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

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

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

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

import com.twitter.ml.api.DataRecord

import com.twitter.ml.api.util.FDsl.\_

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.CandidateFeatureHydrator

import com.twitter.product\_mixer.core.model.common.identifier.FeatureHydratorIdentifier

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

import com.twitter.snowflake.id.SnowflakeId

import com.twitter.stitch.Stitch

import com.twitter.timelines.prediction.features.time\_features.TimeDataRecordFeatures.\_

import com.twitter.util.Duration

import scala.collection.Searching.\_

object TweetTimeDataRecordFeature

extends DataRecordInAFeature[TweetCandidate]

with FeatureWithDefaultOnFailure[TweetCandidate, DataRecord] {

override def defaultValue: DataRecord = new DataRecord()

}

object TweetTimeFeatureHydrator extends CandidateFeatureHydrator[PipelineQuery, TweetCandidate] {

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

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

override def apply(

query: PipelineQuery,

candidate: TweetCandidate,

existingFeatures: FeatureMap

): Stitch[FeatureMap] = {

val tweetFeatures = existingFeatures.getOrElse(EarlybirdFeature, None)

val timeSinceTweetCreation = SnowflakeId.timeFromIdOpt(candidate.id).map(query.queryTime.since)

val timeSinceTweetCreationMs = timeSinceTweetCreation.map(\_.inMillis)

val timeSinceSourceTweetCreationOpt = existingFeatures

.getOrElse(SourceTweetIdFeature, None)

.flatMap { sourceTweetId =>

SnowflakeId.timeFromIdOpt(sourceTweetId).map(query.queryTime.since)

}.orElse(timeSinceTweetCreation)

val lastFavSinceCreationHrs =

tweetFeatures.flatMap(\_.lastFavSinceCreationHrs).map(\_.toDouble)

val lastRetweetSinceCreationHrs =

tweetFeatures.flatMap(\_.lastRetweetSinceCreationHrs).map(\_.toDouble)

val lastReplySinceCreationHrs =

tweetFeatures.flatMap(\_.lastReplySinceCreationHrs).map(\_.toDouble)

val lastQuoteSinceCreationHrs =

tweetFeatures.flatMap(\_.lastQuoteSinceCreationHrs).map(\_.toDouble)

val timeSinceLastFavoriteHrs =

getTimeSinceLastEngagementHrs(lastFavSinceCreationHrs, timeSinceSourceTweetCreationOpt)

val timeSinceLastRetweetHrs =

getTimeSinceLastEngagementHrs(lastRetweetSinceCreationHrs, timeSinceSourceTweetCreationOpt)

val timeSinceLastReplyHrs =

getTimeSinceLastEngagementHrs(lastReplySinceCreationHrs, timeSinceSourceTweetCreationOpt)

val timeSinceLastQuoteHrs =

getTimeSinceLastEngagementHrs(lastQuoteSinceCreationHrs, timeSinceSourceTweetCreationOpt)

val nonPollingTimestampsMs = query.features.get.getOrElse(NonPollingTimesFeature, Seq.empty)

val timeSinceLastNonPollingRequest =

nonPollingTimestampsMs.headOption.map(query.queryTime.inMillis - \_)

val nonPollingRequestsSinceTweetCreation =

if (nonPollingTimestampsMs.nonEmpty && timeSinceTweetCreationMs.isDefined) {

nonPollingTimestampsMs

.search(timeSinceTweetCreationMs.get)(Ordering[Long].reverse)

.insertionPoint

} else 0.0

val tweetAgeRatio =

if (timeSinceTweetCreationMs.exists(\_ > 0.0) && timeSinceLastNonPollingRequest.isDefined) {

timeSinceLastNonPollingRequest.get / timeSinceTweetCreationMs.get.toDouble

} else 0.0

val dataRecord = new DataRecord()

.setFeatureValue(IS\_TWEET\_RECYCLED, false)

.setFeatureValue(TWEET\_AGE\_RATIO, tweetAgeRatio)

.setFeatureValueFromOption(

TIME\_SINCE\_TWEET\_CREATION,

timeSinceTweetCreationMs.map(\_.toDouble)

)

.setFeatureValue(

NON\_POLLING\_REQUESTS\_SINCE\_TWEET\_CREATION,

nonPollingRequestsSinceTweetCreation

)

.setFeatureValueFromOption(LAST\_FAVORITE\_SINCE\_CREATION\_HRS, lastFavSinceCreationHrs)

.setFeatureValueFromOption(LAST\_RETWEET\_SINCE\_CREATION\_HRS, lastRetweetSinceCreationHrs)

.setFeatureValueFromOption(LAST\_REPLY\_SINCE\_CREATION\_HRS, lastReplySinceCreationHrs)

.setFeatureValueFromOption(LAST\_QUOTE\_SINCE\_CREATION\_HRS, lastQuoteSinceCreationHrs)

.setFeatureValueFromOption(TIME\_SINCE\_LAST\_FAVORITE\_HRS, timeSinceLastFavoriteHrs)

.setFeatureValueFromOption(TIME\_SINCE\_LAST\_RETWEET\_HRS, timeSinceLastRetweetHrs)

.setFeatureValueFromOption(TIME\_SINCE\_LAST\_REPLY\_HRS, timeSinceLastReplyHrs)

.setFeatureValueFromOption(TIME\_SINCE\_LAST\_QUOTE\_HRS, timeSinceLastQuoteHrs)

Stitch.value(FeatureMapBuilder().add(TweetTimeDataRecordFeature, dataRecord).build())

}

private def getTimeSinceLastEngagementHrs(

lastEngagementTimeSinceCreationHrsOpt: Option[Double],

timeSinceTweetCreation: Option[Duration]

): Option[Double] = lastEngagementTimeSinceCreationHrsOpt.flatMap { lastEngagementTimeHrs =>

timeSinceTweetCreation.map(\_.inHours - lastEngagementTimeHrs)

}

}