2.1

import java.io.IOException;

import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.FileUtil;

import org.apache.hadoop.fs.FSDataInputStream;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class KMeans {

public static class KMMapper

extends Mapper<Object, Text, IntWritable, Text>{

private double [][] \_centroids;

private IntWritable cid = new IntWritable();

public void setup(Mapper.Context context){

Configuration conf = context.getConfiguration();

String filename = conf.get("Centroids-file");

\_centroids = loadCentroids(filename, conf);

}

public void map(Object key, Text value, Context context

) throws IOException, InterruptedException {

double [] vec = parseVector(value.toString());

cid.set(closest(vec));

context.write(cid, value);

}

private int closest(double [] v){

double mindist = dist(v, \_centroids[0]);

int label =0;

for (int i=1; i<\_centroids.length; i++){

double t = dist(v, \_centroids[i]);

if (mindist>t){

mindist = t;

label = i;

}

}

return label;

}

}

public static class KMReducer

extends Reducer<IntWritable, Text, IntWritable, Text> {

// write output: cid \t centroid\_vector

private Text result = new Text();

public void reduce(IntWritable key, Iterable<Text> vectors,

Context context

) throws IOException, InterruptedException {

double [] sum = null;

int n=0;

for (Text vec : vectors) {

double [] v = parseVector(vec.toString());

if (sum == null) sum = v;

else

for (int i = 0; i < v.length; i++)

sum[i] += v[i];

n ++;

}

String out = Double.toString(sum[0]/n);

for (int i = 1; i < sum.length; i ++ ){

out += "," + Double.toString(sum[i]/n); // csv output

}

result.set(out);

context.write(key, result);

}

}

// compute square Euclidean distance between two vectors v1 and v2

public static double dist(double [] v1, double [] v2){

double sum=0;

for (int i=0; i< v1.length; i++){

double d = v1[i]-v2[i];

sum += d\*d;

}

return Math.sqrt(sum);

}

// check convergence condition

// max{dist(c1[i], c2[i]), i=1..numClusters < threshold

private static boolean converge(double [][] c1, double [][] c2, double threshold){

// c1 and c2 are two sets of centroids

double maxv = 0;

for (int i=0; i< c1.length; i++){

double d= dist(c1[i], c2[i]);

if (maxv<d)

maxv = d;

}

if (maxv <threshold)

return true;

else

return false;

}

public static double [][] loadCentroids(String filename, Configuration conf){

double [][] centroids=null;

Path p = new Path(filename); // Path is used for opening the file.

try{

FileSystem fs = FileSystem.get(conf);//determines local or HDFS

FSDataInputStream file = fs.open(p);

byte[] bs = new byte[file.available()];

file.read(bs);

file.close();

String [] lines = (new String(bs)).split("\n"); //lines are separated by \n

for (String line:lines)

System.out.println(line);

centroids = new double[lines.length][];

for (int i = 0; i < lines.length; i++){

// cid \t centroid

String [] parts = lines[i].split("\t");

int cid = Integer.parseInt(parts[0]);

centroids[cid] = parseVector(parts[1]);

}

}catch(Exception e){

//log.error(e);

System.out.println(e);

}

return centroids;

}

public static double [] parseVector(String s){

String [] itr = s.split(","); // comma separated

double [] v = new double[itr.length];

for (int i = 0; i < itr.length; i++)

v[i] = Double.parseDouble(itr[i]);

return v;

}

public static void main(String[] args) throws Exception {

// usage: hadoop jar km.jar hdfs://localhost:9000/user/your\_home\_directory/centroids data.hdfs output

double [] [] centroid\_1 = new double [0] [2];

double [] [] centroid\_2 = new double [0] [2];

boolean to\_check=false;

int iteration = 5;

for (int i=0 ;i<iteration;i++){

Configuration conf = new Configuration();

conf.set("Centroids-file", args[0]);

System.out.println(conf.get("Centroids-file"));

Job job = Job.getInstance(conf, "KMeans");

job.setJarByClass(KMeans.class);

job.setMapperClass(KMMapper.class);

//job.setCombinerClass(KMCombiner.class);

job.setReducerClass(KMReducer.class);

job.setOutputKeyClass(IntWritable.class);

job.setOutputValueClass(Text.class);

FileInputFormat.addInputPath(job, new Path(args[1]));

FileOutputFormat.setOutputPath(job, new Path(args[2]));

System.exit(job.waitForCompletion(true) ? 0 : 1);

centroid\_1=KMeans.loadCentroids(args[2]+"/part-r-00000",conf);

centroid\_2=KMeans.loadCentroids(args[0],conf);

to\_check = KMeans.converge(centroid\_1,centroid\_2, 0.002);

}

}

}

2.3

import sys

import numpy as np

import os

def mapper(a,b,centroid\_array):

min\_distance = []

min\_distance = np.append(euclidian\_dist(centroid0[0], centroid0[1], a, b)) //distance between two centroids and data points are caluculated and stored in array

print(np.argmin(min\_distance), "\t", a, b) // min distance of both is calculated and label is assigned

def read\_centroids(fname):

data = None

with open(fname, "r") as fd: // opens file and reads data

data = fd.read()

return data

def split\_centroids(centroids\_raw):

centroids = centroids\_raw.split("\r\n") // splits by line

centroid0 = centroids[0].split("\t")[1].split(",") // splits line into words by delimiter , and puts first element into centroid 0

centroid1 = centroids[1].split("\t")[1].split(",")

// splits line into words by delimiter , and puts second element into centroid 1

return centroid0, centroid1

def euclidian\_dist(centroid\_a, centroid\_b, a, b):

centroid\_a = float(centroid\_a)

centroid0\_b = float(centroid\_b)

a = float(a)

b = float(b)

return ((centroid\_a - a) \*\* 2 + (centroid\_b - b) \*\* 2) \*\* 0.5 //distance between centroids and data is caluculated and returned

if \_\_name\_\_ == "\_\_main\_\_":

for line in sys.stdin:

data1, data2 = line.split("\t")[1].split("\n")[0].split(",") // data file is splitted by lines and words and first word assigned to data1 and second word to data2

centroid0, centroid1 = split\_centroids(read\_centroids(sys.argv[1]))

//both split\_centroids and read\_centroids functions are called

centroid\_array=[]

centroid\_array=np.append(centroid0,centroid1) resultant centroid0,centroid1 is stored in centroid\_array

mapper(data1,data2,centroid\_array) // mapper function is called

3.1

//In given purchase file we need to find total sales of each seller so I loaded file into purchaseRDD then splited by \t (since dataset is seperated by \t) and extracted seller and sales column into sale and then used reduceByKey to get total sales of each seller"

val purchaseRDD =sc.textFile("/cloud/purchase") // loading file

val sale = purchaseRDD.map(purchase =>{(purchase.split("\t")(3),purchase.split(4).toInt)})

//sale.take(10).foreach(println)

val result = sale.reduceByKey(\_+\_)

result.take(10).foreach(println)

3.2

//Executed code in Scala

spark-shell

/\*

creating book table by loading file saved on hdfs

\*/

val bookRDD = sc.textFile("/cloud/greesh/book") // loading text file

val bookDF = bookRDD.map(book => { (book.split("\t")(0),book.split("\t")(1))}).toDF("isbn","name") // loading text in file into spark Data frame

bookDF.show() // displays file in form of data frame

bookDF.registerTempTable("book") //creating temporary table book

sqlContext.sql("select \* from book").show()

/\*

creating purchase table by loading file saved on hdfs

\*/

val purchaseRDD = sc.textFile("/cloud/greesh/purchase") // loading text file

val purchaseDF = purchaseRDD.map(purchase => { (purchase.split("\t")(0).toInt,book.split("\t")(1),purchase.split("\t")(2),purchase.split("\t")(3).purchase.split("\t")(4).toInt)}).toDF("year","cid","isbn","seller","price")

// loading text in file into spark Data frame

purchaseDF.show()

purchaseDF.registerTempTable("purchase") //creating temporary table book

sqlContext.sql("select \* from purchase").show()

val seller =sqlContext.sql("select isbn,cid,seller,price from purchase where seller = 'Amazon' order by isbn") //filtering out other sellers as we need only books sold by amazon

seller.registerTempTable("seller")

sqlContext.sql("select seller from seller").show()

val price =sqlContext.sql("select isbn,min(price) as price from purchase group by isbn order by price") //getting lowstest price for each isbn

price.registerTempTable("price")

sqlContext.sql("select price from price").show()

val lp = sqlContext.sql("select p.isbn,s.seller,p.price from price p left join seller s on p.price=s.price where seller is not null) // getting isbn of book's sold by amazon for lowest price

lp.registerTempTable("lp")

val eliminate=sqlContext.sql("select isbn,seller,price from purchase where seller !='Amazon' order by isbn) // we have both amazon and borders selling b1 for 90 so to eliminate b1 from list ,adding other seller's who sold books for lowest price list in a table

eliminate.registerTempTable("eliminate")

val equalremoving=sqlContext.sql("select lp.isbn,lp.seller,lp.price from lp lp left outer join eliminate e on e.price =lp.price where e.isbn is null) // removed b1 from the final list

equalremoving.registerTempTable("equalremoving")

val bookname=sqlContext.sql("select distinct b.name,b.isbn from book b join equalremoving e on e.isbn=b.isbn") // getting book details from table book

bookname.registerTempTable("bookname")

sqlContext.sql("select \* from bookname").show() // displaying final books that amazon sold for lowest price compared to other sellers.