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.hermit.predicate.NamedPredicate

import com.twitter.hermit.predicate.Predicate

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

import com.twitter.frigate.pushservice.params.PushConstants.TweetMediaEmbeddingBQKeyIds

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

import com.twitter.frigate.pushservice.params.PushParams

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

import com.twitter.util.Future

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

object BqmlQualityModelPredicates {

def ingestExtraFeatures(cand: PushCandidate): Unit = {

val tagsCRCountFeature = "tagsCR\_count"

val hasPushOpenOrNtabClickFeature = "has\_PushOpenOrNtabClick"

val onlyPushOpenOrNtabClickFeature = "only\_PushOpenOrNtabClick"

val firstTweetMediaEmbeddingFeature = "media\_embedding\_0"

val tweetMediaEmbeddingFeature =

"media.mediaunderstanding.media\_embeddings.twitter\_clip\_as\_sparse\_continuous\_feature"

if (!cand.numericFeatures.contains(tagsCRCountFeature)) {

cand.numericFeatures(tagsCRCountFeature) = getTagsCRCount(cand)

}

if (!cand.booleanFeatures.contains(hasPushOpenOrNtabClickFeature)) {

cand.booleanFeatures(hasPushOpenOrNtabClickFeature) = isRelatedToMrTwistlyCandidate(cand)

}

if (!cand.booleanFeatures.contains(onlyPushOpenOrNtabClickFeature)) {

cand.booleanFeatures(onlyPushOpenOrNtabClickFeature) = isMrTwistlyCandidate(cand)

}

if (!cand.numericFeatures.contains(firstTweetMediaEmbeddingFeature)) {

val tweetMediaEmbedding = cand.sparseContinuousFeatures

.getOrElse(tweetMediaEmbeddingFeature, Map.empty[String, Double])

Seq.range(0, TweetMediaEmbeddingBQKeyIds.size).foreach { i =>

cand.numericFeatures(s"media\_embedding\_$i") =

tweetMediaEmbedding.getOrElse(TweetMediaEmbeddingBQKeyIds(i).toString, 0.0)

}

}

}

def BqmlQualityModelOonPredicate(

bqmlQualityModelScorer: PushMLModelScorer

)(

implicit stats: StatsReceiver

): NamedPredicate[

PushCandidate with TweetCandidate with RecommendationType

] = {

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

val scopedStatsReceiver = stats.scope(name)

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

val inCandidatesCounter = scopedStatsReceiver.counter("in\_candidates")

val filteredOonCandidatesCounter =

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

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

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

val histogramBinSize = 0.05

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 enableBqmlQualityModelScoreHistogramParam =

target.params(PushFeatureSwitchParams.EnableBqmlQualityModelScoreHistogramParam)

lazy val qualityCandidateScoreHistogramCounters =

bqmlQualityModelScorer.getScoreHistogramCounters(

scopedStatsReceiver,

"quality\_score\_histogram",

histogramBinSize)

if (CandidateUtil.shouldApplyHealthQualityFilters(candidate) && (isOonCandidate || target

.params(PushParams.EnableBqmlReportModelPredictionForF1Tweets))

&& target.params(PushFeatureSwitchParams.EnableBqmlQualityModelPredicateParam)) {

ingestExtraFeatures(candidate)

lazy val shouldFilterFutSeq =

target

.params(PushFeatureSwitchParams.BqmlQualityModelBucketModelIdListParam)

.zip(target.params(PushFeatureSwitchParams.BqmlQualityModelBucketThresholdListParam))

.map {

case (modelId, bucketThreshold) =>

val scoreFutOpt =

bqmlQualityModelScorer.singlePredicationForModelVersion(modelId, candidate)

candidate.populateQualityModelScore(

PushMLModel.FilteringProbability,

modelId,

scoreFutOpt

)

if (isOonCandidate) {

oonCandidatesCounter.incr()

scoreFutOpt.map {

case Some(score) =>

if (score >= bucketThreshold) {

bucketedCandidatesCounter.incr()

if (modelId == target.params(

PushFeatureSwitchParams.BqmlQualityModelTypeParam)) {

if (enableBqmlQualityModelScoreHistogramParam) {

val scoreHistogramBinId =

math.ceil(score / histogramBinSize).toInt

qualityCandidateScoreHistogramCounters(scoreHistogramBinId).incr()

}

if (score >= target.params(

PushFeatureSwitchParams.BqmlQualityModelPredicateThresholdParam)) {

filteredOonCandidatesCounter.incr()

true

} else false

} else false

} else false

case \_ =>

emptyScoreCandidatesCounter.incr()

false

}

} else {

inCandidatesCounter.incr()

Future.False

}

}

Future.collect(shouldFilterFutSeq).flatMap { shouldFilterSeq =>

if (shouldFilterSeq.contains(true)) {

Future.False

} else Future.True

}

} else Future.True

}

.withStats(stats.scope(name))

.withName(name)

}

}