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

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

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

import com.twitter.ml.api.DataRecord

import com.twitter.ml.api.DataRecordMerger

import com.twitter.ml.api.FeatureContext

import com.twitter.ml.api.RichDataRecord

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.model.common.identifier.FeatureHydratorIdentifier

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

import com.twitter.servo.repository.Repository

import com.twitter.stitch.Stitch

import com.twitter.timelines.data\_processing.jobs.timeline\_ranking\_user\_features.TimelinesPartBStoreRegister

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

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

import com.twitter.timelines.prediction.adapters.request\_context.RequestContextAdapter

import com.twitter.timelines.prediction.common.aggregates.TimelinesAggregationConfig

import com.twitter.timelines.suggests.common.dense\_data\_record.thriftscala.DenseFeatureMetadata

import com.twitter.user\_session\_store.thriftjava.UserSession

import com.twitter.util.Time

import javax.inject.Inject

import javax.inject.Named

import javax.inject.Singleton

object PartBAggregateRootFeature extends BaseAggregateRootFeature {

override val aggregateStores: Set[StoreConfig[\_]] = TimelinesPartBStoreRegister.allStores

}

object UserAggregateFeature

extends DataRecordInAFeature[PipelineQuery]

with FeatureWithDefaultOnFailure[PipelineQuery, DataRecord] {

override def defaultValue: DataRecord = new DataRecord()

}

@Singleton

class PartBAggregateQueryFeatureHydrator @Inject() (

@Named(TimelineAggregatePartBRepository)

repository: Repository[Long, Option[UserSession]],

@Named(TimelineAggregateMetadataRepository)

metadataRepository: Repository[Int, Option[DenseFeatureMetadata]])

extends BaseAggregateQueryFeatureHydrator(

repository,

metadataRepository,

PartBAggregateRootFeature

) {

override val identifier: FeatureHydratorIdentifier =

FeatureHydratorIdentifier("PartBAggregateQuery")

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

Set(PartBAggregateRootFeature, UserAggregateFeature)

private val userAggregateFeatureInfo = new AggregateFeatureInfo(

aggregateGroups = Set(

TimelinesAggregationConfig.userAggregatesV2,

TimelinesAggregationConfig.userAggregatesV5Continuous,

TimelinesAggregationConfig.userAggregatesV6,

TimelinesAggregationConfig.twitterWideUserAggregates,

),

aggregateType = AggregateType.User

)

private val userHourAggregateFeatureInfo = new AggregateFeatureInfo(

aggregateGroups = Set(

TimelinesAggregationConfig.userRequestHourAggregates,

),

aggregateType = AggregateType.UserRequestHour

)

private val userDowAggregateFeatureInfo = new AggregateFeatureInfo(

aggregateGroups = Set(

TimelinesAggregationConfig.userRequestDowAggregates

),

aggregateType = AggregateType.UserRequestDow

)

require(

userAggregateFeatureInfo.feature == PartBAggregateRootFeature,

"UserAggregates feature must be provided by the PartB data source.")

require(

userHourAggregateFeatureInfo.feature == PartBAggregateRootFeature,

"UserRequstHourAggregates feature must be provided by the PartB data source.")

require(

userDowAggregateFeatureInfo.feature == PartBAggregateRootFeature,

"UserRequestDowAggregates feature must be provided by the PartB data source.")

override def hydrate(query: PipelineQuery): Stitch[FeatureMap] = {

// Hydrate TimelineAggregatePartBFeature and UserAggregateFeature sequentially.

super.hydrate(query).map { featureMap =>

val time: Time = Time.now

val hourOfDay = RequestContextAdapter.hourFromTimestamp(time.inMilliseconds)

val dayOfWeek = RequestContextAdapter.dowFromTimestamp(time.inMilliseconds)

val dr = featureMap

.get(PartBAggregateRootFeature).map { featuresWithMetadata =>

val userAggregatesDr =

featuresWithMetadata.userAggregatesOpt

.map(featuresWithMetadata.toDataRecord)

val userRequestHourAggregatesDr =

Option(featuresWithMetadata.userRequestHourAggregates.get(hourOfDay))

.map(featuresWithMetadata.toDataRecord)

val userRequestDowAggregatesDr =

Option(featuresWithMetadata.userRequestDowAggregates.get(dayOfWeek))

.map(featuresWithMetadata.toDataRecord)

dropUnknownFeatures(userAggregatesDr, userAggregateFeatureInfo.featureContext)

dropUnknownFeatures(

userRequestHourAggregatesDr,

userHourAggregateFeatureInfo.featureContext)

dropUnknownFeatures(

userRequestDowAggregatesDr,

userDowAggregateFeatureInfo.featureContext)

mergeDataRecordOpts(

userAggregatesDr,

userRequestHourAggregatesDr,

userRequestDowAggregatesDr)

}.getOrElse(new DataRecord())

featureMap + (UserAggregateFeature, dr)

}

}

private val drMerger = new DataRecordMerger

private def mergeDataRecordOpts(dataRecordOpts: Option[DataRecord]\*): DataRecord =

dataRecordOpts.flatten.foldLeft(new DataRecord) { (l, r) =>

drMerger.merge(l, r)

l

}

private def dropUnknownFeatures(

dataRecordOpt: Option[DataRecord],

featureContext: FeatureContext

): Unit =

dataRecordOpt.foreach(new RichDataRecord(\_, featureContext).dropUnknownFeatures())

}