package com.twitter.product\_mixer.component\_library.feature\_hydrator.candidate.tweet\_visibility\_reason

import com.twitter.product\_mixer.component\_library.model.candidate.BaseTweetCandidate

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.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.spam.rtf.{thriftscala => SPAM}

import com.twitter.stitch.Stitch

import com.twitter.stitch.tweetypie.{TweetyPie => TweetypieStitchClient}

import com.twitter.tweetypie.{thriftscala => TP}

import com.twitter.util.Return

import com.twitter.util.Throw

import com.twitter.util.Try

import com.twitter.util.logging.Logging

import javax.inject.Inject

import javax.inject.Singleton

object VisibilityReason

extends FeatureWithDefaultOnFailure[TweetCandidate, Option[SPAM.FilteredReason]] {

override val defaultValue = None

}

/\*\*

\* A [[BulkCandidateFeatureHydrator]] that hydrates TweetCandidates with VisibilityReason features

\* by [[SPAM.SafetyLevel]] when present. The [[VisibilityReason]] feature represents a VisibilityFiltering

\* [[SPAM.FilteredReason]], which contains safety filtering verdict information including action (e.g.

\* Drop, Avoid) and reason (e.g. Misinformation, Abuse). This feature can inform downstream services'

\* handling and presentation of Tweets (e.g. ad avoidance).

\*

\* @param tweetypieStitchClient used to retrieve Tweet fields for BaseTweetCandidates

\* @param safetyLevel specifies VisibilityFiltering SafetyLabel

\*/

@Singleton

case class TweetVisibilityReasonBulkCandidateFeatureHydrator @Inject() (

tweetypieStitchClient: TweetypieStitchClient,

safetyLevel: SPAM.SafetyLevel)

extends BulkCandidateFeatureHydrator[PipelineQuery, BaseTweetCandidate]

with Logging {

override val identifier: FeatureHydratorIdentifier = FeatureHydratorIdentifier(

"TweetVisibilityReason")

override def features: Set[Feature[\_, \_]] = Set(VisibilityReason)

override def apply(

query: PipelineQuery,

candidates: Seq[CandidateWithFeatures[BaseTweetCandidate]]

): Stitch[Seq[FeatureMap]] = {

Stitch

.traverse(candidates.map(\_.candidate.id)) { tweetId =>

tweetypieStitchClient

.getTweetFields(

tweetId = tweetId,

options = TP.GetTweetFieldsOptions(

forUserId = query.getOptionalUserId,

tweetIncludes = Set.empty,

doNotCache = true,

visibilityPolicy = TP.TweetVisibilityPolicy.UserVisible,

safetyLevel = Some(safetyLevel)

)

).liftToTry

}.map { getTweetFieldsResults: Seq[Try[TP.GetTweetFieldsResult]] =>

val tweetFields: Seq[Try[TP.TweetFieldsResultFound]] = getTweetFieldsResults.map {

case Return(TP.GetTweetFieldsResult(\_, TP.TweetFieldsResultState.Found(found), \_, \_)) =>

Return(found)

case Return(TP.GetTweetFieldsResult(\_, resultState, \_, \_)) =>

Throw(

VisibilityReasonFeatureHydrationFailure(

s"Unexpected tweet result state: ${resultState}"))

case Throw(e) =>

Throw(e)

}

tweetFields.map { tweetFieldTry =>

val tweetFilteredReason = tweetFieldTry.map { tweetField =>

tweetField.suppressReason match {

case Some(suppressReason) => Some(suppressReason)

case \_ => None

}

}

FeatureMapBuilder()

.add(VisibilityReason, tweetFilteredReason)

.build()

}

}

}

}

case class VisibilityReasonFeatureHydrationFailure(message: String)

extends Exception(s"VisibilityReasonFeatureHydrationFailure($message)")