BIRCH Clustering of Large Data Sets to Obtain Cluster Features or Clusters

Akshay Kurapaty, Ankit Anand, Rohit Kata, Hemanth Devavarapu

Function of our BIRCH Clustering package

Given a very large set of data points as input, BR_BIRCH package provides the user with a choice between obtaining cluster features or they will have an option to choose through fit function either K-Means or Hclust to obtain clusters as the output after obtaining the cluster features.

Rationale for publishing a package for Birch Clustering

BIRCH has the ability to find a good clustering solution with two scans of the data. BIRCH handles large data sets with a time complexity and space efficiency that is superior to other algorithms.

There is no other package in CRAN R performing BIRCH clustering. Since R is Open Source and packages form the backbone of R programming language, we are creating a package to contribute to the open source community.

BIRCH Clustering Steps

The BIRCH clustering algorithm consists of two main phases:

Phase 1: Build the CF Tree. Load the data into memory by building a *cluster-feature tree.*

Phase 2: Global Clustering. Apply an existing clustering algorithm on the leaves of the CF tree.

In this package, we are focusing on creating a CF tree. Once we obtain the CF tree, we are utilizing Hclust or K-Means based on user preference for which the Cluster Features obtained in phase 1 will serve as input and not the original data.

Implementation Algorithm and Logic of the Code

We utilize a 2 dimensional list. Initially the list is empty. Each element in the list will function as the node and has the capability to store the address of parent node, child node and CF values of the data. The CF values that are stored are:

- *Count*: How many data values in the cluster.
- *Linear Sum*: Sum the individual coordinates. This is a measure of the location of the cluster.
- *Squared Sum*: Sum the squared coordinates. This is a measure of the spread of the Cluster.

The following functions are utilized to implement the cluster:

1. MakeaCFTree = function (x,pagesize,branchingfactor,threshold)

MakeaCFTree function takes data points x (point in n dimensional space) , branchingfactor (maximum number of child nodes allowed) and threshold (Maximum width of the cluster) as input. The first point becomes the root node and as further points are provided as input, the tree grows by utilizing the calculatenearestnode() function & Compute_radius() function. If the new point is within the threshold of an existing cluster, then it will be added to that cluster or else a new cluster is created. For new cluster creation, the branching factor condition is checked and if the number of child nodes are less than branching factor then the new cluster is created to the same parent node. If the number of child nodes are more, then we use the combination of splitNode() function and rearrange() function to split the nodes at required levels to accommodate the new cluster. Createnewnodetop() function is used in the scenario where we need to create a new level for splitting of the CF's. Once the new cluster is formed, recalculateCF() function will recalculate the CF's at all levels where updating is required.

2. calculatenearestnode = function (parentnode,rowvector,depth)

Parentnode, rowvector and depth are required as inputs

3. Compute_radius = function (LS,SS,N)

For the nearest node calculation, it takes the calculated LS, SS and N of the CF nearest node along with the new point. With the updated LS, SS and N, the radius is calculated and is returned by the function which is used to check if the radius is within the threshold or not.

4. splitNode = function(depth, index)

This function is called if the count of the number of clusters retuned by the leaf is equal to the branching factor. It takes the depth and index of the nearest node calculated and

5. rearrange = function(depth, index)

The re-arrange function would take the depth and index of the node at which split has to take place and its child nodes are arranged such they the closer cluster stay in the same branch after splitting. This will ensure that the new points are going the node.

6. createnewnodetop = function(depth,index)

This is created in scenario when a new node has to be created when the branching factor is reached in multiple levels

7. recalculateCF = function(depth,index)

With the inclusion of a new point, the CF of the all the associated parent nodes CF's is updated using this function.

PHASE 1: BUILDING THE CF TREE

1. For each given record, BIRCH compares the location of that record with the location of each CF in the root node, using either the linear sum or the mean of the CF. BIRCH passes the incoming record to the root node CF closest to the incoming record.

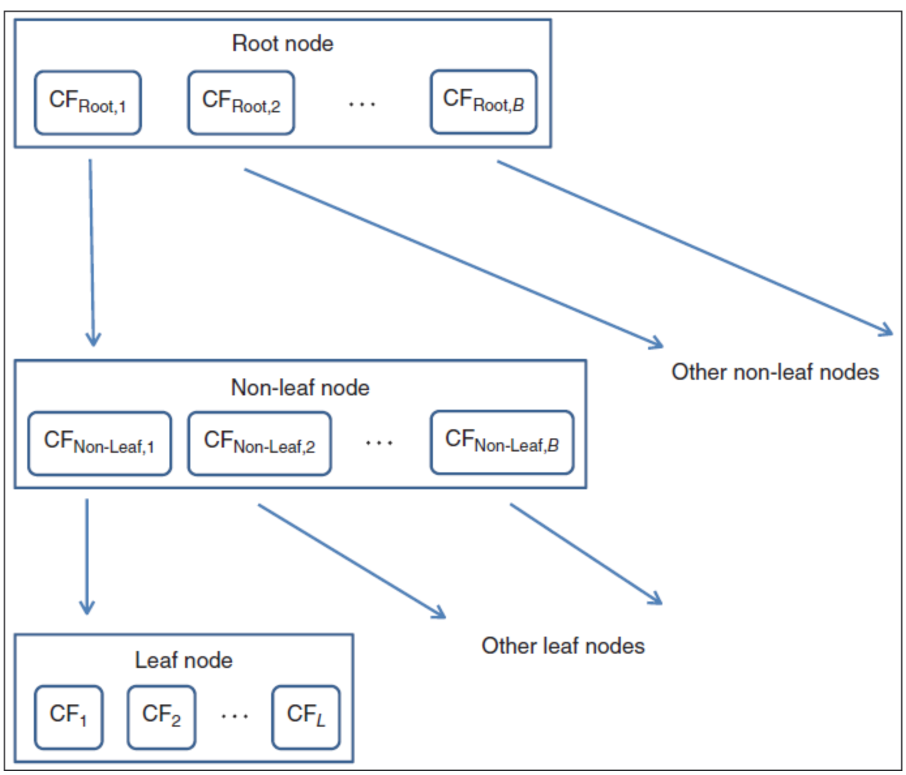
2.The record then descends down to the non-leaf child nodes of the root node CF selected in step 1. BIRCH compares the location of the record with the location of each non-leaf CF. BIRCH passes the incoming record to the non-leaf node CF closest to the incoming record.

3. The record then descends down to the leaf child nodes of the non-leaf node CF selected in step 2. BIRCH compares the location of the record with the location of each leaf. BIRCH tentatively passes the incoming record to the leaf closest to the incoming record.

4. Perform one of (a) or (b):

a. If the radius (defined below) of the chosen leaf including the new record does not exceed the Threshold *T*, then the incoming record is assigned to that leaf. The leaf and all of its parent CFs are updated to account for the new data point.

b. If the radius of the chosen leaf including the newrecord does exceed the Threshold *T*, then a new leaf is formed, consisting of the incoming record only. The parent CFs are updated to account for the new data point.



General structure of a CF tree with branching factor B, and L leafs in each leaf node

PHASE 2: CLUSTERING THE SUB-CLUSTERS

Once the CF tree is built, any existing clustering algorithm may be applied to the sub-clusters (the CF leaf nodes), to combine these sub-clusters into clusters.

IMPLEMENTATION EXAMPLE OF BIRCH CLUSTERING, PHASE 1:

BUILDING THE CF TREE

Let us examine in detail the workings of the BIRCH clustering algorithm as applied to the following one-dimensional toy data set 4

$x1 = 0.5$ $x2 = 0.25$ $x3 = 0$ $x4 = 0.65$ $x5 = 1$ $x6 = 1.4$ $x7 = 1.1$

Let us define our CF tree parameters as follows:

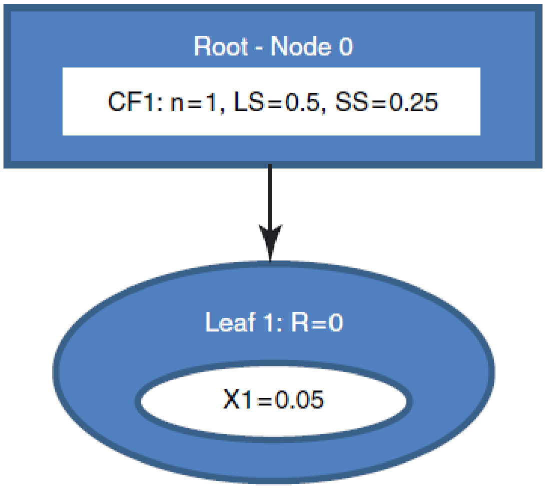
· Threshold T=0.15; no leaf may exceed 0.15 in radius.

· Number of entries in a leaf node L=2.

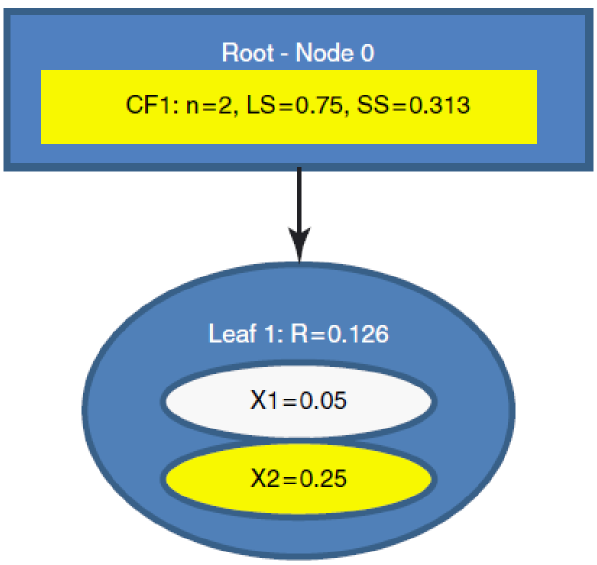
· Branching factor B=2; maximum number of child nodes for each non-leaf node.

The first data value $x1 = 0.5$ *is entered*. The root node is initialized with the CF values of the first data value. A new leaf *Leaf*1 is created, and BIRCH assigns the first record $x1$ to *Leaf*1. Because it contains only one record, the radius of *Leaf*1 is zero, and thus less than T=0.15. *The second data value* $x2 = 0.25$ *is entered*. BIRCH tentatively passes $x2$ =0.25 to *Leaf*1. The radius of *Leaf*1 is now$R = 0.126 < T = 0.15$, so $x2$ is assigned to *Leaf*1. The summary statistics for CF1 are then updated. *The third data value* $x3 = 0$ *is entered*. BIRCH tentatively passes $x3 = 0$ to *Leaf*1. However, the radius of *Leaf*1 now increases to $R = 0.205 > T = 0.15$. Threshold value $T = 0.15$ is exceeded, so $x3$ is *not* assigned to *Leaf*1. Instead, a newleaf is initialized, called *Leaf*2, containing $x3$ only.

The fourth data value $x4 = 0.65$ *is entered*. BIRCH compares $x4$ to the locations of CF1 and CF2. The location is measured by $x = \mathrm{LS}/n$. We have $x\mathrm{CF1} = 0.75/2 = 0.375$ and $x\mathrm{CF2} = 0/1 = 0$. The data point $x4 = 0.65$ is thus closer to CF1 than to CF2.



CF tree after the first data value is entered



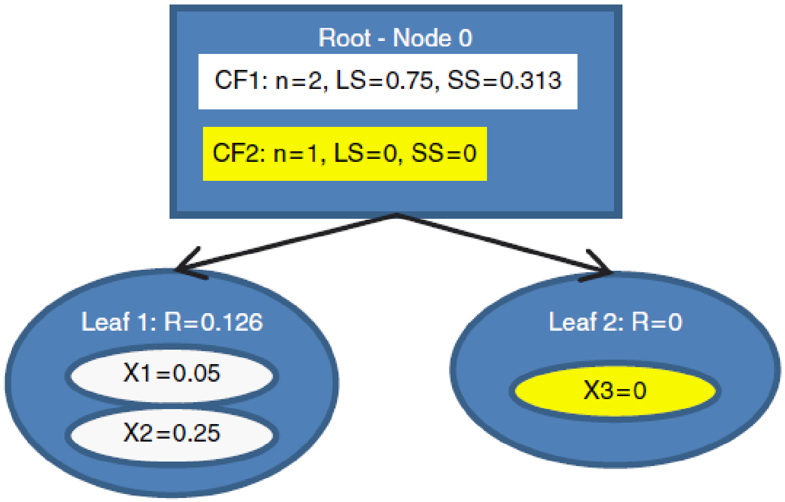
Second data value entered. Summary statistics are updated.

BIRCH tentatively passes $x4$ to CF1. The radius of CF1 now increases to $R = 0.166 > T = 0.15$. The Threshold value $T = 0.15$ is exceeded, so $x4$ is not assigned to CF1. Instead, we would like to initialize a new leaf. However, L=2 means that we cannot have three leafs in a leaf node. We must therefore split the root node into (i) *Node*1, which has as its children *Leaf*1 and *Leaf*2, and (ii) *Node*2, whose only leaf *Leaf*3 contains only $x4$.

Note that the summary statistics for the parent CFs equal the sum of their children CFs.

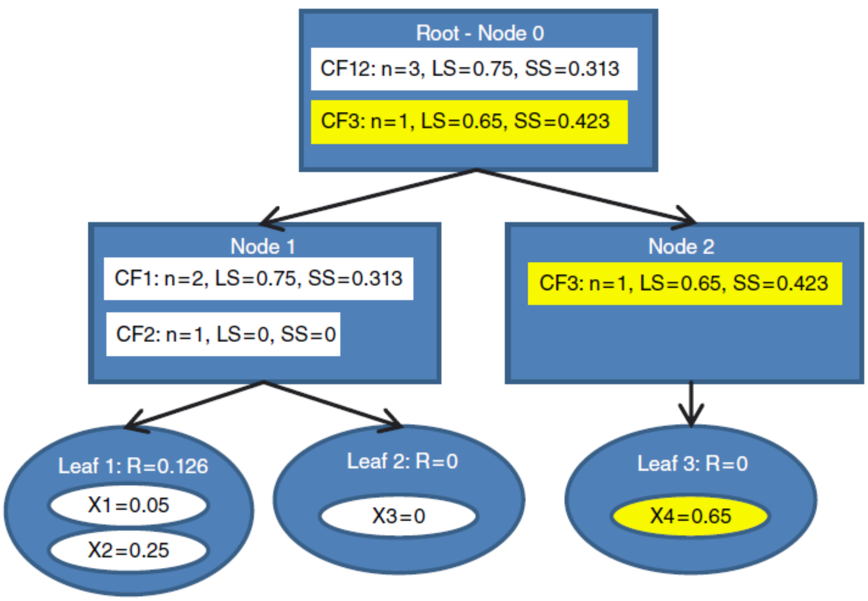
The fifth data value $x5 = 1$ *is entered*. BIRCH compares $x5 = 1$ with the location

of CF12 and CF3. We have xCF12 = 0.75/3 = 0.25 and xCF4 = 0.65/1 = 0.65.

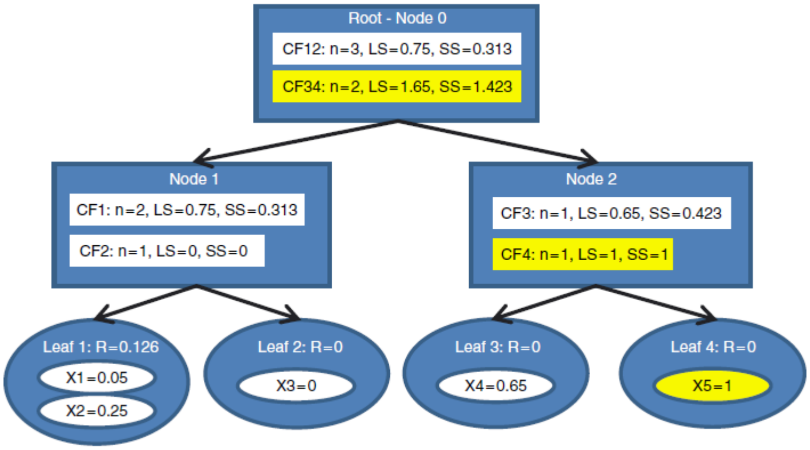


Third data value entered. A new leaf is initialized.

The data point $x5 = 1$ is thus closer to CF3 than to CF12. BIRCH passes $x5$ to CF3. The radius of CF3 now increases to $R = 0.175 > T = 0.15$, so $x5$ cannot be assigned to CF3. Instead, a new leaf in leaf node *Leaf*4 is initialized, with CF, CF4, containing $x5$ only.



Fourth data value entered. The leaf limit L=2 is surpassed, necessitating the creation of new nodes.

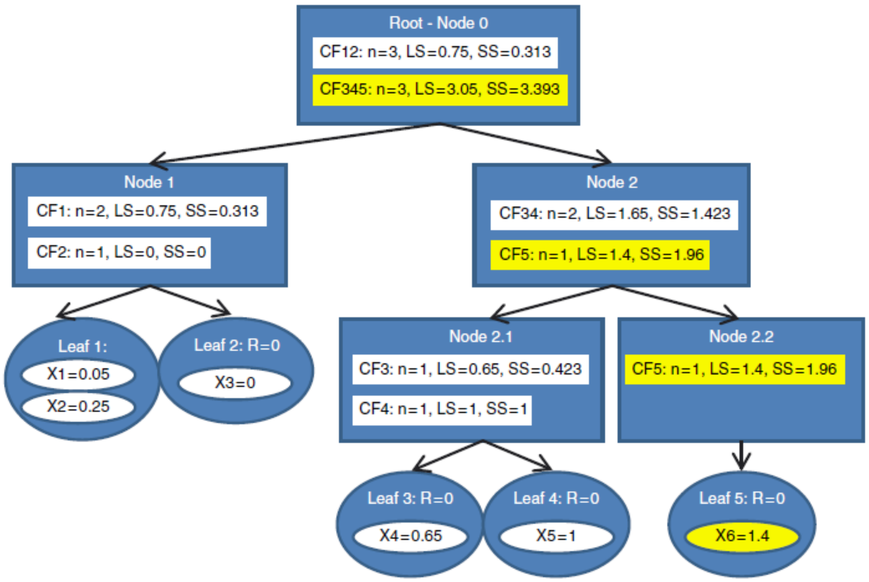


Fifth data value is entered. Another leaf is initialized.

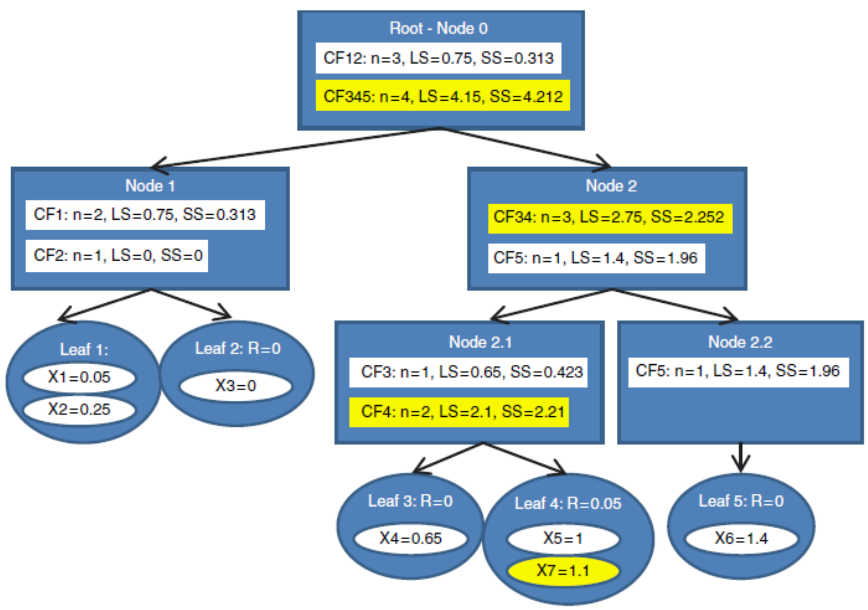
The sixth data value $x6 = 1.4$ *is entered*. At the root node, BIRCH compares $x6 = 1.4$ with the location of CF12 and CF34. We have xCF12 = 0.75/3 = 0.25 and xCF34 = 1.65/2 = 0.825. The data point $x6 = 1.4$ is thus closer to CF34, and BIRCH passes $x6$ to CF34. The record descends to *Node* 2, and BIRCH compares $x6 = 1.4$ with the location of CF3 and CF4. We have xCF3 = 0.65 and xCF4 = 1. The data point $x6 = 1.4$ is thus closer to CF4 than to CF3. BIRCH tentatively passes $x6$ to CF4. The radius of CF4 now increases to $R = 0.2 > T = 0.15$. The Threshold value $T = 0.15$ is exceeded, so $x6$ is not assigned to CF4. But the branching factor B=2 means that we may have at most two leaf nodes branching off of any non-leaf node. Therefore, we will need a new set of non-leaf nodes, *Node*2.1 and *Node*2.2, branching off from *Node*2. *Node*2.1 contains CF3 and CF4, while *Node*2.2 contains the desired new CF5 and the new leaf node *Leaf 5* as its only child, containing only the information from $x6$. *Finally*, *the last data value* $x7 = 1.1$ *is entered*. In the root node, BIRCH compares $x7 = 1.1$ with the location of CF12 and CF345. We have xCF12 = 0.25 and xCF345 = 1.02, so that $x7 = 1.1$ is closer to CF345, and BIRCH passes $x7$ to CF345.

The record then descends down to *Node* 2. The comparison at this node has $x7 = 1.1$ closer to CF34 than to CF5. The record then descends down to *Node* 2.1. Here, $x7 = 1.1$ closer to CF4 than to CF3. BIRCH tentatively passes $x7$ to CF4, and to *Leaf* 4. The radius of *Leaf* 4 becomes $R = 0.05$, which does not exceed the radius threshold value of $T = 0.15$. Therefore, BIRCH assigns $x7$ to *Leaf* 4. The numerical

summaries in all of its parents are updated and we obtain the final form of the CF tree.



Sixth data value entered. A new leaf node is needed, as are a new non-leaf node and a root node.



Seventh data value entered. Final form of CF tree.

Example Code to Check the Output

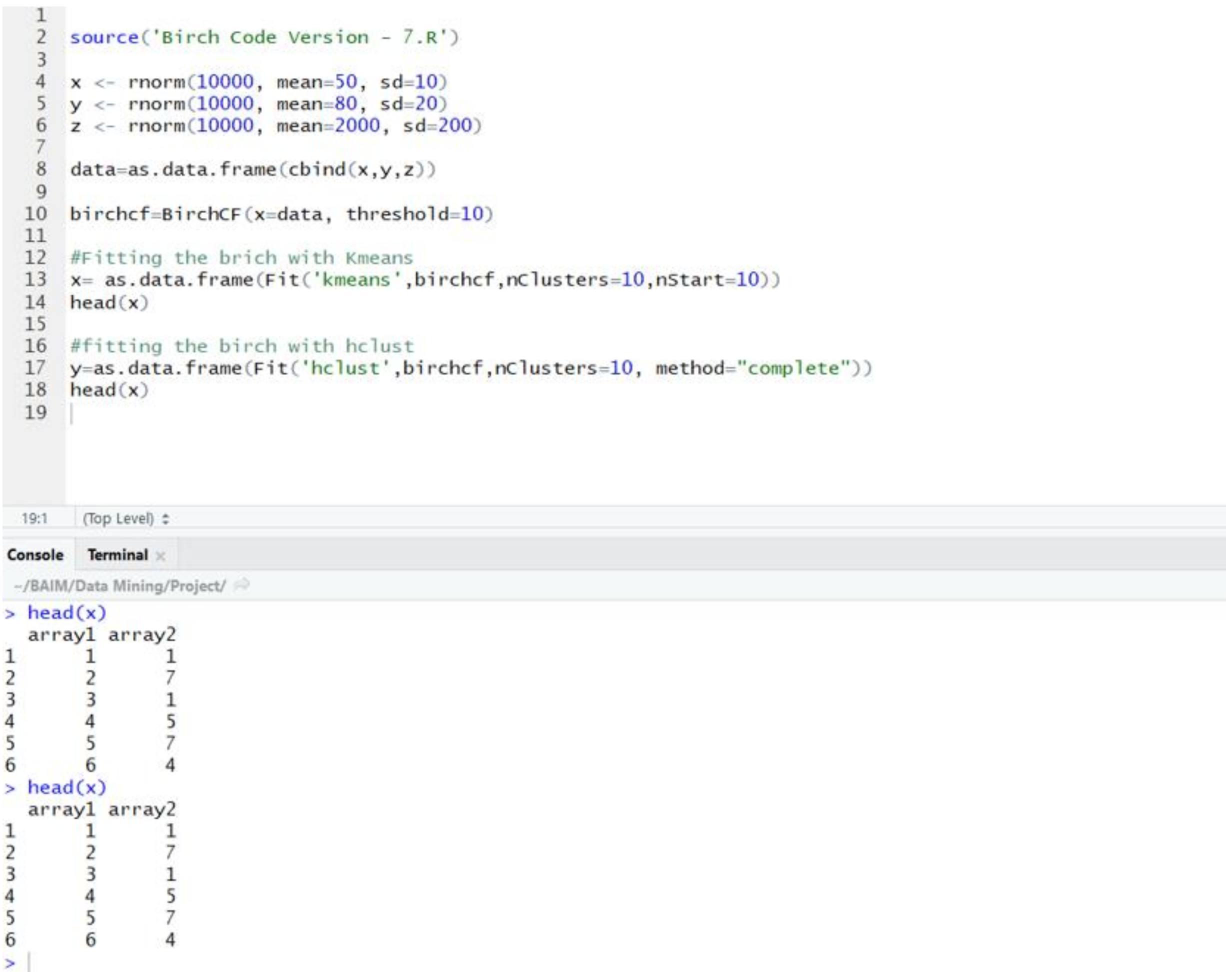
Import the source file before executing the below code

```
x <- rnorm(10000, mean=50, sd=10)
y <- rnorm(10000, mean=80, sd=20)
z <- rnorm(10000, mean=2000, sd=200)

data=as.data.frame(cbind(x,y,z))
```

birchcf=Birchcf(x=data, threshold=10)

#Fitting the brich with Kmeans
fit('kmeans',birchcf,nClusters=3,nStart=10)
#fitting the birch with hclust
fit('hclust',birchcf,nClusters=3, method="Complete")



```r
source('Birch Code Version - 7.R')

x <- rnorm(10000, mean=50, sd=10)
y <- rnorm(10000, mean=80, sd=20)
z <- rnorm(10000, mean=2000, sd=200)

data=as.data.frame(cbind(x,y,z))

birchcf=BirchCF(x=data, threshold=10)

#Fitting the brich with Kmeans
x= as.data.frame(Fit('kmeans',birchcf,nClusters=10,nStart=10))
head(x)

#fitting the birch with hclust
y=as.data.frame(Fit('hclust',birchcf,nClusters=10, method="complete"))
head(x)
```



```
> head(x)
  array1 array2
1      1      1
2      2      7
3      3      1
4      4      5
5      5      7
6      6      4
> head(x)
  array1 array2
1      1      1
2      2      7
3      3      1
4      4      5
5      5      7
6      6      4
>
```