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

import com.twitter.ml.api.DataRecord

import com.twitter.ml.api.FeatureContext

import com.twitter.ml.api.IRecordOneToOneAdapter

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.pipeline.PipelineQuery

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

import com.twitter.stitch.Stitch

import com.twitter.timelines.data\_processing.ml\_util.aggregation\_framework.AggregateGroup

import com.twitter.timelines.data\_processing.ml\_util.aggregation\_framework.AggregateType.AggregateType

import com.twitter.timelines.suggests.common.dense\_data\_record.thriftjava.DenseCompactDataRecord

import java.lang.{Long => JLong}

import java.util.{Map => JMap}

abstract case class BaseEdgeAggregateFeature(

aggregateGroups: Set[AggregateGroup],

aggregateType: AggregateType,

extractMapFn: AggregateFeaturesToDecodeWithMetadata => JMap[JLong, DenseCompactDataRecord],

adapter: IRecordOneToOneAdapter[Seq[DataRecord]],

getSecondaryKeysFn: CandidateWithFeatures[TweetCandidate] => Seq[Long])

extends DataRecordInAFeature[PipelineQuery]

with FeatureWithDefaultOnFailure[PipelineQuery, DataRecord] {

override def defaultValue: DataRecord = new DataRecord

private val rootFeatureInfo = new AggregateFeatureInfo(aggregateGroups, aggregateType)

val featureContext: FeatureContext = rootFeatureInfo.featureContext

val rootFeature: BaseAggregateRootFeature = rootFeatureInfo.feature

}

trait BaseEdgeAggregateFeatureHydrator

extends BulkCandidateFeatureHydrator[PipelineQuery, TweetCandidate] {

def aggregateFeatures: Set[BaseEdgeAggregateFeature]

override def features = aggregateFeatures.asInstanceOf[Set[Feature[\_, \_]]]

override def apply(

query: PipelineQuery,

candidates: Seq[CandidateWithFeatures[TweetCandidate]]

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

val featureMapBuilders: Seq[FeatureMapBuilder] =

for (\_ <- candidates) yield FeatureMapBuilder()

aggregateFeatures.foreach { feature =>

val featureValues = hydrateAggregateFeature(query, candidates, feature)

(featureMapBuilders zip featureValues).foreach {

case (featureMapBuilder, featureValue) => featureMapBuilder.add(feature, featureValue)

}

}

featureMapBuilders.map(\_.build())

}

private def hydrateAggregateFeature(

query: PipelineQuery,

candidates: Seq[CandidateWithFeatures[TweetCandidate]],

feature: BaseEdgeAggregateFeature

): Seq[DataRecord] = {

val rootFeature = feature.rootFeature

val extractMapFn = feature.extractMapFn

val featureContext = feature.featureContext

val secondaryIds: Seq[Seq[Long]] = candidates.map(feature.getSecondaryKeysFn)

val featuresToDecodeWithMetadata = query.features

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

.getOrElse(AggregateFeaturesToDecodeWithMetadata.empty)

// Decode the DenseCompactDataRecords into DataRecords for each required secondary id.

val decoded: Map[Long, DataRecord] = Utils.selectAndTransform(

secondaryIds.flatten.distinct,

featuresToDecodeWithMetadata.toDataRecord,

extractMapFn(featuresToDecodeWithMetadata)

)

// Remove unnecessary features in-place. This is safe because the underlying DataRecords

// are unique and have just been generated in the previous step.

decoded.values.foreach(Utils.filterDataRecord(\_, featureContext))

// Put features into the FeatureMapBuilders

secondaryIds.map { ids =>

val dataRecords = ids.flatMap(decoded.get)

feature.adapter.adaptToDataRecord(dataRecords)

}

}

}