**Project: Unsupervised Framework for Indexing of Big Data**

**Date: Jan 26, 2015**

**People: Ramakrishna Sakhamuri**

**Specific Aims:**

1. *Cleanse and transform the input data and place it in the INPUT directory.*
2. *Store the data using Hadoop file system (HDFS).* 
   1. Implement HDFS on a local machine.
   2. Store the data in Hadoop and then pass the data files to Birch algorithm along with their respective Hadoop addresses.
3. *Birch algorithm.*
   1. Design and implement CF tree and Birch algorithm in python.
   2. Read the files and their respective Hadoop addresses.
   3. Create an instance of Birch algorithm and redirect the input data to the instance.
      1. Input the data to Birch and build the CF tree.
   4. Once the data scanned by Birch archive the data.
4. *Test the Framework with Biological data and send the results for peer review.*

**Specific Aims worked during this Reporting Period:**

1. *Hadoop file system (HDFS).*
   1. We did setup two Hadoop instances on a local machine.
2. *Birch algorithm.*
   1. Performed clustering on a sample set of data with the help of clustering software. Used these observations in implementing Birch algorithm.
   2. Completed the implementation of Birch algorithm.
3. Implemented a client process which takes the input and load it into Hadoop and then passes it to Birch instance.

**Key Accomplishments:**

* Installed two Hadoop instances on local machine. One of it acts as a slave machine and one as a master.
* Processed data into Hadoop with 2 replication factor.
* Designed Birch algorithm and executed with a sample data set. And obtained the output of CF tree for the given data set.
* We are successful in implementing indexing of Big Data using CF tree.

**Position on Project GANTT-Chart:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Tasks | Week7 (02/01/16 – 02/05/16) | Week8  (02/08/16 – 02/12/16) | Week9  (02/15/16 – 02/19/16) | Week10  (02/22/16 – 02/26/16) |
|
| System Testing | Completed | Completed |  |  |
| System Demo |  |  | Completed |  |
| Project Presentation |  |  |  | Completed |

***Appendix 1:***

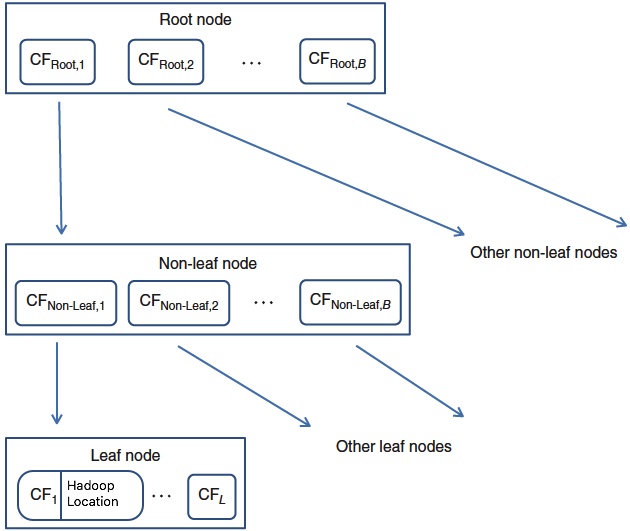
**Specific Aims:**

1. *Birch algorithm:*
2. A set of data points are the input to Birch algorithm. The first phase of Birch algorithm is building a CF tree. Every set of N dimensional data points has a Clustering Feature defined, i.e., a triple CF= (N, LS, SS). LS is the linear sum and SS is the square sum of data points.
3. A CF tree is a height balanced tree. It has two parameters branching factor B, “B determines the maximum children allowed for a non-leaf node” (Larose, 2015).
4. Threshold T, “T is an upper limit to the radius of a cluster in a leaf node” (Larose, 2015).
5. As and how in a B tree CF tree also has leaf nodes and non-leaf nodes.
6. A non Leaf node has at most B entries and each of form [CFi, childi]. CFi is clustering feature of sub cluster, childi is a pointer representing its sub cluster (Larose, 2015).
7. A leaf node has a limit of L entries of form [CFi,] and a Leaf node also has two pointers prev and next which connects all the leaf nodes together (Larose, 2015) and along with these we also store the memory address and node in which it is stored in a Hadoop cluster. Hence all the clustering features are organized and works as an index to the data.
8. And Birch also needs an input of ‘initial\_diameter’ which can be decide by clustering a sample set of data using clustering algorithms available on Mahout (preferably K-means).

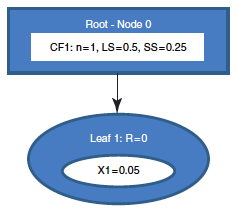
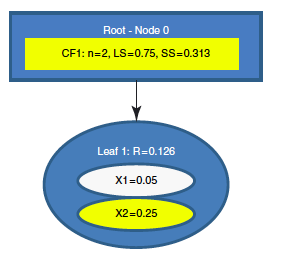
* We have to convert raw sample files into sequence files and then into vectors which is achievable by a sequence of Mahout commands and these are the input files for Mahout K-means clustering.

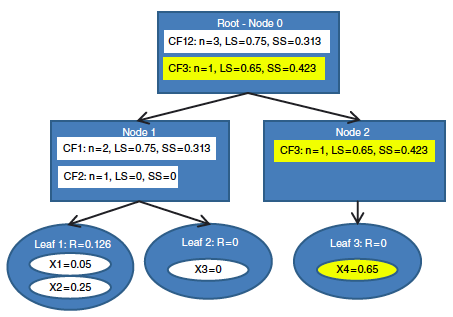
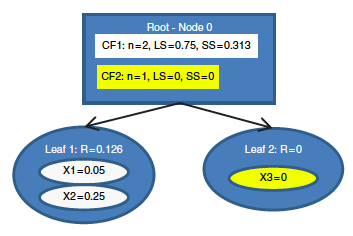
*CF Tree building process* (Larose, 2015)*:*

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 a’. 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 b’. 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 (i) or (ii):
   1. If the radius 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.
   2. If the radius of the chosen leaf including the new record 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.



*CF tree implementation with sample data:* (Larose, 2015)



*Diameter calculation in BIRCH process:*

R =

1. HDFS:
2. HDFS is Hadoop distributed file system.
3. HDFS is a file system designed for storing very large files with streaming data access patterns, running on clusters of commodity hardware.
4. In this context Very large files means files of hundreds of megabytes, gigabytes or terabytes. There are Hadoop clusters that are running today with petabytes of storage.
5. HDFS is built around the idea that the most efficient data processing pattern is a write-once, read-many-times pattern. A dataset is typically generated or copied from source, then various analyses are performed on that dataset over time. Each analysis will involve a large proportion, if not all, of the dataset, so the time to read the whole dataset is more important than the latency in reading the first record.[ (White, 2012)]
6. It is designed to run on Commodity hardware which is a commonly available hardware but not needed to run on expensive, highly reliable hardware.
7. Hadoop stores data in blocks. Usually in any file system, data is stored in block size of 512 bytes but in Hadoop each block is of size 64 MB.
8. The main components of HDFS are Namenode and Datanode.
9. Namenode:
   1. It manages the file system namespace. It maintains the file system tree and the metadata or all the files and directories in the tree.
   2. The namenode also knows the datanodes on which all the blocks for a given file are located, however, it does not store block locations persistently, since this information is reconstructed from datanodes when the system starts.
10. Datanodes:
    1. Are the workhorses of the filesystem.
    2. They store and retrieve blocks when they are told to (by clients or the namenode), and they report back to the namenode periodically with lists of blocks that they are storing.

**Specific Aims worked during this Reporting Period:**

1. *Birch Algorithm Execution:*
2. Birch algorithm is designed in python class.
3. In this algorithm we used cftree library which comes with instances classes like cftree, cfnode, leaf-node and non-leaf node.
4. Initially the dataset is transformed into a list of records where each record is again a list of attributes (columns).
5. And the final dataset looks like a list and also it doesn’t contain any text data as the algorithm deals with the linear and square sums, we restricted the text data and allowed only numerical data to the algorithm.
6. A Birch instance is created and passed all mandatory arguments like the data file and Hadoop address of respective file and few other variables like Branching factor, Threshold limit (initial diameter) and maximum node entries to it.
7. Then the Birch algorithm processed the input data and created a CF tree for the input data and returns the leaf nodes and all its components like the parent node address for each leaf, number of entries in each leaf node and list of all the entries in it.
8. Hadoop Implementation:
   1. Hadoop implementation on local machine is accomplished using following steps.
      1. Hadoop is installed on an Ubuntu server.
      2. Installed two virtual machine on windows.
      3. Installed Ubuntu 14.04 on the Virtual machines.
      4. Downloaded and installed Hadoop packages on the Ubuntu server.
      5. Changed the configuration files to configure
         * Core-site.xml: to set default schema and authority.
         * Hdfs-site.xml: to set the default replication to 2 rather than the default value of replication 3.
      6. Configure one server as master and the other as a slave.
      7. Master machine holds the namenode and one datanode. Slave machine holds only datanode.
      8. Hence the entire system is configured to serve multiple datanodes i.e., Hadoop cluster.

**Key Accomplishments:**

1. As we stated above, we have used few available classes in Birch implementation.

*Linear\_sum & Square\_sum:*

* These are the two methods designed in such a way that it takes an input record and calculates either square or linear sum accordingly and returns it to the main program.

*CFtree:*

* This class contains few variables and some important methods which does the insertion of data into tree. When we say insertion of data, the first record gets inserted right under the root where as next record onwards it has to calculate the linear and square sums to decide where that particular record fits into.
* And it also does a split operation when an incoming record does fits under existing non-leaf node but the non-leaf node’s Branching factor. These are the few important operations designed under this class.

*CFentry*:

* This class is designed with a set of methods which calculates and returns parameters like radius of cluster, diameter and inter and intra cluster distances.

*Birch*:

* + We designed Birch class which comprises of few class variables and methods.
  + It is built by importing cftree class. And birch utilizes predefined methods to process the cftree.
  + Whenever we create an instance of designed Birch class and passes the input dataset and few other variables like initial\_diameter(T), Branching\_Factor(B) and Max\_node\_entries(L) it processes the tree.
  + Once it creates the CFtree it returns the Leaf nodes and its entries along with parent info of each leaf node and Hadoop address of the data files the leaf node data beongs to.

Sample Output:

Leaf-node 0x7f2a21de5908, parent Non-leaf node 0x7f2a21de54a8, parent Non-leaf node 0x7f2a21dc7ba8, parent None, feature CF (65, [354.70000000000005, 1952.907699999999, -5954.093700000001, 413.0, 213.0, 118.0, 33275.0, -589941.0, 87451.0], 6069805515.32) [0x7f2a21de5828], successors: 1, feature CF (21, [201.09999999999997, 634.1873000000002, -1913.7572999999998, 98.0, 47.0, 24.0, 10087.0, -209979.0, 20535.0], 2131224470.28) [0x7f2a21de58d0], successors: 4, feature CF (4, [79.19999999999999, 122.2854, -367.8428, 21.0, 12.0, 4.0, 1486.0, -39996.0, 3642.0], 403982913.69) [0x7f2a21de59e8], entries: 4

CF (1, [6.1, 30.3214, -91.2517, 7.0, 3.0, 0.0, 241.0, -9999.0, 639.0], 100455744.47) [0x7f2a21de5710] Hadoop\_adress ['/user/root/in/2016-02-22/test3.txt']

CF (1, [33.5, 30.8213, -92.6697, 4.0, 3.0, 2.0, 411.0, -9999.0, 1122.0], 101418494.88) [0x7f2a21de5400] Hadoop\_adress ['/user/root/in/2016-02-22/test5.txt']

CF (1, [33.5, 30.8213, -92.6697, 3.0, 3.0, 1.0, 389.0, -9999.0, 869.0], 100897161.88) [0x7f2a21ddaf28] Hadoop\_adress ['/user/root/in/2016-02-22/test3.txt']

CF (1, [6.1, 30.3214, -91.2517, 7.0, 3.0, 1.0, 445.0, -9999.0, 1012.0], 101211512.47) [0x7f2a21de53c8] Hadoop\_adress ['/user/root/in/2016-02-22/test3.txt']

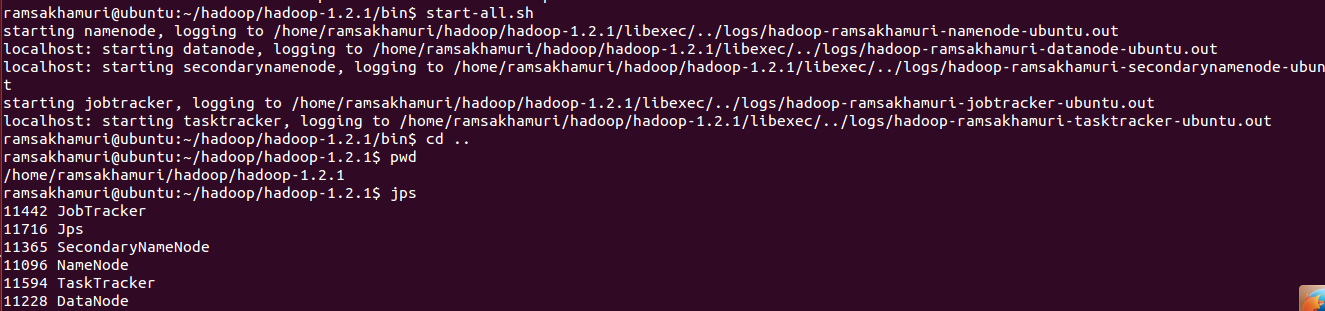
, Leaf-node 0x7f2a21de59b0, parent Non-leaf node 0x7f2a21de54a8, parent Non-leaf node 0x7f2a21dc7ba8, parent None, feature CF (65, [354.70000000000005, 1952.907699999999, -5954.093700000001, 413.0, 213.0, 118.0, 33275.0, -589941.0, 87451.0], 6069805515.32) [0x7f2a21de5828], successors: 1, feature CF (21, [201.09999999999997, 634.1873000000002, -1913.7572999999998, 98.0, 47.0, 24.0, 10087.0, -209979.0, 20535.0], 2131224470.28) [0x7f2a21de58d0], successors: 4, feature CF (2, [12.2, 60.6428, -182.5034, 15.0, 7.0, 4.0, 1253.0, -19998.0, 3362.0], 206680821.94) [0x7f2a21de5a58], entries: 2

CF (1, [6.1, 30.3214, -91.2517, 12.0, 5.0, 2.0, 643.0, -9999.0, 2045.0], 104584931.47) [0x7f2a21de5748] Hadoop\_adress ['/user/root/in/2016-02-22/test3.txt']

CF (1, [6.1, 30.3214, -91.2517, 3.0, 2.0, 2.0, 610.0, -9999.0, 1317.0], 102095890.47) [0x7f2a21de5780] Hadoop\_adress ['/user/root/in/2016-02-22/test3.txt']

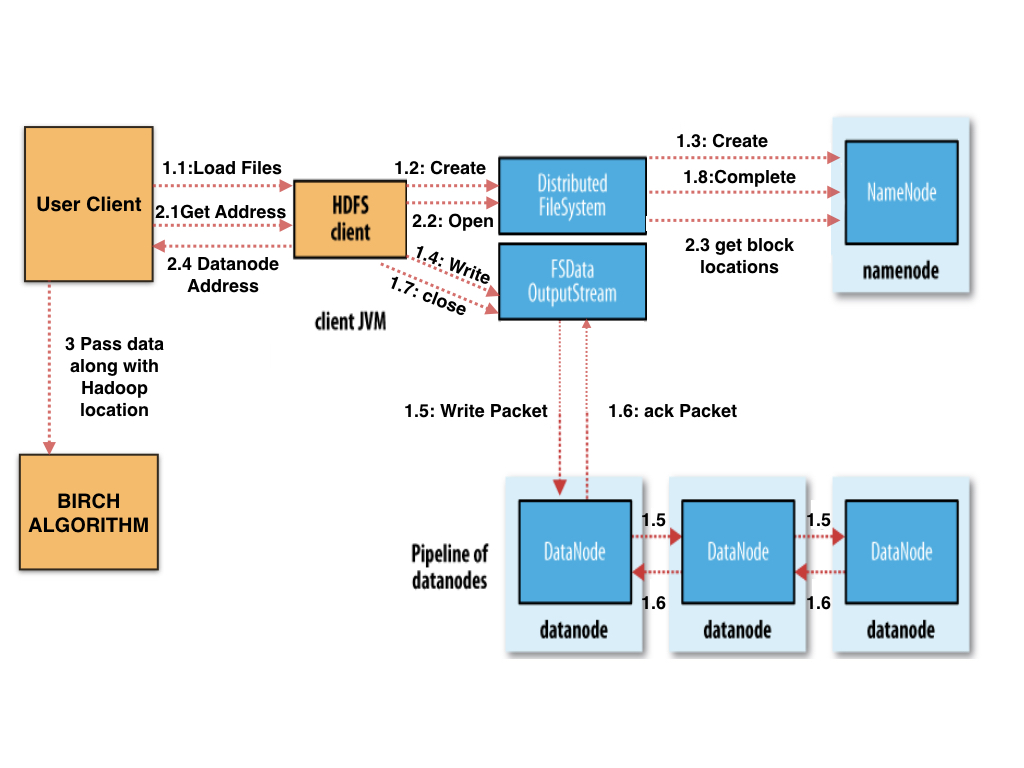
]

1. Successfully installed Hadoop cluster on local machine.
   * And executed few commands to make sure it is working fine. Below executions shows all the running daemon processes running on VM.



1. Designed a client process which automates the entire system process.
   * This system includes a shell script client.sh and a python script client.py. And they both executes hand in hand.
   * Shell script client.sh runs as a background process.
     1. It looks for the files in INPUT directory and once it gets any files, moves them into HADOOP processing directory.
     2. Then it loads the data into Hadoop and handshakes Python process.
   * Python script client.py also runs in the background along with above process.
     1. It looks for the Hadoop processed files and once it gets those files it pulls the address of each file from HDFS.
     2. It gives the data files one by one to birch along with their respective addresses.

Entire system looks like the below picture.



***Appendix 2:***

**Client.sh:**

IN\_DIR='/home/ramsakhamuri/PROJECT/INPUT'

HADOOP\_PROC\_DIR='/home/ramsakhamuri/PROJECT/HADOOP\_PROCESS'

BIRCH\_PROC\_DIR='/home/ramsakhamuri/PROJECT/BIRCH\_PROCESS'

while true

do

DATE=`date +%Y-%m-%d`

HADOOP\_PATH='/user/ramsakhamuri/in/'$DATE

if [[ $(ls $IN\_DIR) != '' ]]

then

while [[ $(ls $HADOOP\_PROC\_DIR) != '' ]]

do

sleep 10

echo "previous files are still in Birch process directory "$HADOOP\_PROC\_DIR

done

mv $IN\_DIR/\* $HADOOP\_PROC\_DIR

hadoop fs -ls $HADOOP\_PATH || hadoop fs -mkdir $HADOOP\_PATH

hadoop fs -copyFromLocal $HADOOP\_PROC\_DIR/\* $HADOOP\_PATH

while [[ $(ls $BIRCH\_PROC\_DIR) != '' ]]

do

sleep 10

echo "previous files are still in Birch process directory "$BIRCH\_PROC\_DIR

done

touch $BIRCH\_PROC\_DIR/$DATE

while [ ! -f $HADOOP\_PROC\_DIR"/DONE" ]

do

sleep 10

done

mv $HADOOP\_PROC\_DIR/\* $BIRCH\_PROC\_DIR

rm $BIRCH\_PROC\_DIR/$DATE

fi

sleep 60

done

**client.py:**

# -\*- coding: utf-8 -\*-

"""

Created on Sat Jan 23

@author: ramsakhamuri

"""

from pyclustering.cluster.birch import birch

import os

import time

import shutil

import subprocess

PROC\_DIR='/home/ramsakhamuri/PROJECT/BIRCH\_PROCESS/'

HADOOP\_HOME='/user/ramsakhamuri/in/'

HADOOP\_PROC\_DIR='/home/ramsakhamuri/PROJECT/HADOOP\_PROCESS/'

PROCESSED\_DIR='/home/ramsakhamuri/PROJECT/PROCESSED/'

LOGFILE\_DIR='/home/ramsakhamuri/PROJECT/LOGDIR/logfile'

birch1=birch([[]],10,[],initial\_diameter=10)

z=0

while True :

while (os.listdir(PROC\_DIR)==[]):

time.sleep(30)

for datefile in os.listdir(PROC\_DIR):

DATE=datefile

HADOOP\_PATH= HADOOP\_HOME + DATE

os.popen("touch "+HADOOP\_PROC\_DIR + "DONE")

#Below loop is to check moving of files from Hadoop dir to Birch Dir is finished

while(os.path.isfile(PROC\_DIR+datefile) == True):

time.sleep(10)

DONEFILE=PROC\_DIR+"DONE"

os.remove(DONEFILE)

d1={}

#d2=[]

d2=os.listdir(PROC\_DIR)

l=[]

c=0

d=0

p = subprocess.Popen(["hadoop","fsck",HADOOP\_PATH,"-files","-blocks","-locations"], stdout=subprocess.PIPE)

(output,error)=p.communicate()

for x in str(output)[2:len(output)-3].split('\\n'):

if d<len(d2):

for y in d2:

if y in x:

d1[y]=c

d=d+1

l.append(x)

c=c+1

for file in d2:

address=[]

data=l[d1[file]+1:l[d1[file]+1:].index('')+d1[file]+1]

for line in data:

address+=line[line.index('[')+1:line.index(']')].replace(' ','').split(',')

address=list(set(address))

d1[file]=address

for birch\_file in d2:

data=[]

file=''

file=PROC\_DIR+birch\_file

for i in open(file,'r').readlines():

k=i.strip().split()

for j in range(0,len(k)):

k[j]=float(k[j])

#k=k[2:]

data.append(k)

birch1.\_birch\_\_pointer\_data=data;

birch1.\_birch\_\_hadoop\_address=d1[birch\_file]

birch1.process()

for file2 in d2:

file2=PROC\_DIR+"/"+file2

shutil.move(file2,PROCESSED\_DIR)

file\_out=LOGFILE\_DIR+str(z)+'.txt'

f1=open(file\_out,'w')

f1.write(str(d2))

f1.write("CF Tree after the previous load")

f1.write("\n")

f1.write(str(birch1.\_birch\_\_tree.leafes))

f1.close()

z=z+1

**birch.py:**

# -\*- coding: utf-8 -\*-

"""

Created on Sat Jan 23

@author: ramsakhamuri

"""

from pyclustering.cluster.birch import birch

import os

import time

import shutil

#import subprocess

############### Configure the below variables for a change in system #################

PROJECT='/root/Desktop/ram/'

HADOOP\_HOME='/user/root/in/'

#####################################################################################

PROC\_DIR= PROJECT + 'PROJECT/BIRCH\_PROCESS/'

HADOOP\_PROC\_DIR= PROJECT + 'PROJECT/HADOOP\_PROCESS/'

PROCESSED\_DIR= PROJECT + 'PROJECT/PROCESSED/'

LOGDIR= PROJECT + 'PROJECT/LOGDIR/logfile'

birch1=birch([[]],10,[],initial\_diameter=10)

z=0

while True :

while (os.listdir(PROC\_DIR)==[]):

time.sleep(30)

for datefile in os.listdir(PROC\_DIR):

DATE=datefile

HADOOP\_PATH= HADOOP\_HOME + DATE +'/'

os.popen("touch "+HADOOP\_PROC\_DIR + "DONE")

#Below loop is to check moving of files from Hadoop dir to Birch Dir is finished

while(os.path.isfile(PROC\_DIR+datefile) == True):

time.sleep(10)

DONEFILE=PROC\_DIR+"DONE"

os.remove(DONEFILE)

d1={}

d2=os.listdir(PROC\_DIR)

for file in d2:

address=[]

address.append(HADOOP\_PATH + file)

d1[file]=address

for birch\_file in d2:

data=[]

file=''

file=PROC\_DIR+birch\_file

for i in open(file,'r').readlines():

k=i.strip().split()

for j in range(0,len(k)):

k[j]=float(k[j])

#k=k[2:]

data.append(k)

birch1.\_birch\_\_pointer\_data=data;

birch1.\_birch\_\_hadoop\_address=d1[birch\_file]

birch1.process()

for file2 in d2:

file2=PROC\_DIR+"/"+file2

shutil.move(file2,PROCESSED\_DIR)

file\_out= LOGDIR +str(z)+'.txt'

f1=open(file\_out,'w')

f1.write(str(d2))

f1.write("CF Tree after the previous load")

f1.write("\n")

f1.write(str(birch1.\_birch\_\_tree.leafes))

f1.close()

z=z+1

# Bibliography

Athman Bouguettaya, Q. Y. (2014). Efficient agglomerative hierarchical clustering. *Expert Systems with Applications* .

Larose, D. T. (2015). *Data MIning and Predictive Analystics.* Wiley.

Layton, R. (2015). *Learning Data Mining with Python.* PACKT PUBLISHING.

Tian Zhang, R. R. (1996). *BIRCH: an efficient data clustering method for very large databases.* NY.

White, T. (2012). *Hadoop: The Definitive Guide* (Third Edition ed.). O'REILLY.