package com.twitter.visibility.builder.tweets

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.stitch.Stitch

import com.twitter.tweetypie.thriftscala.Tweet

import com.twitter.tweetypie.thriftscala.ConversationControl

import com.twitter.visibility.builder.FeatureMapBuilder

import com.twitter.visibility.builder.users.RelationshipFeatures

import com.twitter.visibility.common.InvitedToConversationRepo

import com.twitter.visibility.features.ConversationRootAuthorFollowsViewer

import com.twitter.visibility.features.TweetConversationViewerIsInvited

import com.twitter.visibility.features.TweetConversationViewerIsInvitedViaReplyMention

import com.twitter.visibility.features.TweetConversationViewerIsRootAuthor

import com.twitter.visibility.features.TweetHasByInvitationConversationControl

import com.twitter.visibility.features.TweetHasCommunityConversationControl

import com.twitter.visibility.features.TweetHasFollowersConversationControl

import com.twitter.visibility.features.ViewerFollowsConversationRootAuthor

class ConversationControlFeatures(

relationshipFeatures: RelationshipFeatures,

isInvitedToConversationRepository: InvitedToConversationRepo,

statsReceiver: StatsReceiver) {

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

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

private[this] val tweetCommunityConversationRequest =

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

private[this] val tweetByInvitationConversationRequest =

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

private[this] val tweetFollowersConversationRequest =

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

private[this] val rootAuthorFollowsViewer =

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

private[this] val viewerFollowsRootAuthor =

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

def isCommunityConversation(conversationControl: Option[ConversationControl]): Boolean =

conversationControl

.collect {

case \_: ConversationControl.Community =>

tweetCommunityConversationRequest.incr()

true

}.getOrElse(false)

def isByInvitationConversation(conversationControl: Option[ConversationControl]): Boolean =

conversationControl

.collect {

case \_: ConversationControl.ByInvitation =>

tweetByInvitationConversationRequest.incr()

true

}.getOrElse(false)

def isFollowersConversation(conversationControl: Option[ConversationControl]): Boolean =

conversationControl

.collect {

case \_: ConversationControl.Followers =>

tweetFollowersConversationRequest.incr()

true

}.getOrElse(false)

def conversationRootAuthorId(

conversationControl: Option[ConversationControl]

): Option[Long] =

conversationControl match {

case Some(ConversationControl.Community(community)) =>

Some(community.conversationTweetAuthorId)

case Some(ConversationControl.ByInvitation(byInvitation)) =>

Some(byInvitation.conversationTweetAuthorId)

case Some(ConversationControl.Followers(followers)) =>

Some(followers.conversationTweetAuthorId)

case \_ => None

}

def viewerIsRootAuthor(

conversationControl: Option[ConversationControl],

viewerIdOpt: Option[Long]

): Boolean =

(conversationRootAuthorId(conversationControl), viewerIdOpt) match {

case (Some(rootAuthorId), Some(viewerId)) if rootAuthorId == viewerId => true

case \_ => false

}

def viewerIsInvited(

conversationControl: Option[ConversationControl],

viewerId: Option[Long]

): Boolean = {

val invitedUserIds = conversationControl match {

case Some(ConversationControl.Community(community)) =>

community.invitedUserIds

case Some(ConversationControl.ByInvitation(byInvitation)) =>

byInvitation.invitedUserIds

case Some(ConversationControl.Followers(followers)) =>

followers.invitedUserIds

case \_ => Seq()

}

viewerId.exists(invitedUserIds.contains(\_))

}

def conversationAuthorFollows(

conversationControl: Option[ConversationControl],

viewerId: Option[Long]

): Stitch[Boolean] = {

val conversationAuthorId = conversationControl.collect {

case ConversationControl.Community(community) =>

community.conversationTweetAuthorId

}

conversationAuthorId match {

case Some(authorId) =>

rootAuthorFollowsViewer.incr()

relationshipFeatures.authorFollowsViewer(authorId, viewerId)

case None =>

Stitch.False

}

}

def followsConversationAuthor(

conversationControl: Option[ConversationControl],

viewerId: Option[Long]

): Stitch[Boolean] = {

val conversationAuthorId = conversationControl.collect {

case ConversationControl.Followers(followers) =>

followers.conversationTweetAuthorId

}

conversationAuthorId match {

case Some(authorId) =>

viewerFollowsRootAuthor.incr()

relationshipFeatures.viewerFollowsAuthor(authorId, viewerId)

case None =>

Stitch.False

}

}

def viewerIsInvitedViaReplyMention(

tweet: Tweet,

viewerIdOpt: Option[Long]

): Stitch[Boolean] = {

val conversationIdOpt: Option[Long] = tweet.conversationControl match {

case Some(ConversationControl.Community(community))

if community.inviteViaMention.contains(true) =>

tweet.coreData.flatMap(\_.conversationId)

case Some(ConversationControl.ByInvitation(invitation))

if invitation.inviteViaMention.contains(true) =>

tweet.coreData.flatMap(\_.conversationId)

case Some(ConversationControl.Followers(followers))

if followers.inviteViaMention.contains(true) =>

tweet.coreData.flatMap(\_.conversationId)

case \_ => None

}

(conversationIdOpt, viewerIdOpt) match {

case (Some(conversationId), Some(viewerId)) =>

isInvitedToConversationRepository(conversationId, viewerId)

case \_ => Stitch.False

}

}

def forTweet(tweet: Tweet, viewerId: Option[Long]): FeatureMapBuilder => FeatureMapBuilder = {

requests.incr()

val cc = tweet.conversationControl

\_.withConstantFeature(TweetHasCommunityConversationControl, isCommunityConversation(cc))

.withConstantFeature(TweetHasByInvitationConversationControl, isByInvitationConversation(cc))

.withConstantFeature(TweetHasFollowersConversationControl, isFollowersConversation(cc))

.withConstantFeature(TweetConversationViewerIsRootAuthor, viewerIsRootAuthor(cc, viewerId))

.withConstantFeature(TweetConversationViewerIsInvited, viewerIsInvited(cc, viewerId))

.withFeature(ConversationRootAuthorFollowsViewer, conversationAuthorFollows(cc, viewerId))

.withFeature(ViewerFollowsConversationRootAuthor, followsConversationAuthor(cc, viewerId))

.withFeature(

TweetConversationViewerIsInvitedViaReplyMention,

viewerIsInvitedViaReplyMention(tweet, viewerId))

}

}