package com.twitter.frigate.pushservice.ml

import com.twitter.abuse.detection.scoring.thriftscala.{Model => TweetHealthModel}

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

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

import com.twitter.frigate.common.base.FeatureMap

import com.twitter.frigate.common.base.TweetAuthor

import com.twitter.frigate.common.base.TweetAuthorDetails

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.PushConstants

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

import com.twitter.frigate.pushservice.predicate.HealthPredicates.userHealthSignalValueToDouble

import com.twitter.frigate.pushservice.util.CandidateHydrationUtil

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

import com.twitter.frigate.pushservice.util.MediaAnnotationsUtil

import com.twitter.frigate.thriftscala.UserMediaRepresentation

import com.twitter.hss.api.thriftscala.SignalValue

import com.twitter.hss.api.thriftscala.UserHealthSignal

import com.twitter.hss.api.thriftscala.UserHealthSignal.AgathaCalibratedNsfwDouble

import com.twitter.hss.api.thriftscala.UserHealthSignal.NsfwTextUserScoreDouble

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

import com.twitter.storehaus.ReadableStore

import com.twitter.util.Future

import com.twitter.util.Time

object HealthFeatureGetter {

def getFeatures(

pushCandidate: PushCandidate,

producerMediaRepresentationStore: ReadableStore[Long, UserMediaRepresentation],

userHealthScoreStore: ReadableStore[Long, UserHealthSignalResponse],

tweetHealthScoreStoreOpt: Option[ReadableStore[TweetScoringRequest, TweetScoringResponse]] =

None

): Future[FeatureMap] = {

pushCandidate match {

case cand: PushCandidate with TweetCandidate with TweetAuthor with TweetAuthorDetails =>

val pMediaNsfwRequest =

TweetScoringRequest(cand.tweetId, TweetHealthModel.ExperimentalHealthModelScore4)

val pTweetTextNsfwRequest =

TweetScoringRequest(cand.tweetId, TweetHealthModel.ExperimentalHealthModelScore1)

cand.authorId match {

case Some(authorId) =>

Future

.join(

userHealthScoreStore.get(authorId),

producerMediaRepresentationStore.get(authorId),

tweetHealthScoreStoreOpt.map(\_.get(pMediaNsfwRequest)).getOrElse(Future.None),

tweetHealthScoreStoreOpt.map(\_.get(pTweetTextNsfwRequest)).getOrElse(Future.None),

cand.tweetAuthor

).map {

case (

healthSignalsResponseOpt,

producerMuOpt,

pMediaNsfwOpt,

pTweetTextNsfwOpt,

tweetAuthorOpt) =>

val healthSignalScoreMap = healthSignalsResponseOpt

.map(\_.signalValues).getOrElse(Map.empty[UserHealthSignal, SignalValue])

val agathaNSFWScore = userHealthSignalValueToDouble(

healthSignalScoreMap

.getOrElse(AgathaCalibratedNsfwDouble, SignalValue.DoubleValue(0.5)))

val userTextNSFWScore = userHealthSignalValueToDouble(

healthSignalScoreMap

.getOrElse(NsfwTextUserScoreDouble, SignalValue.DoubleValue(0.15)))

val pMediaNsfwScore = pMediaNsfwOpt.map(\_.score).getOrElse(0.0)

val pTweetTextNsfwScore = pTweetTextNsfwOpt.map(\_.score).getOrElse(0.0)

val mediaRepresentationMap =

producerMuOpt.map(\_.mediaRepresentation).getOrElse(Map.empty[String, Double])

val sumScore: Double = mediaRepresentationMap.values.sum

val nudityRate =

if (sumScore > 0)

mediaRepresentationMap.getOrElse(

MediaAnnotationsUtil.nudityCategoryId,

0.0) / sumScore

else 0.0

val beautyRate =

if (sumScore > 0)

mediaRepresentationMap.getOrElse(

MediaAnnotationsUtil.beautyCategoryId,

0.0) / sumScore

else 0.0

val singlePersonRate =

if (sumScore > 0)

mediaRepresentationMap.getOrElse(

MediaAnnotationsUtil.singlePersonCategoryId,

0.0) / sumScore

else 0.0

val dislikeCt = cand.numericFeatures.getOrElse(

"tweet.magic\_recs\_tweet\_real\_time\_aggregates\_v2.pair.v2.magicrecs.realtime.is\_ntab\_disliked.any\_feature.Duration.Top.count",

0.0)

val sentCt = cand.numericFeatures.getOrElse(

"tweet.magic\_recs\_tweet\_real\_time\_aggregates\_v2.pair.v2.magicrecs.realtime.is\_sent.any\_feature.Duration.Top.count",

0.0)

val dislikeRate = if (sentCt > 0) dislikeCt / sentCt else 0.0

val authorDislikeCt = cand.numericFeatures.getOrElse(

"tweet\_author\_aggregate.pair.label.ntab.isDisliked.any\_feature.28.days.count",

0.0)

val authorReportCt = cand.numericFeatures.getOrElse(

"tweet\_author\_aggregate.pair.label.reportTweetDone.any\_feature.28.days.count",

0.0)

val authorSentCt = cand.numericFeatures

.getOrElse(

"tweet\_author\_aggregate.pair.any\_label.any\_feature.28.days.count",

0.0)

val authorDislikeRate =

if (authorSentCt > 0) authorDislikeCt / authorSentCt else 0.0

val authorReportRate =

if (authorSentCt > 0) authorReportCt / authorSentCt else 0.0

val (isNsfwAccount, authorAccountAge) = tweetAuthorOpt match {

case Some(tweetAuthor) =>

(

CandidateHydrationUtil.isNsfwAccount(

tweetAuthor,

cand.target.params(PushFeatureSwitchParams.NsfwTokensParam)),

(Time.now - Time.fromMilliseconds(tweetAuthor.createdAtMsec)).inHours

)

case \_ => (false, 0)

}

val tweetSemanticCoreIds = cand.sparseBinaryFeatures

.getOrElse(PushConstants.TweetSemanticCoreIdFeature, Set.empty[String])

val continuousFeatures = Map[String, Double](

"agathaNsfwScore" -> agathaNSFWScore,

"textNsfwScore" -> userTextNSFWScore,

"pMediaNsfwScore" -> pMediaNsfwScore,

"pTweetTextNsfwScore" -> pTweetTextNsfwScore,

"nudityRate" -> nudityRate,

"beautyRate" -> beautyRate,

"singlePersonRate" -> singlePersonRate,

"numSources" -> CandidateUtil.getTagsCRCount(cand),

"favCount" -> cand.numericFeatures

.getOrElse("tweet.core.tweet\_counts.favorite\_count", 0.0),

"activeFollowers" -> cand.numericFeatures

.getOrElse("RecTweetAuthor.User.ActiveFollowers", 0.0),

"favorsRcvd28Days" -> cand.numericFeatures

.getOrElse("RecTweetAuthor.User.FavorsRcvd28Days", 0.0),

"tweets28Days" -> cand.numericFeatures

.getOrElse("RecTweetAuthor.User.Tweets28Days", 0.0),

"dislikeCount" -> dislikeCt,

"dislikeRate" -> dislikeRate,

"sentCount" -> sentCt,

"authorDislikeCount" -> authorDislikeCt,

"authorDislikeRate" -> authorDislikeRate,

"authorReportCount" -> authorReportCt,

"authorReportRate" -> authorReportRate,

"authorSentCount" -> authorSentCt,

"authorAgeInHour" -> authorAccountAge.toDouble

)

val booleanFeatures = Map[String, Boolean](

"isSimclusterBased" -> RecTypes.simclusterBasedTweets

.contains(cand.commonRecType),

"isTopicTweet" -> RecTypes.isTopicTweetType(cand.commonRecType),

"isHashSpace" -> RecTypes.tagspaceTypes.contains(cand.commonRecType),

"isFRS" -> RecTypes.frsTypes.contains(cand.commonRecType),

"isModelingBased" -> RecTypes.mrModelingBasedTypes.contains(cand.commonRecType),

"isGeoPop" -> RecTypes.GeoPopTweetTypes.contains(cand.commonRecType),

"hasPhoto" -> cand.booleanFeatures

.getOrElse("RecTweet.TweetyPieResult.HasPhoto", false),

"hasVideo" -> cand.booleanFeatures

.getOrElse("RecTweet.TweetyPieResult.HasVideo", false),

"hasUrl" -> cand.booleanFeatures

.getOrElse("RecTweet.TweetyPieResult.HasUrl", false),

"isMrTwistly" -> CandidateUtil.isMrTwistlyCandidate(cand),

"abuseStrikeTop2Percent" -> tweetSemanticCoreIds.contains(

PushConstants.AbuseStrike\_Top2Percent\_Id),

"abuseStrikeTop1Percent" -> tweetSemanticCoreIds.contains(

PushConstants.AbuseStrike\_Top1Percent\_Id),

"abuseStrikeTop05Percent" -> tweetSemanticCoreIds.contains(

PushConstants.AbuseStrike\_Top05Percent\_Id),

"abuseStrikeTop025Percent" -> tweetSemanticCoreIds.contains(

PushConstants.AbuseStrike\_Top025Percent\_Id),

"allSpamReportsPerFavTop1Percent" -> tweetSemanticCoreIds.contains(

PushConstants.AllSpamReportsPerFav\_Top1Percent\_Id),

"reportsPerFavTop1Percent" -> tweetSemanticCoreIds.contains(

PushConstants.ReportsPerFav\_Top1Percent\_Id),

"reportsPerFavTop2Percent" -> tweetSemanticCoreIds.contains(

PushConstants.ReportsPerFav\_Top2Percent\_Id),

"isNudity" -> tweetSemanticCoreIds.contains(

PushConstants.MediaUnderstanding\_Nudity\_Id),

"beautyStyleFashion" -> tweetSemanticCoreIds.contains(

PushConstants.MediaUnderstanding\_Beauty\_Id),

"singlePerson" -> tweetSemanticCoreIds.contains(

PushConstants.MediaUnderstanding\_SinglePerson\_Id),

"pornList" -> tweetSemanticCoreIds.contains(PushConstants.PornList\_Id),

"pornographyAndNsfwContent" -> tweetSemanticCoreIds.contains(

PushConstants.PornographyAndNsfwContent\_Id),

"sexLife" -> tweetSemanticCoreIds.contains(PushConstants.SexLife\_Id),

"sexLifeOrSexualOrientation" -> tweetSemanticCoreIds.contains(

PushConstants.SexLifeOrSexualOrientation\_Id),

"profanity" -> tweetSemanticCoreIds.contains(PushConstants.ProfanityFilter\_Id),

"isVerified" -> cand.booleanFeatures

.getOrElse("RecTweetAuthor.User.IsVerified", false),

"hasNsfwToken" -> isNsfwAccount

)

val stringFeatures = Map[String, String](

"tweetLanguage" -> cand.categoricalFeatures

.getOrElse("tweet.core.tweet\_text.language", "")

)

FeatureMap(

booleanFeatures = booleanFeatures,

numericFeatures = continuousFeatures,

categoricalFeatures = stringFeatures)

}

case \_ => Future.value(FeatureMap())

}

case \_ => Future.value(FeatureMap())

}

}

}