Implementation of random forest algorithm with parallel computing in R

Implementation of random forest algorithm with parallel computing in R

N Azizah, L S Riza* and Y Wihardi

Department of Computer Science Education, Universitas Pendidikan Indonesia, Indonesia, Bandung, Indonesia

*Corresponding author's email: lala.s.riza@upi.edu

Abstract. Random forest is a method for building models by combining decision trees or decision trees generated from bootstrap samples and random features. A common problem that often occurs when implementing random forest is long processing time because it uses a lot of data and build many tree models to form random trees because it uses single processor. This research proposes random forest method with parallel computing and implemented in R programming language. Some of the cases used in this research are Iris flower dataset, wine quality and diabetes diagnosis data of Pima Indian woman. The results obtained from the entire study show that the computational time used when running random forest with parallel computing is shorter than when running a regular random forest using only a single processor.

1. Introduction

Random forest was first introduced by Breiman in 2001. In his research shows the advantages of random forest, among others, can produce a lower error, provide good results in the classification, can handle the training data in a very large amount efficiently, and effective methods to estimate missing data [1]. Previous research on random forest was conducted by [2] conducting research on web caching by comparing classification accuracy using CART, MARS, random forest and Tree Net methods. Research on the application of random forest methods in driver analysis [3]. In the research of ensemble method on poverty classification in Jombang Regency, it is found that random forest gives best classification accuracy [4].

However, the common problem that often occurs when implementing random forest is the long processing time when using large amounts of data to build many tree models to form random trees when using single processor. Therefore, a random forest design with parallel computing is proposed. Parallel computing is the union of several computers or servers into a single unit that can work on the process simultaneously or simultaneously. Parallel computing makes programs and processes run faster as more CPUs are used [5]. Parallel computing was once used to improve the performance of advanced encryption standards [6].

There are several studies on parallel computing, one of which is the use of parallel computing to improve computer performance [7]. The programming language R is a programming language and software environment for statistical computing and is supported by R Foundation for Statistical Computing [8]. According to [9] R programming is an integrated software suite facility for data manipulation, calculation and graphical display.

This research is aimed to develop random forest to be used in parallel computing in R. To achieve this objective, we utilize the "foreach" package. It is a package that supports the foreach looping construct [10]. We use this package because it can be used easily for repeating the same procedures along with all cores. Actually, the use of the foreach package for parallel computing can be found in the several literature, such as, [11, 12] and it has been implemented in the gradDescent package [13].

2. Methods

2.1. Implementation of Random Forest in R High Performance Computing

The programming language R or R programming is a programming language and software environment for statistical computing and is supported by R Foundation for Statistical Computing [8]. The R language is an implementation of the S programming language combined with lexical scoping semantics inspired by the scheme.

The first step in this research is import data. In this process the orbit element data is inserted into the R programming language. Here is how to import data directly from the website, to keep in mind is the computer must be connected to the internet.

```
library("httr")
a <- GET("https://archive.ics.uci.edu/ml/machine-learning databases/wine/wine.data")
wine <- read.csv(textConnection(content(a)), header=F)
colnames(wine) <- c('Alcohol', 'Malic', 'Ash',  'Alcalinity', 'Magnesium','Phenols',              'Flavanoids',
                    'Nonflavanoids',              'Proanthocyanins', 'Color', 'Hue', 'Dilution', 'Proline', 'Type')
```

Figure 1. Import the wine dataset directly from the website.

Figures 1 shows the process of retrieving the dataset from the online repository and then stored in CSV form, the next being the naming of the columns of each variable because the dataset retrieved does not have an attribute name so if it is not defined it will by default be named V1, V2, V3, and so on.

Once the orbital element's dataset is imported, the next step is to build the bootstrap function.

```
## shuffle dataset
irisData <- iris[sample(nrow(iris)),]

##set the first 120 rows to be training dataset
iris.tra <- irisData[1:120, ]

## set testing dataset
iris.tst <- irisData[121:nrow(irisData), -ncol(irisData)]

## get real values of testing dataset
real.iris <- irisData[121:nrow(irisData), ncol(irisData)]
```

Figure 2. Code of the data retrieval program at random.

Figure 2 is the program code for retrieving data randomly from the available dataset, the above example is data retrieval from the iris datset. The variable "irisData" is used to store randomly-captured data, "iris.tra" is the name of a variable that contains data for training data, "iris.tst" is a variable name containing data for test data whereas " real.iris "is the name of the variable that contains the original data from the test data (testing). In this step, some steps are taken to run random forest with parallel computing. There are several packages needed to run this stage ie packages "foreach", "doParallel" and "entropy". The first step is to install the three packages, then import or call packages as in Figure 3.

```
# Import packages "foreach"
library(forech)
# Import packages "doParallel"
library(doParallel)
```

Figure 3. Import the R packages.

Next is to build the parallel random forest function shown in Figure 4. Basically, this code contains several components, as follows: to collect parameters, to define number of cores, to perform the foreach package along with numbers of trees, to call the decision tree procedure, and to aggregate the final results.

2.2. Experimental Design

In order to conduct an efficient experiment, a suitable experimental design was created so that a satisfactory trial result was obtained. In this research there are two scenarios, the first scenario is to do the calculation of random forest without parallel computing, while for the second scenario is to do the calculation of random forest with parallel computing.

The first experimental scenario of computing random forest without parallel computing using only one processor to test the case of Iris flower dataset, the quality of wine and diabetes of Pima Indian women. The numTree parameter value is set to 20 and 100 and numFeature is set 2 and 4.

The second experimental scenario to test the case of the Iris dataset, Wine and Pima using a parallel random forest with two processors. Modify the parameters as in the first scenario of numTree 20 and 100 and numFeature 2 and 4.

The third experimental scenario to test the case of the Iris dataset, Wine and Pima using a parallel random forest with three processors. Modify the parameters as in the first and second scenarios of numTree 20 and 100 and numFeature 2 and 4. Recent experimental scenarios to test the case of the Iris, Wine and Pima datasets use a parallel random forest with four processors. Modify the parameters as in the previous scenario of numTree 20 and 100 and numFeature 2 and 4.

Each experimental scenario uses the Iris flower species dataset consisting of 5 variables (4 input variables and 1 output variable) and 150 rows of data. Next use the wine quality dataset consisting of 14 variables (13 input variables and 1 output variable) and 178 rows of data. Lastly, by using the Pima Indian female diabetes dataset consisting of 9 columns (8 input variables and 1 output variable) and 798 rows of data.

```
parRandForest <- function(dataset, typeTask = "classification", controlPRF = list(), controlTree = list()){
  ## set default values of all parameters
  controlTree <- setDefaultParametersIfMissing(controlTree, list(nameFeatures = NULL, typeFeatures = NULL, paramPercent = 0.5, typeEntropy =
"ML", maxDepthTree = 3, typeSplitting = "random", minNumData = 3,  sampleFeature = TRUE, nameClasses = NULL))
  controlPRF <- setDefaultParametersIfMissing(controlPRF, list(numTree = 10, numProcessor = 2, numFeature = round(sqrt(ncol(dataset)-1))))
  numTree <- controlPRF$numTree
  numProcessor <- controlPRF$numProcessor
  numFeature <- controlPRF$numFeature
  typeFeatures <- controlTree$typeFeatures
  paramPercent <- controlTree$paramPercent
  typeEntropy <- controlTree$typeEntropy
  maxDepthTree <- controlTree$maxDepthTree
  typeSplitting <- controlTree$typeSplitting
  nameFeatures <- controlTree$nameFeatures
  minNumData <- controlTree$minNumData
  nameClasses <- controlTree$nameClasses

  ## check colnames
  if (is.null(nameFeatures)){
    nameFeatures <- names(dataset)
  }
  else {
    names(dataset) <- nameFeatures
  }

  ## save names of input features
  if (any(typeTask == c("classification", "regression"))){
    inputFeatures <- nameFeatures[-length(nameFeatures)]
  }
  else {
    inputFeatures <- nameFeatures
  }

  ## set nameClasses!
  if ((typeTask == "classification") && (is.null(nameClasses))){
    nameClasses <- c(as.character(sort(unique(dataset[, ncol(dataset)]))))
  }

  ## build trees
  cl <- makeCluster(numProcessor)
  registerDoParallel(cl)
```

```
  modelTree <- list()

  modelTree <- foreach(icount(numTree), .combine=append, .export=ls(envir=globalenv())) %dopar%{

    ## generate bootstrapped data and features
    sampleData <- dataset[sample(nrow(dataset), size = nrow(dataset), replace = TRUE), ]

    ## building trees
    modelTree <- list(decTree(sampleData, typeTask = typeTask, controlTree = list(typeFeatures = typeFeatures, paramPercent = paramPercent,
typeEntropy = typeEntropy, maxDepthTree = maxDepthTree, typeSplitting = typeSplitting, minNumData = minNumData, sampleFeature = TRUE,
numFeature = numFeature, nameClasses = nameClasses)))
  }

  stopCluster(cl)


  attr(modelTree, "typeTask") <- typeTask
  attr(modelTree, "inputFeatures") <- inputFeatures
  attr(modelTree, "numTree") <- numTree
  return(ObjectFactory(modelTree, "randForest"))
}
```

Figure 4. Implementation of Parallel Random Forest in R

3. Results and Discussion

This chapter describes some experimental results that have been made predictions of iris species, wine quality and diagnosis of diabetes Pima Indians by using conventional random forest methods as well as random forest methods combined with parallel computing

The experimental results obtained after running the test case studies that have been determined in this study that is the case of Iris flower dataset, the dataset of the quality of wine and the dataset of diabetes Pima Indian women. The following experimental results were made to predict 3 cases of Iris flower, wine quality and diagnosis of diabetes of Pima Indian women by random forest method in parallel computing with modification of number of processor used, number of tree and number of features.

Table 1. Results of parallel random forest on the Iris dataset.

No	Numbers of Cores	Numbers of Tree	Numbers of Features	Cost (s)	Error (%)
1	2	20	2	1.368404	10.0
2	3	20	2	1.327446	10.0
3	4	20	2	1.440552	10.0
4	2	100	2	4.294907	3.3
5	3	100	2	3.351274	3.3
6	4	100	2	3.207866	3.3
7	2	20	4	1.894165	10.0
8	3	20	4	1.788924	10.0
9	4	20	4	1.703791	10.0
10	2	100	4	6.498522	6.6
11	3	100	4	4.964609	6.6
12	4	100	4	4.470230	6.6

Table 1 shows the time obtained when running a random forest method with parallel computing to predict Iris flower species. Based on the data obtained and displayed in the table can be seen that the more processors are used then the processing time becomes shorter. But that does not mean the faster the error generated smaller. The fastest time is 1.368404 seconds when using two processors with numTree 20 and numFeature 2 but the resulting error is 10.0%. The smallest error is 3.3% at numTree 100 and numFeature 2 using two, three and four processors. The resulting time is 4.294907 seconds,

3.351274 seconds and 3.207866 seconds. The longest time is 6.964609 seconds when using three processors with numTree 100 and num Feature and the resulting error is 6.6%.

Table 2. Results of parallel random forest on the wine dataset.

No	Numbers of Cores	Numbers of Tree	Numbers of Features	Cost (s)	Error (%)
1	2	20	2	3.339464	0.000
2	3	20	2	2.965031	0.000
3	4	20	2	2.570818	0.000
4	2	100	2	13.052	2.857
5	3	100	2	9.62039	2.857
6	4	100	2	7.812652	2.857
7	2	20	4	4.94744	2.857
8	3	20	4	3.883546	0.000
9	4	20	4	3.486539	0.000
10	2	100	4	21.5716	2.857
11	3	100	4	15.30717	0.000
12	4	100	4	12.97523	0.000

Table 2 shows the computational results obtained when running a random forest method with parallel computing to predict the quality of the wine with different parameter values. This affects the processing time spent. The fastest time is obtained when computing with four processors with numTree 20 and numFeature 2 parameters, the resulting error is 0.000%. Errors resulting from the overall wine dataset prediction are relatively small, the highest error is only 2.857%. Overall, the computing time gets shorter when using more processors.

Table 3. Results of parallel random forest on the Pima dataset.

No	Numbers of Cores	Numbers of Tree	Numbers of Features	Cost (s)	Error (%)
1	2	20	2	16.22235	36.364
2	3	20	2	11.97611	35.714
3	4	20	2	9.927734	29.078
4	2	100	2	74.96628	30.519
5	3	100	2	54.40325	29.870
6	4	100	2	45.03	29.221
7	2	20	4	28.772	30.519
8	3	20	4	20.83696	29.870
9	4	20	4	17.41449	30.519
10	2	100	4	147.9651	35.065
11	3	100	4	103.69848	34.416
12	4	100	4	77.15244	35.065

Table 3 shows the results obtained when running the random forest method with parallel computing to predict the quality of the wine with different parameter values. When numTree 100 and numFeature 4 the time gained becomes larger than when numTree 20 and numFeature 2. However, it can be concluded that when the value of the numProcessor parameter increases, the processing time becomes shorter. Errors resulting from the overall experiment are relatively high. The lowest error obtained

when computation dijalanka in four processors with numTree 20 and numFeature 2 is 29.078% with the shortest time also is 9.927734 seconds.

4. Conclusion

Based on the results of random forest implementation implementation with parallel computing in R, obtained some of the following conclusions: This research has produced a system that can predict some cases of Iris flower species, wine quality and diabetes diagnosis of Pima Indian women by applying random forest method with parallel computing. According to the experiments, it can be stated that computational time of random forest with parallel computing is much faster than conventional random forest and predicted results have no significant difference.

5. References

[1] Breiman L 2001 Random forests. *Machine learning* **45** 1 5-32
[2] Sulaiman S, Shamsuddin SM and Abraham A 2011 Intelligent web caching using adaptive regression trees, splines, random forests and tree net. *InData Mining and Optimization (DMO), 2011 3rd Conference on 2011 Jun 28* 108-114
[3] Dewi NK, Syafitri UD, Mulyadi SY 2011 Penerapan Metode Random Forest dalam Driver Analysis. *InForum Statistika dan Komputasi* **16** 1
[4] Muttaqin M J 2013 *Metode Ensemble pada CART Untuk Perbaikan Klasifikasi Kemiskinan di Kabupaten Jombang* (Surabaya: Institut Teknologi Sepuluh Nopember)
[5] Kumar V, Grama A, Gupta A, Karypis G 1994 *Introduction to parallel computing: design and analysis of algorithms*. (Redwood City: Benjamin/Cummings)
[6] Pachori V, Ansari G, Chaudhary N 2012 Improved performance of advance encryption standard using parallel computing. *International Journal of Engineering Research and Applications (IJERA)* **2** 1 9 67-71
[7] Haynes LS, Lau RL, Siewiorek DP, Mizell DW 1982 A survey of highly parallel computing. *Computer* **1** 1 9-24
[8] Matloff N 2011 The art of R programming: A tour of statistical software design. *No Starch Press*
[9] Dalgaard P 2008 *Introductory statistics with R* (Springer Science & Business Media)
[10] Analytics R, Weston S 2014 doParallel: Foreach parallel adaptor for the parallel package. *R package version***1** 8
[11] Riza LS, Asyari AH, Prabawa HW, Kusnendar J, Rahman EF 2018 Parallel Particle Swarm Optimization for Determining Pressure on Water Distribution Systems in R. *Advanced Science Letters* **24** 10 7501-7506
[12] Riza LS, Utama JA, Putra SM, Simatupang FM, Nugroho EP 2018 Parallel Exponential Smoothing Using the Bootstrap Method in R for Forecasting Asteroid's Orbital Elements. *Pertanika Journal of Science & Technology* **26** 1 441-462
[13] Riza LS, Nasrulloh IF, Junaeti E, Zain R, Nandiyanto AB 2016 gradDescentR: An R package implementing gradient descent and its variants for regression tasks. *InInformation Technology, Information Systems and Electrical Engineering (ICITISEE), International Conference* **23** 125-129