package com.twitter.cr\_mixer.util

import com.twitter.cr\_mixer.model.RankedCandidate

import com.twitter.cr\_mixer.model.SimilarityEngineInfo

import com.twitter.cr\_mixer.model.SourceInfo

import com.twitter.cr\_mixer.thriftscala.MetricTag

import com.twitter.cr\_mixer.thriftscala.SimilarityEngineType

import com.twitter.cr\_mixer.thriftscala.SourceType

object MetricTagUtil {

def buildMetricTags(candidate: RankedCandidate): Seq[MetricTag] = {

val interestedInMetricTag = isFromInterestedIn(candidate)

val cgInfoMetricTags = candidate.potentialReasons

.flatMap { cgInfo =>

val sourceMetricTag = cgInfo.sourceInfoOpt.flatMap { sourceInfo =>

toMetricTagFromSource(sourceInfo.sourceType)

}

val similarityEngineTags = toMetricTagFromSimilarityEngine(

cgInfo.similarityEngineInfo,

cgInfo.contributingSimilarityEngines)

val combinedMetricTag = cgInfo.sourceInfoOpt.flatMap { sourceInfo =>

toMetricTagFromSourceAndSimilarityEngine(sourceInfo, cgInfo.similarityEngineInfo)

}

Seq(sourceMetricTag) ++ similarityEngineTags ++ Seq(combinedMetricTag)

}.flatten.toSet

(interestedInMetricTag ++ cgInfoMetricTags).toSeq

}

/\*\*\*

\* match a sourceType to a metricTag

\*/

private def toMetricTagFromSource(sourceType: SourceType): Option[MetricTag] = {

sourceType match {

case SourceType.TweetFavorite => Some(MetricTag.TweetFavorite) // Personalized Topics in Home

case SourceType.Retweet => Some(MetricTag.Retweet) // Personalized Topics in Home

case SourceType.NotificationClick =>

Some(MetricTag.PushOpenOrNtabClick) // Health Filter in MR

case SourceType.OriginalTweet =>

Some(MetricTag.OriginalTweet)

case SourceType.Reply =>

Some(MetricTag.Reply)

case SourceType.TweetShare =>

Some(MetricTag.TweetShare)

case SourceType.UserFollow =>

Some(MetricTag.UserFollow)

case SourceType.UserRepeatedProfileVisit =>

Some(MetricTag.UserRepeatedProfileVisit)

case SourceType.TwiceUserId =>

Some(MetricTag.TwiceUserId)

case \_ => None

}

}

/\*\*\*

\* If the SEInfo is built by a unified sim engine, we un-wrap the contributing sim engines.

\* If not, we log the sim engine as usual.

\* @param seInfo (CandidateGenerationInfo.similarityEngineInfo): SimilarityEngineInfo,

\* @param cseInfo (CandidateGenerationInfo.contributingSimilarityEngines): Seq[SimilarityEngineInfo]

\*/

private def toMetricTagFromSimilarityEngine(

seInfo: SimilarityEngineInfo,

cseInfo: Seq[SimilarityEngineInfo]

): Seq[Option[MetricTag]] = {

seInfo.similarityEngineType match {

case SimilarityEngineType.TweetBasedUnifiedSimilarityEngine => // un-wrap the unified sim engine

cseInfo.map { contributingSimEngine =>

toMetricTagFromSimilarityEngine(contributingSimEngine, Seq.empty)

}.flatten

case SimilarityEngineType.ProducerBasedUnifiedSimilarityEngine => // un-wrap the unified sim engine

cseInfo.map { contributingSimEngine =>

toMetricTagFromSimilarityEngine(contributingSimEngine, Seq.empty)

}.flatten

// SimClustersANN can either be called on its own, or be called under unified sim engine

case SimilarityEngineType.SimClustersANN => // the old "UserInterestedIn" will be replaced by this. Also, OfflineTwice

Seq(Some(MetricTag.SimClustersANN), seInfo.modelId.flatMap(toMetricTagFromModelId(\_)))

case SimilarityEngineType.ConsumerEmbeddingBasedTwHINANN =>

Seq(Some(MetricTag.ConsumerEmbeddingBasedTwHINANN))

case SimilarityEngineType.TwhinCollabFilter => Seq(Some(MetricTag.TwhinCollabFilter))

// In the current implementation, TweetBasedUserTweetGraph/TweetBasedTwHINANN has a tag when

// it's either a base SE or a contributing SE. But for now they only show up in contributing SE.

case SimilarityEngineType.TweetBasedUserTweetGraph =>

Seq(Some(MetricTag.TweetBasedUserTweetGraph))

case SimilarityEngineType.TweetBasedTwHINANN =>

Seq(Some(MetricTag.TweetBasedTwHINANN))

case \_ => Seq.empty

}

}

/\*\*\*

\* pass in a model id, and match it with the metric tag type.

\*/

private def toMetricTagFromModelId(

modelId: String

): Option[MetricTag] = {

val pushOpenBasedModelRegex = "(.\*\_Model20m145k2020\_20220819)".r

modelId match {

case pushOpenBasedModelRegex(\_\*) =>

Some(MetricTag.RequestHealthFilterPushOpenBasedTweetEmbedding)

case \_ => None

}

}

private def toMetricTagFromSourceAndSimilarityEngine(

sourceInfo: SourceInfo,

seInfo: SimilarityEngineInfo

): Option[MetricTag] = {

sourceInfo.sourceType match {

case SourceType.Lookalike

if seInfo.similarityEngineType == SimilarityEngineType.ConsumersBasedUserTweetGraph =>

Some(MetricTag.LookalikeUTG)

case \_ => None

}

}

/\*\*

\* Special use case: used by Notifications team to generate the UserInterestedIn CRT push copy.

\*

\* if we have different types of InterestedIn (eg. UserInterestedIn, NextInterestedIn),

\* this if statement will have to be refactored to contain the real UserInterestedIn.

\* @return

\*/

private def isFromInterestedIn(candidate: RankedCandidate): Set[MetricTag] = {

if (candidate.reasonChosen.sourceInfoOpt.isEmpty

&& candidate.reasonChosen.similarityEngineInfo.similarityEngineType == SimilarityEngineType.SimClustersANN) {

Set(MetricTag.UserInterestedIn)

} else Set.empty

}

}