package com.twitter.frigate.pushservice.predicate

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.frigate.common.base.\_

import com.twitter.frigate.common.candidate.TargetABDecider

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

import com.twitter.frigate.data\_pipeline.features\_common.MrRequestContextForFeatureStore

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.ml.featurestore.lib.dynamic.DynamicFeatureStoreClient

import com.twitter.util.Future

import com.twitter.frigate.pushservice.predicate.PostRankingPredicateHelper.\_

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

object OutOfNetworkCandidatesQualityPredicates {

def getTweetCharLengthThreshold(

target: TargetUser with TargetABDecider,

language: String,

useMediaThresholds: Boolean

): Double = {

lazy val sautOonWithMediaTweetLengthThreshold =

target.params(PushFeatureSwitchParams.SautOonWithMediaTweetLengthThresholdParam)

lazy val nonSautOonWithMediaTweetLengthThreshold =

target.params(PushFeatureSwitchParams.NonSautOonWithMediaTweetLengthThresholdParam)

lazy val sautOonWithoutMediaTweetLengthThreshold =

target.params(PushFeatureSwitchParams.SautOonWithoutMediaTweetLengthThresholdParam)

lazy val nonSautOonWithoutMediaTweetLengthThreshold =

target.params(PushFeatureSwitchParams.NonSautOonWithoutMediaTweetLengthThresholdParam)

val moreStrictForUndefinedLanguages =

target.params(PushFeatureSwitchParams.OonTweetLengthPredicateMoreStrictForUndefinedLanguages)

val isSautLanguage = if (moreStrictForUndefinedLanguages) {

isTweetLanguageInSautOrUndefined(language)

} else isTweetLanguageInSaut(language)

(useMediaThresholds, isSautLanguage) match {

case (true, true) =>

sautOonWithMediaTweetLengthThreshold

case (true, false) =>

nonSautOonWithMediaTweetLengthThreshold

case (false, true) =>

sautOonWithoutMediaTweetLengthThreshold

case (false, false) =>

nonSautOonWithoutMediaTweetLengthThreshold

case \_ => -1

}

}

def getTweetWordLengthThreshold(

target: TargetUser with TargetABDecider,

language: String,

useMediaThresholds: Boolean

): Double = {

lazy val argfOonWithMediaTweetWordLengthThresholdParam =

target.params(PushFeatureSwitchParams.ArgfOonWithMediaTweetWordLengthThresholdParam)

lazy val esfthOonWithMediaTweetWordLengthThresholdParam =

target.params(PushFeatureSwitchParams.EsfthOonWithMediaTweetWordLengthThresholdParam)

lazy val argfOonCandidatesWithMediaCondition =

isTweetLanguageInArgf(language) && useMediaThresholds

lazy val esfthOonCandidatesWithMediaCondition =

isTweetLanguageInEsfth(language) && useMediaThresholds

lazy val afirfOonCandidatesWithoutMediaCondition =

isTweetLanguageInAfirf(language) && !useMediaThresholds

val afirfOonCandidatesWithoutMediaTweetWordLengthThreshold = 5

if (argfOonCandidatesWithMediaCondition) {

argfOonWithMediaTweetWordLengthThresholdParam

} else if (esfthOonCandidatesWithMediaCondition) {

esfthOonWithMediaTweetWordLengthThresholdParam

} else if (afirfOonCandidatesWithoutMediaCondition) {

afirfOonCandidatesWithoutMediaTweetWordLengthThreshold

} else -1

}

def oonTweetLengthBasedPrerankingPredicate(

characterBased: Boolean

)(

implicit stats: StatsReceiver

): NamedPredicate[OutOfNetworkTweetCandidate with TargetInfo[

TargetUser with TargetABDecider

]] = {

val name = "oon\_tweet\_length\_based\_preranking\_predicate"

val scopedStats = stats.scope(s"${name}\_charBased\_$characterBased")

Predicate

.fromAsync {

cand: OutOfNetworkTweetCandidate with TargetInfo[TargetUser with TargetABDecider] =>

cand match {

case candidate: TweetAuthorDetails =>

val target = candidate.target

val crt = candidate.commonRecType

val updatedMediaLogic =

target.params(PushFeatureSwitchParams.OonTweetLengthPredicateUpdatedMediaLogic)

val updatedQuoteTweetLogic =

target.params(PushFeatureSwitchParams.OonTweetLengthPredicateUpdatedQuoteTweetLogic)

val useMediaThresholds = if (updatedMediaLogic || updatedQuoteTweetLogic) {

val hasMedia = updatedMediaLogic && (candidate.hasPhoto || candidate.hasVideo)

val hasQuoteTweet = updatedQuoteTweetLogic && candidate.quotedTweet.nonEmpty

hasMedia || hasQuoteTweet

} else RecTypes.isMediaType(crt)

val enableFilter =

target.params(PushFeatureSwitchParams.EnablePrerankingTweetLengthPredicate)

val language = candidate.tweet.flatMap(\_.language.map(\_.language)).getOrElse("")

val tweetTextOpt = candidate.tweet.flatMap(\_.coreData.map(\_.text))

val (length: Double, threshold: Double) = if (characterBased) {

(

tweetTextOpt.map(\_.size.toDouble).getOrElse(9999.0),

getTweetCharLengthThreshold(target, language, useMediaThresholds))

} else {

(

tweetTextOpt.map(getTweetWordLength).getOrElse(999.0),

getTweetWordLengthThreshold(target, language, useMediaThresholds))

}

scopedStats.counter("threshold\_" + threshold.toString).incr()

CandidateUtil.shouldApplyHealthQualityFiltersForPrerankingPredicates(candidate).map {

case true if enableFilter =>

length > threshold

case \_ => true

}

case \_ =>

scopedStats.counter("author\_is\_not\_hydrated").incr()

Future.True

}

}.withStats(scopedStats)

.withName(name)

}

private def isTweetLanguageInAfirf(candidateLanguage: String): Boolean = {

val setAFIRF: Set[String] = Set("")

setAFIRF.contains(candidateLanguage)

}

private def isTweetLanguageInEsfth(candidateLanguage: String): Boolean = {

val setESFTH: Set[String] = Set("")

setESFTH.contains(candidateLanguage)

}

private def isTweetLanguageInArgf(candidateLanguage: String): Boolean = {

val setARGF: Set[String] = Set("")

setARGF.contains(candidateLanguage)

}

private def isTweetLanguageInSaut(candidateLanguage: String): Boolean = {

val setSAUT = Set("")

setSAUT.contains(candidateLanguage)

}

private def isTweetLanguageInSautOrUndefined(candidateLanguage: String): Boolean = {

val setSautOrUndefined = Set("")

setSautOrUndefined.contains(candidateLanguage)

}

def containTargetNegativeKeywords(text: String, denylist: Seq[String]): Boolean = {

if (denylist.isEmpty)

false

else {

denylist

.map { negativeKeyword =>

text.toLowerCase().contains(negativeKeyword)

}.reduce(\_ || \_)

}

}

def NegativeKeywordsPredicate(

postRankingFeatureStoreClient: DynamicFeatureStoreClient[MrRequestContextForFeatureStore]

)(

implicit stats: StatsReceiver

): NamedPredicate[

PushCandidate with TweetCandidate with RecommendationType

] = {

val name = "negative\_keywords\_predicate"

val scopedStatsReceiver = stats.scope(name)

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

val filteredOonCandidatesCounter = scopedStatsReceiver.counter("filtered\_oon\_candidates")

val tweetLanguageFeature = "RecTweet.TweetyPieResult.Language"

Predicate

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

val target = candidate.target

val crt = candidate.commonRecType

val isTwistlyCandidate = RecTypes.twistlyTweets.contains(crt)

lazy val enableNegativeKeywordsPredicateParam =

target.params(PushFeatureSwitchParams.EnableNegativeKeywordsPredicateParam)

lazy val negativeKeywordsPredicateDenylist =

target.params(PushFeatureSwitchParams.NegativeKeywordsPredicateDenylist)

lazy val candidateLanguage =

candidate.categoricalFeatures.getOrElse(tweetLanguageFeature, "")

if (CandidateUtil.shouldApplyHealthQualityFilters(candidate) && candidateLanguage.equals(

"en") && isTwistlyCandidate && enableNegativeKeywordsPredicateParam) {

allOonCandidatesCounter.incr()

val tweetTextFuture: Future[String] =

getTweetText(candidate, postRankingFeatureStoreClient)

tweetTextFuture.map { tweetText =>

val containsNegativeWords =

containTargetNegativeKeywords(tweetText, negativeKeywordsPredicateDenylist)

candidate.cachePredicateInfo(

name,

if (containsNegativeWords) 1.0 else 0.0,

0.0,

containsNegativeWords)

if (containsNegativeWords) {

filteredOonCandidatesCounter.incr()

false

} else true

}

} else Future.True

}

.withStats(stats.scope(name))

.withName(name)

}

}