package com.twitter.frigate.pushservice.predicate

import com.twitter.abuse.detection.scoring.thriftscala.Model

import com.twitter.abuse.detection.scoring.thriftscala.TweetScoringRequest

import com.twitter.abuse.detection.scoring.thriftscala.TweetScoringResponse

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.frigate.common.base.TweetCandidate

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.params.PushParams

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

import com.twitter.hermit.predicate.NamedPredicate

import com.twitter.hermit.predicate.Predicate

import com.twitter.storehaus.ReadableStore

import com.twitter.util.Future

object PNegMultimodalPredicates {

def healthSignalScorePNegMultimodalPredicate(

tweetHealthScoreStore: ReadableStore[TweetScoringRequest, TweetScoringResponse]

)(

implicit stats: StatsReceiver

): NamedPredicate[PushCandidate with TweetCandidate] = {

val name = "pneg\_multimodal\_predicate"

val statsScope = stats.scope(name)

val oonCandidatesCounter = statsScope.counter("oon\_candidates")

val nonEmptyModelScoreCounter = statsScope.counter("non\_empty\_model\_score")

val bucketedCounter = statsScope.counter("bucketed\_oon\_candidates")

val filteredCounter = statsScope.counter("filtered\_oon\_candidates")

Predicate

.fromAsync { candidate: PushCandidate with TweetCandidate =>

val target = candidate.target

val crt = candidate.commonRecType

val isOonCandidate = RecTypes.isOutOfNetworkTweetRecType(crt) ||

RecTypes.outOfNetworkTopicTweetTypes.contains(crt)

lazy val enablePNegMultimodalPredicateParam =

target.params(PushFeatureSwitchParams.EnablePNegMultimodalPredicateParam)

lazy val pNegMultimodalPredicateModelThresholdParam =

target.params(PushFeatureSwitchParams.PNegMultimodalPredicateModelThresholdParam)

lazy val pNegMultimodalPredicateBucketThresholdParam =

target.params(PushFeatureSwitchParams.PNegMultimodalPredicateBucketThresholdParam)

val pNegMultimodalEnabledForF1Tweets =

target.params(PushParams.EnablePnegMultimodalPredictionForF1Tweets)

if (CandidateUtil.shouldApplyHealthQualityFilters(

candidate) && (isOonCandidate || pNegMultimodalEnabledForF1Tweets) && enablePNegMultimodalPredicateParam) {

val pNegMultimodalRequest = TweetScoringRequest(candidate.tweetId, Model.PNegMultimodal)

tweetHealthScoreStore.get(pNegMultimodalRequest).map {

case Some(tweetScoringResponse) =>

nonEmptyModelScoreCounter.incr()

val pNegMultimodalScore = 1.0 - tweetScoringResponse.score

candidate

.cacheExternalScore("PNegMultimodalScore", Future.value(Some(pNegMultimodalScore)))

if (isOonCandidate) {

oonCandidatesCounter.incr()

if (pNegMultimodalScore > pNegMultimodalPredicateBucketThresholdParam) {

bucketedCounter.incr()

if (pNegMultimodalScore > pNegMultimodalPredicateModelThresholdParam) {

filteredCounter.incr()

false

} else true

} else true

} else {

true

}

case \_ => true

}

} else {

Future.True

}

}

.withStats(stats.scope(name))

.withName(name)

}

}