package com.twitter.interaction\_graph.scio.agg\_all

import collection.JavaConverters.\_

import com.spotify.scio.values.SCollection

import com.twitter.algebird.mutable.PriorityQueueMonoid

import com.twitter.interaction\_graph.scio.common.GraphUtil

import com.twitter.interaction\_graph.thriftscala.Edge

import com.twitter.scalding\_internal.multiformat.format.keyval.KeyVal

import com.twitter.timelines.real\_graph.thriftscala.RealGraphFeatures

import com.twitter.timelines.real\_graph.thriftscala.RealGraphFeaturesTest

import com.twitter.timelines.real\_graph.v1.thriftscala.{RealGraphFeatures => RealGraphFeaturesV1}

import com.twitter.user\_session\_store.thriftscala.UserSession

import com.twitter.interaction\_graph.scio.common.ConversionUtil.\_

object InteractionGraphAggregationTransform {

val ordering: Ordering[Edge] = Ordering.by(-\_.weight.getOrElse(0.0))

// converts our Edge thrift into timelines' thrift

def getTopKTimelineFeatures(

scoredAggregatedEdge: SCollection[Edge],

maxDestinationIds: Int

): SCollection[KeyVal[Long, UserSession]] = {

scoredAggregatedEdge

.filter(\_.weight.exists(\_ > 0))

.keyBy(\_.sourceId)

.groupByKey

.map {

case (sourceId, edges) =>

val (inEdges, outEdges) = edges.partition(GraphUtil.isFollow)

val inTopK =

if (inEdges.isEmpty) Nil

else {

val inTopKQueue =

new PriorityQueueMonoid[Edge](maxDestinationIds)(ordering)

inTopKQueue

.build(inEdges).iterator().asScala.toList.flatMap(

toRealGraphEdgeFeatures(hasTimelinesRequiredFeatures))

}

val outTopK =

if (outEdges.isEmpty) Nil

else {

val outTopKQueue =

new PriorityQueueMonoid[Edge](maxDestinationIds)(ordering)

outTopKQueue

.build(outEdges).iterator().asScala.toList.flatMap(

toRealGraphEdgeFeatures(hasTimelinesRequiredFeatures))

}

KeyVal(

sourceId,

UserSession(

userId = Some(sourceId),

realGraphFeatures = Some(RealGraphFeatures.V1(RealGraphFeaturesV1(inTopK, outTopK))),

realGraphFeaturesTest =

Some(RealGraphFeaturesTest.V1(RealGraphFeaturesV1(inTopK, outTopK)))

)

)

}

}

}