HPC PROJECT— Implementation of Random Forest in R on MIT Supercloud

December 17, 2019

Submitted by :
Group members :
Anushree Chopde 01865258
Laukik Upadhye 01833608
Akshay Reddy 01861852

1 Abstract:

High performance scientific computing and machine learning are coming togather in many upcoming applications, for example natural lanuage processing, various prediction and detection models. In our HPC project we are combining HPC with the machine learning prediction model called random forest model, by parallelizing it in R and deploying it on MIT Supercloud.

We have created two versions of our R code for random forest. One for serial execution and the other for parallel execution. Our main aim is to show how the speed of execution increases with increase in core.

We focused on the optimizing the speed of execution of the program by multiprocessing in R, and achieved the expected results.

2 Introduction and background:

Random Forest:

Random forest is developed by aggregating decision trees. In random forest, we aggregate the predictions made by the number of decision trees and ensemble it. This technique is called Ensemble learning.

Objective and Working:

To improve the decision tree technique we can create a number of decision tree classifiers, each on different random subset of training set. To make a prediction, we obtain predictions of different individual decision trees, and predict the class as the class that gets maximum votes.

Random forest can be used for classification in case of discrete or factor variables, and regression in case of continuous variables.

It avoids overfitting and can deal with large number of features.

Helps with feature selection based on importance.

Random forest has only two free parameters as follows :

1) number of trees - ntree, 500 by default
2) variables randomly sampled at each split - mtry, default is square root (p) for classification and p/3 for regression, where p is number of features.

Algorithm:

1) Draw ntree bootstrap samples.

2) For each bootstrap sample, grow un-pruned tree by choosing the best split based on a random sample of mtry predictors at each node.

3) Predict new data using majority votes for classification and average based on ntree trees.

Random Forest working:

We will proceed as follows to train the Random Forest:

Step 1) Import the data
Step 2) Data partition for training and testing
Step 3) Train the model
Step 4) Construct accuracy function
Step 5) Visualize the model
Step 6) Evaluate the model
Step 7) Visualize result for training and testing
Step 8) Visualize results

Our approach to the project:

We have chosen language as R in order to converge data science models with high performance scientific computing. R is a popular programming language used by data scientists for statistical computations and data analysis.

In our project we have considered an example of CTG medical data available online, and performed prediction analysis on it using random forest. Introduction to our data and work is explained in section 4.

We have created two versions of our R code for random forest. One for serial execution and the other for parallel execution. The goal of executing random forest parallely is to reduce the execution time of the model.

There is not much work done on parallel executions in R. While doing our research, we found only few references and research papers with parallelization in R. The motivation behind choosing this topic was to implement the learnt parallelization concepts in the class in C language in different language and provide an R example to MIT Supercloud as there is no example implemented in R on MIT Supercloud.

3 MIT Supercloud:

The MIT Supercloud is intended to support research collaboration between MIT Lincoln Laboratory and students, faculty and researchers at MIT and other academic institutions. The system is designed to support research projects that require significant compute, memory or big data resources.[2]

3.1 Steps:

1) Request an account on MIT Supercloud : https://supercloud.mit.edu/requesting-account.
2) Login with the username. To connect put the command on termminal: ssh username@txe1-login.mit.edu
3) Run program. (This is discussed more in section 3.2.)

3.2 Working with MIT Supercloud:

MIT Supercloud is a cloud platform that provides different platforms on which we can work. For our project, we requested for R platform.

There are three ways of deploying projects on MIT Supercloud :

* On terminal
* On jyupiter notebook on MIT Supercloud.
* Uploading our R script directly on MIT Supercloud.

We chose the third way of deploying that is uploading our R scripts for serial and parallel execution of random forest and dataset file directly on MIT Supercloud, as we were already finished the programming and testing on local machine.

To run the R programs on MIT Supercloud, we created a batch script submit.sh, which we run on the terminal using the command : LLsub submit.sh. Command to directly run the R script is : Rscript rf.R. Images of the visualizations after running the batch script are saved in pdf format.

Futher we have added our work to MIT Supercloud github page : https://github.com/AnushreeChopde/teaching-examples/tree/master/R

4 Experimental work:

Data set used : CTG Data.
Data set imported : CTG.csv.

About data :

* Measurement of fetal heart rate (FHR) and Uterine Contraction (UC) features of cardiogram.
* 2126 CTGs automatically processed and diagnostic features measured.
* CTG is classified by three labels as Normal (1), Suspect (2) and Pathologic (3).
* NSP is categorical variable with three categories as 1 for Normal, 2 for Suspect and 3 for Pathologic.
*The output of the prediction model will be the prediction whether the person is Normal, Suspect or Pathologic.

We started with programming of the random forest prediction model in R. For partitioning we have taken 70 percent for training and 30 percent for testing and set seed as 123.

In our project we have executed random forest model in serially as well as parallely. The modeules 4.1 and 4.2 shows the serial and parallel execution respectively.

4.1 Serial execution of random forest:

In serial computing a problem is divided into instructions. The instructions are executed sequentially one after the other on single proccessor, and only one instruction can be executed at a time.[1]

In serial execution of random forest, number of decision trees are generated sequentially are aggregated and the class with maximum votes of predictions from all decision trees is the predicted class.

The following image gives the working of random forest in general.

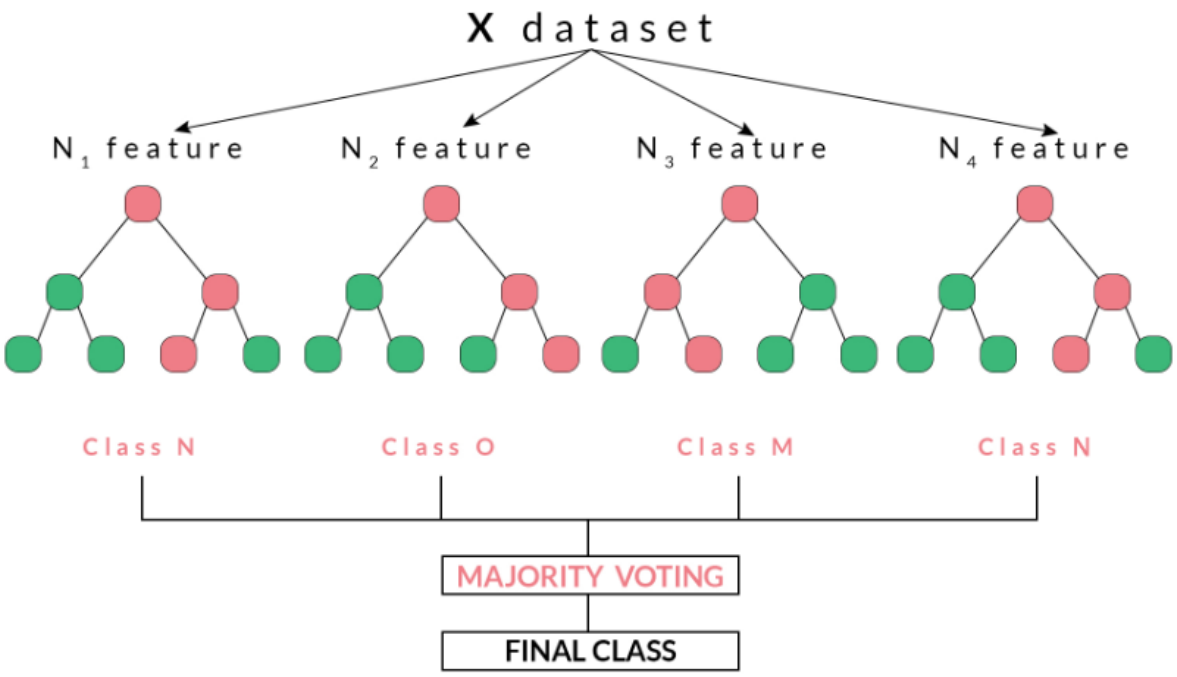

Figure 1: Serial execution of random forest.[3]

From the above figure, we understand the serial implementation of the program. Our CTG dataset is accepted by the program, number of decision trees are formed based on the features and each decision tree classifier predicts a class as Normal, Suspect or Pathalogic. After that the random forest model outputs the final class as the majority predicted classes by the decision trees.

Since the program runs serially, the decision trees are formed and the prediction of each decision tree is done serially, hence results in taking some time for execution. The randomForest() function is used to generate random forest model in R. Next, we have measured execution time by the Sys.time() function in R.

Time required for serial execution of random forest is 8.558022 seconds and the training accuracy was 99.87 percent and testing accuracy was 0.9334 percent.

4.2 Parallel execution of random forest:

In Parallel computing, a problem is broken into discrete parts that can be solved simultaneously. Each part is further broken down into a series of instructions. Instructions from each part execute concurrently on different processors, for which an overall control and coordination mechanism is employed.[1] There are different types of parallel computing like multi-threading and multi-processing.

For our parallel model, we are working with multi-processing, that is the work of the program, that is building a number of decision trees and aggregating them in order to predict the class, is divided into various cores, cores being inputted from the program. In other words, various decision trees are built simultaneously by different cores. This will be explained further in detail in the next section.

The parallel execution of random forest is demonstrated in the figure below.

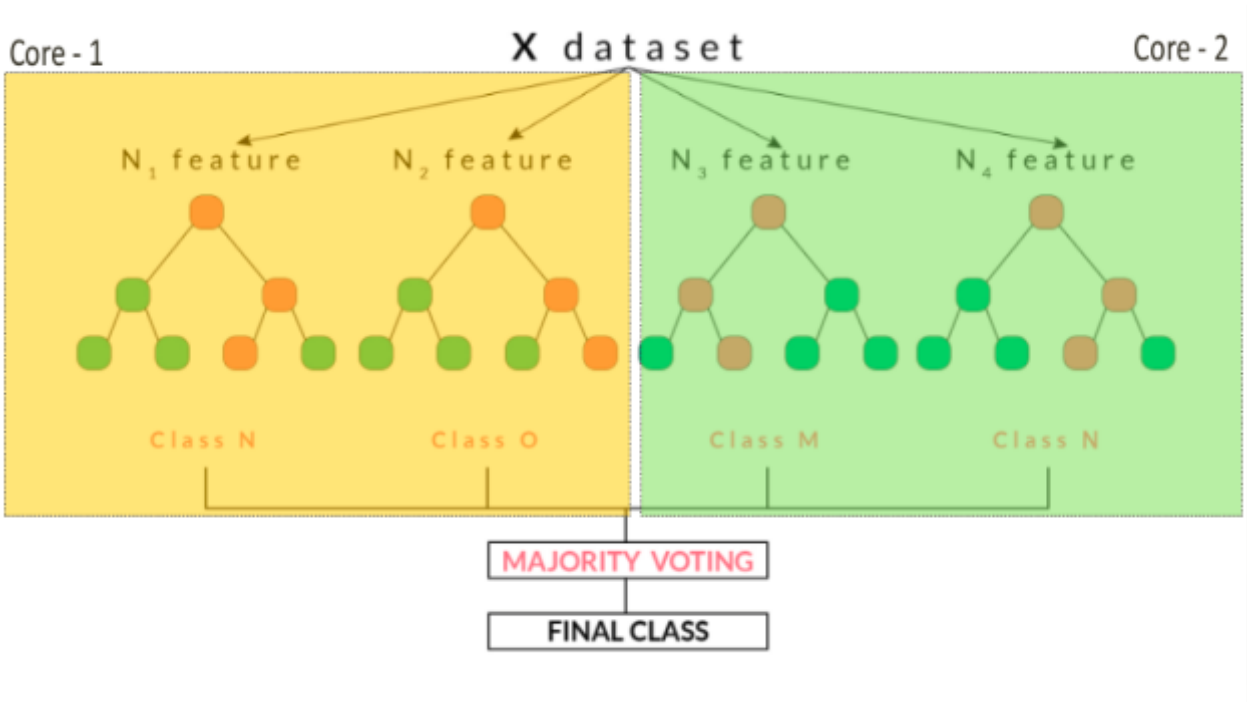

Figure 2: Parallel execution of random forest.[3]

From Fig2, we can understand the execution of random forest in parallel. The CTG dataset is accepted by the program, number of decision trees are generated and predictions from those decision trees are done simultaneously by number of cores, in this case 2 cores, that is the work is divided and the cores work on them simultaneously giving prediction from the decision trees in different cores at the same time. This reduces the execution time of the program.

Following are the specifications on what we have used for the parallel execution of random forest in R. Additional packages used are randomForest, foreach, caret and doParallel, which has parallelization built in their functions. Fig3 gives the purpose of each package used for parallelization.

Package	Purpose
Foreach	It is true horsepower of parallel processing in R. Uses *%dopar%* to parallelize tasks and returns it at a list of results vectors.
doParallel	Provides parallel backend for *%dopar%* function.
randomForest	Builds a number of decision trees. Uses *foreach* and the *combine* function to get parallelization.
Caret	Provides framework to find optimal model by trying multiple models with resampling.

Figure 3: Packages used for parallel execution of random forest in R.[4]

By using *registerDoParallel(cores = n)* and *getDoParWorkers()* we can set the number of cores and run multiple times to get system time as done in the serial version of the code.

Sys.time() gives the execution time of the program in R. We chose to run it multiple times and compared the execution time of serial and parallel execution. We observe that, parallel execution gives output in less time as compared to serial execution, as the work is divided into cores by processes.

Following is the code snippet for parallelization part of the program:

```
# Random Forest
library(randomForest)
library(e1071)
library(ggplot2)
library(doParallel)
library(foreach)
set.seed(222)

#loop to run for multiple cores. In our batch script submit.sh Parallel_output0.sh has the
    execution times for different number of cores from 2 to 16. We used this time to plot
    the scaling plot and we notice that with increase in number of cores, execution time is
    decreased.

for(i in 2:16)
{
    registerDoParallel(cores = i)
    getDoParWorkers()

    start_time <- Sys.time()

    rf <- foreach(ntree=rep(200, 5), .combine=randomForest::combine,
              .multicombine=TRUE, .packages='randomForest') %dopar% {
                randomForest(NSP~., data=train, ntree=ntree, mtry=8)
              }
    end_time <- Sys.time()

    time <- print(end_time - start_time)

}

#Optional part : To run individually for different number of cores, we can uncomment the
    below lines putting as many number of cores we need. In this example, we tried both with
     loop to print multiple values and individual runs with different number of cores. Same
    can be noticed from the commented part in the submit.sh batch script.

#registerDoParallel(cores = 2) #Parallel_output1.txt in submit.sh
#registerDoParallel(cores = 4) #Parallel_output2.txt in submit.sh
#registerDoParallel(cores = 8) #Parallel_output3.txt in submit.sh
#registerDoParallel(cores = 16) #Parallel_output4.txt in submit.sh
#registerDoParallel(cores = 32) #Parallel_output5.txt in submit.sh
#registerDoParallel(cores = 64) #Parallel_output6.txt in submit.sh

#getDoParWorkers()

#start_time <- Sys.time()

#rf <- foreach(ntree=rep(200, 5), .combine=randomForest::combine,
#              .multicombine=TRUE, .packages='randomForest') %dopar% {
#                randomForest(NSP~., data=train, ntree=ntree, mtry=8)
#              }
#end_time <- Sys.time()

#time <- print(end_time - start_time)

#getDoParWorkers()

#start_time <- Sys.time()

#rf <- foreach(ntree=rep(200, 5), .combine=randomForest::combine,
#              .multicombine=TRUE, .packages='randomForest') %dopar% {
#                randomForest(NSP~., data=train, ntree=ntree, mtry=8)
#              }
#end_time <- Sys.time()

#time <- print(end_time - start_time)
```

Program 1: Code snippet for parallel execution of random forest in R.

The above program snippet does the main parallelization of our model. We have installed and called the libraries - randomforest, e10701, ggplot2, doParallel, foreach. A seed is set to 222 to select features randomly.

To check the execution performance of our parallel program, initially, we gave different number of cores one by one to the program (commented part in optional part in above snippet) by *registerDoParallel(cores = number of cores)* and *getDoParallel()* and then running the *rf* functions in the code, we found that the execution time taken by parallel program is much less as compared to serial program.

In order to generate speedup and scaleup graphs and to get clear picture of the execution time with respect to cores at one place, we used for loop to fetch the execution time required for different number of cores from 2 to 16.

Time required for parallel execution of random forest is 2.598843 seconds to 0.8281538 secconds with increase in number of cores and the training accuracy was 99.87 percent and testing accuracy was 93.34 percent.

We have put the results for serial and parallel execution of the random forest model in section 4.3 and the scaleup and speedup graphs in section 4.4.

4.3 Results for serial and parallel execution of random forest in R:

```
1   1    2    3
2 1655  295  176
3 Time difference of 8.558022 secs
```

Program 2: Execution time results for serial execution on MIT Supercloud.

```
1   1    2    3
2 1655  295  176
3 Time difference of 2.598843 secs
4 Time difference of 1.862376 secs
5 Time difference of 1.616494 secs
6 Time difference of 0.8240888 secs
7 Time difference of 0.9192913 secs
8 Time difference of 0.8292949 secs
9 Time difference of 0.8659461 secs
10 Time difference of 0.8430467 secs
11 Time difference of 0.9184213 secs
12 Time difference of 0.8838823 secs
13 Time difference of 0.8281538 secs
14 Time difference of 0.8977926 secs
15 Time difference of 0.8997848 secs
16 Time difference of 0.8915453 secs
17 Time difference of 0.8688896 secs
```

Program 3: Execution time results for parallel execution on MIT Supercloud.

4.4 Scaleup and speedup for parallel execution of random forest in R:

When workload is increased with increasing number of cores is called strong scaling.
Following is the graph for scaleup for parallel execution of random forest in R.

Speedup is the ratio of time required by one core and time required by different number of cores i.e. $\frac{T_1}{T_p}$.

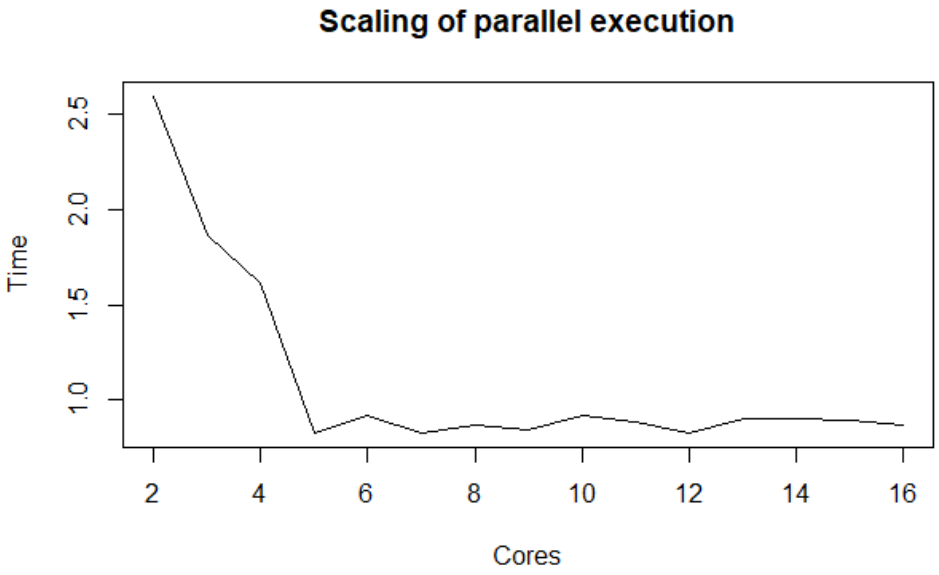

Figure 4: Scaleup for parallel execution of random forest in R.

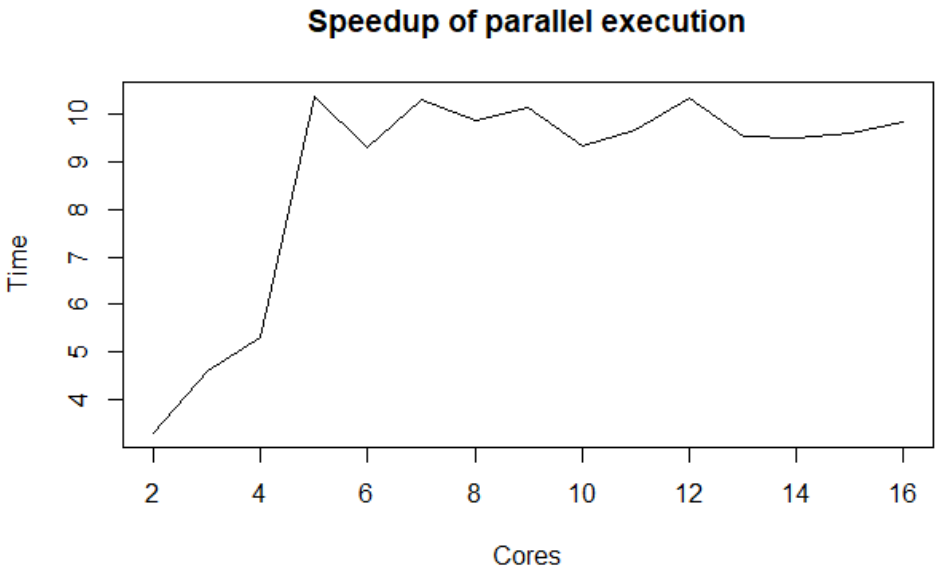

Figure 5: Speedup for parallel execution of random forest in R.

5 Conclusion:

From the results section 4.3 and scaleup and speedup section 4.4 of the report, it is clear that time required for serial execution of random forest is approximately 10 times of the time required for parallel execution of random forest in R. Also we observe that the training and testing accuracy are constant which

are very good, both above 92 percent, but the execution time reduces drastically for parallel execution of random forest in R, hence we achieve the proposed and expected results.

6 References:

[1] https://computing.llnl.gov/tutorials/parallel_comp/
[2] https://supercloud.mit.edu/\\
[3] https://www.google.com/search?q=Random+forest+images&client=ubuntu&hs=5ho&channel=fs&sxsrf=ACYBGNTtFjAxMc4_WCC01KjJ5rQhb5DRQw:1576642661309&source=lnms&tbm=isch&sa=X&ved=2ahUKEwjahI_Oq77mAhXBpFkKHYEoCn4Q_AUoAXoECA4QAw&biw=1299&bih=639\\
[4] MIT Supercoud parallel documentation on https://github.com/AnushreeChopde/teaching-examples/tree/master/R