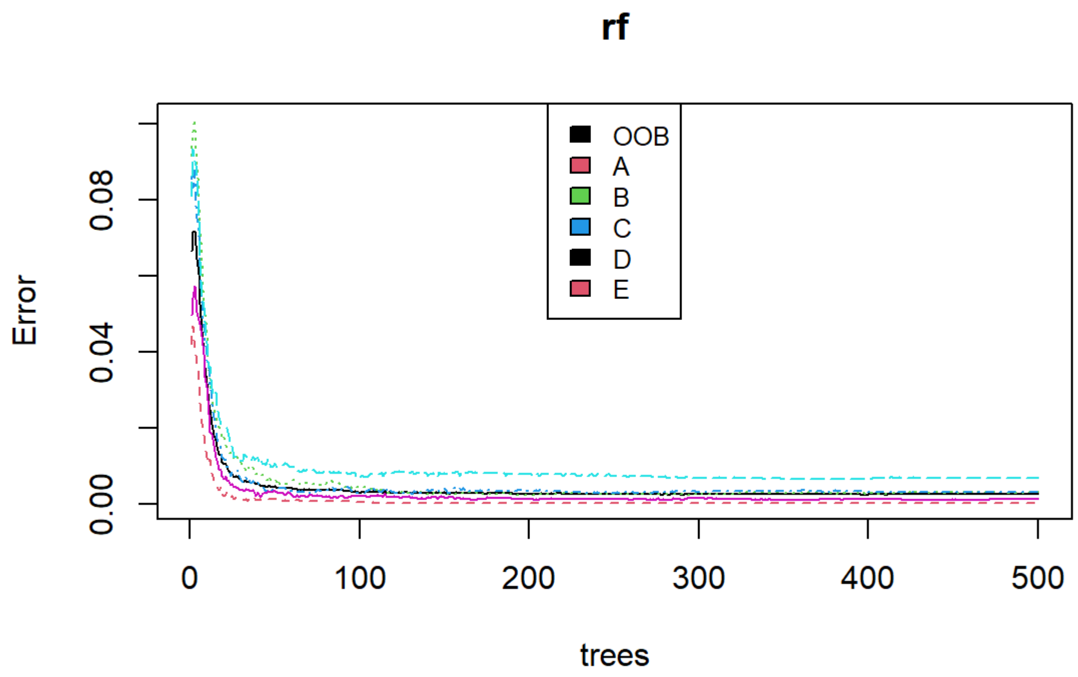
```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    A    B    C    D    E
##          A 5580    0    0    0    0
##          B    0 3797    0    0    0
##          C    0    0 3422    0    0
##          D    0    0    0 3216    0
##          E    0    0    0    0 3607
## 
## Overall Statistics
##                                      
##                Accuracy : 1          
##                  95% CI : (0.9998, 1)
##     No Information Rate : 0.2844     
##     P-Value [Acc > NIR] : < 2.2e-16  
##                                      
##                   Kappa : 1          
##                                      
##  Mcnemar's Test P-Value : NA         
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   1.0000   1.0000   1.0000   1.0000
## Specificity            1.0000   1.0000   1.0000   1.0000   1.0000
## Pos Pred Value         1.0000   1.0000   1.0000   1.0000   1.0000
## Neg Pred Value         1.0000   1.0000   1.0000   1.0000   1.0000
## Prevalence             0.2844   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2844   0.1935   0.1744   0.1639   0.1838
## Detection Prevalence   0.2844   0.1935   0.1744   0.1639   0.1838
## Balanced Accuracy      1.0000   1.0000   1.0000   1.0000   1.0000
```

This prediction is made with the same data to build it, therefore, the accuracy is always high.

Error Rate of Prediction Model

In the following graph of error rate of the random forest, it can be observed that error rate is reduced when the number of trees is increased. In this case, error rate is stabilized or constant when more than 100 trees have been tested and voted.

This information can be used to reconfigure the random forest to consider 100 trees instead of its default: 500 trees. It could help to improve its performance during calculations.

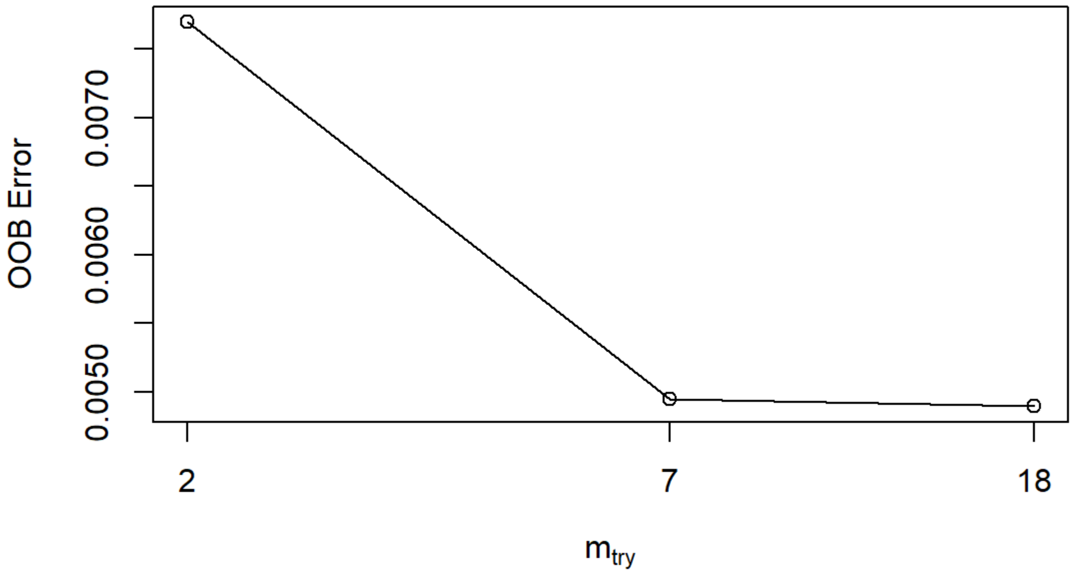


Tune prediction model.

TuneRF is a function that helps to identify the optimal value of mtry or number of variables randomly sampled as candidates at each split to consider for building the trees.

```
t <- tuneRF(dfTraining[,-53],dfTraining[,53],
            stepFactor = 0.4,
            plot = TRUE,
            ntreetry = 100,
            trace = TRUE,
            improve = 0.5
            )
```

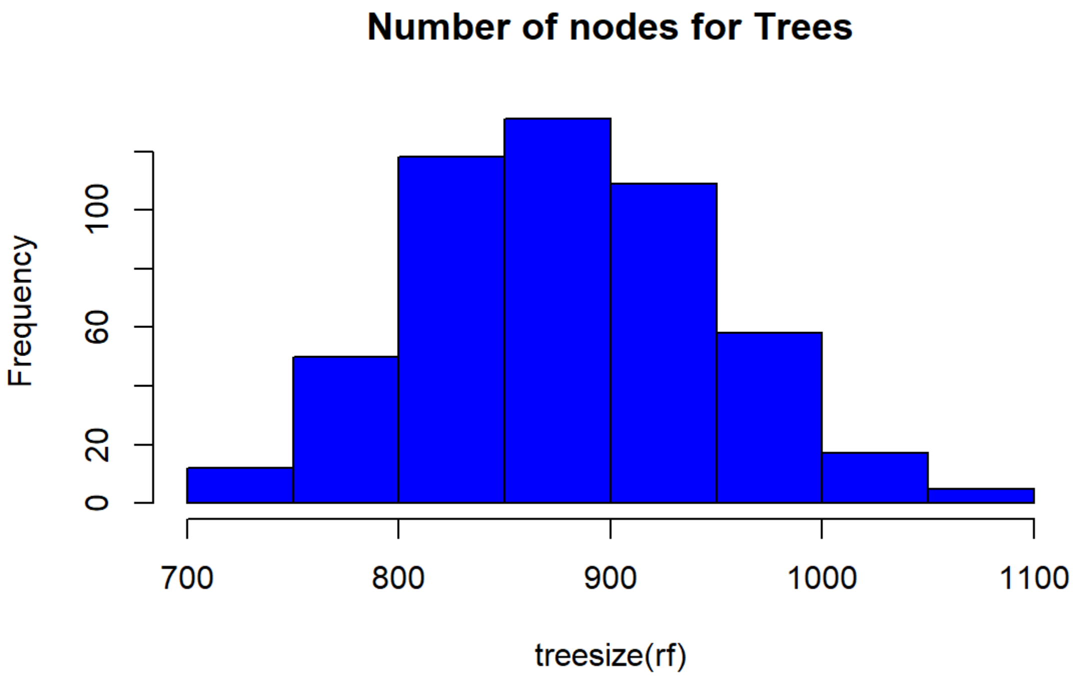
```
## mtry = 7  OOB error = 0.49%
## Searching left ...
## mtry = 18    OOB error = 0.49%
## 0.01030928 0.5
## Searching right ...
## mtry = 2     OOB error = 0.77%
## -0.556701 0.5
```



In this case, it is confirmed that 7 is the optimal number.

Number of nodes for the trees

In the following histogram is showed that more than 300 trees have between 800 and 950 nodes. This information give us an idea about how big the trees are.

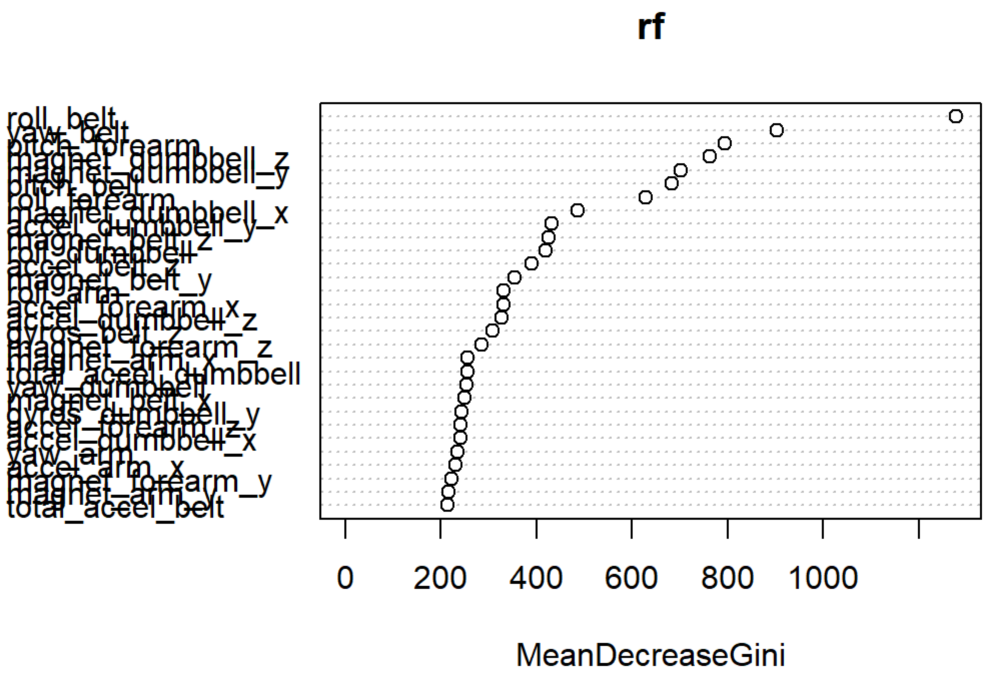


## Variable of

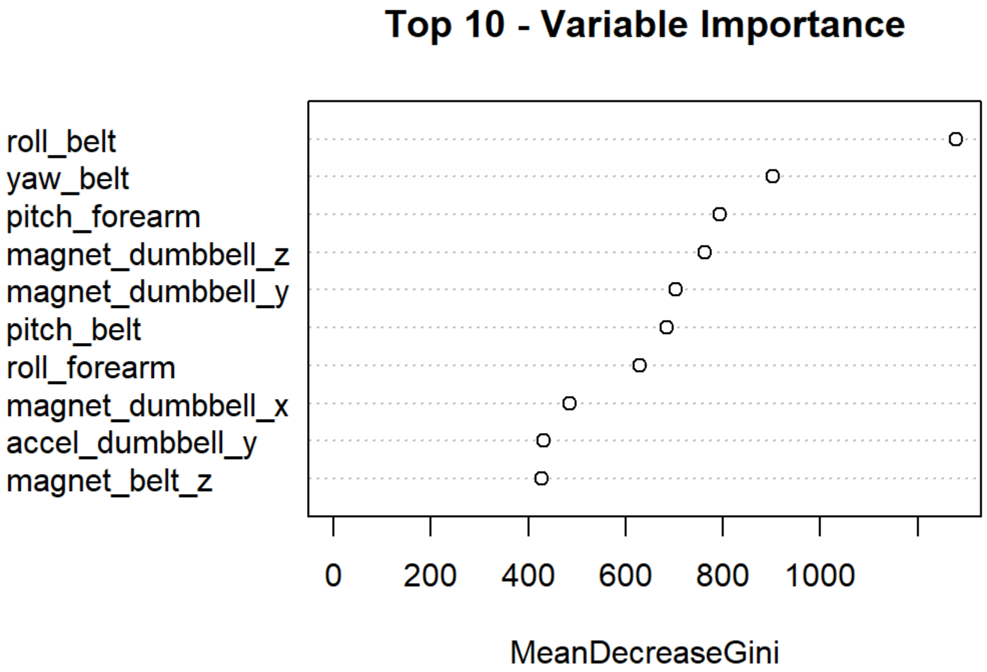
Importance. Random forest can determine which variables have more impact on the outcome of the values, they are called variable of importance.

In the following graphs can be observed all 52 variables and its top 10. Remember that model can be obtained by working with 7 of them.

```
varImpPlot(rf)
```



```
varImpPlot(rf,
           sort = T,
           n.var = 10,
           main = "Top 10 - Variable Importance")
```



Cross Validation.

By using this procedure, the prediction performance of models can be cross validated by reducing the number of predictors, ranked by variable of importance.

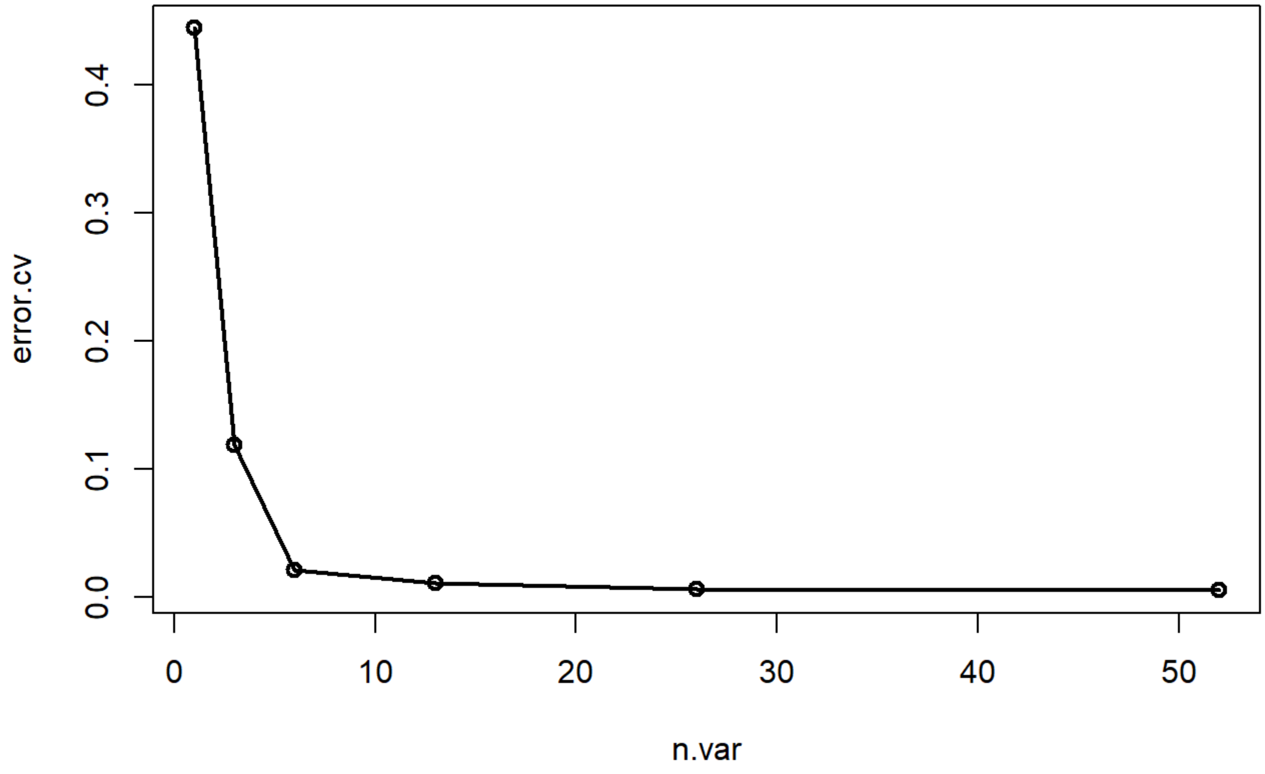
```
result <- rfcv(dfTraining[,-53],dfTraining[,53],cv.fold = 3)
result$n.var
```

```
## [1] 52 26 13  6  3  1
```

```
result$error.cv
```

```
##          52          26          13           6           3           1
## 0.005504026 0.006064621 0.010906126 0.021608399 0.119202935 0.444602997
```

```
with(result, plot(n.var, error.cv, type="o", lwd=2))
```



In this model, error rates increase over 10% when less than 6 variables are used. This confirm the 7 variables used in this report to build the model and obtain an error rate of 0,26%

Forecasting on new data

Finally, this model can be used to forecast on Test data set and gives the following output:

```
p2 <- predict(rf,dfTest)
p2
```

```
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 
##  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
```

Bibliography

Ugulino, W.; Cardador, D.; Vega, K.; Velloso, E.; Milidiu, R.; Fuks, H. Wearable Computing: Accelerometers' Data Classification of Body Postures and Movements. Proceedings of 21st Brazilian Symposium on Artificial Intelligence. Advances in Artificial Intelligence - SBIA 2012. In: Lecture Notes in Computer Science. , pp. 52-61. Curitiba, PR: Springer Berlin / Heidelberg, 2012. ISBN 978-3-642-34458-9. DOI: 10.1007/978-3-642-34459-6_6. http://groupware.les.inf.puc-rio.br/har#weight_lifting_exercises#ixzz6fHf135Td (http://groupware.les.inf.puc-rio.br/har#weight_lifting_exercises#ixzz6fHf135Td)

Leo Breiman and Adele Cutler, Random Forest
https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm
(https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm)

Niklas Donges, A COMPLETE GUIDE TO THE RANDOM FOREST ALGORITHM.https://builtin.com/data-science/random-forest-algorithm (https://builtin.com/data-science/random-forest-algorithm)

Dr. Bharatendra Rai, Random Forest in R - Classification and Prediction Example with Definition & Steps, https://www.youtube.com/watch?v=dJclNIN-TPo (https://www.youtube.com/watch?v=dJclNIN-TPo)

Quotations

1, 2 Human Activity Recognition: http://groupware.les.inf.puc-rio.br/har#weight_lifting_exercises
(http://groupware.les.inf.puc-rio.br/har#weight_lifting_exercises)

3 Near-zero variance predictors. Should we remove them?:
https://tagteam.harvard.edu/hub_feeds/1981/feed_items/367058
(https://tagteam.harvard.edu/hub_feeds/1981/feed_items/367058)

4 https://builtin.com/data-science/random-forest-algorithm (https://builtin.com/data-science/random-forest-algorithm)