package com.twitter.representationscorer.twistlyfeatures

import com.twitter.decider.SimpleRecipient

import com.twitter.finagle.stats.Stat

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.representationscorer.common.\_

import com.twitter.representationscorer.twistlyfeatures.Engagements.\_

import com.twitter.simclusters\_v2.common.SimClustersEmbeddingId.LongInternalId

import com.twitter.stitch.Stitch

import com.twitter.strato.generated.client.recommendations.user\_signal\_service.SignalsClientColumn

import com.twitter.strato.generated.client.recommendations.user\_signal\_service.SignalsClientColumn.Value

import com.twitter.usersignalservice.thriftscala.BatchSignalRequest

import com.twitter.usersignalservice.thriftscala.SignalRequest

import com.twitter.usersignalservice.thriftscala.SignalType

import com.twitter.util.Time

import scala.collection.mutable.ArrayBuffer

import com.twitter.usersignalservice.thriftscala.ClientIdentifier

class UserSignalServiceRecentEngagementsClient(

stratoClient: SignalsClientColumn,

decider: RepresentationScorerDecider,

stats: StatsReceiver) {

import UserSignalServiceRecentEngagementsClient.\_

private val signalStats = stats.scope("user-signal-service", "signal")

private val signalTypeStats: Map[SignalType, Stat] =

SignalType.list.map(s => (s, signalStats.scope(s.name).stat("size"))).toMap

def get(userId: UserId): Stitch[Engagements] = {

val request = buildRequest(userId)

stratoClient.fetcher.fetch(request).map(\_.v).lowerFromOption().map { response =>

val now = Time.now

val sevenDaysAgo = now - SevenDaysSpan

val thirtyDaysAgo = now - ThirtyDaysSpan

Engagements(

favs7d = getUserSignals(response, SignalType.TweetFavorite, sevenDaysAgo),

retweets7d = getUserSignals(response, SignalType.Retweet, sevenDaysAgo),

follows30d = getUserSignals(response, SignalType.AccountFollowWithDelay, thirtyDaysAgo),

shares7d = getUserSignals(response, SignalType.TweetShareV1, sevenDaysAgo),

replies7d = getUserSignals(response, SignalType.Reply, sevenDaysAgo),

originalTweets7d = getUserSignals(response, SignalType.OriginalTweet, sevenDaysAgo),

videoPlaybacks7d =

getUserSignals(response, SignalType.VideoView90dPlayback50V1, sevenDaysAgo),

block30d = getUserSignals(response, SignalType.AccountBlock, thirtyDaysAgo),

mute30d = getUserSignals(response, SignalType.AccountMute, thirtyDaysAgo),

report30d = getUserSignals(response, SignalType.TweetReport, thirtyDaysAgo),

dontlike30d = getUserSignals(response, SignalType.TweetDontLike, thirtyDaysAgo),

seeFewer30d = getUserSignals(response, SignalType.TweetSeeFewer, thirtyDaysAgo),

)

}

}

private def getUserSignals(

response: Value,

signalType: SignalType,

earliestValidTimestamp: Time

): Seq[UserSignal] = {

val signals = response.signalResponse

.getOrElse(signalType, Seq.empty)

.view

.filter(\_.timestamp > earliestValidTimestamp.inMillis)

.map(s => s.targetInternalId.collect { case LongInternalId(id) => (id, s.timestamp) })

.collect { case Some((id, engagedAt)) => UserSignal(id, engagedAt) }

.take(EngagementsToScore)

.force

signalTypeStats(signalType).add(signals.size)

signals

}

private def buildRequest(userId: Long) = {

val recipient = Some(SimpleRecipient(userId))

// Signals RSX always fetches

val requestSignals = ArrayBuffer(

SignalRequestFav,

SignalRequestRetweet,

SignalRequestFollow

)

// Signals under experimentation. We use individual deciders to disable them if necessary.

// If experiments are successful, they will become permanent.

if (decider.isAvailable(FetchSignalShareDeciderKey, recipient))

requestSignals.append(SignalRequestShare)

if (decider.isAvailable(FetchSignalReplyDeciderKey, recipient))

requestSignals.append(SignalRequestReply)

if (decider.isAvailable(FetchSignalOriginalTweetDeciderKey, recipient))

requestSignals.append(SignalRequestOriginalTweet)

if (decider.isAvailable(FetchSignalVideoPlaybackDeciderKey, recipient))

requestSignals.append(SignalRequestVideoPlayback)

if (decider.isAvailable(FetchSignalBlockDeciderKey, recipient))

requestSignals.append(SignalRequestBlock)

if (decider.isAvailable(FetchSignalMuteDeciderKey, recipient))

requestSignals.append(SignalRequestMute)

if (decider.isAvailable(FetchSignalReportDeciderKey, recipient))

requestSignals.append(SignalRequestReport)

if (decider.isAvailable(FetchSignalDontlikeDeciderKey, recipient))

requestSignals.append(SignalRequestDontlike)

if (decider.isAvailable(FetchSignalSeeFewerDeciderKey, recipient))

requestSignals.append(SignalRequestSeeFewer)

BatchSignalRequest(userId, requestSignals, Some(ClientIdentifier.RepresentationScorerHome))

}

}

object UserSignalServiceRecentEngagementsClient {

val FetchSignalShareDeciderKey = "representation\_scorer\_fetch\_signal\_share"

val FetchSignalReplyDeciderKey = "representation\_scorer\_fetch\_signal\_reply"

val FetchSignalOriginalTweetDeciderKey = "representation\_scorer\_fetch\_signal\_original\_tweet"

val FetchSignalVideoPlaybackDeciderKey = "representation\_scorer\_fetch\_signal\_video\_playback"

val FetchSignalBlockDeciderKey = "representation\_scorer\_fetch\_signal\_block"

val FetchSignalMuteDeciderKey = "representation\_scorer\_fetch\_signal\_mute"

val FetchSignalReportDeciderKey = "representation\_scorer\_fetch\_signal\_report"

val FetchSignalDontlikeDeciderKey = "representation\_scorer\_fetch\_signal\_dont\_like"

val FetchSignalSeeFewerDeciderKey = "representation\_scorer\_fetch\_signal\_see\_fewer"

val EngagementsToScore = 10

private val engagementsToScoreOpt: Option[Long] = Some(EngagementsToScore)

val SignalRequestFav: SignalRequest =

SignalRequest(engagementsToScoreOpt, SignalType.TweetFavorite)

val SignalRequestRetweet: SignalRequest = SignalRequest(engagementsToScoreOpt, SignalType.Retweet)

val SignalRequestFollow: SignalRequest =

SignalRequest(engagementsToScoreOpt, SignalType.AccountFollowWithDelay)

// New experimental signals

val SignalRequestShare: SignalRequest =

SignalRequest(engagementsToScoreOpt, SignalType.TweetShareV1)

val SignalRequestReply: SignalRequest = SignalRequest(engagementsToScoreOpt, SignalType.Reply)

val SignalRequestOriginalTweet: SignalRequest =

SignalRequest(engagementsToScoreOpt, SignalType.OriginalTweet)

val SignalRequestVideoPlayback: SignalRequest =

SignalRequest(engagementsToScoreOpt, SignalType.VideoView90dPlayback50V1)

// Negative signals

val SignalRequestBlock: SignalRequest =

SignalRequest(engagementsToScoreOpt, SignalType.AccountBlock)

val SignalRequestMute: SignalRequest =

SignalRequest(engagementsToScoreOpt, SignalType.AccountMute)

val SignalRequestReport: SignalRequest =

SignalRequest(engagementsToScoreOpt, SignalType.TweetReport)

val SignalRequestDontlike: SignalRequest =

SignalRequest(engagementsToScoreOpt, SignalType.TweetDontLike)

val SignalRequestSeeFewer: SignalRequest =

SignalRequest(engagementsToScoreOpt, SignalType.TweetSeeFewer)

}