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 TweetEngagementRatioPredicate {

def QTtoNtabClickBasedPredicate(

)(

implicit stats: StatsReceiver

): NamedPredicate[

PushCandidate with TweetCandidate with RecommendationType

] = {

val name = "oon\_tweet\_engagement\_filter\_qt\_to\_ntabclick\_ratio\_based\_predicate"

val scopedStatsReceiver = stats.scope(name)

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

val filteredCandidatesCounter =

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

val quoteCountFeature =

"tweet.core.tweet\_counts.quote\_count"

val ntabClickCountFeature =

"tweet.magic\_recs\_tweet\_real\_time\_aggregates\_v2.pair.v2.magicrecs.realtime.is\_ntab\_clicked.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 QTtoNtabClickRatioThreshold =

target.params(PushFeatureSwitchParams.TweetQTtoNtabClickRatioThresholdParam)

lazy val quoteCount = candidate.numericFeatures.getOrElse(quoteCountFeature, 0.0)

lazy val ntabClickCount = candidate.numericFeatures.getOrElse(ntabClickCountFeature, 0.0)

lazy val quoteRate = if (ntabClickCount > 0) quoteCount / ntabClickCount else 1.0

if (isOonCandidate) allOonCandidatesCounter.incr()

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

val ntabClickThreshold = 1000

candidate.cachePredicateInfo(

name + "\_count",

ntabClickCount,

ntabClickThreshold,

ntabClickCount >= ntabClickThreshold)

candidate.cachePredicateInfo(

name + "\_ratio",

quoteRate,

QTtoNtabClickRatioThreshold,

quoteRate < QTtoNtabClickRatioThreshold)

if (ntabClickCount >= ntabClickThreshold && quoteRate < QTtoNtabClickRatioThreshold) {

filteredCandidatesCounter.incr()

Future.False

} else Future.True

} else Future.True

}

.withStats(stats.scope(name))

.withName(name)

}

def TweetReplyLikeRatioPredicate(

)(

implicit stats: StatsReceiver

): NamedPredicate[PushCandidate with TweetCandidate] = {

val name = "tweet\_reply\_like\_ratio"

val scopedStatsReceiver = stats.scope(name)

val allCandidatesCounter = scopedStatsReceiver.counter("all\_candidates")

val filteredCandidatesCounter = scopedStatsReceiver.counter("filtered\_candidates")

val bucketedCandidatesCounter = scopedStatsReceiver.counter("bucketed\_candidates")

Predicate

.fromAsync { candidate: PushCandidate =>

allCandidatesCounter.incr()

val target = candidate.target

val likeCount = candidate.numericFeatures

.getOrElse(PushConstants.TweetLikesFeatureName, 0.0)

val replyCount = candidate.numericFeatures

.getOrElse(PushConstants.TweetRepliesFeatureName, 0.0)

val ratio = replyCount / likeCount.max(1)

val isOonCandidate = RecTypes.isOutOfNetworkTweetRecType(candidate.commonRecType) ||

RecTypes.outOfNetworkTopicTweetTypes.contains(candidate.commonRecType)

if (isOonCandidate

&& CandidateUtil.shouldApplyHealthQualityFilters(candidate)

&& replyCount > target.params(

PushFeatureSwitchParams.TweetReplytoLikeRatioReplyCountThreshold)) {

bucketedCandidatesCounter.incr()

if (ratio > target.params(

PushFeatureSwitchParams.TweetReplytoLikeRatioThresholdLowerBound)

&& ratio < target.params(

PushFeatureSwitchParams.TweetReplytoLikeRatioThresholdUpperBound)) {

filteredCandidatesCounter.incr()

Future.False

} else {

Future.True

}

} else {

Future.True

}

}

.withStats(stats.scope(s"predicate\_$name"))

.withName(name)

}

}