package com.twitter.visibility.builder.tweets

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.snowflake.id.SnowflakeId

import com.twitter.stitch.Stitch

import com.twitter.tweetypie.thriftscala.CollabControl

import com.twitter.tweetypie.thriftscala.Tweet

import com.twitter.util.Duration

import com.twitter.util.Time

import com.twitter.visibility.builder.FeatureMapBuilder

import com.twitter.visibility.common.SafetyLabelMapSource

import com.twitter.visibility.common.TweetId

import com.twitter.visibility.common.UserId

import com.twitter.visibility.features.\_

import com.twitter.visibility.models.SemanticCoreAnnotation

import com.twitter.visibility.models.TweetSafetyLabel

object TweetFeatures {

def FALLBACK\_TIMESTAMP: Time = Time.epoch

def tweetIsSelfReply(tweet: Tweet): Boolean = {

tweet.coreData match {

case Some(coreData) =>

coreData.reply match {

case Some(reply) =>

reply.inReplyToUserId == coreData.userId

case None =>

false

}

case None =>

false

}

}

def tweetReplyToParentTweetDuration(tweet: Tweet): Option[Duration] = for {

coreData <- tweet.coreData

reply <- coreData.reply

inReplyToStatusId <- reply.inReplyToStatusId

replyTime <- SnowflakeId.timeFromIdOpt(tweet.id)

repliedToTime <- SnowflakeId.timeFromIdOpt(inReplyToStatusId)

} yield {

replyTime.diff(repliedToTime)

}

def tweetReplyToRootTweetDuration(tweet: Tweet): Option[Duration] = for {

coreData <- tweet.coreData

if coreData.reply.isDefined

conversationId <- coreData.conversationId

replyTime <- SnowflakeId.timeFromIdOpt(tweet.id)

rootTime <- SnowflakeId.timeFromIdOpt(conversationId)

} yield {

replyTime.diff(rootTime)

}

def tweetTimestamp(tweetId: Long): Time =

SnowflakeId.timeFromIdOpt(tweetId).getOrElse(FALLBACK\_TIMESTAMP)

def tweetSemanticCoreAnnotations(tweet: Tweet): Seq[SemanticCoreAnnotation] = {

tweet.escherbirdEntityAnnotations

.map(a =>

a.entityAnnotations.map { annotation =>

SemanticCoreAnnotation(

annotation.groupId,

annotation.domainId,

annotation.entityId

)

}).toSeq.flatten

}

def tweetIsNullcast(tweet: Tweet): Boolean = {

tweet.coreData match {

case Some(coreData) =>

coreData.nullcast

case None =>

false

}

}

def tweetAuthorUserId(tweet: Tweet): Option[UserId] = {

tweet.coreData.map(\_.userId)

}

}

sealed trait TweetLabels {

def forTweet(tweet: Tweet): Stitch[Seq[TweetSafetyLabel]]

def forTweetId(tweetId: TweetId): Stitch[Seq[TweetSafetyLabel]]

}

class StratoTweetLabelMaps(safetyLabelSource: SafetyLabelMapSource) extends TweetLabels {

override def forTweet(tweet: Tweet): Stitch[Seq[TweetSafetyLabel]] = {

forTweetId(tweet.id)

}

def forTweetId(tweetId: TweetId): Stitch[Seq[TweetSafetyLabel]] = {

safetyLabelSource

.fetch(tweetId).map(

\_.map(

\_.labels

.map(

\_.map(sl => TweetSafetyLabel.fromTuple(sl.\_1, sl.\_2)).toSeq

).getOrElse(Seq())

).getOrElse(Seq()))

}

}

object NilTweetLabelMaps extends TweetLabels {

override def forTweet(tweet: Tweet): Stitch[Seq[TweetSafetyLabel]] = Stitch.Nil

override def forTweetId(tweetId: TweetId): Stitch[Seq[TweetSafetyLabel]] = Stitch.Nil

}

class TweetFeatures(tweetLabels: TweetLabels, statsReceiver: StatsReceiver) {

private[this] val scopedStatsReceiver = statsReceiver.scope("tweet\_features")

private[this] val requests = scopedStatsReceiver.counter("requests")

private[this] val tweetSafetyLabels =

scopedStatsReceiver.scope(TweetSafetyLabels.name).counter("requests")

private[this] val tweetTakedownReasons =

scopedStatsReceiver.scope(TweetTakedownReasons.name).counter("requests")

private[this] val tweetIsSelfReply =

scopedStatsReceiver.scope(TweetIsSelfReply.name).counter("requests")

private[this] val tweetTimestamp =

scopedStatsReceiver.scope(TweetTimestamp.name).counter("requests")

private[this] val tweetReplyToParentTweetDuration =

scopedStatsReceiver.scope(TweetReplyToParentTweetDuration.name).counter("requests")

private[this] val tweetReplyToRootTweetDuration =

scopedStatsReceiver.scope(TweetReplyToRootTweetDuration.name).counter("requests")

private[this] val tweetSemanticCoreAnnotations =

scopedStatsReceiver.scope(TweetSemanticCoreAnnotations.name).counter("requests")

private[this] val tweetId =

scopedStatsReceiver.scope(TweetId.name).counter("requests")

private[this] val tweetHasNsfwUser =

scopedStatsReceiver.scope(TweetHasNsfwUser.name).counter("requests")

private[this] val tweetHasNsfwAdmin =

scopedStatsReceiver.scope(TweetHasNsfwAdmin.name).counter("requests")

private[this] val tweetIsNullcast =

scopedStatsReceiver.scope(TweetIsNullcast.name).counter("requests")

private[this] val tweetHasMedia =

scopedStatsReceiver.scope(TweetHasMedia.name).counter("requests")

private[this] val tweetIsCommunity =

scopedStatsReceiver.scope(TweetIsCommunityTweet.name).counter("requests")

private[this] val tweetIsCollabInvitation =

scopedStatsReceiver.scope(TweetIsCollabInvitationTweet.name).counter("requests")

def forTweet(tweet: Tweet): FeatureMapBuilder => FeatureMapBuilder = {

forTweetWithoutSafetyLabels(tweet)

.andThen(\_.withFeature(TweetSafetyLabels, tweetLabels.forTweet(tweet)))

}

def forTweetWithoutSafetyLabels(tweet: Tweet): FeatureMapBuilder => FeatureMapBuilder = {

requests.incr()

tweetTakedownReasons.incr()

tweetIsSelfReply.incr()

tweetTimestamp.incr()

tweetReplyToParentTweetDuration.incr()

tweetReplyToRootTweetDuration.incr()

tweetSemanticCoreAnnotations.incr()

tweetId.incr()

tweetHasNsfwUser.incr()

tweetHasNsfwAdmin.incr()

tweetIsNullcast.incr()

tweetHasMedia.incr()

tweetIsCommunity.incr()

tweetIsCollabInvitation.incr()

\_.withConstantFeature(TweetTakedownReasons, tweet.takedownReasons.getOrElse(Seq.empty))

.withConstantFeature(TweetIsSelfReply, TweetFeatures.tweetIsSelfReply(tweet))

.withConstantFeature(TweetTimestamp, TweetFeatures.tweetTimestamp(tweet.id))

.withConstantFeature(

TweetReplyToParentTweetDuration,

TweetFeatures.tweetReplyToParentTweetDuration(tweet))

.withConstantFeature(

TweetReplyToRootTweetDuration,

TweetFeatures.tweetReplyToRootTweetDuration(tweet))

.withConstantFeature(

TweetSemanticCoreAnnotations,

TweetFeatures.tweetSemanticCoreAnnotations(tweet))

.withConstantFeature(TweetId, tweet.id)

.withConstantFeature(TweetHasNsfwUser, tweetHasNsfwUser(tweet))

.withConstantFeature(TweetHasNsfwAdmin, tweetHasNsfwAdmin(tweet))

.withConstantFeature(TweetIsNullcast, TweetFeatures.tweetIsNullcast(tweet))

.withConstantFeature(TweetHasMedia, tweetHasMedia(tweet))

.withConstantFeature(TweetIsCommunityTweet, tweetHasCommunity(tweet))

.withConstantFeature(TweetIsCollabInvitationTweet, tweetIsCollabInvitation(tweet))

}

def tweetHasNsfwUser(tweet: Tweet): Boolean =

tweet.coreData.exists(\_.nsfwUser)

def tweetHasNsfwAdmin(tweet: Tweet): Boolean =

tweet.coreData.exists(\_.nsfwAdmin)

def tweetHasMedia(tweet: Tweet): Boolean =

tweet.coreData.exists(\_.hasMedia.getOrElse(false))

def tweetHasCommunity(tweet: Tweet): Boolean = {

tweet.communities.exists(\_.communityIds.nonEmpty)

}

def tweetIsCollabInvitation(tweet: Tweet): Boolean = {

tweet.collabControl.exists(\_ match {

case CollabControl.CollabInvitation(\_) => true

case \_ => false

})

}

}