**K-Means clustering in Big Data**

DESCRIPTION

**Description**

This project is based on the k-means clustering which is a type of unsupervised learning algorithm. The goal of this algorithm is to find groups in the data, with the number of groups represented by the variable K. The algorithm works iteratively to assign each data point to one of the K groups based on the features that are provided and data points are clustered based on feature similarity.

**Background of Problem Statement**

LoudAcre Mobile is a mobile phone service provider which has introduced a new open network campaign. As a part of this campaign, the company has invited users to raise a request to initiate a complaint about the towers in their locality, if they face issues with their mobile network. LoudAcre has collected the dataset of users who had raised the complaint.

**Domain**: Telecommunication

**Dataset Description**

 The data fields are as follows:

|  |  |
| --- | --- |
| 1. | Date: Format (YYYY:MM:DD:HH: MM: SS) |
| 2. | Area name |
| 3. | Complaint ID |
| 4. | Latitude |
| 5. | Longitude |

**Analysis to be done:**

The fourth and the fifth field of the dataset has latitude and longitude of users which is an important information for the company. You have to find this information of latitude and longitude on the basis of available dataset and create three clusters of users with a k-means algorithm. This will help Loudacre maximize the coverage for its users.

import scala.math.pow

def ds(p1: (Double,Double), p2: (Double,Double)) = {

pow(p1.\_1 - p2.\_1,2) + pow(p1.\_2 - p2.\_2,2 )

}

def ap(p1: (Double,Double), p2: (Double,Double)) = {

(p1.\_1 + p2.\_1, p1.\_2 + p2.\_2)

}

def clsPt(p: (Double,Double), longlat: Array[(Double,Double)]): Int = {

var index = 0

var bestIndex = 0

var closest = Double.PositiveInfinity

for (i <- 0 until longlat.length) {

val dist = ds(p,longlat(i))

if (dist < closest) {

closest = dist

bestIndex = i

}

}

bestIndex

}

val K = 3

val convergeDist = .1

val longlat = sc.textFile("/user/brishirevanth\_gmail/Project\_2\_Dataset.txt").

map(line => line.split(',')).

map(fields => (fields(3).toDouble,fields(4).toDouble)).

filter(point => !((point.\_1 == 0) && (point.\_2 == 0))).

persist()

val kPoints = longlat.takeSample(false, K, 34)

println("Starting K longlat:")

kPoints.foreach(println)

var tempDist = Double.PositiveInfinity

while (tempDist > convergeDist) {

val closest = longlat.map(p => (clsPt(p, kPoints), (p, 1)))

val pointStats = closest.reduceByKey{case ((point1,n1),(point2,n2))

=> (ap(point1,point2),n1+n2) }

val newPoints = pointStats.map{case (i,(point,n)) =>

(i,(point.\_1/n,point.\_2/n))}.collectAsMap()

tempDist = 0.0

for (i <- 0 until K) {

tempDist += ds(kPoints(i),newPoints(i))

}

println("Distance between iterations: "+tempDist)

for (i <- 0 until K) {

kPoints(i) = newPoints(i)

}

}

println("Final K longlat: " )

kPoints.foreach(println)