Scalable and Distributed Computing

Assignment 1 - Parallel Programming in R

Rafaela Becerra Robalino, Marta Cortés Ocaña, Anna Subirós de Arriaga 15/02/2020

Contents

1 Introduction

The aim of the present work is the address of a serial problem to a parallelization process using R. The need for parallelization is derived from the time cost of computational complex analysis. In this case, the target is to do a clusterization of 37,554 computers based on their similarities on a ten-category description. There are several clustering methodologies that could be used, for simplicity the K-means tool from R will be used. After proposing a serial version of the program, a parallelization will be done in order to seek for a more efficient alternative. Different ways of parallelization will be tested in order to do a benchmarck comparing the serial version and the parallel programs.

The theory behind the algorithms and the computational development of both programs will be described in the next sections.

2 Theoretical Framework

Parallelization

Parallelization can be understood as the simultaneous execution of a large computation divided across multiple cores with the final idea that there exists a speed-up in the computational time. (Peng, 2019)

R provides two implementations of parallelism: forking and socket. On one hand, forking copies the existing version of R and therefore, everything (variables, libraries, functions,...) is inherited from the master process. The main advantage of using forking is its velocity as it runs faster. However, it is not compatible with all the operation systems due to the fact that it does not work on Windows. On the other hand, socket will be performed in all operating systems but launch on each core a new version of R without sharing objects or variables between them, which will only be passed explicitly from the master process. This is the reason why it is slower than forking.

Parallelization will be easy because of packages build in R like 'parallel'. 'Parallel' is a combination of the multicore and snow packages, the first one build to perform forking while the second is made to support sockets parallel computations. The functions that will be used from this package will be `detectCores` which identifies the number of the cores in the CPU, and `makeCluster` which will create the specified parallel socket and forking clusters. In order to share the objects when using the socket implementation, the function `clusterExport` is used to export functions and libraries, respectively. This package will be used with other packages like `foreach` and `doParallel`. The first one will allow iterating over elements without the use of a loop, similar to a lappy function. The use of this package will need `doParallel` which is considered as an adaptor for the `parallel` package.

Clusterization by K-means

Clustering analysis is unsupervised learning tool which aims to find homogeneous subgroups in the data. K-means clustering will partition the observations into a pre-specified number of clusters which is denoted by K.

The algorithm behind the K-means clustering is based on randomly assigning each observation to a cluster number, compute the cluster centroid or the vector of the features means of the observations in each cluster, and again assign each observation to the closest centroid. This will continue to be iterated until, there are no changes in the assignations of the observations to the cluster, so we will get the best solution of partition. The k-means function of R compute the distances between each centroid and the observations by calculating the squared Euclidean measure, which is given by:

$$W\left(C_k\right) = \frac{1}{|C_k|} \sum_{i,j' \in C_k} \sum_{j=1}^{p} \left(x_{ij} - x_{i'j}\right)^2 \qquad (1)$$

Where:

C is the number of observations in the Kth cluster, p the number of features, x each observation, then can be interpret as the within cluster variation given by the squared distance between the observations divided by the number of total observations of each cluster (James, 2015, p.387)

The final algorithm of the k-means can be defined as:

$$\underset{C_1,\ldots,C_K}{\text{minimize}} \left\{ \sum_{k=1}^{K} W\left(C_k\right) \right\} \tag{2}$$

The main problem of this methodology is to define the optimal number of K clusters in which the data will be partitioned. There are several methods to address the problem, two main methodologies are the elbow graph and the silhouette.

The elbow graph involves plotting the within-clusters sum of squares (wss) or the squared Euclidean distance which we are trying to minimize. The K optimal will be driven by the number from which there is a trend that the wss will start to decrease linearly.

Conversely, the average silhouette measures how far or close are the observations from its assigned cluster with each K. Then, the optimal K will be the one that obtains the higher value.

After obtaining the optimal K, the R function "kmeans" will allow getting directly the clusterization. Moreover, the results could be interpreted considering the two dimensions approach which involves the Principal Component Analysis or PCA. PCA will allow finding a low-dimensional representation of the observations that explains a representative fraction of the total variance.

3 Methodology

3.1 Preprocessing

The dataset contains 37,554 observations and eleven columns that correspond to computers and their specifications (id, price, speed, hd, ram, screen, cd, multi, premium, ads, and trend).

Given that there are some categorical variables: cd, multi, and premium, these must be substitute with values 0 for no and 1 for yes.

Additionally, there were no NA identify. Table 1 shows a descriptive analysis performed in order to see some insights about the dataset.

Variable	Mean	Standard Deviation	Variable	Mean	Standard Deviation
price	3563	1185	speed	52.01	21.16
hd	416.6	225.5	ram	8.287	5.631
screen	14.61	0.9051	cd	0.4646	0.4988
multi	0.1395	0.3465	premium	0.9022	0.2970
ads	221.3	74.83	trend	3.514	1.714

Table 1: Summary dataset

Moreover, since the units of the variables are different, standardization is required. Standardization is considered as the process of re-scaling variables by normalizing them resulting in each to have mean zero and standard deviation one, this process is imperative when performing *k-means* clusterization because if not, we could obtain clusters that are dominated by the variables with highest variance (Chen et al, p.277)

3.2 Serial Programming

To compute the serial program to find the best number of clusters and classify our data, we have created a function `kmeans_k` which applies a value (K) into the `k-means` R function, extracting afterwards the total within-cluster sum of squares.

Then, we have defined an array of a minimum and maximum value of K (2:15) and we have used the function `sapply` which employs each value of K into the `kmeans_k` function and give us a vector with the total within-cluster sum of squares for each K value.

As we have said before, the elbow method allows you to select the optimal value of K through visualization, exactly, where there is a starting trend of linearity decreasing of the wss (total within-cluster sum of squares). Therefore, we have plotted the wss vector which can be appreciated in the next figure.

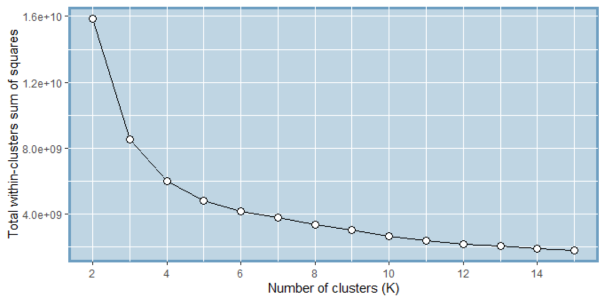

Figure 1: Elbow graph to determine the optimal value of clusters

It can be said that the starting inclination of the curve is produced at k=4, but to be sure we have also implemented the silhouette graph to discard that the best number of clusters was not 3 or 5.

In this way, we have created the function `silhouette_score` which also calculates the average silhouette for each K value of clusters returning a vector. By plotting this vector that has also been obtained using the `sapply` function, we have get the next figure from where we can confirm that the optimal number of clusters is 4.

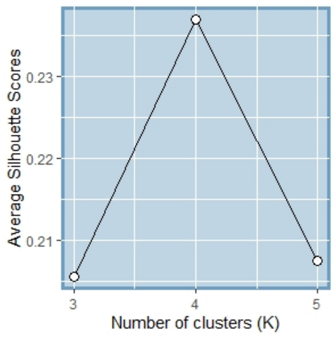

Figure 2: Silhouette graph to determine the optimal value of clusters

3.3 Parallel Programming

Regarding the parallel version of the program above, we have used the two approaches, Forking and Socked, to check which one is faster.

First of all, we have installed the packages `Paralell`, `doParallel` and `foreach` which are indispensable in the execution of this version.

After that, we have obtained the numbers of cores of our system which is 4. Therefore, we have got four simultaneous execution of our computation that has been contributing to decrease the time of the processing.

By applying the function `makeCluster` we have created a cluster of nodes for the parallel computation, using whether Forking or Socket types. Also, we have compared two types of functions: `parSapply` and `foreach` in order to see which one has the best performance.

For the socket approach we must compute `clusterExport` function first, which copy our data frame to each of the cluster members. However, this is not needed when using forking approach.

To apply `parLapply` function is simple and only has 3 arguments: the cluster of nodes, the values of K that we want to apply and the function where that K values are tested. Regarding `foreach` function when combining it with the `%dopar%` we can simulate a construction that will work as a `for` and a `lapply` function. This allows to get a parallelization with almost the same structure that a function could take in the serial program.

Then, we have used parallelization for two operations, the calculus of the optimal value of K (using the elbow plot) and the calculation of the mean of the clusters in order to obtain the highest.

3.4 Clusterization

After obtaining that the K optimal is four, the `k-means` function was used to cluster the data set. Since this case considers a multivariate analysis, the calculation of the principal components can be useful to visualize the results.

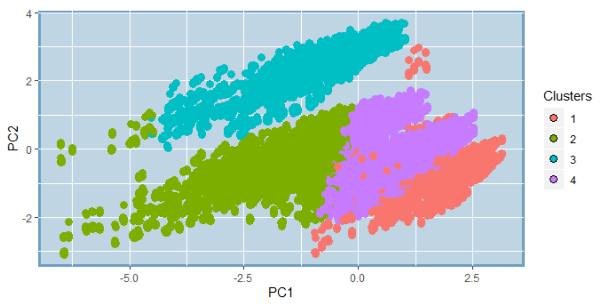

Figure 3: First two principal components visualization

As seen in the plot, the configuration of the two principal components denotes two mayor clusters with very marked difference to the right, and two other groups that have observations that cross the boundaries. These two PCAs represent almost 36% of the total variability of the dataset, which is not a high value but it can be use to have a more interpretable result.

Moreover, we have done a heatmap that conjugates the value of the sample mean for each variable and cluster. This shows that there are some variables that differ substantially among groups and, conversely, others like 'price' and 'trend' that remain almost the same. For example, we can see how Cluster 1 has the highest value for 'multi' and 'cd while presents the lower values for the rest of the variables or, how, Cluster 4, presents the highest absolute value for 'premium' which means no premium computers would belong to this cluster.

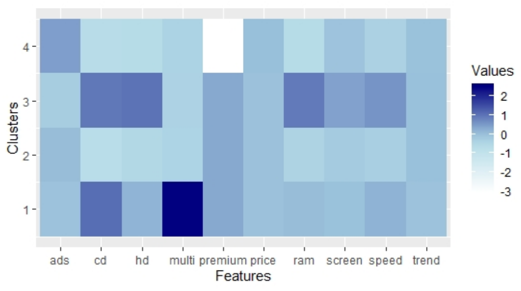

Figure 4: Heatmap of variables and kmeans centers

When we see the variable price, the average of the values is higher in cluster 2 but by a small difference. This in fact can be explained as there are other variables that become more relevant to separate the computers if we give them a similar weight from the beginning of the analysis by normalization.

Cluster 1	**Cluster 2**	**Cluster 3**	**Cluster 4**
3586.315	3561.168	3560.793	3560.674

Table 2: Average price per cluster

These results are the same for both programs because parallelization just changes the way the functions are run, but does not affect in the outputs. The main goal of parallelization will be the cutting of the time that takes running high demanding computations.

4 Results

In this section, the results of the two parallelization outputs will be shown in the following table where the difference between the operations, approaches and functions can be seen:

Operations	Serial	Parallel			
		FORK		PSOCK	
	Sapply	Sapply	Foreach	Sapply	Foreach
Wss	125.4895	65.09868	51.02128	66.94231	50.95334
Mean	0.03519344	/	2.472537	/	0.1505201

Table 3: Summary of time results (in seconds) regarding the operation, method, approaches and functions

5 Conclusions

The optimal K to use in the k-means is $k = 4$ as we have seen in both the elbow and silhouette graphs. From the heatmap, we can observe that the price is not an important variable in the clusterization. As a conclusion, we can see that the mean of the price does not present many differences depending on the clusters being the cluster number 1 the one that has a higher mean.

The use of parallelization can result in a less time consuming program, consequently, more complex analysis can be performed by taking the advantage of multiple cores to make multiple computations at the same time. That is the case of the 'Wss' computation, which is around 53% faster than the serial programming.

However, in the execution of the 'Mean' operation, the parallel version does not work better than the serial because only the time spent to find and create the number of clusters is higher to the time employed in the computation of `Sapply` function into the serial form.

Fortunately, the type of parallelization that performs better in terms of time is socket implementation, which is the one that is available for all operating systems. Although the difference is not to remarkable in the 'Wss' operation.

Finally, regarding the function implemented, `foreach` works quite better than the `Sapply`, this could be because the first one has the local variables available at all cores by default.

6 Bibliography

Chen, L.,Liu, C., Liu, Q. & Deng, K. (2009). Database Systems for Advanced Applications: DASFAA 2009 International. Springer.

Cran-r-project. Foreach. Retrieved from: https://cran.r-project.org/web/packages/foreach/index.html

DoParallel. Retrieved from: https://cran.r-project.org/web/packages/doParallel/index.html

James, G., Witten, D., Hastie, T., and Tibshirani, R. (2015). An Introduction to Statistical Learning with Applications in R. Springer. Heidelberg Dordrecht, London.

Peng, R. (2019). R Programming for Data Science. https://bookdown.org/rdpeng/rprogdatascience/#stay-in-touch

RDocumentation. Support for Parallel computation in R. Retrieved from: https://www.rdocumentation.org/packages/parallel/versions/3.6.2