

ATHENS UNIVERSITY OF ECONOMICS AND  
BUSINESS

---

## K-Means clustering algorithm in MapReduce

---

*Authors:*  
Vaggelis Malandrakis  
Kleio Fragkedaki

*Course:* Big Data Management Systems

*Professor:* Damianos Chatziantoniou

Department of Management Science and Technology

April 7, 2019

# Contents

<b>1</b>	<b>Hadoop Installation</b>	<b>1</b>
<b>2</b>	<b>Dataset Creation</b>	<b>2</b>
2.1	createDataset.py . . . . .	2
2.2	Plot of data points . . . . .	4
<b>3</b>	<b>K-means Clustering Algorithm</b>	<b>5</b>
3.1	Instructions for running k-means in Cloudera . . . . .	5
3.2	run.sh & reader.py . . . . .	5
3.2.1	run.sh . . . . .	6
3.2.2	reader.py . . . . .	6
3.3	MapReduce . . . . .	7
3.3.1	mapper.py . . . . .	7
3.3.2	reducer.py . . . . .	8
3.4	Plot Representation . . . . .	9

## Chapter 1

# Hadoop Installation

The first task is to install hadoop in your local machine. We chose to install Cloudera distribution (which includes hadoop) in order to implement k-means algorithm. For this purpose, you first need to download **Oracle Virtual box** [11] based on your operating system and **Cloudera Quickstart VM** [12].

When everything is installed, you need to create a Virtual Machine with Linux operating system and version ad Ubuntu 64 bit in your VB. After that, set up the newly created VM with cloudera-quickstart-vm-5.13.0-0-virtualbox.ovf file.

## Chapter 2

# Dataset Creation

### 2.1 createDataset.py

We created a text file, containing more than 1M data points in the form of (x, y), where x and y are real numbers. The generation of was biased toward the creation of three clusters. In other words, we chose randomly three centers (10, 25), (2, -9) and (-15, -35) and then, we generated the rest of the points around these, using some random distance following a skewed distribution.

For this purpose we first found distances by using `scipy.stats.skewnorm` library and then we used these distance to choose random points as shown bellow.

---

```

1 from scipy.stats import skewnorm
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import pandas as pd
5 from random import random, uniform
6 from math import sqrt, pow
7
8
9 # get random points that follows skew distribution and
10 # use them as distance
11 def skew():
12
13     fig, ax = plt.subplots(1, 1) # This function creates
14                                   # a figure and a grid of subplots
15
16     a = 5 # skewness parameter, when a = 0 the
17           # distribution is identical to a normal distribution
18
19     d = skewnorm.rvs(a, size=350000)
20
21     # histogram of numbers generated
22     plt.show()
23     ax.hist(d, density=True, histtype='stepfilled', alpha=0.2)
24
25     # return all the numbers generated with skew distribution
26     return d
27
28 # get random point from a circle that has as center
29 # given centroid and as radius the distance found in
30 # skew() function

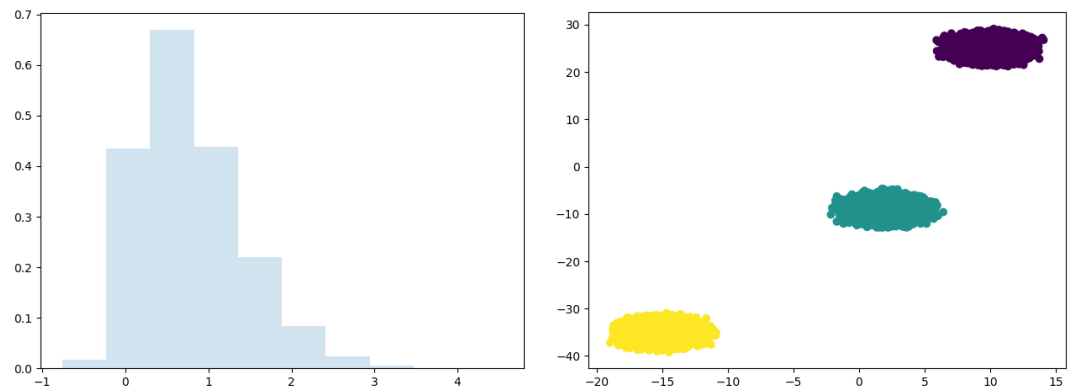
```

```
31 def get_random_point(x, y, cluster):
32
33     array = []
34     distance = skew()
35
36     for d in distance:
37         while True:
38             angle = random() * math.pi * 2
39             x1 = x + math.cos (angle ) * d
40             y1 = y + math.sin (angle ) * d
41
42             if [x1, y1, cluster] not in array:
43                 print('yeah')
44                 break # if the pair of (x1,y1) exist find another pair
of (x1,y1)
45
46             array.append([x1, y1, cluster])
47             print(x1, y1, cluster, len(array))
48
49     return array
50
51
52 if __name__ == "__main__":
53
54     # pre-define three centers for clusters
55     centers = [[10, 25], [2, -9], [-15, -35]]
56     data = [] # array with all the final data
57
58     for center in centers:
59         # concat the results of these
60         # three clusters in one array
61         data = data + get_random_point ( center[0], center[1],
centers.index(center)) print(len(data))
62         print(data)
63
64     plt.scatter([item[0] for item in data], [item[1] for item in
data], c=[item[2] for item in data])
65     plt.show()
66
67     df = pd.DataFrame(data)
68     df.drop(df.columns[[2]], axis=1, inplace=True)
69     df.to_csv("datasetNew", header=False, index= False, sep="," )
```

## 2.2 Plot of data points

The distances were generated with the use of `skew()` function and follow skew distribution as left image reveals.

The data points that were generated are shown in the right image.



## Chapter 3

# K-means Clustering Algorithm

K-means algorithm is the most well-known and commonly used clustering method. It takes the input parameter,  $k$ , and partitions a set of  $n$  objects into  $k$  clusters so that similarity of points inside cluster is high. Cluster similarity is measured according to the mean.

### 3.1 Instructions for running k-means in Cloudera

Before running k-means algorithm, we need to follow some steps. First of all, *download MapReduce folder* from our gitHub repository.

MapReduce folder includes six main files which are `mapper.py`, `reducer.py`, `reader.py`, `run.sh`, `centroids.txt`, `dataset.txt`. `centroids.txt` file includes three points in the shape of  $(x,y)$ , that k-means algorithm gets as initial centroids. These points were selected in a way that k-means to be executed more than one times, but less than five so as the whole process not to be that long. Furthermore, the `dataset.txt` file is the one that was created in the previous chapter. These six files are necessary for k-means and they need to be in the same folder.

After that, you need to enter MapReduce folder from terminal by using the command `cd /path-to-MapReduce-Folder/MapReduce`. Secondly, you need to *install Python3 in Cloudera*. For this purpose, you can read [10].

In addition, we need to create a folder named "testMapReduce" in hdfs and transfer `dataset.txt` file inside this folder. In order to create a folder in hdfs you can execute `hadoop fs -mkdir /testMapReduce` and to transfer dataset `hadoop fs -copyFromLocal dataset.txt /testMapReduce/dataset.txt` command.

Finally, to run k-means algorithm you need to type `sh run.sh`.

### 3.2 run.sh & reader.py

In order to run k-means algorithm a lot of times, we created a bash shell script that starts mapreduce in hadoop with different `centroid.txt` files every time it is running. This shell script ends when previous centroids minus newly generated ones have distance less than one (check `reader.py` script bellow).

More specifically, `i` variable is used for the creation of different outputs in hdfs and the idea of this script is to change `centroid.txt` file in local folder with the one generated as output and saved in hdfs from mapreduce process.

### 3.2.1 run.sh

```

1  #!/bin/bash
2
3  i=1
4  while :
5  do
6      hadoop jar ../../../../usr/lib/hadoop-mapreduce/hadoop-
        streaming.jar -file centroids.txt -file ./mapper.py -mapper
        ./mapper.py -file ./reducer.py -reducer ./reducer.py -input
        /testMapReduce/dataset -output /testMapReduce/mapreduce-
        output$i
7
8      rm -f centroids1.txt
9
10     hadoop fs -copyToLocal /testMapReduce/mapreduce-output$i/part
        -00000 centroids1.txt
11
12     seeiftrue='python reader.py'
13     if [ $seeiftrue = 1 ]
14     then
15         rm centroids.txt
16         hadoop fs -copyToLocal /testMapReduce/mapreduce-output$i/
        part-00000 centroids.txt
17         break
18     else
19         rm centroids.txt
20         hadoop fs -copyToLocal /testMapReduce/mapreduce-output$i/
        part-00000 centroids.txt
21     fi
22     i=$((i+1))
23 done

```

### 3.2.2 reader.py

```

1  __authors__ = "Vaggelis Malandrakis, Kleio Fragkedaki"
2
3  from mapper import getCentroids
4
5  #check if distance of centroids and centroids1 is less than 1
6  def checkCentroidsDistance(centroids, centroids1):
7      f1x = abs(centroids[0][0] - centroids1[0][0]) < 1
8      f1y = abs(centroids[0][1] - centroids1[0][1]) < 1
9      f2x = abs(centroids[1][0] - centroids1[1][0]) < 1
10     f2y = abs(centroids[1][1] - centroids1[1][1]) < 1
11     f3x = abs(centroids[2][0] - centroids1[2][0]) < 1
12     f3y = abs(centroids[2][1] - centroids1[2][1]) < 1
13
14     if f1x and f1y and f2x and f2y and f3x and f3y:
15         print(1)
16     else:
17         print(0)
18
19  if __name__ == "__main__":
20     centroids = getCentroids('centroids.txt')
21     centroids1 = getCentroids('centroids1.txt')
22
23     checkCentroidsDistance(centroids, centroids1)

```



## 3.3 MapReduce

To implement mapreduce in hadoop, we created two files, mapper.py and reducer.py, as described in [4] and [8].

### 3.3.1 mapper.py

As regards mapper, mapper's job is to create the clusters. More specifically, every point of the dataset matches with one of the centroids that are in the centroid.txt file at the time. So, in the end 3 clusters are generated.

---

```
1 #!/usr/bin/env python
2 """mapper.py"""
3
4 __authors__ = "Vaggelis Malandrakis, Kleio Fragkedaki"
5
6 import sys
7 from math import sqrt
8
9 # get initial centroids from a txt file and add them in an array
10 def getCentroids(filepath):
11     centroids = []
12
13     with open(filepath) as fp:
14         line = fp.readline()
15         while line:
16             if line:
17                 try:
18                     line = line.strip()
19                     cord = line.split(',')
20                     # cord[0] is x and cord[1] is y point of a
21                     centroid
22                     centroids.append([float(cord[0]), float(cord[1])
23                                     ])
24                 except:
25                     break
26             else:
27                 break
28         line = fp.readline()
29     fp.close()
30     return centroids
31
32 # create clusters based on initial centroids
33 def createClusters(centroids):
34
35     #read dataset.txt
36     for line in sys.stdin:
37         line = line.strip()
38         cord = line.split(',')
39         min_dist = 1000000000000000
40         index = -1
41
```

```

42     for centroid in centroids:
43         try:
44             cord[0] = float(cord[0])
45             cord[1] = float(cord[1])
46         except ValueError:
47             # float was not a number, so silently
48             # ignore/discard this line
49             continue
50
51         # euclidian distance from every point of dataset
52         # to every centroid
53         cur_dist = sqrt(pow(cord[0] - centroid[0], 2) + pow(
54             cord[1] - centroid[1], 2))
55
56         # find the centroid which is closer to the point
57         if cur_dist <= min_dist:
58             min_dist = cur_dist
59             index = centroids.index(centroid)
60
61         var = "%s\t%s\t%s" % (index, cord[0], cord[1])
62         print(var)
63
64 if __name__ == "__main__":
65     centroids = getCentroids('centroids.txt')
66     createClusters(centroids)

```

### 3.3.2 reducer.py

On the other hand, reducer's job is to find the average centroids from the newly given cluster map. More specifically, every point of each cluster is being added in order the center of this cluster to be found. So, in the end the new centroids of each cluster are generated.

---

```

1  #!/usr/bin/env python
2  """reducer.py"""
3
4  __authors__ = "Vaggelis Malandrakis, Kleio Fragkedaki"
5
6  import sys
7
8  def calculateNewCentroids():
9      current_centroid = None
10     sum_x = 0
11     sum_y = 0
12     count = 0
13
14     # input comes from STDIN
15     for line in sys.stdin:
16
17         # parse the input of mapper.py
18         centroid_index, x, y = line.split('\t')
19
20         # convert x and y (currently a string) to float
21         try:

```

```

22     x = float(x)
23     y = float(y)
24 except ValueError:
25     # float was not a number, so silently
26     # ignore/discard this line
27     continue
28
29 # this IF-switch only works because Hadoop sorts map output
30 # by key (here: word) before it is passed to the reducer
31 if current_centroid == centroid_index:
32     count += 1
33     sum_x += x
34     sum_y += y
35 else:
36     if count != 0:
37         # print the average of every cluster to get
38         # new centroids
39         print(str(sum_x / count) + ", " + str(sum_y / count))
40
41     current_centroid = centroid_index
42     sum_x = x
43     sum_y = y
44     count = 1
45
46 # print last cluster's centroids
47 if current_centroid == centroid_index and count != 0:
48     print(str(sum_x / count) + ", " + str(sum_y / count))
49
50 if __name__ == "__main__":
51     calculateNewCentroids()

```

### 3.4 Plot Representation

Last but not least, we wanted to visualize the implementation of k-means in hadoop so as to ensure that our results are the correct ones.

For this purpose, we created the following python script named **printer.py** which generates the images shown bellow, starting from the first generated plot.

```

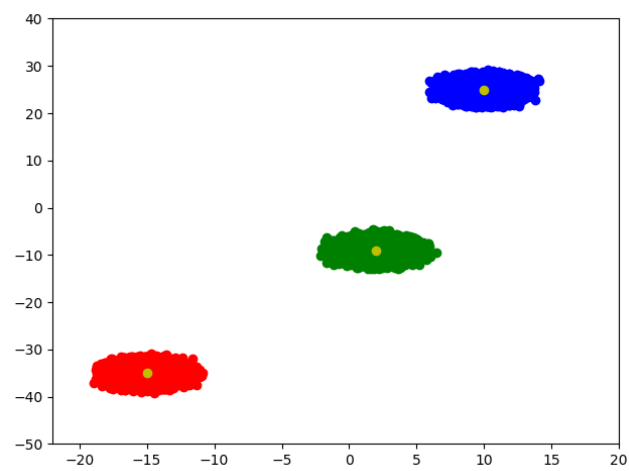
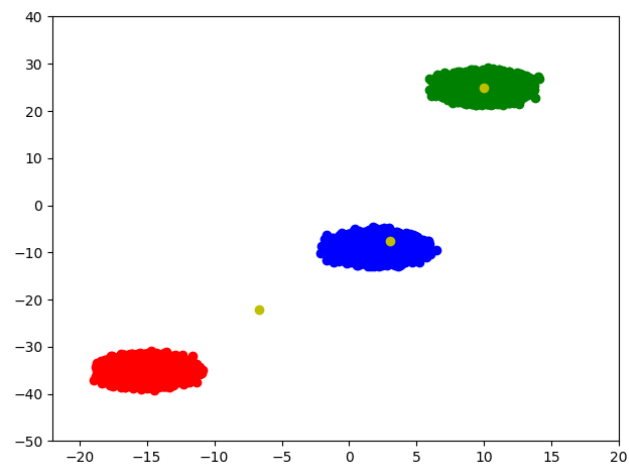
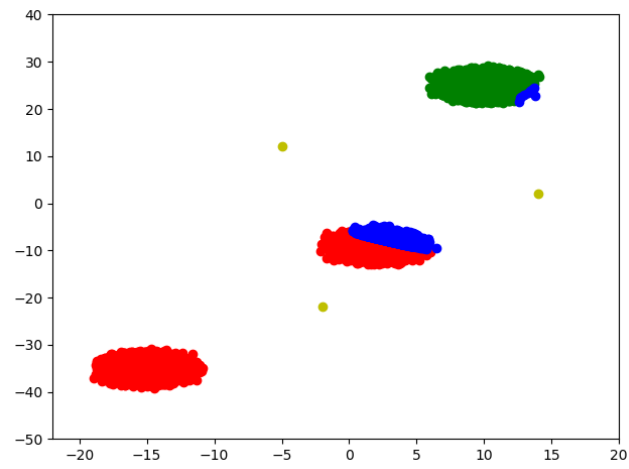
1 import matplotlib.pyplot as plt
2 from scipy.spatial import distance
3
4 centroids=[]
5
6 # initiliaze list for points
7 x=[[[]],[[]],[[]],[[]],[[]],[[]]]
8 # initiliaze list for centroids
9 M=[[[]],[[]]]
10
11 # import centroids.txt file
12 filepath = 'centroids.txt'
13 with open(filepath) as fp:
14     # read line by line

```

```

15     line = fp.readline()
16     while line:
17         if line:
18             # delete blanks for each lines
19             line = line.strip()
20             # extract centroids coordinates
21             cord = line.split(',')
22             centroids.append((float(cord[0]), float(cord[1])))
23             M[0].append((float(cord[0])))
24             M[1].append((float(cord[1])))
25         else:
26             break
27     line = fp.readline()
28
29 fp.close()
30
31 # import dataset.txt file
32 filepath = 'dataset.txt'
33 with open(filepath) as fp:
34     # read line by line
35     line = fp.readline()
36     while line:
37         if line:
38             # delete blanks for each lines
39             line = line.strip()
40             # extract points coordinates
41             cord = line.split(',')
42             x = (float(cord[0]), float(cord[1]))
43             # implement k means loop
44             # find the nearest centroid
45             dist = 1000000000000000
46             selected_m = -1
47             for m in centroids:
48                 test_distance = distance.euclidean(x, m)
49                 if test_distance < dist:
50                     dist = test_distance
51                     selected_m = centroids.index(m)
52
53                 X[selected_m][0].append(x[0])
54                 X[selected_m][1].append(x[1])
55             else:
56                 break
57     line = fp.readline()
58
59 fp.close()
60
61 # print cluster 0 with red color
62 plt.plot(X[0][0], X[0][1], 'ro')
63 # print cluster 1 with green color
64 plt.plot(X[1][0], X[1][1], 'go')
65 # print cluster 2 with blue color
66 plt.plot(X[2][0], X[2][1], 'bo')
67 # print centroids with yellow color
68 plt.plot(M[0], M[1], 'yo')
69 plt.axis([-22, 20, -50, 40])
70 plt.show()

```



# References

- [1] Scipy documentation. <https://docs.scipy.org/doc/scipy-1.2.0/reference/generated/scipy.stats.skewnorm.html>
- [2] Github of scipy library. [https://github.com/scipy/scipy/blob/v1.2.1/scipy/stats/\\_continuous\\_distns.py](https://github.com/scipy/scipy/blob/v1.2.1/scipy/stats/_continuous_distns.py)
- [3] Circle- choose a random point from a circle.  
<https://stackoverflow.com/questions/14096138/find-the-point-on-a-circle-with-given-center-point-radius-and-degree>
- [4] Max Bodoia *MapReduce Algorithms for k-means Clustering*. [https://stanford.edu/~rezab/classes/cme323/S16/projects\\_reports/bodoia.pdf?fbclid=IwAR1mhpyJotsJjK39AGCq0ttm-woRy-l-T9VK7Ssmib7oZduxju7-lqkni3k/](https://stanford.edu/~rezab/classes/cme323/S16/projects_reports/bodoia.pdf?fbclid=IwAR1mhpyJotsJjK39AGCq0ttm-woRy-l-T9VK7Ssmib7oZduxju7-lqkni3k/)
- [5] MapReduce in Python. <https://www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python/>
- [6] MapReduce in Python. <http://www.quuxlabs.com/tutorials/writing-an-hadoop-mapreduce-program-in-python/>
- [7] MapReduce in Python. <https://blog.matthewrathbone.com/2013/11/17/python-map-reduce-on-hadoop-a-beginners-tutorial.html>
- [8] Weizhong Zhao, Huifang Ma and Qing He.  
*Parallel K-Means Clustering Based on MapReduce* [https://www.cs.ucsb.edu/~veronika/MAE/parallelkmeansmapreduce\\_zhao.pdf](https://www.cs.ucsb.edu/~veronika/MAE/parallelkmeansmapreduce_zhao.pdf)
- [9] K-means MapReduce Python Hadoop. [https://github.com/ArsMing276/Kmeans\\_Implementation\\_with\\_Mapreduce](https://github.com/ArsMing276/Kmeans_Implementation_with_Mapreduce)
- [10] Install Python3 on CENT OS 7.  
<https://linuxize.com/post/how-to-install-python-3-on-centos-7/>
- [11] Oracle Virtual Box <https://www.virtualbox.org/wiki/Downloads>
- [12] Cloudera Quickstart VM [https://www.cloudera.com/downloads/quickstart\\_vms/5-13.html](https://www.cloudera.com/downloads/quickstart_vms/5-13.html)