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

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.home\_mixer.model.ContentFeatures

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

import com.twitter.home\_mixer.product.scored\_tweets.feature\_hydrator.adapters.content.InReplyToContentFeatureAdapter

import com.twitter.home\_mixer.product.scored\_tweets.feature\_hydrator.adapters.earlybird.InReplyToEarlybirdAdapter

import com.twitter.home\_mixer.util.ReplyRetweetUtil

import com.twitter.ml.api.DataRecord

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.model.common.identifier.FeatureHydratorIdentifier

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

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

import com.twitter.search.common.features.thriftscala.ThriftTweetFeatures

import com.twitter.snowflake.id.SnowflakeId

import com.twitter.stitch.Stitch

import com.twitter.timelines.conversation\_features.v1.thriftscala.ConversationFeatures

import com.twitter.timelines.conversation\_features.{thriftscala => cf}

import com.twitter.timelines.prediction.adapters.conversation\_features.ConversationFeaturesAdapter

import com.twitter.util.Duration

import com.twitter.util.Time

import javax.inject.Inject

import javax.inject.Singleton

import scala.collection.JavaConverters.\_

object InReplyToTweetHydratedEarlybirdFeature

extends Feature[TweetCandidate, Option[ThriftTweetFeatures]]

object ConversationDataRecordFeature

extends DataRecordInAFeature[TweetCandidate]

with FeatureWithDefaultOnFailure[TweetCandidate, DataRecord] {

override def defaultValue: DataRecord = new DataRecord()

}

object InReplyToEarlybirdDataRecordFeature

extends DataRecordInAFeature[TweetCandidate]

with FeatureWithDefaultOnFailure[TweetCandidate, DataRecord] {

override def defaultValue: DataRecord = new DataRecord()

}

object InReplyToTweetypieContentDataRecordFeature

extends DataRecordInAFeature[TweetCandidate]

with FeatureWithDefaultOnFailure[TweetCandidate, DataRecord] {

override def defaultValue: DataRecord = new DataRecord()

}

/\*\*

\* The purpose of this hydrator is to

\* 1) hydrate simple features into replies and their ancestor tweets

\* 2) keep both the normal replies and ancestor source candidates, but hydrate into the candidates

\* features useful for predicting the quality of the replies and source ancestor tweets.

\*/

@Singleton

class ReplyFeatureHydrator @Inject() (statsReceiver: StatsReceiver)

extends BulkCandidateFeatureHydrator[PipelineQuery, TweetCandidate] {

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

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

ConversationDataRecordFeature,

InReplyToTweetHydratedEarlybirdFeature,

InReplyToEarlybirdDataRecordFeature,

InReplyToTweetypieContentDataRecordFeature

)

private val defaulDataRecord: DataRecord = new DataRecord()

private val DefaultFeatureMap = FeatureMapBuilder()

.add(ConversationDataRecordFeature, defaulDataRecord)

.add(InReplyToTweetHydratedEarlybirdFeature, None)

.add(InReplyToEarlybirdDataRecordFeature, defaulDataRecord)

.add(InReplyToTweetypieContentDataRecordFeature, defaulDataRecord)

.build()

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

private val hydratedReplyCounter = scopedStatsReceiver.counter("hydratedReply")

private val hydratedAncestorCounter = scopedStatsReceiver.counter("hydratedAncestor")

override def apply(

query: PipelineQuery,

candidates: Seq[CandidateWithFeatures[TweetCandidate]]

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

val replyToInReplyToTweetMap =

ReplyRetweetUtil.replyTweetIdToInReplyToTweetMap(candidates)

val candidatesWithRepliesHydrated = candidates.map { candidate =>

replyToInReplyToTweetMap

.get(candidate.candidate.id).map { inReplyToTweet =>

hydratedReplyCounter.incr()

hydratedReplyCandidate(candidate, inReplyToTweet)

}.getOrElse((candidate, None, None))

}

/\*\*

\* Update ancestor tweets with descendant replies and hydrate simple features from one of

\* the descendants.

\*/

val ancestorTweetToDescendantRepliesMap =

ReplyRetweetUtil.ancestorTweetIdToDescendantRepliesMap(candidates)

val candidatesWithRepliesAndAncestorTweetsHydrated = candidatesWithRepliesHydrated.map {

case (

maybeAncestorTweetCandidate,

updatedReplyConversationFeatures,

inReplyToTweetEarlyBirdFeature) =>

ancestorTweetToDescendantRepliesMap

.get(maybeAncestorTweetCandidate.candidate.id)

.map { descendantReplies =>

hydratedAncestorCounter.incr()

val (ancestorTweetCandidate, updatedConversationFeatures): (

CandidateWithFeatures[TweetCandidate],

Option[ConversationFeatures]

) =

hydrateAncestorTweetCandidate(

maybeAncestorTweetCandidate,

descendantReplies,

updatedReplyConversationFeatures)

(ancestorTweetCandidate, inReplyToTweetEarlyBirdFeature, updatedConversationFeatures)

}

.getOrElse(

(

maybeAncestorTweetCandidate,

inReplyToTweetEarlyBirdFeature,

updatedReplyConversationFeatures))

}

candidatesWithRepliesAndAncestorTweetsHydrated.map {

case (candidate, inReplyToTweetEarlyBirdFeature, updatedConversationFeatures) =>

val conversationDataRecordFeature = updatedConversationFeatures

.map(f => ConversationFeaturesAdapter.adaptToDataRecord(cf.ConversationFeatures.V1(f)))

.getOrElse(defaulDataRecord)

val inReplyToEarlybirdDataRecord =

InReplyToEarlybirdAdapter

.adaptToDataRecords(inReplyToTweetEarlyBirdFeature).asScala.head

val inReplyToContentDataRecord = InReplyToContentFeatureAdapter

.adaptToDataRecords(

inReplyToTweetEarlyBirdFeature.map(ContentFeatures.fromThrift)).asScala.head

FeatureMapBuilder()

.add(ConversationDataRecordFeature, conversationDataRecordFeature)

.add(InReplyToTweetHydratedEarlybirdFeature, inReplyToTweetEarlyBirdFeature)

.add(InReplyToEarlybirdDataRecordFeature, inReplyToEarlybirdDataRecord)

.add(InReplyToTweetypieContentDataRecordFeature, inReplyToContentDataRecord)

.build()

case \_ => DefaultFeatureMap

}

}

private def hydratedReplyCandidate(

replyCandidate: CandidateWithFeatures[TweetCandidate],

inReplyToTweetCandidate: CandidateWithFeatures[TweetCandidate]

): (

CandidateWithFeatures[TweetCandidate],

Option[ConversationFeatures],

Option[ThriftTweetFeatures]

) = {

val tweetedAfterInReplyToTweetInSecs =

(

originalTweetAgeFromSnowflake(inReplyToTweetCandidate),

originalTweetAgeFromSnowflake(replyCandidate)) match {

case (Some(inReplyToTweetAge), Some(replyTweetAge)) =>

Some((inReplyToTweetAge - replyTweetAge).inSeconds.toLong)

case \_ => None

}

val existingConversationFeatures = Some(

replyCandidate.features

.getOrElse(ConversationFeature, None).getOrElse(ConversationFeatures()))

val updatedConversationFeatures = existingConversationFeatures match {

case Some(v1) =>

Some(

v1.copy(

tweetedAfterInReplyToTweetInSecs = tweetedAfterInReplyToTweetInSecs,

isSelfReply = Some(

replyCandidate.features.getOrElse(

AuthorIdFeature,

None) == inReplyToTweetCandidate.features.getOrElse(AuthorIdFeature, None))

)

)

case \_ => None

}

// Note: if inReplyToTweet is a retweet, we need to read early bird feature from the merged

// early bird feature field from RetweetSourceTweetFeatureHydrator class.

// But if inReplyToTweet is a reply, we return its early bird feature directly

val inReplyToTweetThriftTweetFeaturesOpt = {

if (inReplyToTweetCandidate.features.getOrElse(IsRetweetFeature, false)) {

inReplyToTweetCandidate.features.getOrElse(SourceTweetEarlybirdFeature, None)

} else {

inReplyToTweetCandidate.features.getOrElse(EarlybirdFeature, None)

}

}

(replyCandidate, updatedConversationFeatures, inReplyToTweetThriftTweetFeaturesOpt)

}

private def hydrateAncestorTweetCandidate(

ancestorTweetCandidate: CandidateWithFeatures[TweetCandidate],

descendantReplies: Seq[CandidateWithFeatures[TweetCandidate]],

updatedReplyConversationFeatures: Option[ConversationFeatures]

): (CandidateWithFeatures[TweetCandidate], Option[ConversationFeatures]) = {

// Ancestor could be a reply. For example, in thread: tweetA -> tweetB -> tweetC,

// tweetB is a reply and ancestor at the same time. Hence, tweetB's conversation feature

// will be updated by hydratedReplyCandidate and hydrateAncestorTweetCandidate functions.

val existingConversationFeatures =

if (updatedReplyConversationFeatures.nonEmpty)

updatedReplyConversationFeatures

else

Some(

ancestorTweetCandidate.features

.getOrElse(ConversationFeature, None).getOrElse(ConversationFeatures()))

val updatedConversationFeatures = existingConversationFeatures match {

case Some(v1) =>

Some(

v1.copy(

hasDescendantReplyCandidate = Some(true),

hasInNetworkDescendantReply =

Some(descendantReplies.exists(\_.features.getOrElse(InNetworkFeature, false)))

))

case \_ => None

}

(ancestorTweetCandidate, updatedConversationFeatures)

}

private def originalTweetAgeFromSnowflake(

candidate: CandidateWithFeatures[TweetCandidate]

): Option[Duration] = {

SnowflakeId

.timeFromIdOpt(

candidate.features

.getOrElse(SourceTweetIdFeature, None).getOrElse(candidate.candidate.id))

.map(Time.now - \_)

}

}