package com.twitter.simclusters\_v2.common.clustering

import com.twitter.sbf.graph.ConnectedComponents

import com.twitter.sbf.graph.Graph

import com.twitter.util.Stopwatch

import it.unimi.dsi.fastutil.ints.IntSet

import scala.collection.SortedMap

import scala.jdk.CollectionConverters.\_

/\*\*

\* Aggregate entities into clusters such that a cluster contains all embeddings with a similarity

\* above a configurable threshold to any other embedding.

\*

\* @param similarityThreshold: When building the edges between entities, edges with weight

\* less than or equal to this threshold will be filtered out.

\*/

class ConnectedComponentsClusteringMethod(

similarityThreshold: Double)

extends ClusteringMethod {

import ClusteringStatistics.\_

def cluster[T](

embeddings: Map[Long, T],

similarityFn: (T, T) => Double,

recordStatCallback: (String, Long) => Unit = (\_, \_) => ()

): Set[Set[Long]] = {

val timeSinceGraphBuildStart = Stopwatch.start()

// com.twitter.sbf.graph.Graph expects neighbors to be sorted in ascending order.

val sourcesById = SortedMap(embeddings.zipWithIndex.map {

case (source, idx) => idx -> source

}.toSeq: \_\*)

val neighbours = sourcesById.map {

case (srcIdx, (\_, src)) =>

sourcesById

.collect {

case (dstIdx, (\_, dst)) if srcIdx != dstIdx => // avoid self-edges

val similarity = similarityFn(src, dst)

recordStatCallback(

StatComputedSimilarityBeforeFilter,

(similarity \* 100).toLong // preserve up to two decimal points

)

if (similarity > similarityThreshold)

Some(dstIdx)

else None

}.flatten.toArray

}.toArray

recordStatCallback(StatSimilarityGraphTotalBuildTime, timeSinceGraphBuildStart().inMilliseconds)

val timeSinceClusteringAlgRunStart = Stopwatch.start()

val nEdges = neighbours.map(\_.length).sum / 2 // Graph expects count of undirected edges

val graph = new Graph(sourcesById.size, nEdges, neighbours)

val clusters = ConnectedComponents

.connectedComponents(graph).asScala.toSet

.map { i: IntSet => i.asScala.map(sourcesById(\_).\_1).toSet }

recordStatCallback(

StatClusteringAlgorithmRunTime,

timeSinceClusteringAlgRunStart().inMilliseconds)

clusters

}

}