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.features.tweet.thriftscala.GraphFeaturesForQuery

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

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

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

import com.twitter.recos.user\_tweet\_graph.util.GetAllInternalTweetIdsUtil

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

import com.twitter.recos.user\_tweet\_graph.util.SampleLHSUsersUtil

import com.twitter.recos.util.Action

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 tweetBasedRelatedTweets.

\*

\*/

class TweetBasedRelatedTweetsHandler(bipartiteGraph: BipartiteGraph, statsReceiver: StatsReceiver)

extends RequestHandler[TweetBasedRelatedTweetRequest, RelatedTweetResponse] {

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

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

trackFutureBlockStats(stats) {

val internalQueryTweetIds =

GetAllInternalTweetIdsUtil.getAllInternalTweetIds(request.tweetId, bipartiteGraph)

val response = internalQueryTweetIds match {

case head +: Nil => getRelatedTweets(request, head)

case \_ => RelatedTweetResponse()

}

Future.value(response)

}

}

private def getRelatedTweets(

request: TweetBasedRelatedTweetRequest,

maskedTweetId: Long

): RelatedTweetResponse = {

val maxNumSamplesPerNeighbor = request.maxNumSamplesPerNeighbor.getOrElse(100)

val maxResults = request.maxResults.getOrElse(200)

val minScore = request.minScore.getOrElse(0.5)

val maxTweetAge = request.maxTweetAgeInHours.getOrElse(48)

val minResultDegree = request.minResultDegree.getOrElse(50)

val minQueryDegree = request.minQueryDegree.getOrElse(10)

val minCooccurrence = request.minCooccurrence.getOrElse(3)

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

val queryTweetDegree = bipartiteGraph.getRightNodeDegree(maskedTweetId)

stats.stat("queryTweetDegree").add(queryTweetDegree)

if (queryTweetDegree < minQueryDegree) {

stats.counter("queryTweetDegreeLessThanMinQueryDegree").incr()

RelatedTweetResponse()

} else {

val sampledLHSuserIds =

SampleLHSUsersUtil.sampleLHSUsers(maskedTweetId, maxNumSamplesPerNeighbor, bipartiteGraph)

val rHStweetIds = FetchRHSTweetsUtil.fetchRHSTweets(

sampledLHSuserIds,

bipartiteGraph,

Set(Action.Favorite, Action.Retweet)

)

val scorePreFactor =

queryTweetDegree / math.log(queryTweetDegree) / sampledLHSuserIds.distinct.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)

RelatedTweetResponse(

tweets = relatedTweets,

queryTweetGraphFeatures = Some(GraphFeaturesForQuery(degree = Some(queryTweetDegree))))

}

}

}