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

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.graph\_feature\_service.{thriftscala => gfs}

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

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

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

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

import com.twitter.home\_mixer.util.CandidatesUtil

import com.twitter.home\_mixer.util.ObservedKeyValueResultHandler

import com.twitter.ml.api.DataRecord

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.FeatureWithDefaultOnFailure

import com.twitter.product\_mixer.core.feature.datarecord.DataRecordInAFeature

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.identifier.FeatureHydratorIdentifier

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

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

import com.twitter.servo.repository.KeyValueRepository

import com.twitter.stitch.Stitch

import com.twitter.timelines.prediction.adapters.two\_hop\_features.TwoHopFeaturesAdapter

import com.twitter.util.Try

import javax.inject.Inject

import javax.inject.Named

import javax.inject.Singleton

import scala.collection.JavaConverters.\_

object GraphTwoHopFeature

extends DataRecordInAFeature[TweetCandidate]

with FeatureWithDefaultOnFailure[TweetCandidate, DataRecord] {

override def defaultValue: DataRecord = new DataRecord()

}

@Singleton

class GraphTwoHopFeatureHydrator @Inject() (

@Named(GraphTwoHopRepository) client: KeyValueRepository[(Seq[Long], Long), Long, Seq[

gfs.IntersectionValue

]],

override val statsReceiver: StatsReceiver)

extends BulkCandidateFeatureHydrator[PipelineQuery, TweetCandidate]

with ObservedKeyValueResultHandler {

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

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

override val statScope: String = identifier.toString

private val twoHopFeaturesAdapter = new TwoHopFeaturesAdapter

private val FollowFeatureType = gfs.FeatureType(gfs.EdgeType.Following, gfs.EdgeType.FollowedBy)

override def apply(

query: PipelineQuery,

candidates: Seq[CandidateWithFeatures[TweetCandidate]]

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

// Apply filters to in network candidates for retweets only.

val (inNetworkCandidates, oonCandidates) = candidates.partition { candidate =>

candidate.features.getOrElse(FromInNetworkSourceFeature, false)

}

val inNetworkCandidatesToHydrate =

inNetworkCandidates.filter(\_.features.getOrElse(IsRetweetFeature, false))

val candidatesToHydrate = (inNetworkCandidatesToHydrate ++ oonCandidates)

.flatMap(candidate => CandidatesUtil.getOriginalAuthorId(candidate.features)).distinct

val response = client((candidatesToHydrate, query.getRequiredUserId))

response.map { result =>

candidates.map { candidate =>

val originalAuthorId = CandidatesUtil.getOriginalAuthorId(candidate.features)

val value = observedGet(key = originalAuthorId, keyValueResult = result)

val transformedValue = postTransformer(value)

val followedByUserIds = value.toOption

.flatMap(getFollowedByUserIds(\_))

.getOrElse(Seq.empty)

FeatureMapBuilder()

.add(GraphTwoHopFeature, transformedValue)

.add(FollowedByUserIdsFeature, followedByUserIds)

.build()

}

}

}

private def getFollowedByUserIds(input: Option[Seq[gfs.IntersectionValue]]): Option[Seq[Long]] =

input.map(\_.filter(\_.featureType == FollowFeatureType).flatMap(\_.intersectionIds).flatten)

private def postTransformer(input: Try[Option[Seq[gfs.IntersectionValue]]]): Try[DataRecord] =

input.map(twoHopFeaturesAdapter.adaptToDataRecords(\_).asScala.head)

}