# Question 1: MapReduce framework

## MapReduce discussion

MapReduce framework is a part of Hadoop ecosystem that was designed to perform batch analysis on data which is distributed among the cluster of computers. It is a powerful tool that takes advantage of a big number of machines [1] and perform calculations in a parallel manner, though there are several weak sides that should be considered. Hadoop is a representative of a master-worker architecture, and it applies to MapReduce framework as well: Job Tracker is located at the master node and the Task Trackers are located on each work node. Job Tracker takes care of assigning Map and Reduce tasks to the workers - Task Trackers. Mapper is responsible for cleaning and labeling data, whereas Reducer covers assigned calculations. It is important to mention that framework follows the principal that calculations have to be brought to data, not data to calculations. On one hand it contributes to efficiency, but on the other hand requires proper scheduling, resource management and monitoring. This management processes are both time consuming - they take a significant part of time that algorithm works [2], and resource consuming – Job Tracker may overload the Master Node which may lead to the failure of the latter, and this is a serious problem.Another limitation is type of data which requires analysis/calculation. As was mentioned above, MapReduce is designed to perform batch analysis, so tasks where data is constantly updating (streaming) cannot be covered but the framework. Also, MapReduce requires input data to be independent, which also shorten the list of algorithm possibilities. However, tasks of SELECT type perfectly suits. For example, finding e-mail addresses of all customers in USA that spend more than 100$ at once in 2020 on Amazon or finding home addresses of people older than 18 to send invitations for President Election.

## Description

Question 1 covers implementation of MapReduce framework to find some statistics metrics of the chosen data set. The dataset contains weather data which was collected daily throughout the months of April 2007 in the USA. Data is organized in table manner and saved as .txt, it has a variety of features/columns of different types; however, this question will be focused only on 2 of them: data stamps and related temperature. The task is to calculate average, minimum and maximum temperature for each day in April, as well as daily standard deviation.

## Pseudo code

This section presents the logic behind performed calculations and serves as base to construct the actual python code.

### Mapper



### Reducer



## Python Code

This section contains python code for mapper and reducer, as well as script to run at terminal on cluster. The main idea is to clean, prepare and emit data in the form of tuple (date, value). Meanwhile reducer performs assigned calculations.

### Mapper



### Reducer



### MapReduce script

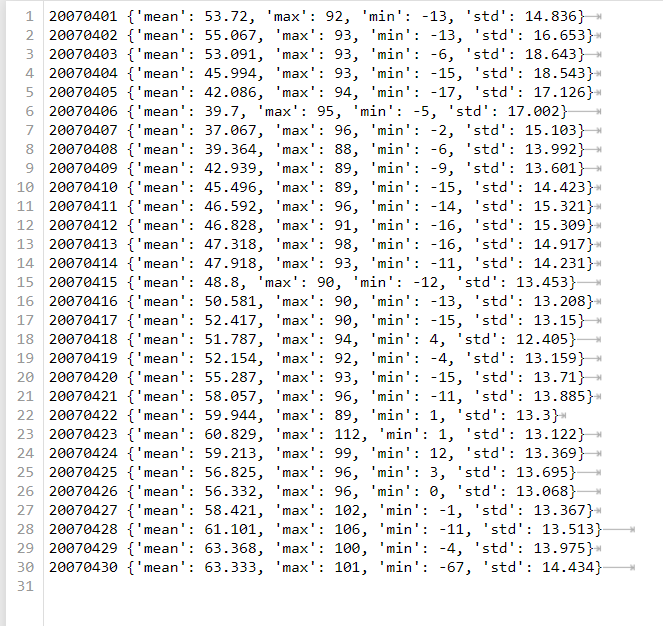


## Results and interpretation

Result of the calculations is stored to the part-00000 file that could be find on cluster inside the folder CW\_Q1\_output:



Screenshot of the results is attached below – requested metrics calculated for each day of month of April 2007.



# Question 2: K-means with Apache Mahout

## Description

K-means is a clustering algorithm that belongs to non-supervised machine learning algorithms. For implementing K-means towards big data Hadoop suggested a special member of its ecosystem – Mahout. It is a machine learning library which is designed to work with Big Data. This question contains steps of k-means implementation with Mahout and HDFS, as well as detailed comments and explanation. The task is to cluster the set of French plays (presented as .txt files) and find the best hyperparameters for the algorithm by testifying different options.

## K-means clustering implementation

#### Step 1: Preparation

Everything starts with creation of a proper directories hierarchy and placing the dataset. Firstly *k-means\_CW* is created to place the dataset to the NameNode on Lena, and then – copy data to HDFS. *CW\_final* is created to store some .sh files, that help to automatize further calculations:

Diagram

Description automatically generated

Diagram 1. Directories hierarchy on the NameNode after preparation step

Here are the commands used to complete the preparation stage:



#### Step 2: Creation of sequence file(s)

At this stage all the members of the dataset (documents) are to be converted into a sequence file(s) for further manipulations, namely - creation of a vectorized data description .



#### Step 3: Sparse representation

Now, with the use of sequence files, a vectorized documents representation (TF-IDF vectors) is implemented.



After implementation of steps 1-3, HDFS contains the following directories and files:

Diagram

Description automatically generated

Diagram 2. HDFS directories hierarchy after first 3 steps

#### Step 4: Create canopies (approximate centroids for k-means)

At this step canopy clustering (built-in Mahout method) is implemented to approximate centroids for the further k-means algorithm implementation. Canopy method requires 2 specific hyperparameters -t1 (“looseness”) and t2 (“tightness”). Their values may vary depending on the type of distance measure and the result one would like to obtain[3]. In this course work 2 types of distance will be tested – Cosine and Euclidean. To approximate best values, some testing was performed. Its details can be found in the table below. To have more options to explore, there have been chosen 2 pairs of t1/t2 for each distance type and then tested on k-means algorithm.

Best t1 and t2 values approximation (double-click to open):



According to the test with French plays dataset and 10 clusters , best pairs t1 and t2 are: [3, 1.5] and [3, 0.5] for Cosine distance, [3, 1.5] and [2000, 1500] for Euclidean distance. To create 4 respective versions of canopy centroids, automation was out in place. Here are 2 python scripts that generates bash files with mahout canopy clustering commands.

For Cosine distance:

****

Canopy clustering for Euclidean distance (double-click to open):

****

After running python scripts above, files were copied to the server and run simultaneously from terminal. Commands from the files are listed below. In yellow is highlighted parts that differs (double-click to open):









The structure of NameNode after step 3 was as follows:

Diagram

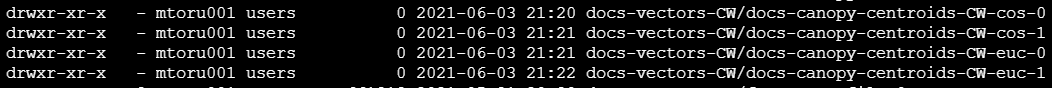
Description automatically generated

Diagram 5. NameNode directories hierarchy after canopy clustering stage

Detailed explanation on how to perform canopy clustering and approximate centroids:



As a result of the actions above, 4 files with centroids approximation were added to the respective output directory. This is an important step it has direct influence on performance of algorithm: it may converge differently or on different iteration depending on where centroids were initialized. The output of canopy clustering script looks as follows:



#### Step 5: K-means algorithm itself

As mentioned before, the main task of the question no. 2 is to explore how the varying of hyperparameters influences on the result. This course work testifies the following hyper-parameters:

* quantity of clusters k = [8, 10, 14, 16, 18, 20, 22, 24, 26, 30],
* measure of distance [Euclidean, Cosine]
* t1/t2 values – “loose”/”tight” distance for canopy clustering

If consider 10 cluster values, 2 distances and 2 canopy centroids options for each distance – 40 different options of k-means algorithm and therefore 40 mahout commands in total are needed. Manual implementation might be time-consuming, so there has been developed a python code that generates commands in separate bash files. Double-click to open:



After been generated, these files have to be copied to the server and simultaneously run with the help of the loop. Here are 4 examples (out of total of 40) on how commands inside files look like. In yellow is highlighted parts of the code that differs. Double-click to open the code:

1. Distance: Cosine, centroids: version 0 (t1 = 3, t2 = 1.5),

quantity of clusters: 8



1. Distance: Cosine, centroids: version 1 (t1 = 3, t2 = 0.5),

quantity of clusters: 8



1. Distance: Euclidean, centroids: version 0 (t1 = 3, t2 = 1.5),

quantity of clusters: 8



1. Distance: Euclidean, centroids: version 0 (t1 = 2000, t2 = 1500),

quantity of clusters: 8



For the sake of efficiency set of commands were divided into 2 groups by type of distance measure when storing on NameNode – KMEANS\_COS and KMEANS\_EUC directories respectively:

Diagram

Description automatically generated

Diagram 6. NameNode directories hierarchy after k-means clustering stage

From the programming perspective, the process of running k-means algorithm contains the following steps:



As a result of running k-means clustering algorithm, 40 output directories with some results were created at HDFS. The explanatory scheme is attached below.

Diagram

Description automatically generated

Diagram 7. NameNode directories hierarchy after k-means clustering step

#### Step 6: Evaluation

The final step is to evaluate k-means algorithm and interpret its results that stored in output directories. It would be time-consuming to type one by one mahout evaluation command 40 times, as well as to check each time on each iteration a particular k-mean algorithm converged (to pass correct directory input). Hence, the automation was put in place.

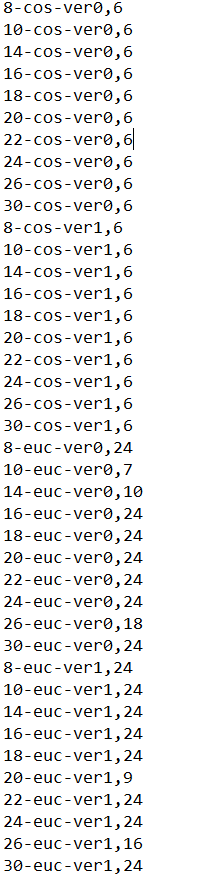
Firstly, here is the helper script, that finds number of iterations. It generates a .sh file which contains 40 bash scripts, each checks the quantity of files in respective k-mean algorithm output directory and store result into .txt file [5]:



Inside this file contains the follows (short example):



After running the script above in server terminal, the .txt file with results is generated: (identifier, num of files in k-means algorithm output directory). Here is an example:



Knowing the number of files, it is easy to calculate the number of iterations and pass correct output directory to cluster-dump command. Python script that generates 40 cluster-dump commands:

 Below is the example of how one of the generated file looks like. In yellow are highlighted parts that are different in each file.



After last step NameNode has the following hierarchy of both helper, and output files:

Diagram

Description automatically generated

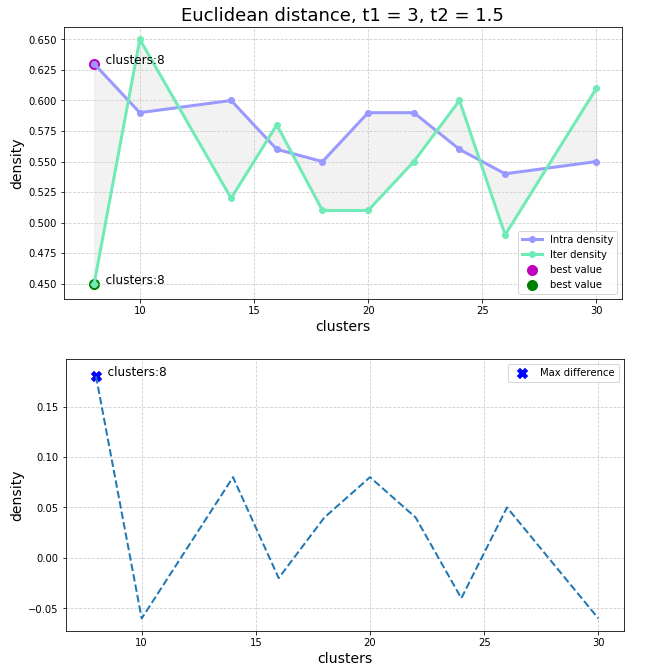
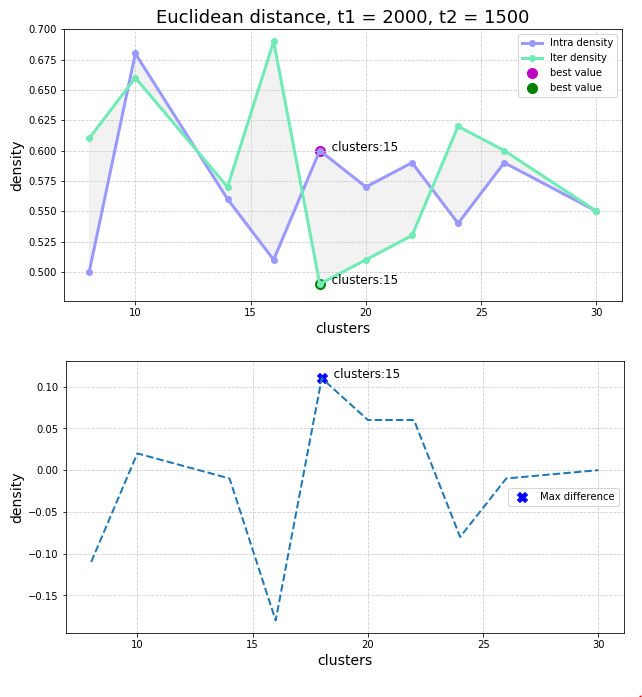
Here is the list of taken actions to perform the final step – evaluation:



## Results and take away

After evaluation, it is explored that out of two tested distances, Cosine performs much better than Euclidean. It can be explained by 3 factors:

1. Euclidean distance works better with lower dimensions[5].
2. Euclidean distance is not suitable for documents clustering problem.
3. Wrong hyperparameters were chosen for centroids approximation in canopy clustering – namely t1 and t2.

Statement that Euclidean distance performs worser becomes evident from the charts to the right. Overall, we can judge about performance of the clustering algorithm considering 2 metrics: average inter- and intra-cluster density. The first refers to density of data points among different clusters, and the latter – density inside the cluster. The higher intra-cluster density and lower inter-cluster density are, the better algorithm performs. With Euclidean distance we can see that for some k density is higher among different clusters rather than inside one, which signals that algorithm failed. We can observe it is happening with no regard to chosen t1 and t2. As for Cosine distance, both choices of t1 and t2 show that best performance of the algorithm gives k = 22 clusters and k = 30 clusters. As explored data set is a corpus of French plays, I assume that 22 clusters may refer to 22 storylines/plots and 30 clusters refers to roughly 30 authors.

The biggest difference between inter- and intra-cluster density and hence the best performing algorithm is obtained by:

* Distance measure: **Cosine**
* Canopy centroids that were approximated with: **t1 = 3** and **t2 = 1.5**
* Quantity of clusters **k = 22**

It is also evident, that on chosen distance of clusters there is no correlation between raising quantity of clusters and better performance. And, for sure, the best result of all (not presented here) would be obtained with the quantity of clusters equal to the quantity of plays.

## References

1. Topic 3, DSM010-2021-APR, Goldsmith, the University of London
2. Vladimir Vlasov, *MapReduce: Limitations, Optimizations and Open Issues* <https://www.researchgate.net/publication/261339577_MapReduce_Limitations_Optimizations_and_Open_Issues>
3. Chapter 9, Sean Owen, Robin Anil, Ted Dunning, Ellen Friedman, *Mahout in Action* [*https://livebook.manning.com/book/mahout-in-action/chapter-9/*](https://livebook.manning.com/book/mahout-in-action/chapter-9/)
4. Loop in bash: <https://stackoverflow.com/questions/41079143/run-all-shell-scripts-in-folder>
5. Bash: write to file <https://linuxize.com/post/bash-write-to-file/>
6. Charu C. Aggarwal, Alexandr Hinneburg, Daniel A. Keim, *On the Surprising Behavior of Distance Metrics in High Dimensional Space* <https://bib.dbvis.de/uploadedFiles/155.pdf>

The following materials was used:

1. Chapter 10, Sean Owen, Robin Anil, Ted Dunning, Ellen Friedman, *Mahout in Action* [*https://livebook.manning.com/book/mahout-in-action/chapter-10/*](https://livebook.manning.com/book/mahout-in-action/chapter-10/)
2. K-means document clustering using Apache Mahout command line: <https://datasciencetutos.wordpress.com/2016/08/04/k-means-document-clustering-using-apache-mahout-command-line/>
3. Mahout Apache <https://mahout.apache.org/>