package com.twitter.frigate.pushservice.predicate.ntab\_caret\_fatigue

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.frigate.common.predicate.ntab\_caret\_fatigue.NtabCaretClickFatiguePredicateHelper

import com.twitter.notificationservice.thriftscala.CaretFeedbackDetails

import com.twitter.util.Duration

import com.twitter.conversions.DurationOps.\_

import scala.math.min

import com.twitter.util.Time

import com.twitter.frigate.thriftscala.{CommonRecommendationType => CRT}

object NtabCaretClickFatigueUtils {

private def pushCapForFeedback(

feedbackDetails: Seq[CaretFeedbackDetails],

feedbacks: Seq[FeedbackModel],

param: ContinuousFunctionParam,

statsReceiver: StatsReceiver

): Double = {

val stats = statsReceiver.scope("mr\_seelessoften\_contfn\_pushcap")

val pushCapTotal = stats.counter("pushcap\_total")

val pushCapInvalid =

stats.counter("pushcap\_invalid")

pushCapTotal.incr()

val timeSinceMostRecentDislikeMs =

NtabCaretClickFatiguePredicateHelper.getDurationSinceMostRecentDislike(feedbackDetails)

val mostRecentFeedbackTimestamp: Option[Long] =

feedbacks

.map { feedback =>

feedback.timestampMs

}.reduceOption(\_ max \_)

val timeSinceMostRecentFeedback: Option[Duration] =

mostRecentFeedbackTimestamp.map(Time.now - Time.fromMilliseconds(\_))

val nTabDislikePushCap = timeSinceMostRecentDislikeMs match {

case Some(lastDislikeTimeMs) => {

ContinuousFunction.safeEvaluateFn(lastDislikeTimeMs.inDays.toDouble, param, stats)

}

case \_ => {

pushCapInvalid.incr()

param.defaultValue

}

}

val feedbackPushCap = timeSinceMostRecentFeedback match {

case Some(lastDislikeTimeVal) => {

ContinuousFunction.safeEvaluateFn(lastDislikeTimeVal.inDays.toDouble, param, stats)

}

case \_ => {

pushCapInvalid.incr()

param.defaultValue

}

}

min(nTabDislikePushCap, feedbackPushCap)

}

def durationToFilterForFeedback(

feedbackDetails: Seq[CaretFeedbackDetails],

feedbacks: Seq[FeedbackModel],

param: ContinuousFunctionParam,

defaultPushCap: Double,

statsReceiver: StatsReceiver

): Duration = {

val pushCap = min(

pushCapForFeedback(feedbackDetails, feedbacks, param, statsReceiver),

defaultPushCap

)

if (pushCap <= 0) {

Duration.Top

} else {

24.hours / pushCap

}

}

def hasUserDislikeInLast90Days(feedbackDetails: Seq[CaretFeedbackDetails]): Boolean = {

val timeSinceMostRecentDislike =

NtabCaretClickFatiguePredicateHelper.getDurationSinceMostRecentDislike(feedbackDetails)

timeSinceMostRecentDislike.exists(\_ < 90.days)

}

def feedbackModelFilterByCRT(

crts: Set[CRT]

): Seq[FeedbackModel] => Seq[

FeedbackModel

] = { feedbacks =>

feedbacks.filter { feedback =>

feedback.notification match {

case Some(notification) => crts.contains(notification.commonRecommendationType)

case None => false

}

}

}

def feedbackModelExcludeCRT(

crts: Set[CRT]

): Seq[FeedbackModel] => Seq[

FeedbackModel

] = { feedbacks =>

feedbacks.filter { feedback =>

feedback.notification match {

case Some(notification) => !crts.contains(notification.commonRecommendationType)

case None => true

}

}

}

}