package com.twitter.visibility.builder.tweets

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.stitch.Stitch

import com.twitter.visibility.builder.FeatureMapBuilder

import com.twitter.visibility.builder.users.ViewerVerbsAuthor

import com.twitter.visibility.common.UserId

import com.twitter.visibility.common.UserRelationshipSource

import com.twitter.visibility.features.\_

import com.twitter.visibility.models.TweetSafetyLabel

import com.twitter.visibility.models.ViolationLevel

class FosnrPefetchedLabelsRelationshipFeatures(

userRelationshipSource: UserRelationshipSource,

statsReceiver: StatsReceiver) {

private[this] val scopedStatsReceiver =

statsReceiver.scope("fonsr\_prefetched\_relationship\_features")

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

private[this] val viewerFollowsAuthorOfViolatingTweet =

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

private[this] val viewerDoesNotFollowAuthorOfViolatingTweet =

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

def forNonFosnr(): FeatureMapBuilder => FeatureMapBuilder = {

requests.incr()

\_.withConstantFeature(ViewerFollowsAuthorOfViolatingTweet, false)

.withConstantFeature(ViewerDoesNotFollowAuthorOfViolatingTweet, false)

}

def forTweetWithSafetyLabelsAndAuthorId(

safetyLabels: Seq[TweetSafetyLabel],

authorId: Long,

viewerId: Option[Long]

): FeatureMapBuilder => FeatureMapBuilder = {

requests.incr()

\_.withFeature(

ViewerFollowsAuthorOfViolatingTweet,

viewerFollowsAuthorOfViolatingTweet(safetyLabels, authorId, viewerId))

.withFeature(

ViewerDoesNotFollowAuthorOfViolatingTweet,

viewerDoesNotFollowAuthorOfViolatingTweet(safetyLabels, authorId, viewerId))

}

def viewerFollowsAuthorOfViolatingTweet(

safetyLabels: Seq[TweetSafetyLabel],

authorId: UserId,

viewerId: Option[UserId]

): Stitch[Boolean] = {

if (safetyLabels

.map(ViolationLevel.fromTweetSafetyLabelOpt).collect {

case Some(level) => level

}.isEmpty) {

return Stitch.False

}

ViewerVerbsAuthor(

authorId,

viewerId,

userRelationshipSource.follows,

viewerFollowsAuthorOfViolatingTweet)

}

def viewerDoesNotFollowAuthorOfViolatingTweet(

safetyLabels: Seq[TweetSafetyLabel],

authorId: UserId,

viewerId: Option[UserId]

): Stitch[Boolean] = {

if (safetyLabels

.map(ViolationLevel.fromTweetSafetyLabelOpt).collect {

case Some(level) => level

}.isEmpty) {

return Stitch.False

}

ViewerVerbsAuthor(

authorId,

viewerId,

userRelationshipSource.follows,

viewerDoesNotFollowAuthorOfViolatingTweet).map(following => !following)

}

}