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

import com.twitter.dal.personal\_data.{thriftjava => pd}

import com.twitter.finagle.stats.StatsReceiver

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

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.datarecord.DataRecordOptionalFeature

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

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.stitch.Stitch

import com.twitter.strato.catalog.Fetch

import com.twitter.strato.generated.client.ml.featureStore.SimClustersUserInterestedInTweetEmbeddingDotProduct20M145K2020OnUserTweetEdgeClientColumn

import javax.inject.Inject

import javax.inject.Singleton

object SimClustersUserInterestedInTweetEmbeddingDataRecordFeature

extends DataRecordOptionalFeature[TweetCandidate, Double]

with DoubleDataRecordCompatible {

override val featureName: String =

"user-tweet.recommendations.sim\_clusters\_scores.user\_interested\_in\_tweet\_embedding\_dot\_product\_20m\_145k\_2020"

override val personalDataTypes: Set[pd.PersonalDataType] =

Set(pd.PersonalDataType.InferredInterests)

}

@Singleton

class SimClustersUserTweetScoresHydrator @Inject() (

simClustersColumn: SimClustersUserInterestedInTweetEmbeddingDotProduct20M145K2020OnUserTweetEdgeClientColumn,

statsReceiver: StatsReceiver)

extends BulkCandidateFeatureHydrator[PipelineQuery, TweetCandidate] {

override val identifier: FeatureHydratorIdentifier =

FeatureHydratorIdentifier("SimClustersUserTweetScores")

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

SimClustersUserInterestedInTweetEmbeddingDataRecordFeature)

private val scopedStatsReceiver = statsReceiver.scope(getClass.getSimpleName)

private val keyFoundCounter = scopedStatsReceiver.counter("key/found")

private val keyLossCounter = scopedStatsReceiver.counter("key/loss")

private val keyFailureCounter = scopedStatsReceiver.counter("key/failure")

private val keySkipCounter = scopedStatsReceiver.counter("key/skip")

private val DefaultFeatureMap = FeatureMapBuilder()

.add(SimClustersUserInterestedInTweetEmbeddingDataRecordFeature, None)

.build()

private val MinFavToHydrate = 9

override def apply(

query: PipelineQuery,

candidates: Seq[CandidateWithFeatures[TweetCandidate]]

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

Stitch.run {

Stitch.collect {

candidates.map { candidate =>

val ebFeatures = candidate.features.getOrElse(EarlybirdFeature, None)

val favCount = ebFeatures.flatMap(\_.favCountV2).getOrElse(0)

if (ebFeatures.isEmpty || favCount >= MinFavToHydrate) {

simClustersColumn.fetcher

.fetch((query.getRequiredUserId, candidate.candidate.id), Unit)

.map {

case Fetch.Result(response, \_) =>

if (response.nonEmpty) keyFoundCounter.incr() else keyLossCounter.incr()

FeatureMapBuilder()

.add(SimClustersUserInterestedInTweetEmbeddingDataRecordFeature, response)

.build()

case \_ =>

keyFailureCounter.incr()

DefaultFeatureMap

}

} else {

keySkipCounter.incr()

Stitch.value(DefaultFeatureMap)

}

}

}

}

}

}