package com.twitter.follow\_recommendations.common.candidate\_sources.sims\_expansion

import com.google.inject.Singleton

import com.twitter.follow\_recommendations.common.candidate\_sources.sims.SwitchingSimsSource

import com.twitter.follow\_recommendations.common.models.CandidateUser

import com.twitter.follow\_recommendations.common.models.HasRecentFollowedUserIds

import com.twitter.hermit.model.Algorithm

import com.twitter.product\_mixer.core.model.common.identifier.CandidateSourceIdentifier

import com.twitter.stitch.Stitch

import com.twitter.timelines.configapi.HasParams

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.follow\_recommendations.common.clients.socialgraph.SocialGraphClient

import com.twitter.product\_mixer.core.model.marshalling.request.HasClientContext

import javax.inject.Inject

object RecentFollowingSimilarUsersSource {

val Identifier = CandidateSourceIdentifier(Algorithm.NewFollowingSimilarUser.toString)

}

@Singleton

class RecentFollowingSimilarUsersSource @Inject() (

socialGraph: SocialGraphClient,

switchingSimsSource: SwitchingSimsSource,

statsReceiver: StatsReceiver)

extends SimsExpansionBasedCandidateSource[

HasParams with HasRecentFollowedUserIds with HasClientContext

](switchingSimsSource) {

val identifier = RecentFollowingSimilarUsersSource.Identifier

private val stats = statsReceiver.scope(identifier.name)

private val maxResultsStats = stats.scope("max\_results")

private val calibratedScoreCounter = stats.counter("calibrated\_scores\_counter")

override def firstDegreeNodes(

request: HasParams with HasRecentFollowedUserIds with HasClientContext

): Stitch[Seq[CandidateUser]] = {

if (request.params(RecentFollowingSimilarUsersParams.TimestampIntegrated)) {

val recentFollowedUserIdsWithTimeStitch =

socialGraph.getRecentFollowedUserIdsWithTime(request.clientContext.userId.get)

recentFollowedUserIdsWithTimeStitch.map { results =>

val first\_degree\_nodes = results

.sortBy(-\_.timeInMs).take(

request.params(RecentFollowingSimilarUsersParams.MaxFirstDegreeNodes))

val max\_timestamp = first\_degree\_nodes.head.timeInMs

first\_degree\_nodes.map {

case userIdWithTime =>

CandidateUser(

userIdWithTime.userId,

score = Some(userIdWithTime.timeInMs.toDouble / max\_timestamp))

}

}

} else {

Stitch.value(

request.recentFollowedUserIds

.getOrElse(Nil).take(

request.params(RecentFollowingSimilarUsersParams.MaxFirstDegreeNodes)).map(

CandidateUser(\_, score = Some(1.0)))

)

}

}

override def maxSecondaryDegreeNodes(

req: HasParams with HasRecentFollowedUserIds with HasClientContext

): Int = {

req.params(RecentFollowingSimilarUsersParams.MaxSecondaryDegreeExpansionPerNode)

}

override def maxResults(

req: HasParams with HasRecentFollowedUserIds with HasClientContext

): Int = {

val firstDegreeNodes = req.params(RecentFollowingSimilarUsersParams.MaxFirstDegreeNodes)

val maxResultsNum = req.params(RecentFollowingSimilarUsersParams.MaxResults)

maxResultsStats

.stat(

s"RecentFollowingSimilarUsersSource\_firstDegreeNodes\_${firstDegreeNodes}\_maxResults\_${maxResultsNum}")

.add(1)

maxResultsNum

}

override def scoreCandidate(sourceScore: Double, similarToScore: Double): Double = {

sourceScore \* similarToScore

}

override def calibrateDivisor(

req: HasParams with HasRecentFollowedUserIds with HasClientContext

): Double = {

req.params(DBV2SimsExpansionParams.RecentFollowingSimilarUsersDBV2CalibrateDivisor)

}

override def calibrateScore(

candidateScore: Double,

req: HasParams with HasRecentFollowedUserIds with HasClientContext

): Double = {

calibratedScoreCounter.incr()

candidateScore / calibrateDivisor(req)

}

}