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.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 OONTweetNegativeFeedbackBasedPredicate {

def ntabDislikeBasedPredicate(

)(

implicit stats: StatsReceiver

): NamedPredicate[

PushCandidate with TweetCandidate with RecommendationType

] = {

val name = "oon\_tweet\_dislike\_based\_predicate"

val scopedStatsReceiver = stats.scope(name)

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

val oonCandidatesImpressedCounter =

scopedStatsReceiver.counter("oon\_candidates\_impressed")

val filteredCandidatesCounter =

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

val ntabDislikeCountFeature =

"tweet.magic\_recs\_tweet\_real\_time\_aggregates\_v2.pair.v2.magicrecs.realtime.is\_ntab\_disliked.any\_feature.Duration.Top.count"

val sentFeature =

"tweet.magic\_recs\_tweet\_real\_time\_aggregates\_v2.pair.v2.magicrecs.realtime.is\_sent.any\_feature.Duration.Top.count"

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 ntabDislikeCountThreshold =

target.params(PushFeatureSwitchParams.TweetNtabDislikeCountThresholdParam)

lazy val ntabDislikeRateThreshold =

target.params(PushFeatureSwitchParams.TweetNtabDislikeRateThresholdParam)

lazy val ntabDislikeCountThresholdForMrTwistly =

target.params(PushFeatureSwitchParams.TweetNtabDislikeCountThresholdForMrTwistlyParam)

lazy val ntabDislikeRateThresholdForMrTwistly =

target.params(PushFeatureSwitchParams.TweetNtabDislikeRateThresholdForMrTwistlyParam)

val isMrTwistly = CandidateUtil.isMrTwistlyCandidate(candidate)

lazy val dislikeCount = candidate.numericFeatures.getOrElse(ntabDislikeCountFeature, 0.0)

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

lazy val dislikeRate = if (sentCount > 0) dislikeCount / sentCount else 0.0

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

allOonCandidatesCounter.incr()

val (countThreshold, rateThreshold) = if (isMrTwistly) {

(ntabDislikeCountThresholdForMrTwistly, ntabDislikeRateThresholdForMrTwistly)

} else {

(ntabDislikeCountThreshold, ntabDislikeRateThreshold)

}

candidate.cachePredicateInfo(

name + "\_count",

dislikeCount,

countThreshold,

dislikeCount > countThreshold)

candidate.cachePredicateInfo(

name + "\_rate",

dislikeRate,

rateThreshold,

dislikeRate > rateThreshold)

if (dislikeCount > countThreshold && dislikeRate > rateThreshold) {

filteredCandidatesCounter.incr()

Future.False

} else Future.True

} else Future.True

}

.withStats(stats.scope(name))

.withName(name)

}

}