package com.twitter.visibility.builder.users

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.gizmoduck.thriftscala.User

import com.twitter.stitch.Stitch

import com.twitter.visibility.builder.FeatureMapBuilder

import com.twitter.visibility.common.UserId

import com.twitter.visibility.common.UserRelationshipSource

import com.twitter.visibility.features.\_

class RelationshipFeatures(

userRelationshipSource: UserRelationshipSource,

statsReceiver: StatsReceiver) {

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

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

private[this] val authorFollowsViewer =

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

private[this] val viewerFollowsAuthor =

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

private[this] val authorBlocksViewer =

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

private[this] val viewerBlocksAuthor =

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

private[this] val authorMutesViewer =

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

private[this] val viewerMutesAuthor =

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

private[this] val authorHasReportedViewer =

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

private[this] val viewerHasReportedAuthor =

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

private[this] val viewerMutesRetweetsFromAuthor =

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

def forAuthorId(

authorId: Long,

viewerId: Option[Long]

): FeatureMapBuilder => FeatureMapBuilder = {

requests.incr()

\_.withFeature(AuthorFollowsViewer, authorFollowsViewer(authorId, viewerId))

.withFeature(ViewerFollowsAuthor, viewerFollowsAuthor(authorId, viewerId))

.withFeature(AuthorBlocksViewer, authorBlocksViewer(authorId, viewerId))

.withFeature(ViewerBlocksAuthor, viewerBlocksAuthor(authorId, viewerId))

.withFeature(AuthorMutesViewer, authorMutesViewer(authorId, viewerId))

.withFeature(ViewerMutesAuthor, viewerMutesAuthor(authorId, viewerId))

.withFeature(AuthorReportsViewerAsSpam, authorHasReportedViewer(authorId, viewerId))

.withFeature(ViewerReportsAuthorAsSpam, viewerHasReportedAuthor(authorId, viewerId))

.withFeature(ViewerMutesRetweetsFromAuthor, viewerMutesRetweetsFromAuthor(authorId, viewerId))

}

def forNoAuthor(): FeatureMapBuilder => FeatureMapBuilder = {

\_.withConstantFeature(AuthorFollowsViewer, false)

.withConstantFeature(ViewerFollowsAuthor, false)

.withConstantFeature(AuthorBlocksViewer, false)

.withConstantFeature(ViewerBlocksAuthor, false)

.withConstantFeature(AuthorMutesViewer, false)

.withConstantFeature(ViewerMutesAuthor, false)

.withConstantFeature(AuthorReportsViewerAsSpam, false)

.withConstantFeature(ViewerReportsAuthorAsSpam, false)

.withConstantFeature(ViewerMutesRetweetsFromAuthor, false)

}

def forAuthor(author: User, viewerId: Option[Long]): FeatureMapBuilder => FeatureMapBuilder = {

requests.incr()

\_.withFeature(AuthorFollowsViewer, authorFollowsViewer(author, viewerId))

.withFeature(ViewerFollowsAuthor, viewerFollowsAuthor(author, viewerId))

.withFeature(AuthorBlocksViewer, authorBlocksViewer(author, viewerId))

.withFeature(ViewerBlocksAuthor, viewerBlocksAuthor(author, viewerId))

.withFeature(AuthorMutesViewer, authorMutesViewer(author, viewerId))

.withFeature(ViewerMutesAuthor, viewerMutesAuthor(author, viewerId))

.withFeature(AuthorReportsViewerAsSpam, authorHasReportedViewer(author.id, viewerId))

.withFeature(ViewerReportsAuthorAsSpam, viewerHasReportedAuthor(author.id, viewerId))

.withFeature(ViewerMutesRetweetsFromAuthor, viewerMutesRetweetsFromAuthor(author, viewerId))

}

def viewerFollowsAuthor(authorId: UserId, viewerId: Option[UserId]): Stitch[Boolean] =

ViewerVerbsAuthor(authorId, viewerId, userRelationshipSource.follows, viewerFollowsAuthor)

def viewerFollowsAuthor(author: User, viewerId: Option[UserId]): Stitch[Boolean] =

ViewerVerbsAuthor(

author,

viewerId,

p => p.following,

userRelationshipSource.follows,

viewerFollowsAuthor)

def authorFollowsViewer(authorId: UserId, viewerId: Option[UserId]): Stitch[Boolean] =

AuthorVerbsViewer(authorId, viewerId, userRelationshipSource.follows, authorFollowsViewer)

def authorFollowsViewer(author: User, viewerId: Option[UserId]): Stitch[Boolean] =

AuthorVerbsViewer(

author,

viewerId,

p => p.followedBy,

userRelationshipSource.follows,

authorFollowsViewer)

def viewerBlocksAuthor(authorId: UserId, viewerId: Option[UserId]): Stitch[Boolean] =

ViewerVerbsAuthor(authorId, viewerId, userRelationshipSource.blocks, viewerBlocksAuthor)

def viewerBlocksAuthor(author: User, viewerId: Option[UserId]): Stitch[Boolean] =

ViewerVerbsAuthor(

author,

viewerId,

p => p.blocking,

userRelationshipSource.blocks,

viewerBlocksAuthor)

def authorBlocksViewer(authorId: UserId, viewerId: Option[UserId]): Stitch[Boolean] =

ViewerVerbsAuthor(authorId, viewerId, userRelationshipSource.blockedBy, authorBlocksViewer)

def authorBlocksViewer(author: User, viewerId: Option[UserId]): Stitch[Boolean] =

ViewerVerbsAuthor(

author,

viewerId,

p => p.blockedBy,

userRelationshipSource.blockedBy,

authorBlocksViewer)

def viewerMutesAuthor(authorId: UserId, viewerId: Option[UserId]): Stitch[Boolean] =

ViewerVerbsAuthor(authorId, viewerId, userRelationshipSource.mutes, viewerMutesAuthor)

def viewerMutesAuthor(author: User, viewerId: Option[UserId]): Stitch[Boolean] =

ViewerVerbsAuthor(

author,

viewerId,

p => p.muting,

userRelationshipSource.mutes,

viewerMutesAuthor)

def authorMutesViewer(authorId: UserId, viewerId: Option[UserId]): Stitch[Boolean] =

ViewerVerbsAuthor(authorId, viewerId, userRelationshipSource.mutedBy, authorMutesViewer)

def authorMutesViewer(author: User, viewerId: Option[UserId]): Stitch[Boolean] =

ViewerVerbsAuthor(

author,

viewerId,

p => p.mutedBy,

userRelationshipSource.mutedBy,

authorMutesViewer)

def viewerHasReportedAuthor(authorId: UserId, viewerId: Option[UserId]): Stitch[Boolean] =

ViewerVerbsAuthor(

authorId,

viewerId,

userRelationshipSource.reportsAsSpam,

viewerHasReportedAuthor)

def authorHasReportedViewer(authorId: UserId, viewerId: Option[UserId]): Stitch[Boolean] =

ViewerVerbsAuthor(

authorId,

viewerId,

userRelationshipSource.reportedAsSpamBy,

authorHasReportedViewer)

def viewerMutesRetweetsFromAuthor(authorId: UserId, viewerId: Option[UserId]): Stitch[Boolean] =

ViewerVerbsAuthor(

authorId,

viewerId,

userRelationshipSource.noRetweetsFrom,

viewerMutesRetweetsFromAuthor)

def viewerMutesRetweetsFromAuthor(author: User, viewerId: Option[UserId]): Stitch[Boolean] =

ViewerVerbsAuthor(

author,

viewerId,

p => p.noRetweetsFrom,

userRelationshipSource.noRetweetsFrom,

viewerMutesRetweetsFromAuthor)

}