package com.twitter.timelines.prediction.common.aggregates.real\_time

import com.twitter.ml.api.DataRecord

import com.twitter.ml.featurestore.lib.TweetId

import com.twitter.ml.featurestore.lib.data.PredictionRecord

import com.twitter.ml.featurestore.lib.entity.Entity

import com.twitter.ml.featurestore.lib.online.{FeatureStoreClient, FeatureStoreRequest}

import com.twitter.storehaus.ReadableStore

import com.twitter.timelines.prediction.common.adapters.TimelinesAdapterBase

import com.twitter.util.Future

import scala.collection.JavaConverters.\_

class TweetFeaturesReadableStore(

featureStoreClient: FeatureStoreClient,

tweetEntity: Entity[TweetId],

tweetFeaturesAdapter: TimelinesAdapterBase[PredictionRecord])

extends ReadableStore[Set[Long], DataRecord] {

override def multiGet[K <: Set[Long]](keys: Set[K]): Map[K, Future[Option[DataRecord]]] = {

val orderedKeys: Seq[K] = keys.toSeq

val featureStoreRequests: Seq[FeatureStoreRequest] = getFeatureStoreRequests(orderedKeys)

val predictionRecordsFut: Future[Seq[PredictionRecord]] = featureStoreClient(

featureStoreRequests)

getDataRecordMap(orderedKeys, predictionRecordsFut)

}

private def getFeatureStoreRequests[K <: Set[Long]](

orderedKeys: Seq[K]

): Seq[FeatureStoreRequest] = {

orderedKeys.map { key: Set[Long] =>

FeatureStoreRequest(

entityIds = key.map { tweetId => tweetEntity.withId(TweetId(tweetId)) }.toSeq

)

}

}

private def getDataRecordMap[K <: Set[Long]](

orderedKeys: Seq[K],

predictionRecordsFut: Future[Seq[PredictionRecord]]

): Map[K, Future[Option[DataRecord]]] = {

orderedKeys.zipWithIndex.map {

case (tweetIdSet, index) =>

val dataRecordFutOpt: Future[Option[DataRecord]] = predictionRecordsFut.map {

predictionRecords =>

predictionRecords.lift(index).flatMap { predictionRecordAtIndex: PredictionRecord =>

tweetFeaturesAdapter.adaptToDataRecords(predictionRecordAtIndex).asScala.headOption

}

}

(tweetIdSet, dataRecordFutOpt)

}.toMap

}

}