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

Chapter 1

Hadoop Installation

The first task is to install hadoop in your machine. We chose to install Cloudera 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.

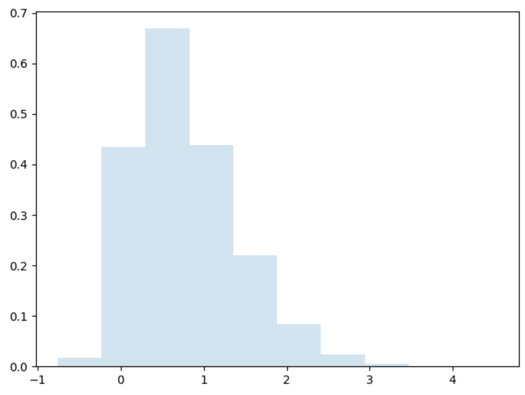
For this purpose we first found distances by using scipy.stats.skewnorm library and then we used these distance to choose random points as shown bellow.

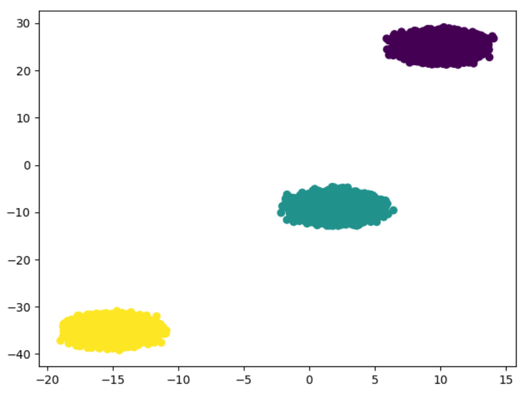
```python
from scipy.stats import skewnorm
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from random import random, uniform
from math import sqrt, pow


# get random points that follows skew distribution and
# use them as distance
def skew():

   fig, ax = plt.subplots(1, 1) # This function creates
          # a figure and a grid of subplots

   a = 5 # skewness parameter, when a = 0 the
         # distribution is identical to a normal distribution

   d = skewnorm.rvs(a, size=350000)

    # histogram of numbers generated
   plt.show()
   ax.hist(d, density=True, histtype='stepfilled', alpha=0.2)

   # return all the numbers generated with skew distribution
   return d


# get random point from a circle that has as center
# given centroid and as radius the distance found in
# skew() function
```

```python
def get_random_point(x, y, cluster):

    array = []
    distance = skew()

    for d in distance:
        while True:
        angle = random() * math.pi * 2
        x1 = x + math.cos (angle ) * d
        y1 = y + math.sin (angle ) * d

        if [x1, y1, cluster] not in array:
            print('yeah')
            break # if the pair of (x1,y1) exist find another pair of (x1,y1)

        array.append([x1, y1, cluster])
        print(x1, y1, cluster, len(array))

    return array


if __name__ == "__main__":

    # pre-define three centers for clusters
    centers = [[10, 25], [2, -9], [-15, -35]]
    data = [] # array with all the final data

    for center in centers:
        # concat the results of these
        # three clusters in one array
        data = data + get_random_point ( center[0], center[1], centers.index(center)) print(len(data))
        print(data)

    plt.scatter([item[0] for item in data], [item[1] for item in data], c=[item[2] for item in data])
    plt.show()

    df = pd.DataFrame(data)
    df.drop(df.columns[[2]], axis=1, inplace=True)
    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 outside 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. As regards, 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. On the other hand, the dataset.txt file is the one that was created in the previous chapter.These six files are necessary for k-means to be implemented and they are needed 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 parse dataset.txt file inside this folder. In order to create a folder in hdfs you can execute *hadoop fs –mkdir /testMapReduce* and to parse 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 cendroid.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

```bash
#!/bin/bash

i=1
while :
do
   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

   rm -f centroids1.txt

   hadoop fs -copyToLocal /testMapReduce/mapreduce-output$i/part-00000 centroids1.txt

   seeiftrue=`python reader.py`
   if [ $seeiftrue = 1 ]
   then
      rm centroids.txt
      hadoop fs -copyToLocal /testMapReduce/mapreduce-output$i/part-00000 centroids.txt
      break
   else
      rm centroids.txt
      hadoop fs -copyToLocal /testMapReduce/mapreduce-output$i/part-00000 centroids.txt
   fi
   i=$((i+1))
done
```

3.2.2 reader.py

```python
__authors__ = "Vaggelis Malandrakis, KLeio Fragkedaki"

from mapper import getCentroids

#check if distance of centroids and centroids1 is less than 1
def checkCentroidsDistance(centroids, centroids1):
   f1x = abs(centroids[0][0] - centroids1[0][0])<1
   f1y = abs(centroids[0][1] - centroids1[0][1])<1
   f2x = abs(centroids[1][0] - centroids1[1][0])<1
   f2y = abs(centroids[1][1] - centroids1[1][1])<1
   f3x = abs(centroids[2][0] - centroids1[2][0])<1
   f3y = abs(centroids[2][1] - centroids1[2][1])<1

   if f1x and f1y and f2x and f2y and f3x and f3y:
      print(1)
   else:
      print(0)

if __name__ == "__main__":
   centroids = getCentroids('centroids.txt')
   centroids1 = getCentroids('centroids1.txt')

   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

A regards mapper, mapper's job is to create the clusters. More specifically, every point of the dataset is matching with one of the centroids that are in the centroid.txt file at the time. So, in the end clusters are generated, which in our case are three in number.

```python
#!/usr/bin/env python
"""mapper.py"""

__authors__ = "Vaggelis Malandrakis, KLeio Fragkedaki"

import sys
from math import sqrt

# get initial centroids from a txt file and add them in an array
def getCentroids(filepath):
   centroids = []

   with open(filepath) as fp:
      line = fp.readline()
      while line:
         if line:
            try:
               line = line.strip()
               cord = line.split(', ')
               # cord[0] is x and cord[1] is y point of a
   centroid
               centroids.append([float(cord[0]), float(cord[1])
   ])
            except:
               break
         else:
            break

      line = fp.readline()

   fp.close()
   return centroids

# create clusters based on initial centroids
def createClusters(centroids):

   #read dataset.txt
   for line in sys.stdin:
      line = line.strip()
      cord = line.split(',')
      min_dist = 100000000000000
      index = -1
```

```python

        for centroid in centroids:
            try:
                cord[0] = float(cord[0])
                cord[1] = float(cord[1])
            except ValueError:
                # float was not a number, so silently
                # ignore/discard this line
                continue

            # euclidian distance from every point of dataset
            # to every centroid
            cur_dist = sqrt(pow(cord[0] - centroid[0], 2) + pow(cord[1] - centroid[1], 2))

            # find the centroid which is closer to the point
            if cur_dist <= min_dist:
                min_dist = cur_dist
                index = centroids.index(centroid)

        var = "%s\t%s\t%s" % (index, cord[0], cord[1])
        print(var)

if __name__ == "__main__":
    centroids = getCentroids('centroids.txt')
    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 adding in order the center of this cluster to be found. So, in the end new centroids of each cluster are generated.

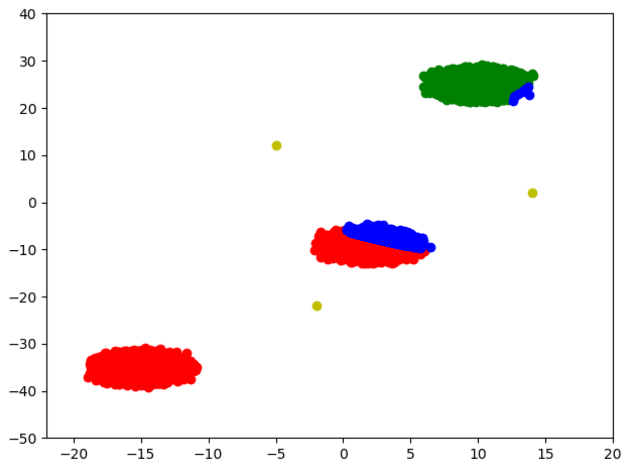
```python
#!/usr/bin/env python
"""reducer.py"""

__authors__ = "Vaggelis Malandrakis, KLeio Fragkedaki"

import sys

def calculateNewCentroids():
    current_centroid = None
    sum_x = 0
    sum_y = 0
    count = 0

    # input comes from STDIN
    for line in sys.stdin:

    # parse the input of mapper.py
    centroid_index, x, y = line.split('\t')

    # convert x and y (currently a string) to float
    try:
        x = float(x)
        y = float(y)
    except ValueError:
        # float was not a number, so silently
        # ignore/discard this line
        continue

    # this IF-switch only works because Hadoop sorts map output
    # by key (here: word) before it is passed to the reducer
    if current_centroid == centroid_index:
        count += 1
        sum_x += x
        sum_y += y
    else:
        if count != 0:
            # print the average of every cluster to get
            # new centroids
            print(str(sum_x / count) + ", " + str(sum_y / count))

        current_centroid = centroid_index
        sum_x = x
        sum_y = y
        count = 1

    # print last cluster's centroids
    if current_centroid == centroid_index and count != 0:
        print(str(sum_x / count) + ", " + str(sum_y / count))

if __name__ == "__main__":
    calculateNewCentroids()
```

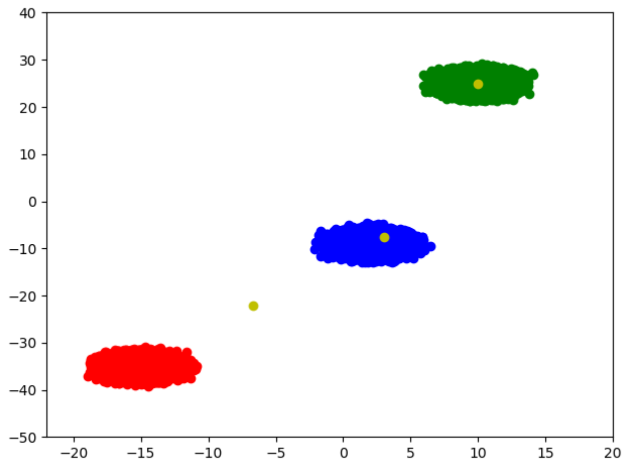
3.4 Plot Representation

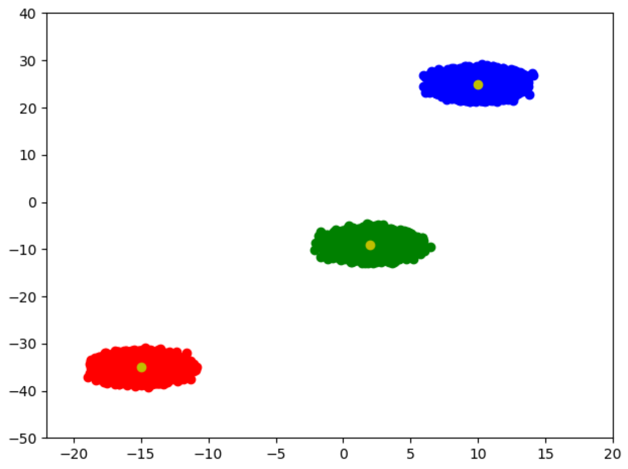
Last but not least, we wanted to visualize the implementation of k-means in hadoop so as to ensure that our results is the correct ones.

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

```python
import matplotlib.pyplot as plt
from scipy.spatial import distance

centroids=[]

# initiliaze list for points
X=[[[],[]],[[],[]],[[],[]]]
# initiliaze list for centroids
M=[[],[]]

# import centroids.txt file
filepath = 'centroids.txt'
with open(filepath) as fp:
    # read line by line
    line = fp.readline()
    while line:
        if line:
            # delete blanks for each lines
            line = line.strip()
            # extract centroids coordinates
            cord = line.split(', ')
            centroids.append((float(cord[0]), float(cord[1])))
            M[0].append((float(cord[0])))
            M[1].append((float(cord[1])))
        else:
            break
    line = fp.readline()

fp.close()

# import dataset.txt file
filepath = 'dataset.txt'
with open(filepath) as fp:
    # read line by line
    line = fp.readline()
    while line:
        if line:
            # delete blanks for each lines
            line = line.strip()
            # extract points coordinates
            cord = line.split(',')
            x = (float(cord[0]), float(cord[1]))
            # implement k means loop
            # find the nearest centroid
            dist = 100000000000000
            selected_m = -1
            for m in centroids:
                test_distance = distance.euclidean(x, m)
                if test_distance < dist:
                    dist = test_distance
                    selected_m = centroids.index(m)

                    X[selected_m][0].append(x[0])
                    X[selected_m][1].append(x[1])
        else:
            break
        line = fp.readline()

fp.close()

# print cluster 0 with red color
plt.plot(X[0][0], X[0][1], 'ro')
# print cluster 1 with green color
plt.plot(X[1][0], X[1][1], 'go')
# print cluster 2 with blue color
plt.plot(X[2][0], X[2][1], 'bo')
# print centroids with yellow color
plt.plot(M[0], M[1], 'yo')
plt.axis([-22, 20, -50, 40])
plt.show()
```


40
30
20
10
0
−10
−20
−30
−40
−50
−20
−15
−10
−5
0
5
10
15
20


40
30
20
10
0
−10
−20
−30
−40
−50
−20
−15
−10
−5
0
5
10
15
20


40
30
20
10
0
−10
−20
−30
−40
−50
−20
−15
−10
−5
0
5
10
15
20

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

[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/

[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

[9] K-means MapReduce Python Hadoop. 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