package com.twitter.frigate.pushservice.predicate

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.\_

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

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

import com.twitter.frigate.pushservice.ml.HealthFeatureGetter

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

import com.twitter.hermit.predicate.NamedPredicate

import com.twitter.hermit.predicate.Predicate

import com.twitter.frigate.pushservice.ml.PushMLModelScorer

import com.twitter.frigate.pushservice.params.PushMLModel

import com.twitter.util.Future

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

import com.twitter.frigate.thriftscala.UserMediaRepresentation

import com.twitter.hss.api.thriftscala.UserHealthSignalResponse

import com.twitter.storehaus.ReadableStore

object BqmlHealthModelPredicates {

def healthModelOonPredicate(

bqmlHealthModelScorer: PushMLModelScorer,

producerMediaRepresentationStore: ReadableStore[Long, UserMediaRepresentation],

userHealthScoreStore: ReadableStore[Long, UserHealthSignalResponse],

tweetHealthScoreStore: ReadableStore[TweetScoringRequest, TweetScoringResponse]

)(

implicit stats: StatsReceiver

): NamedPredicate[

PushCandidate with TweetCandidate with RecommendationType with TweetAuthor

] = {

val name = "bqml\_health\_model\_based\_predicate"

val scopedStatsReceiver = stats.scope(name)

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

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

val filteredOonCandidatesCounter =

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

val emptyScoreCandidatesCounter = scopedStatsReceiver.counter("empty\_score\_candidates")

val healthScoreStat = scopedStatsReceiver.stat("health\_model\_dist")

Predicate

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

val target = candidate.target

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

RecTypes.outOfNetworkTopicTweetTypes.contains(candidate.commonRecType)

lazy val enableBqmlHealthModelPredicateParam =

target.params(PushFeatureSwitchParams.EnableBqmlHealthModelPredicateParam)

lazy val enableBqmlHealthModelPredictionForInNetworkCandidates =

target.params(

PushFeatureSwitchParams.EnableBqmlHealthModelPredictionForInNetworkCandidatesParam)

lazy val bqmlHealthModelPredicateFilterThresholdParam =

target.params(PushFeatureSwitchParams.BqmlHealthModelPredicateFilterThresholdParam)

lazy val healthModelId = target.params(PushFeatureSwitchParams.BqmlHealthModelTypeParam)

lazy val enableBqmlHealthModelScoreHistogramParam =

target.params(PushFeatureSwitchParams.EnableBqmlHealthModelScoreHistogramParam)

val healthModelScoreFeature = "bqml\_health\_model\_score"

val histogramBinSize = 0.05

lazy val healthCandidateScoreHistogramCounters =

bqmlHealthModelScorer.getScoreHistogramCounters(

scopedStatsReceiver,

"health\_score\_histogram",

histogramBinSize)

candidate match {

case candidate: PushCandidate with TweetAuthor with TweetAuthorDetails

if enableBqmlHealthModelPredicateParam && (isOonCandidate || enableBqmlHealthModelPredictionForInNetworkCandidates) =>

HealthFeatureGetter

.getFeatures(

candidate,

producerMediaRepresentationStore,

userHealthScoreStore,

Some(tweetHealthScoreStore))

.flatMap { healthFeatures =>

allCandidatesCounter.incr()

candidate.mergeFeatures(healthFeatures)

val healthModelScoreFutOpt =

if (candidate.numericFeatures.contains(healthModelScoreFeature)) {

Future.value(candidate.numericFeatures.get(healthModelScoreFeature))

} else

bqmlHealthModelScorer.singlePredicationForModelVersion(

healthModelId,

candidate

)

candidate.populateQualityModelScore(

PushMLModel.HealthNsfwProbability,

healthModelId,

healthModelScoreFutOpt

)

healthModelScoreFutOpt.map {

case Some(healthModelScore) =>

healthScoreStat.add((healthModelScore \* 10000).toFloat)

if (enableBqmlHealthModelScoreHistogramParam) {

healthCandidateScoreHistogramCounters(

math.ceil(healthModelScore / histogramBinSize).toInt).incr()

}

if (CandidateUtil.shouldApplyHealthQualityFilters(

candidate) && isOonCandidate) {

oonCandidatesCounter.incr()

val threshold = bqmlHealthModelPredicateFilterThresholdParam

candidate.cachePredicateInfo(

name,

healthModelScore,

threshold,

healthModelScore > threshold)

if (healthModelScore > threshold) {

filteredOonCandidatesCounter.incr()

false

} else true

} else true

case \_ =>

emptyScoreCandidatesCounter.incr()

true

}

}

case \_ => Future.True

}

}

.withStats(stats.scope(name))

.withName(name)

}

}