package com.twitter.follow\_recommendations.common.candidate\_sources.stp

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

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

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.functional\_component.candidate\_source.CandidateSource

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

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

import com.twitter.socialgraph.thriftscala.RelationshipType

import com.twitter.stitch.Stitch

import com.twitter.strato.generated.client.onboarding.userrecs.StrongTiePredictionFeaturesOnUserClientColumn

import javax.inject.Singleton

import javax.inject.Inject

/\*\*

\* Returns mutual follows. It first gets mutual follows from recent 100 follows and followers, and then unions this

\* with mutual follows from STP features dataset.

\*/

@Singleton

class MutualFollowStrongTiePredictionSource @Inject() (

sgsClient: SocialGraphClient,

strongTiePredictionFeaturesOnUserClientColumn: StrongTiePredictionFeaturesOnUserClientColumn)

extends CandidateSource[HasClientContext with HasRecentFollowedUserIds, CandidateUser] {

val identifier: CandidateSourceIdentifier =

MutualFollowStrongTiePredictionSource.Identifier

override def apply(

target: HasClientContext with HasRecentFollowedUserIds

): Stitch[Seq[CandidateUser]] = {

target.getOptionalUserId match {

case Some(userId) =>

val newFollowings = target.recentFollowedUserIds

.getOrElse(Nil)

.take(MutualFollowStrongTiePredictionSource.NumOfRecentFollowings)

val newFollowersStitch =

sgsClient

.getRecentEdges(RecentEdgesQuery(userId, Seq(RelationshipType.FollowedBy))).map(

\_.take(MutualFollowStrongTiePredictionSource.NumOfRecentFollowers))

val mutualFollowsStitch =

strongTiePredictionFeaturesOnUserClientColumn.fetcher

.fetch(userId).map(\_.v.flatMap(\_.topMutualFollows.map(\_.map(\_.userId))).getOrElse(Nil))

Stitch.join(newFollowersStitch, mutualFollowsStitch).map {

case (newFollowers, mutualFollows) => {

(newFollowings.intersect(newFollowers) ++ mutualFollows).distinct

.map(id => CandidateUser(id, Some(CandidateUser.DefaultCandidateScore)))

}

}

case \_ => Stitch.Nil

}

}

}

object MutualFollowStrongTiePredictionSource {

val Identifier: CandidateSourceIdentifier = CandidateSourceIdentifier(

Algorithm.MutualFollowSTP.toString)

val NumOfRecentFollowings = 100

val NumOfRecentFollowers = 100

}