|  |  |
| --- | --- |
| Lab 5 – Apache Spark | Name: Shuyue JiaID: 56846018 |

1. After the previous labs, you have learned how to implement k-means using Python.
2. In this lab, you are asked to implement k-means using Scala commonly found in Apache Spark.
3. You are given the k-means template file “kmeans.scala”.
4. In the file, there are several spark functions to be implemented by yourself.
5. The following table summarizes the meanings of those functions.

|  |  |
| --- | --- |
| **Function** | **Description** |
| def distance(p:Vector[Double], q:Vector[Double]) : Double | It calculates the distance between two points “p” and “q”. |
| def clostestpoint(q: Vector[Double], candidates: Array[Vector[Double]]): Vector[Double] | Given a query point “q”, it finds the nearest point among “candidates”. |
| def add\_vec(v1: Vector[Double], v2: Vector[Double]): Vector[Double] = | It performs the addition of two points “v1” and “v2”. |
| def average(cluster: Iterable[Vector[Double]]): Vector[Double] | It finds the centroid of “cluster”. |

Answer:

**def** **distance**(p**:**Vector[**Double**], q**:**Vector[**Double**])**:** **Double** **=** {

// var dist = math.sqrt(p.zip(q).map(pair => math.pow((pair.\_1 - pair.\_2), 2)).reduce(\_ + \_));

// return dist

**var** dist **=** p.zip(q).map(pair **=>** distance(pair.\_1, pair.\_2)).reduce(\_ **+** \_);

dist **=** dist **/** p.length

**return** dist

}

**def** **clostestpoint**(q: Vector[**Double**], candidates: Array[Vector[**Double**]])**:** Vector[**Double**] **=** {

Vector[**Double**] **=** {

**var** bestIndex **=** **0**

**var** closest **=** **Double**.PositiveInfinity

**for** (i **<0** until candidates.length) {

**val** tempDist **=** distance(q, candidates(i))

**if** (tempDist **<** closest) {

closest **=** tempDist

bestIndex **=** i

}

}

**return** candidates(bestIndex)

}

}

**def** **add\_vec**(v1: Vector[**Double**], v2: Vector[**Double**])**:** Vector[**Double**] **=** {

**var** newVector **=** v1.zip(v2).map(pair **=>** (pair.\_1 **+** pair.\_2));

**return** newVector

}

**def** **average**(cluster: Iterable[Vector[**Double**]])**:** Vector[**Double**] **=** {

**val** numVectors **=** cluster.size

**var** out **=** Vector(**0.0**, **0.0**)

**var** it **=** cluster.toIterator

**while** (it.hasNext) {

out **=** add\_vec(out, it.next())

}

**var** ret **=** out.map(x **=>** (x **/** numVectors))

**return** ret

}

////////////////////////

// Main Program Below //

////////////////////////

**var** lines **=** sc.textFile("clustering\_dataset.txt");

**var** data **=** lines.map (l **=>** Vector.empty **++** l.split (**'\t'**).map(\_.toDouble))

**var** k **=** **3**

**var** centers **=** data.takeSample(**false**, k, **99**)

**var** centersBefore**=** centers

**var** minDist **=** **1e-6**

**var** d **=** **1** **+** minDist

**do**{

**var** closest **=** data.map(p **=>** (closestpoint(p, centers), p))

**var** pointsgroup **=** closest.groupByKey()

**var** newCenters **=** pointsgroup.mapValues(ps **=>** average(ps))

**var** editedCenters **=** newCenters.values

d **=** distance(centers, editedCenters.collect)

centers **=** editedCenters.collect

println("Distance = ")

println(d)

} **while** (d **>** minDist)

println ("Initial cluster centers")

centersBefore.foreach(println)

print ("Centers after K-means cluster")

centers.foreach(println)

System.exit(**0**)

1. After that, you are asked to use the functions to implement k-means using Scala.
2. Once you have finished the above coding, you are asked to code and run your k-means on the given data file “clustering\_dataset.txt” with k=3.
3. Please report the 3 cluster centroids you have found in the below table.

|  |  |
| --- | --- |
|  | **Data Values** |
| Centroid 1 | Vector (12.972570894104999, 19.430958569806002) |
| Centroid 2 | Vector (6.676557700004402, 13.166919895269002) |
| Centroid 3 | Vector (6.8945984549098975, 2.11317553301257) |

1. Please upload your “kmeans.scala” to the submission system.
2. Please also upload this sheet with your answers to the submission system.
3. This is the end.