package com.twitter.frigate.pushservice.predicate

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.frigate.common.base.\_

import com.twitter.frigate.common.rec\_types.RecTypes

import com.twitter.frigate.pushservice.model.PushTypes.PushCandidate

import com.twitter.frigate.pushservice.params.PushConstants.\_

import com.twitter.frigate.pushservice.params.PushFeatureSwitchParams

import com.twitter.frigate.pushservice.util.CandidateUtil

import com.twitter.hermit.predicate.NamedPredicate

import com.twitter.hermit.predicate.Predicate

import com.twitter.util.Future

object OONSpreadControlPredicate {

def oonTweetSpreadControlPredicate(

)(

implicit stats: StatsReceiver

): NamedPredicate[

PushCandidate with TweetCandidate with RecommendationType

] = {

val name = "oon\_tweet\_spread\_control\_predicate"

val scopedStatsReceiver = stats.scope(name)

val allOonCandidatesCounter = scopedStatsReceiver.counter("all\_oon\_candidates")

val filteredCandidatesCounter =

scopedStatsReceiver.counter("filtered\_oon\_candidates")

Predicate

.fromAsync { candidate: PushCandidate with TweetCandidate with RecommendationType =>

val target = candidate.target

val crt = candidate.commonRecType

val isOonCandidate = RecTypes.isOutOfNetworkTweetRecType(crt) ||

RecTypes.outOfNetworkTopicTweetTypes.contains(crt)

lazy val minTweetSendsThreshold =

target.params(PushFeatureSwitchParams.MinTweetSendsThresholdParam)

lazy val spreadControlRatio =

target.params(PushFeatureSwitchParams.SpreadControlRatioParam)

lazy val favOverSendThreshold =

target.params(PushFeatureSwitchParams.FavOverSendThresholdParam)

lazy val sentCount = candidate.numericFeatures.getOrElse(sentFeatureName, 0.0)

lazy val followerCount =

candidate.numericFeatures.getOrElse(authorActiveFollowerFeatureName, 0.0)

lazy val favCount = candidate.numericFeatures.getOrElse(favFeatureName, 0.0)

lazy val favOverSends = favCount / (sentCount + 1.0)

if (CandidateUtil.shouldApplyHealthQualityFilters(candidate) && isOonCandidate) {

allOonCandidatesCounter.incr()

if (sentCount > minTweetSendsThreshold &&

sentCount > spreadControlRatio \* followerCount &&

favOverSends < favOverSendThreshold) {

filteredCandidatesCounter.incr()

Future.False

} else Future.True

} else Future.True

}

.withStats(stats.scope(name))

.withName(name)

}

def oonAuthorSpreadControlPredicate(

)(

implicit stats: StatsReceiver

): NamedPredicate[

PushCandidate with TweetCandidate with RecommendationType

] = {

val name = "oon\_author\_spread\_control\_predicate"

val scopedStatsReceiver = stats.scope(name)

val allOonCandidatesCounter = scopedStatsReceiver.counter("all\_oon\_candidates")

val filteredCandidatesCounter =

scopedStatsReceiver.counter("filtered\_oon\_candidates")

Predicate

.fromAsync { candidate: PushCandidate with TweetCandidate with RecommendationType =>

val target = candidate.target

val crt = candidate.commonRecType

val isOonCandidate = RecTypes.isOutOfNetworkTweetRecType(crt) ||

RecTypes.outOfNetworkTopicTweetTypes.contains(crt)

lazy val minAuthorSendsThreshold =

target.params(PushFeatureSwitchParams.MinAuthorSendsThresholdParam)

lazy val spreadControlRatio =

target.params(PushFeatureSwitchParams.SpreadControlRatioParam)

lazy val reportRateThreshold =

target.params(PushFeatureSwitchParams.AuthorReportRateThresholdParam)

lazy val dislikeRateThreshold =

target.params(PushFeatureSwitchParams.AuthorDislikeRateThresholdParam)

lazy val authorSentCount =

candidate.numericFeatures.getOrElse(authorSendCountFeatureName, 0.0)

lazy val authorReportCount =

candidate.numericFeatures.getOrElse(authorReportCountFeatureName, 0.0)

lazy val authorDislikeCount =

candidate.numericFeatures.getOrElse(authorDislikeCountFeatureName, 0.0)

lazy val followerCount = candidate.numericFeatures

.getOrElse(authorActiveFollowerFeatureName, 0.0)

lazy val reportRate =

authorReportCount / (authorSentCount + 1.0)

lazy val dislikeRate =

authorDislikeCount / (authorSentCount + 1.0)

if (CandidateUtil.shouldApplyHealthQualityFilters(candidate) && isOonCandidate) {

allOonCandidatesCounter.incr()

if (authorSentCount > minAuthorSendsThreshold &&

authorSentCount > spreadControlRatio \* followerCount &&

(reportRate > reportRateThreshold || dislikeRate > dislikeRateThreshold)) {

filteredCandidatesCounter.incr()

Future.False

} else Future.True

} else Future.True

}

.withStats(stats.scope(name))

.withName(name)

}

}