package com.twitter.frigate.pushservice.adaptor

import com.twitter.contentrecommender.thriftscala.MetricTag

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

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

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

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

import com.twitter.cr\_mixer.thriftscala.{MetricTag => CrMixerMetricTag}

import com.twitter.finagle.stats.Stat

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.frigate.common.base.AlgorithmScore

import com.twitter.frigate.common.base.CandidateSource

import com.twitter.frigate.common.base.CandidateSourceEligible

import com.twitter.frigate.common.base.CrMixerCandidate

import com.twitter.frigate.common.base.TopicCandidate

import com.twitter.frigate.common.base.TopicProofTweetCandidate

import com.twitter.frigate.common.base.TweetCandidate

import com.twitter.frigate.common.predicate.CommonOutNetworkTweetCandidatesSourcePredicates.filterOutInNetworkTweets

import com.twitter.frigate.common.predicate.CommonOutNetworkTweetCandidatesSourcePredicates.filterOutReplyTweet

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

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

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

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

import com.twitter.frigate.pushservice.store.CrMixerTweetStore

import com.twitter.frigate.pushservice.store.UttEntityHydrationStore

import com.twitter.frigate.pushservice.util.AdaptorUtils

import com.twitter.frigate.pushservice.util.PushDeviceUtil

import com.twitter.frigate.pushservice.util.TopicsUtil

import com.twitter.frigate.pushservice.util.TweetWithTopicProof

import com.twitter.frigate.thriftscala.CommonRecommendationType

import com.twitter.hermit.predicate.socialgraph.RelationEdge

import com.twitter.product\_mixer.core.thriftscala.ClientContext

import com.twitter.stitch.tweetypie.TweetyPie.TweetyPieResult

import com.twitter.storehaus.ReadableStore

import com.twitter.topiclisting.utt.LocalizedEntity

import com.twitter.tsp.thriftscala.TopicSocialProofRequest

import com.twitter.tsp.thriftscala.TopicSocialProofResponse

import com.twitter.util.Future

import scala.collection.Map

case class ContentRecommenderMixerAdaptor(

crMixerTweetStore: CrMixerTweetStore,

tweetyPieStore: ReadableStore[Long, TweetyPieResult],

edgeStore: ReadableStore[RelationEdge, Boolean],

topicSocialProofServiceStore: ReadableStore[TopicSocialProofRequest, TopicSocialProofResponse],

uttEntityHydrationStore: UttEntityHydrationStore,

globalStats: StatsReceiver)

extends CandidateSource[Target, RawCandidate]

with CandidateSourceEligible[Target, RawCandidate] {

override val name: String = this.getClass.getSimpleName

private[this] val stats = globalStats.scope("ContentRecommenderMixerAdaptor")

private[this] val numOfValidAuthors = stats.stat("num\_of\_valid\_authors")

private[this] val numOutOfMaximumDropped = stats.stat("dropped\_due\_out\_of\_maximum")

private[this] val totalInputRecs = stats.counter("input\_recs")

private[this] val totalOutputRecs = stats.stat("output\_recs")

private[this] val totalRequests = stats.counter("total\_requests")

private[this] val nonReplyTweetsCounter = stats.counter("non\_reply\_tweets")

private[this] val totalOutNetworkRecs = stats.counter("out\_network\_tweets")

private[this] val totalInNetworkRecs = stats.counter("in\_network\_tweets")

/\*\*

\* Builds OON raw candidates based on input OON Tweets

\*/

def buildOONRawCandidates(

inputTarget: Target,

oonTweets: Seq[TweetyPieResult],

tweetScoreMap: Map[Long, Double],

tweetIdToTagsMap: Map[Long, Seq[CrMixerMetricTag]],

maxNumOfCandidates: Int

): Option[Seq[RawCandidate]] = {

val cands = oonTweets.flatMap { tweetResult =>

val tweetId = tweetResult.tweet.id

generateOONRawCandidate(

inputTarget,

tweetId,

Some(tweetResult),

tweetScoreMap,

tweetIdToTagsMap

)

}

val candidates = restrict(

maxNumOfCandidates,

cands,

numOutOfMaximumDropped,

totalOutputRecs

)

Some(candidates)

}

/\*\*

\* Builds a single RawCandidate With TopicProofTweetCandidate

\*/

def buildTopicTweetRawCandidate(

inputTarget: Target,

tweetWithTopicProof: TweetWithTopicProof,

localizedEntity: LocalizedEntity,

tags: Option[Seq[MetricTag]],

): RawCandidate with TopicProofTweetCandidate = {

new RawCandidate with TopicProofTweetCandidate {

override def target: Target = inputTarget

override def topicListingSetting: Option[String] = Some(

tweetWithTopicProof.topicListingSetting)

override def tweetId: Long = tweetWithTopicProof.tweetId

override def tweetyPieResult: Option[TweetyPieResult] = Some(

tweetWithTopicProof.tweetyPieResult)

override def semanticCoreEntityId: Option[Long] = Some(tweetWithTopicProof.topicId)

override def localizedUttEntity: Option[LocalizedEntity] = Some(localizedEntity)

override def algorithmCR: Option[String] = tweetWithTopicProof.algorithmCR

override def tagsCR: Option[Seq[MetricTag]] = tags

override def isOutOfNetwork: Boolean = tweetWithTopicProof.isOON

}

}

/\*\*

\* Takes a group of TopicTweets and transforms them into RawCandidates

\*/

def buildTopicTweetRawCandidates(

inputTarget: Target,

topicProofCandidates: Seq[TweetWithTopicProof],

tweetIdToTagsMap: Map[Long, Seq[CrMixerMetricTag]],

maxNumberOfCands: Int

): Future[Option[Seq[RawCandidate]]] = {

val semanticCoreEntityIds = topicProofCandidates

.map(\_.topicId)

.toSet

TopicsUtil

.getLocalizedEntityMap(inputTarget, semanticCoreEntityIds, uttEntityHydrationStore)

.map { localizedEntityMap =>

val rawCandidates = topicProofCandidates.collect {

case topicSocialProof: TweetWithTopicProof

if localizedEntityMap.contains(topicSocialProof.topicId) =>

// Once we deprecate CR calls, we should replace this code to use the CrMixerMetricTag

val tags = tweetIdToTagsMap.get(topicSocialProof.tweetId).map {

\_.flatMap { tag => MetricTag.get(tag.value) }

}

buildTopicTweetRawCandidate(

inputTarget,

topicSocialProof,

localizedEntityMap(topicSocialProof.topicId),

tags

)

}

val candResult = restrict(

maxNumberOfCands,

rawCandidates,

numOutOfMaximumDropped,

totalOutputRecs

)

Some(candResult)

}

}

private def generateOONRawCandidate(

inputTarget: Target,

id: Long,

result: Option[TweetyPieResult],

tweetScoreMap: Map[Long, Double],

tweetIdToTagsMap: Map[Long, Seq[CrMixerMetricTag]]

): Option[RawCandidate with TweetCandidate] = {

val tagsFromCR = tweetIdToTagsMap.get(id).map { \_.flatMap { tag => MetricTag.get(tag.value) } }

val candidate = new RawCandidate with CrMixerCandidate with TopicCandidate with AlgorithmScore {

override val tweetId = id

override val target = inputTarget

override val tweetyPieResult = result

override val localizedUttEntity = None

override val semanticCoreEntityId = None

override def commonRecType =

getMediaBasedCRT(

CommonRecommendationType.TwistlyTweet,

CommonRecommendationType.TwistlyPhoto,

CommonRecommendationType.TwistlyVideo)

override def tagsCR = tagsFromCR

override def algorithmScore = tweetScoreMap.get(id)

override def algorithmCR = None

}

Some(candidate)

}

private def restrict(

maxNumToReturn: Int,

candidates: Seq[RawCandidate],

numOutOfMaximumDropped: Stat,

totalOutputRecs: Stat

): Seq[RawCandidate] = {

val newCandidates = candidates.take(maxNumToReturn)

val numDropped = candidates.length - newCandidates.length

numOutOfMaximumDropped.add(numDropped)

totalOutputRecs.add(newCandidates.size)

newCandidates

}

private def buildCrMixerRequest(

target: Target,

countryCode: Option[String],

language: Option[String],

seenTweets: Seq[Long]

): CrMixerTweetRequest = {

CrMixerTweetRequest(

clientContext = ClientContext(

userId = Some(target.targetId),

countryCode = countryCode,

languageCode = language

),

product = Product.Notifications,

productContext = Some(ProductContext.NotificationsContext(NotificationsContext())),

excludedTweetIds = Some(seenTweets)

)

}

private def selectCandidatesToSendBasedOnSettings(

isRecommendationsEligible: Boolean,

isTopicsEligible: Boolean,

oonRawCandidates: Option[Seq[RawCandidate]],

topicTweetCandidates: Option[Seq[RawCandidate]]

): Option[Seq[RawCandidate]] = {

if (isRecommendationsEligible && isTopicsEligible) {

Some(topicTweetCandidates.getOrElse(Seq.empty) ++ oonRawCandidates.getOrElse(Seq.empty))

} else if (isRecommendationsEligible) {

oonRawCandidates

} else if (isTopicsEligible) {

topicTweetCandidates

} else None

}

override def get(target: Target): Future[Option[Seq[RawCandidate]]] = {

Future

.join(

target.seenTweetIds,

target.countryCode,

target.inferredUserDeviceLanguage,

PushDeviceUtil.isTopicsEligible(target),

PushDeviceUtil.isRecommendationsEligible(target)

).flatMap {

case (seenTweets, countryCode, language, isTopicsEligible, isRecommendationsEligible) =>

val request = buildCrMixerRequest(target, countryCode, language, seenTweets)

crMixerTweetStore.getTweetRecommendations(request).flatMap {

case Some(response) =>

totalInputRecs.incr(response.tweets.size)

totalRequests.incr()

AdaptorUtils

.getTweetyPieResults(

response.tweets.map(\_.tweetId).toSet,

tweetyPieStore).flatMap { tweetyPieResultMap =>

filterOutInNetworkTweets(

target,

filterOutReplyTweet(tweetyPieResultMap.toMap, nonReplyTweetsCounter),

edgeStore,

numOfValidAuthors).flatMap {

outNetworkTweetsWithId: Seq[(Long, TweetyPieResult)] =>

totalOutNetworkRecs.incr(outNetworkTweetsWithId.size)

totalInNetworkRecs.incr(response.tweets.size - outNetworkTweetsWithId.size)

val outNetworkTweets: Seq[TweetyPieResult] = outNetworkTweetsWithId.map {

case (\_, tweetyPieResult) => tweetyPieResult

}

val tweetIdToTagsMap = response.tweets.map { tweet =>

tweet.tweetId -> tweet.metricTags.getOrElse(Seq.empty)

}.toMap

val tweetScoreMap = response.tweets.map { tweet =>

tweet.tweetId -> tweet.score

}.toMap

val maxNumOfCandidates =

target.params(PushFeatureSwitchParams.NumberOfMaxCrMixerCandidatesParam)

val oonRawCandidates =

buildOONRawCandidates(

target,

outNetworkTweets,

tweetScoreMap,

tweetIdToTagsMap,

maxNumOfCandidates)

TopicsUtil

.getTopicSocialProofs(

target,

outNetworkTweets,

topicSocialProofServiceStore,

edgeStore,

PushFeatureSwitchParams.TopicProofTweetCandidatesTopicScoreThreshold).flatMap {

tweetsWithTopicProof =>

buildTopicTweetRawCandidates(

target,

tweetsWithTopicProof,

tweetIdToTagsMap,

maxNumOfCandidates)

}.map { topicTweetCandidates =>

selectCandidatesToSendBasedOnSettings(

isRecommendationsEligible,

isTopicsEligible,

oonRawCandidates,

topicTweetCandidates)

}

}

}

case \_ => Future.None

}

}

}

/\*\*

\* For a user to be available the following news to happen

\*/

override def isCandidateSourceAvailable(target: Target): Future[Boolean] = {

Future

.join(

PushDeviceUtil.isRecommendationsEligible(target),

PushDeviceUtil.isTopicsEligible(target)

).map {

case (isRecommendationsEligible, isTopicsEligible) =>

(isRecommendationsEligible || isTopicsEligible) &&

target.params(PushParams.ContentRecommenderMixerAdaptorDecider)

}

}

}