# [SRC] 09.1. Cluster Users by Followers and Friends

val tweets = spark.read.parquet("/dataset/twitter/reference")

import org.apache.spark.mllib.linalg.Vectors

val data = tweets

.select("user.friendsCount", "user.followersCount")

.rdd

.map{r =>

Vectors.dense(r.getAs[Long]("friendsCount").toDouble, r.getAs[Long]("followersCount").toDouble)

}

import org.apache.spark.mllib.clustering.KMeans

val model = KMeans.train(data = data, k = 4, maxIterations = 20)

val points = data.collect.map(d => (d, model.predict(d)))

println("%table x\ty\tcluster")

points.foreach(r=>

println(Seq(r.\_1(0), r.\_1(1), r.\_2).mkString("\t"))

)

val pyData = points.map(r => Seq(r.\_1(0), r.\_1(1), r.\_2).toArray)

z.put("pyData", pyData)

%pyspark

data = z.get("pyData")

import io

import matplotlib

import matplotlib.pyplot as plt

matplotlib.use('Agg')

def show(p):

img = io.StringIO()

p.savefig(img, format='svg')

img.seek(0)

print("%html <div style='width:600px'>" + img.getvalue() + "</div>")

plt.clf()

x = []

y = []

k = []

for i in range(0, len(data)):

x.append(data[i][0])

y.append(data[i][1])

k.append(data[i][2])

plt.scatter(x,y, c=k, alpha=0.5)

show(plt)

val WSSSE = model.computeCost(data)

println("Within Set Sum of Squared Errors = " + WSSSE)

val numIterations = 20

val costs = scala.collection.mutable.ArrayBuffer.empty[Double]

for (numClusters <- 2 to 5) {

val clusters = KMeans

.train(data, numClusters, numIterations)

costs += clusters.computeCost(data)

}

println("%table x\ty")

costs

.zipWithIndex

.foreach(r => {

println(Seq(r.\_2 + 2, r.\_1).mkString("\t"))

})

# [SRC] 09.2. Cluster Users By Location

val tweets = spark.read.parquet("/dataset/twitter/reference")

import org.apache.spark.ml.linalg.SQLDataTypes.VectorType

import org.apache.spark.sql.Row

import org.apache.spark.ml.linalg.{Vectors => MLVectors}

import org.apache.spark.sql.types.{StructField, StructType}

val rdd = tweets

.select("geoLocation.longitude", "geoLocation.latitude")

.rdd

.map{r =>

Row(MLVectors.dense(r.getAs[Double]("longitude"), r.getAs[Double]("latitude")))

}

val schema = StructType(Array(StructField("features", VectorType, false)))

val data = spark.createDataFrame(rdd, schema)

data.show

import org.apache.spark.ml.clustering.{KMeans => MLKmeans}

val kmeans = new MLKmeans()

.setK(4)

.setMaxIter(20)

val model = kmeans.fit(data)

val fitted = model.transform(data)

fitted.show

model.clusterCenters.foreach(println)

import org.apache.spark.ml.linalg.Vector

println("%table x\ty\tcluster")

fitted.collect.foreach(r => {

val v = r.getAs[Vector]("features")

println(Seq(v(0), v(1), r.getAs[Int]("prediction")).mkString("\t"))

})

val pyData = fitted.map(r => {

val v = r.getAs[Vector]("features")

Seq(v(0), v(1), r.getAs[Int]("prediction")).toArray

}).collect

z.put("pyData", pyData)

%pyspark

data = z.get("pyData")

import matplotlib

import io

import matplotlib.pyplot as plt

matplotlib.use('Agg')

def show(p):

img = io.StringIO()

p.savefig(img, format='svg')

img.seek(0)

print("%html <div style='width:600px'>" + img.getvalue() + "</div>")

plt.clf()

x = []

y = []

k = []

for i in range(0, len(data)):

x.append(data[i][0])

y.append(data[i][1])

k.append(data[i][2])

plt.scatter(x,y, c=k, alpha=0.5)

show(plt)

val numIterations = 20

val costs = scala.collection.mutable.ArrayBuffer.empty[Double]

for (numClusters <- 2 to 5) {

val kmeans = new MLKmeans()

.setK(numClusters)

.setMaxIter(numIterations)

val model = kmeans.fit(data)

costs += model.computeCost(data)

}

println("%table x\ty")

costs

.zipWithIndex

.foreach(r => {

println(Seq(r.\_2 + 2, r.\_1).mkString("\t"))

})

# [SRC] 09.3. Run KMeans on a Stream

System.setProperty("twitter4j.oauth.consumerKey", "CJ63hFzuu3KLovHt8LmFMw")

System.setProperty("twitter4j.oauth.consumerSecret", "2W373bWM1Oca9zyDk38qeUoVamdJ78ZMfexq6mQsLb8")

System.setProperty("twitter4j.oauth.accessToken", "101229446-5qSO2hRsVfjO3nhopQGFT4FjPpb2XmBGDAHkxBg")

System.setProperty("twitter4j.oauth.accessTokenSecret", "D7TDqjm1TBpjF1Gs6OmIhWDC13WQJrLM86SoSznH8SQ")

import org.apache.spark.mllib.clustering.StreamingKMeans

import org.apache.spark.mllib.feature.StandardScaler

import org.apache.spark.mllib.linalg.{Vector, Vectors}

import org.apache.spark.streaming.twitter.TwitterUtils

import org.apache.spark.streaming.{Seconds, StreamingContext}

import twitter4j.Status

val ssc = new StreamingContext(sc, Seconds(5))

val stream = TwitterUtils.createStream(ssc, None)

def printCenters(kmeans: StreamingKMeans) = {

val centers = kmeans.latestModel.clusterCenters

println("centers:")

centers.map(println)

}

val model = new StreamingKMeans()

.setK(3)

.setHalfLife(5, "batches")

.setRandomCenters(2, 0.0)

val data = stream.filter(s => (

s.isRetweet

))

data.foreachRDD { rdd =>

if (rdd.count > 0) {

def featurize(status: Status): Vector = {

val s = status.getRetweetedStatus

Vectors.dense(

s.getRetweetCount.toDouble,

s.getUser.getFollowersCount.toDouble

)

}

val f = rdd.map(featurize)

val scaledData = new StandardScaler(false, true)

.fit(f)

.transform(f)

model

.latestModel

.update(scaledData, model.decayFactor, model.timeUnit)

val fitted = model.latestModel.predict(scaledData).collect

printCenters(model)

}

}

import java.util.concurrent.TimeUnit

ssc.start()

ssc.awaitTerminationOrTimeout(TimeUnit.SECONDS.toMillis(10))

printCenters(model)