MapReduce for K-means Clustering on Hadoop:

Let's solve a clustering problem using MapReduce on Hadoop.

To be more specific, we'll be clustering text documents containing different words and context. So, let's begin.

Step 1: Open Cloudera Quickstart VM.

Step 2: Clone the following repository on your local host machine.

`www.github.com/NSTiwari/Hadoop-MapReduce-Programs`

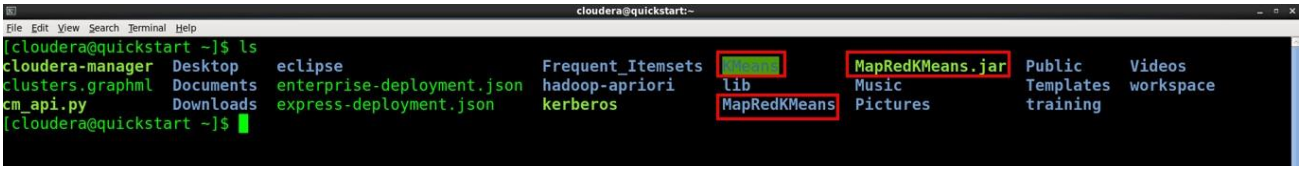
You'll see the **`KMeansClustering`** folder inside this repository along with other folders. The **`KMeansClustering`** folder contains two sub-folders – **`KMeans`** and **`MapRedKMeans`** respectively. Copy these sub-folders on the guest machine inside **`/home/cloudera`** directory.

The **`MapRedKMeans`** sub-directory contains a file named **`MapRedKMeans.jar.`** Copy or move this file to **`/home/cloudera.`**

Overall, two directories – **`Kmeans`** and **`MapRedKMeans`** and one file – **`MapRedKMeans.jar`** should be present inside **`/home/cloudera`**.

Check if these directories and file are present or not.

ls



As you can see, all three entities are present.

Step 3: Create a corpus of text documents as a dataset containing 'k' no. of clusters you want.

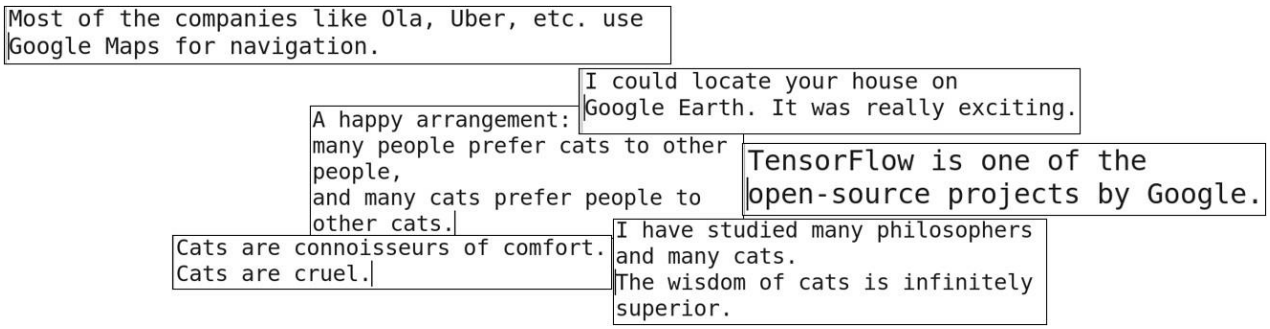
In this example, we have used text documents for two classes – Cats and Google. In other words, the dataset would form two clusters.

Create a directory named **`data`** inside **`/home/cloudera/KMeans.`** Next, create 'k' no. of sub-folders inside the **`data`** directory. Each sub-folder would contain certain text documents.

For our example, since k=2, we create two sub-directories. Each sub-directory contains 10 text documents. The directory structure looks something like the following.

```
KMeans
|__data
    |__cats
    |    |__cats1
    |    |__cats2
    |    |…
    |    |__cats10
    |
    |__google
        |__google1
        |__google2
        |…
        |__google10
```

Following are some sample text documents of each sub-directory.



These text documents contain the words ‘cat’ and ‘Google’.

Step 4: Vectorize the corpus.

Now, we need to vectorize the corpus. In other words, we represent every document from the corpus as a vector whose length is equal to the vocabulary of the corpus.

To do so, navigate to **`KMeans`** directory and run the following command.

java -jar ProcessCorpus.jar

```
[cloudera@quickstart ~]$ cd KMeans
[cloudera@quickstart KMeans]$ ls
20_newsgroups  data  GetCentroids.jar  GetDistribution.jar  KMeans.jar  ProcessCorpus.jar  SeqKMeans
[cloudera@quickstart KMeans]$
[cloudera@quickstart KMeans]$ java -jar ProcessCorpus.jar
Enter the directory where the corpus is located: data
Enter the name of the file to write the result to: vectors
Enter the max number of docs to use in each subdirectory: 15
data
Counting the number of occurs of each word in the corpus..Found 137 unique words in the corpus.
How many of those words do you want to use to process the docs? 100
Done creating the dictionary.
Converting the corpus to a list of vectors.Done vectorizing all of the docs!
[cloudera@quickstart KMeans]$
```

You’ll be prompted to specify the corpus directory and the name of the resultant vector file.

Our corpus is located inside **`data`** directory and we named our resultant vector file as **`vectors.`** Each sub-directory inside the **`data`** folder contains 10 documents.

Note:

> We provided 15 as the parameter for the maximum documents that can be used (max documents taken into consideration) for K-means clustering. Since 15>10, all the documents are considered for clustering. You can give any other number as long as it is greater than zero.

It is found that there are 137 unique words across all the 20 documents in both the sub-directories. We choose 100 out of these 137 words at random, for processing. This means that, the vector length is 100.

Every term in the corpus is vectorized mathematically by considering the term frequency. The value for each term lies between 0 and 1.

Consider the following **`vectors`** file generated.

```
key: data/cats/cats7~; value: cats; len: 100; 1:0.14285714285714285; 4:0.14285714285714285;
8:0.14285714285714285; 27:0.14285714285714285; 31:0.14285714285714285; 59:0.14285714285714285;
66:0.14285714285714285;
key: data/cats/cats10~; value: cats; len: 100; 1:0.07692307692307693; 8:0.15384615384615385;
9:0.07692307692307693; 12:0.07692307692307693; 13:0.07692307692307693; 15:0.15384615384615385;
18:0.07692307692307693; 19:0.15384615384615385; 49:0.07692307692307693; 56:0.07692307692307693;
key: data/cats/cats4; value: cats; len: 100; 1:0.14285714285714285; 2:0.07142857142857142;
3:0.07142857142857142; 5:0.14285714285714285; 6:0.07142857142857142; 7:0.07142857142857142;
9:0.07142857142857142; 39:0.07142857142857142; 65:0.07142857142857142; 72:0.07142857142857142;
86:0.07142857142857142; 88:0.07142857142857142;
```

Let's understand this file.

Interpretation:

After vectorization of documents, the 27th word out of 100 uniquely chosen words has a value of 0.1428 and it lies inside **`cats7`** document. Similarly, the 49th unique word that lies inside **`cats10`** document has value 0.0769 and the 39th unique word that lies inside **`cats4`** document has value 0.0714.

So, now we have a vectorized version of the corpus for each term with a value that determines how frequently that term repeats in the subsequent documents.

Step 5: Generate centroids.

Now, we'll generate a file named **`clusters`** that will choose initial set of centroids from the data.

Run the following command:

java -jar GetCentroids.jar

```
[cloudera@quickstart KMeans]$ java -jar GetCentroids.jar
Enter the data file to select the clusters from: vectors
Enter the name of the file to write the result to: clusters
Enter the number of clusters to select: 2
.Done selecting centroids.
[cloudera@quickstart KMeans]$
```

This will create a file called **`clusters`** which has two cluster centroids randomly generated as shown below.

```
key: cluster0; value: google; len: 100; 0:0.09090909090909091; 14:0.09090909090909091;
20:0.18181818181818182; 30:0.09090909090909091; 46:0.09090909090909091; 54:0.09090909090909091;
55:0.09090909090909091; 63:0.09090909090909091; 73:0.09090909090909091; 95:0.09090909090909091;
key: cluster1; value: cats; len: 100; 1:0.07692307692307693; 8:0.15384615384615385;
9:0.07692307692307693; 12:0.07692307692307693; 13:0.07692307692307693; 15:0.15384615384615385;
18:0.07692307692307693; 19:0.15384615384615385; 49:0.07692307692307693; 56:0.07692307692307693;
```

As you can see, the points 0, 14, 20, 30, 46, 54, 55, 63, 73 and 95 belong to first cluster (cluster0) and the points 1, 8, 9, 12, 13, 15, 18, 19, 49, and 56 belong to second cluster (cluster1).

Note that they were assigned to these clusters randomly by generating centroids from the data.

Now, we need to apply K-means clustering algorithm on this data to actually determine what clusters do these points (or words) belong to.

Step 6: Apply K-means Clustering.

Run the following command:

java -jar KMeans.jar

```
[cloudera@quickstart KMeans]$ java -jar KMeans.jar
Enter the file with the data vectors: vectors
Enter the name of the file where the clusers are loated: clusters
Enter the number of iterations to run: 2
.Done with pass thru data.
.Done with pass thru data.
[cloudera@quickstart KMeans]$
```

Step 7: Get the final distribution of points.

Now, after applying K-means Clustering, the points are clustered according to their frequency. Run the following command to find the distribution.

java -jar GetDistribution.jar

```
[cloudera@quickstart KMeans]$ java -jar GetDistribution.jar
Enter the file with the data vectors: vectors
Enter the name of the file where the clusers are loated: clusters
.Done with pass thru data.
******* cluster0 ******* google: 13; cats: 3;

******* cluster1 ******* cats: 12; google: 2;
[cloudera@quickstart KMeans]$
```

So, the distribution says that 13 words that belonged to **`google`** sub-directory and 3 words that belonged to **`cats`** sub-directory are clustered together in **`cluster0`.**

12 words from **`cats`** sub-directory and 2 words from **`google`** sub-directory are clustered together in **`cluster1`**.

Thus, it can be concluded that **`cluster0`** refers to **`Google`** text documents and **`cluster1`** refers to **`Cats`** text document respectively.

If the results are good enough, we are good to move forward and apply K-means Clustering on Hadoop.

Step 8: Create input directories on HDFS.

Create two directories named **data** and **clusters** on HDFS.

sudo -u hdfs hadoop fs -mkdir /data
sudo -u hdfs hadoop fs -mkdir /clusters
hdfs dfs -ls /

```
[cloudera@quickstart ~]$ sudo -u hdfs hadoop fs -mkdir /data
[cloudera@quickstart ~]$ sudo -u hdfs hadoop fs -mkdir /clusters
[cloudera@quickstart ~]$ hdfs dfs -ls /
Found 7 items
drwxr-xr-x   - hdfs  supergroup          0 2021-04-01 00:43 /clusters
drwxr-xr-x   - hdfs  supergroup          0 2021-04-01 00:42 /data
drwxr-xr-x   - hbase supergroup          0 2021-03-26 07:37 /hbase
drwxr-xr-x   - solr  solr                0 2015-06-09 03:38 /solr
drwxrwxrwx   - hdfs  supergroup          0 2021-03-26 09:26 /tmp
drwxr-xr-x   - hdfs  supergroup          0 2021-03-31 03:07 /user
drwxr-xr-x   - hdfs  supergroup          0 2015-06-09 03:36 /var
[cloudera@quickstart ~]$
```

Step 9: Copy input files on HDFS.

Copy the **`vectors`** file (created in Step 4) and **`clusters`** file (created in Step 5) to **`/data`** directory and **`/clusters`** directory on HDFS respectively.

sudo -u hdfs hadoop fs -put vectors /data
sudo -u hdfs hadoop fs -put clusters /clusters

Check if the files were copied successfully.

hdfs dfs -ls /data
hdfs dfs -ls /clusters

```
[cloudera@quickstart ~]$ hdfs dfs -ls /data
Found 1 items
-rw-r--r--   1 hdfs supergroup       7329 2021-04-01 00:49 /data/vectors
[cloudera@quickstart ~]$
[cloudera@quickstart ~]$ hdfs dfs -ls /clusters
Found 1 items
-rw-r--r--   1 hdfs supergroup       2776 2021-04-01 00:49 /clusters/clusters
[cloudera@quickstart ~]$
```

Both the files are copied successfully to their respective destinations on HDFS.

Step 10: Configure permissions to run MapReduce for K-means clustering on Hadoop.

Before we run the MapReduce job on Hadoop, we need to give permission to read, write and execute the MapReduce program. We also need to provide permission for the default user (cloudera) to write the output file inside HDFS.

Run the following commands:

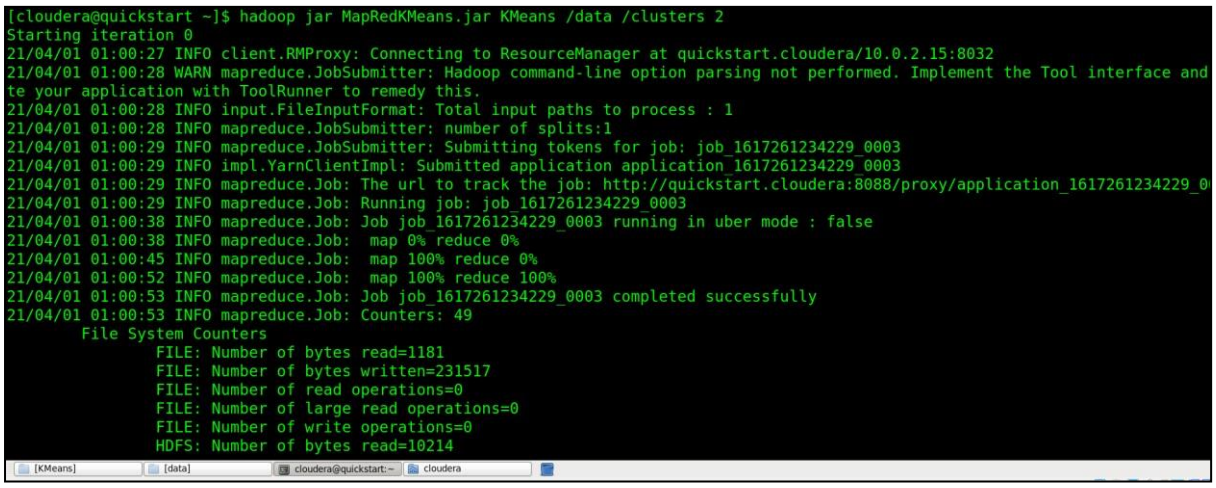
```
[cloudera@quickstart ~]$ chmod 777 vectors clusters KMeans/
[cloudera@quickstart ~]$ sudo -u hdfs hadoop fs -chown cloudera /data /clusters
[cloudera@quickstart ~]$
```

Step 11: Run MapReduce on Hadoop.

Now, we are all set to run the MapReduce job(s) on Hadoop.

hadoop jar MapRedKMeans.jar KMeans /data /clusters 2

This will run 2 iterations of the K-means algorithm on top of all 20 documents in the **`Cats`** and **`Google`** data set. "**`/data`**" is the directory in HDFS where the data are located, "**`/clusters`**" is the directory where the initial clusters are located, and "2" is the number of iterations to run; this means that two separate MapReduce jobs will be run in sequence.



Step 12: Read the MapReduce output.

hdfs dfs -ls /

```
[cloudera@quickstart ~]$ hdfs dfs -ls /
Found 9 items
drwxr-xr-x   - cloudera supergroup          0 2021-04-01 00:59 /clusters
drwxr-xr-x   - hdfs     supergroup          0 2021-04-01 01:00 /clusters1
drwxr-xr-x   - hdfs     supergroup          0 2021-04-01 01:01 /clusters2
drwxr-xr-x   - cloudera supergroup          0 2021-04-01 00:59 /data
drwxr-xr-x   - hbase    supergroup          0 2021-03-26 07:37 /hbase
drwxr-xr-x   - solr     solr                0 2015-06-09 03:38 /solr
drwxrwxrwx   - hdfs     supergroup          0 2021-03-26 09:26 /tmp
drwxr-xr-x   - hdfs     supergroup          0 2021-03-31 03:07 /user
drwxr-xr-x   - hdfs     supergroup          0 2015-06-09 03:36 /var
```

So, **clusters1** and **clusters2** are the two directories created. The final output is stored in **clusters2.**

Let's copy both these directories on local machine and investigate what's inside them.

hdfs dfs -copyToLocal /clusters1
hdfs dfs -copyToLocal /clusters2

```
        key: cluster0; value: 16; len: 100; 0:0.13518125913516962; 1:0.0066650390625; 2:0.046435546875;
3:0.052231233016304335; 4:0.03887939453125; 5:0.01666259765625; 6:0.005795686141304348; 7:0.03813883463541667;
9:0.0228515625; 10:0.012118252840909092; 11:0.03173014322916667; 13:0.019775390625; 14:0.024578978044713436;
15:0.009521484375; 16:0.016049592391304348; 20:0.024236505681818184; 21:0.016049592391304348;
22:0.024332682291666668; 24:0.028881835937499996; 25:0.035546875000000006; 29:0.012118252840909092;
30:0.012118252840909092; 32:0.005795686141304348; 33:0.012118252840909092; 34:0.005795686141304348; 35:0.01025390625;
36:0.01025390625; 37:0.005795686141304348; 40:0.014811197916666666; 41:0.01025390625; 42:0.01025390625;
43:0.022216796875; 45:0.012118252840909092; 46:0.012118252840909092; 47:0.014811197916666666; 48:0.022216796875;
51:0.014811197916666666; 52:0.012118252840909092; 53:0.014811197916666666; 54:0.012118252840909092;
55:0.012118252840909092; 57:0.005795686141304348; 58:0.012118252840909092; 60:0.01025390625; 63:0.012118252840909092;
64:0.005795686141304348; 68:0.005795686141304348; 71:0.012118252840909092; 73:0.012118252840909092; 75:0.01025390625;
76:0.005795686141304348; 80:0.014811197916666666; 82:0.012118252840909092; 83:0.022216796875;
84:0.005795686141304348; 87:0.012118252840909092; 90:0.012118252840909092; 91:0.005795686141304348;
92:0.0066650390625; 95:0.012118252840909092; 96:0.005795686141304348;
        key: cluster1; value: 14; len: 100; 1:0.11673681639060644; 2:0.03295501874219075; 3:0.021970012494793835;
4:0.05144644592530889; 5:0.03222268499236428; 6:0.027096348743579062; 7:0.026364014993752603; 8:0.045630025950725656;
9:0.022815012975362828; 10:0.03460276967930029; 12:0.027209015474321595; 13:0.01183000672796591;
15:0.02366001345593182; 17:0.030208767180341524; 18:0.027209015474321592; 19:0.02366001345593182;
23:0.030758017492711374; 26:0.015379008746355687; 27:0.02197001249479383; 28:0.015379008746355687;
31:0.02197001249479383; 38:0.015379008746355687; 39:0.010985006247396916; 44:0.010985006247396916;
49:0.01183000672796591; 50:0.019223760932944603; 56:0.01183000672796591; 59:0.02197001249479383;
61:0.019223760932944603; 62:0.019223760932944603; 65:0.010985006247396916; 66:0.02197001249479383;
67:0.010985006247396916; 69:0.010985006247396916; 70:0.015379008746355687; 72:0.010985006247396916;
```

All the points highlighted in red belong to **cluster0** and all the points highlighted in blue belong to **cluster1**.

Congratulations, you've just implemented K-means Clustering on textual data using MapReduce on Hadoop.