package com.twitter.recos.user\_tweet\_graph.relatedTweetHandlers

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.graphjet.bipartite.api.BipartiteGraph

import com.twitter.recos.user\_video\_graph.thriftscala.\_

import com.twitter.recos.user\_video\_graph.util.FetchRHSTweetsUtil

import com.twitter.recos.user\_video\_graph.util.FilterUtil

import com.twitter.recos.user\_video\_graph.util.GetRelatedTweetCandidatesUtil

import com.twitter.recos.util.Stats.\_

import com.twitter.servo.request.\_

import com.twitter.util.Duration

import com.twitter.util.Future

import scala.concurrent.duration.HOURS

/\*\*

\* Implementation of the Thrift-defined service interface for consumersTweetBasedRelatedTweets.

\* given a list of consumer userIds, find the tweets they co-engaged with (we're treating input userIds as consumers therefore "consumersTweetBasedRelatedTweets" )

\* example use case: given a list of user's contacts in their address book, find tweets those contacts engaged with

\*/

class ConsumersBasedRelatedTweetsHandler(

bipartiteGraph: BipartiteGraph,

statsReceiver: StatsReceiver)

extends RequestHandler[ConsumersBasedRelatedTweetRequest, RelatedTweetResponse] {

private val stats = statsReceiver.scope(this.getClass.getSimpleName)

override def apply(request: ConsumersBasedRelatedTweetRequest): Future[RelatedTweetResponse] = {

trackFutureBlockStats(stats) {

val maxResults = request.maxResults.getOrElse(200)

val minScore = request.minScore.getOrElse(0.0)

val maxTweetAge = request.maxTweetAgeInHours.getOrElse(48)

val minResultDegree = request.minResultDegree.getOrElse(50)

val minCooccurrence = request.minCooccurrence.getOrElse(3)

val excludeTweetIds = request.excludeTweetIds.getOrElse(Seq.empty).toSet

val consumerSeedSet = request.consumerSeedSet.distinct.filter { userId =>

val userDegree = bipartiteGraph.getLeftNodeDegree(userId)

// constrain to users that have <100 engagements to avoid spammy behavior

userDegree < 100

}

val rhsTweetIds = FetchRHSTweetsUtil.fetchRHSTweets(

consumerSeedSet,

bipartiteGraph

)

val scorePreFactor = 1000.0 / consumerSeedSet.size

val relatedTweetCandidates = GetRelatedTweetCandidatesUtil.getRelatedTweetCandidates(

rhsTweetIds,

minCooccurrence,

minResultDegree,

scorePreFactor,

bipartiteGraph)

val relatedTweets = relatedTweetCandidates

.filter(relatedTweet =>

FilterUtil.tweetAgeFilter(

relatedTweet.tweetId,

Duration(maxTweetAge, HOURS)) && (relatedTweet.score > minScore) && (!excludeTweetIds

.contains(relatedTweet.tweetId))).take(maxResults)

stats.stat("response\_size").add(relatedTweets.size)

Future.value(RelatedTweetResponse(tweets = relatedTweets))

}

}

}