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

import com.twitter.escherbird.{thriftscala => esb}

import com.twitter.finagle.stats.Stat

import com.twitter.finagle.stats.StatsReceiver

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

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

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

import com.twitter.home\_mixer.product.scored\_tweets.feature\_hydrator.adapters.content.ContentFeatureAdapter

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

import com.twitter.home\_mixer.util.tweetypie.content.FeatureExtractionHelper

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.keyvalue.KeyValueResult

import com.twitter.servo.repository.KeyValueRepository

import com.twitter.stitch.Stitch

import com.twitter.timelines.prediction.common.util.MediaUnderstandingAnnotations

import com.twitter.tweetypie.{thriftscala => tp}

import com.twitter.util.Future

import com.twitter.util.Return

import com.twitter.util.Throw

import com.twitter.util.Try

import javax.inject.Inject

import javax.inject.Named

import javax.inject.Singleton

import scala.collection.JavaConverters.\_

object TweetypieContentDataRecordFeature

extends DataRecordInAFeature[TweetCandidate]

with FeatureWithDefaultOnFailure[TweetCandidate, DataRecord] {

override def defaultValue: DataRecord = new DataRecord()

}

@Singleton

class TweetypieContentFeatureHydrator @Inject() (

@Named(TweetypieContentRepository) client: KeyValueRepository[Seq[Long], Long, tp.Tweet],

override val statsReceiver: StatsReceiver)

extends BulkCandidateFeatureHydrator[PipelineQuery, TweetCandidate]

with ObservedKeyValueResultHandler {

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

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

MediaUnderstandingAnnotationIdsFeature,

TweetypieContentDataRecordFeature

)

override val statScope: String = identifier.toString

private val bulkRequestLatencyStat =

statsReceiver.scope(statScope).scope("bulkRequest").stat("latency\_ms")

private val postTransformerLatencyStat =

statsReceiver.scope(statScope).scope("postTransformer").stat("latency\_ms")

private val bulkPostTransformerLatencyStat =

statsReceiver.scope(statScope).scope("bulkPostTransformer").stat("latency\_ms")

private val DefaultDataRecord: DataRecord = new DataRecord()

override def apply(

query: PipelineQuery,

candidates: Seq[CandidateWithFeatures[TweetCandidate]]

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

val tweetIdsToHydrate = candidates.map(getCandidateOriginalTweetId).distinct

val response: Future[KeyValueResult[Long, tp.Tweet]] = Stat.timeFuture(bulkRequestLatencyStat) {

if (tweetIdsToHydrate.isEmpty) Future.value(KeyValueResult.empty)

else client(tweetIdsToHydrate)

}

response.flatMap { result =>

Stat.timeFuture(bulkPostTransformerLatencyStat) {

OffloadFuturePools

.parallelize[CandidateWithFeatures[TweetCandidate], Try[(Seq[Long], DataRecord)]](

candidates,

parTransformer(result, \_),

parallelism = 32,

default = Return((Seq.empty, DefaultDataRecord))

).map {

\_.map {

case Return(result) =>

FeatureMapBuilder()

.add(MediaUnderstandingAnnotationIdsFeature, result.\_1)

.add(TweetypieContentDataRecordFeature, result.\_2)

.build()

case Throw(e) =>

FeatureMapBuilder()

.add(MediaUnderstandingAnnotationIdsFeature, Throw(e))

.add(TweetypieContentDataRecordFeature, Throw(e))

.build()

}

}

}

}

}

private def parTransformer(

result: KeyValueResult[Long, tp.Tweet],

candidate: CandidateWithFeatures[TweetCandidate]

): Try[(Seq[Long], DataRecord)] = {

val originalTweetId = Some(getCandidateOriginalTweetId(candidate))

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

Stat.time(postTransformerLatencyStat)(postTransformer(value))

}

private def postTransformer(

result: Try[Option[tp.Tweet]]

): Try[(Seq[Long], DataRecord)] = {

result.map { tweet =>

val transformedValue = tweet.map(FeatureExtractionHelper.extractFeatures)

val semanticAnnotations = transformedValue

.flatMap { contentFeatures =>

contentFeatures.semanticCoreAnnotations.map {

getNonSensitiveHighRecallMediaUnderstandingAnnotationEntityIds

}

}.getOrElse(Seq.empty)

val dataRecord = ContentFeatureAdapter.adaptToDataRecords(transformedValue).asScala.head

(semanticAnnotations, dataRecord)

}

}

private def getCandidateOriginalTweetId(

candidate: CandidateWithFeatures[TweetCandidate]

): Long = {

candidate.features

.getOrElse(SourceTweetIdFeature, None).getOrElse(candidate.candidate.id)

}

private def getNonSensitiveHighRecallMediaUnderstandingAnnotationEntityIds(

semanticCoreAnnotations: Seq[esb.TweetEntityAnnotation]

): Seq[Long] =

semanticCoreAnnotations

.filter(MediaUnderstandingAnnotations.isEligibleNonSensitiveHighRecallMUAnnotation)

.map(\_.entityId)

}