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

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

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

import com.twitter.home\_mixer.product.scored\_tweets.feature\_hydrator.RealGraphViewerAuthorFeatureHydrator.getCombinedRealGraphFeatures

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

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.CandidateFeatureHydrator

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.stitch.Stitch

import com.twitter.timelines.prediction.adapters.real\_graph.RealGraphEdgeFeaturesCombineAdapter

import com.twitter.timelines.prediction.adapters.real\_graph.RealGraphFeaturesAdapter

import com.twitter.timelines.real\_graph.v1.{thriftscala => v1}

import com.twitter.timelines.real\_graph.{thriftscala => rg}

import com.twitter.util.Throw

import javax.inject.Inject

import javax.inject.Singleton

import scala.collection.JavaConverters.\_

object RealGraphViewerAuthorDataRecordFeature

extends DataRecordInAFeature[TweetCandidate]

with FeatureWithDefaultOnFailure[TweetCandidate, DataRecord] {

override def defaultValue: DataRecord = new DataRecord()

}

object RealGraphViewerAuthorsDataRecordFeature

extends DataRecordInAFeature[TweetCandidate]

with FeatureWithDefaultOnFailure[TweetCandidate, DataRecord] {

override def defaultValue: DataRecord = new DataRecord()

}

@Singleton

class RealGraphViewerAuthorFeatureHydrator @Inject() ()

extends CandidateFeatureHydrator[PipelineQuery, TweetCandidate] {

override val identifier: FeatureHydratorIdentifier =

FeatureHydratorIdentifier("RealGraphViewerAuthor")

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

Set(RealGraphViewerAuthorDataRecordFeature, RealGraphViewerAuthorsDataRecordFeature)

private val realGraphEdgeFeaturesAdapter = new RealGraphFeaturesAdapter

private val realGraphEdgeFeaturesCombineAdapter =

new RealGraphEdgeFeaturesCombineAdapter(prefix = "authors.realgraph")

private val MissingKeyFeatureMap = FeatureMapBuilder()

.add(RealGraphViewerAuthorDataRecordFeature, Throw(MissingKeyException))

.add(RealGraphViewerAuthorsDataRecordFeature, Throw(MissingKeyException))

.build()

override def apply(

query: PipelineQuery,

candidate: TweetCandidate,

existingFeatures: FeatureMap

): Stitch[FeatureMap] = OffloadFuturePools.offload {

val viewerId = query.getRequiredUserId

val realGraphFeatures = query.features

.flatMap(\_.getOrElse(RealGraphFeatures, None))

.getOrElse(Map.empty[Long, v1.RealGraphEdgeFeatures])

existingFeatures.getOrElse(AuthorIdFeature, None) match {

case Some(authorId) =>

val realGraphAuthorFeatures =

getRealGraphViewerAuthorFeatures(viewerId, authorId, realGraphFeatures)

val realGraphAuthorDataRecord = realGraphEdgeFeaturesAdapter

.adaptToDataRecords(realGraphAuthorFeatures).asScala.headOption.getOrElse(new DataRecord)

val combinedRealGraphFeaturesDataRecord = for {

inReplyToAuthorId <- existingFeatures.getOrElse(InReplyToUserIdFeature, None)

} yield {

val combinedRealGraphFeatures =

getCombinedRealGraphFeatures(Seq(authorId, inReplyToAuthorId), realGraphFeatures)

realGraphEdgeFeaturesCombineAdapter

.adaptToDataRecords(Some(combinedRealGraphFeatures)).asScala.headOption

.getOrElse(new DataRecord)

}

FeatureMapBuilder()

.add(RealGraphViewerAuthorDataRecordFeature, realGraphAuthorDataRecord)

.add(

RealGraphViewerAuthorsDataRecordFeature,

combinedRealGraphFeaturesDataRecord.getOrElse(new DataRecord))

.build()

case \_ => MissingKeyFeatureMap

}

}

private def getRealGraphViewerAuthorFeatures(

viewerId: Long,

authorId: Long,

realGraphEdgeFeaturesMap: Map[Long, v1.RealGraphEdgeFeatures]

): rg.UserRealGraphFeatures = {

realGraphEdgeFeaturesMap.get(authorId) match {

case Some(realGraphEdgeFeatures) =>

rg.UserRealGraphFeatures(

srcId = viewerId,

features = rg.RealGraphFeatures.V1(

v1.RealGraphFeatures(edgeFeatures = Seq(realGraphEdgeFeatures))))

case \_ =>

rg.UserRealGraphFeatures(

srcId = viewerId,

features = rg.RealGraphFeatures.V1(v1.RealGraphFeatures(edgeFeatures = Seq.empty)))

}

}

}

object RealGraphViewerAuthorFeatureHydrator {

def getCombinedRealGraphFeatures(

userIds: Seq[Long],

realGraphEdgeFeaturesMap: Map[Long, v1.RealGraphEdgeFeatures]

): rg.RealGraphFeatures = {

val edgeFeatures = userIds.flatMap(realGraphEdgeFeaturesMap.get)

rg.RealGraphFeatures.V1(v1.RealGraphFeatures(edgeFeatures = edgeFeatures))

}

}