package com.twitter.home\_mixer.product.scored\_tweets.feature\_hydrator

import com.twitter.home\_mixer.model.HomeFeatures.FavoritedByCountFeature

import com.twitter.home\_mixer.model.HomeFeatures.FavoritedByUserIdsFeature

import com.twitter.home\_mixer.model.HomeFeatures.FromInNetworkSourceFeature

import com.twitter.home\_mixer.model.HomeFeatures.InReplyToTweetIdFeature

import com.twitter.home\_mixer.model.HomeFeatures.RealGraphInNetworkScoresFeature

import com.twitter.home\_mixer.model.HomeFeatures.RepliedByCountFeature

import com.twitter.home\_mixer.model.HomeFeatures.RepliedByEngagerIdsFeature

import com.twitter.home\_mixer.model.HomeFeatures.RetweetedByCountFeature

import com.twitter.home\_mixer.model.HomeFeatures.RetweetedByEngagerIdsFeature

import com.twitter.home\_mixer.model.HomeFeatures.SourceTweetIdFeature

import com.twitter.home\_mixer.param.HomeMixerInjectionNames.UtegSocialProofRepository

import com.twitter.product\_mixer.component\_library.model.candidate.TweetCandidate

import com.twitter.product\_mixer.core.feature.Feature

import com.twitter.product\_mixer.core.feature.featuremap.FeatureMap

import com.twitter.product\_mixer.core.feature.featuremap.FeatureMapBuilder

import com.twitter.product\_mixer.core.functional\_component.feature\_hydrator.BulkCandidateFeatureHydrator

import com.twitter.product\_mixer.core.model.common.CandidateWithFeatures

import com.twitter.product\_mixer.core.model.common.Conditionally

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

import com.twitter.product\_mixer.core.pipeline.PipelineQuery

import com.twitter.product\_mixer.core.util.OffloadFuturePools

import com.twitter.recos.recos\_common.{thriftscala => rc}

import com.twitter.recos.user\_tweet\_entity\_graph.{thriftscala => uteg}

import com.twitter.servo.keyvalue.KeyValueResult

import com.twitter.servo.repository.KeyValueRepository

import com.twitter.stitch.Stitch

import javax.inject.Inject

import javax.inject.Named

import javax.inject.Singleton

@Singleton

class UtegFeatureHydrator @Inject() (

@Named(UtegSocialProofRepository) client: KeyValueRepository[

(Seq[Long], (Long, Map[Long, Double])),

Long,

uteg.TweetRecommendation

]) extends BulkCandidateFeatureHydrator[PipelineQuery, TweetCandidate]

with Conditionally[PipelineQuery] {

override val identifier: FeatureHydratorIdentifier = FeatureHydratorIdentifier("Uteg")

override val features: Set[Feature[\_, \_]] = Set(

FavoritedByUserIdsFeature,

RetweetedByEngagerIdsFeature,

RepliedByEngagerIdsFeature,

FavoritedByCountFeature,

RetweetedByCountFeature,

RepliedByCountFeature

)

override def onlyIf(query: PipelineQuery): Boolean = query.features

.exists(\_.getOrElse(RealGraphInNetworkScoresFeature, Map.empty[Long, Double]).nonEmpty)

override def apply(

query: PipelineQuery,

candidates: Seq[CandidateWithFeatures[TweetCandidate]]

): Stitch[Seq[FeatureMap]] = OffloadFuturePools.offloadFuture {

val seedUserWeights = query.features.map(\_.get(RealGraphInNetworkScoresFeature)).get

val sourceTweetIds = candidates.flatMap(\_.features.getOrElse(SourceTweetIdFeature, None))

val inReplyToTweetIds = candidates.flatMap(\_.features.getOrElse(InReplyToTweetIdFeature, None))

val tweetIds = candidates.map(\_.candidate.id)

val tweetIdsToSend = (tweetIds ++ sourceTweetIds ++ inReplyToTweetIds).distinct

val utegQuery = (tweetIdsToSend, (query.getRequiredUserId, seedUserWeights))

client(utegQuery).map(handleResponse(candidates, \_))

}

private def handleResponse(

candidates: Seq[CandidateWithFeatures[TweetCandidate]],

results: KeyValueResult[Long, uteg.TweetRecommendation],

): Seq[FeatureMap] = {

candidates.map { candidate =>

val inNetwork = candidate.features.getOrElse(FromInNetworkSourceFeature, false)

val candidateProof = results(candidate.candidate.id).toOption.flatten

val sourceProof = candidate.features

.getOrElse(SourceTweetIdFeature, None).flatMap(results(\_).toOption.flatten)

val proofs = Seq(candidateProof, sourceProof).flatten.map(\_.socialProofByType)

val favoritedBy = proofs.flatMap(\_.get(rc.SocialProofType.Favorite)).flatten

val retweetedBy = proofs.flatMap(\_.get(rc.SocialProofType.Retweet)).flatten

val repliedBy = proofs.flatMap(\_.get(rc.SocialProofType.Reply)).flatten

val (favoritedByCount, retweetedByCount, repliedByCount) =

if (!inNetwork) {

(favoritedBy.size.toDouble, retweetedBy.size.toDouble, repliedBy.size.toDouble)

} else { (0.0, 0.0, 0.0) }

FeatureMapBuilder()

.add(FavoritedByUserIdsFeature, favoritedBy)

.add(RetweetedByEngagerIdsFeature, retweetedBy)

.add(RepliedByEngagerIdsFeature, repliedBy)

.add(FavoritedByCountFeature, favoritedByCount)

.add(RetweetedByCountFeature, retweetedByCount)

.add(RepliedByCountFeature, repliedByCount)

.build()

}

}

}