package com.twitter.cr\_mixer.candidate\_generation

import com.twitter.contentrecommender.thriftscala.TweetInfo

import com.twitter.cr\_mixer.config.TimeoutConfig

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

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

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

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

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

import com.twitter.cr\_mixer.param.TopicTweetParams

import com.twitter.cr\_mixer.similarity\_engine.CertoTopicTweetSimilarityEngine

import com.twitter.cr\_mixer.similarity\_engine.SkitHighPrecisionTopicTweetSimilarityEngine

import com.twitter.cr\_mixer.similarity\_engine.SkitTopicTweetSimilarityEngine

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

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

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.finagle.util.DefaultTimer

import com.twitter.frigate.common.util.StatsUtil

import com.twitter.servo.util.MemoizingStatsReceiver

import com.twitter.simclusters\_v2.common.TweetId

import com.twitter.simclusters\_v2.thriftscala.TopicId

import com.twitter.snowflake.id.SnowflakeId

import com.twitter.storehaus.ReadableStore

import com.twitter.util.Duration

import com.twitter.util.Future

import com.twitter.util.Time

import javax.inject.Inject

import javax.inject.Singleton

/\*\*

\* Formerly CrTopic in legacy Content Recommender. This generator finds top Tweets per Topic.

\*/

@Singleton

class TopicTweetCandidateGenerator @Inject() (

certoTopicTweetSimilarityEngine: CertoTopicTweetSimilarityEngine,

skitTopicTweetSimilarityEngine: SkitTopicTweetSimilarityEngine,

skitHighPrecisionTopicTweetSimilarityEngine: SkitHighPrecisionTopicTweetSimilarityEngine,

tweetInfoStore: ReadableStore[TweetId, TweetInfo],

timeoutConfig: TimeoutConfig,

globalStats: StatsReceiver) {

private val timer = DefaultTimer

private val stats: StatsReceiver = globalStats.scope(this.getClass.getCanonicalName)

private val fetchCandidatesStats = stats.scope("fetchCandidates")

private val filterCandidatesStats = stats.scope("filterCandidates")

private val tweetyPieFilteredStats = filterCandidatesStats.stat("tweetypie\_filtered")

private val memoizedStatsReceiver = new MemoizingStatsReceiver(stats)

def get(

query: TopicTweetCandidateGeneratorQuery

): Future[Map[Long, Seq[TopicTweet]]] = {

val maxTweetAge = query.params(TopicTweetParams.MaxTweetAge)

val product = query.product

val allStats = memoizedStatsReceiver.scope("all")

val perProductStats = memoizedStatsReceiver.scope("perProduct", product.name)

StatsUtil.trackMapValueStats(allStats) {

StatsUtil.trackMapValueStats(perProductStats) {

val result = for {

retrievedTweets <- fetchCandidates(query)

initialTweetCandidates <- convertToInitialCandidates(retrievedTweets)

filteredTweetCandidates <- filterCandidates(

initialTweetCandidates,

maxTweetAge,

query.isVideoOnly,

query.impressedTweetList)

rankedTweetCandidates = rankCandidates(filteredTweetCandidates)

hydratedTweetCandidates = hydrateCandidates(rankedTweetCandidates)

} yield {

hydratedTweetCandidates.map {

case (topicId, topicTweets) =>

val topKTweets = topicTweets.take(query.maxNumResults)

topicId -> topKTweets

}

}

result.raiseWithin(timeoutConfig.topicTweetEndpointTimeout)(timer)

}

}

}

private def fetchCandidates(

query: TopicTweetCandidateGeneratorQuery

): Future[Map[TopicId, Option[Seq[TopicTweetWithScore]]]] = {

Future.collect {

query.topicIds.map { topicId =>

topicId -> StatsUtil.trackOptionStats(fetchCandidatesStats) {

Future

.join(

certoTopicTweetSimilarityEngine.get(CertoTopicTweetSimilarityEngine

.fromParams(topicId, query.isVideoOnly, query.params)),

skitTopicTweetSimilarityEngine

.get(SkitTopicTweetSimilarityEngine

.fromParams(topicId, query.isVideoOnly, query.params)),

skitHighPrecisionTopicTweetSimilarityEngine

.get(SkitHighPrecisionTopicTweetSimilarityEngine

.fromParams(topicId, query.isVideoOnly, query.params))

).map {

case (certoTopicTweets, skitTfgTopicTweets, skitHighPrecisionTopicTweets) =>

val uniqueCandidates = (certoTopicTweets.getOrElse(Nil) ++

skitTfgTopicTweets.getOrElse(Nil) ++

skitHighPrecisionTopicTweets.getOrElse(Nil))

.groupBy(\_.tweetId).map {

case (\_, dupCandidates) => dupCandidates.head

}.toSeq

Some(uniqueCandidates)

}

}

}.toMap

}

}

private def convertToInitialCandidates(

candidatesMap: Map[TopicId, Option[Seq[TopicTweetWithScore]]]

): Future[Map[TopicId, Seq[InitialCandidate]]] = {

val initialCandidates = candidatesMap.map {

case (topicId, candidatesOpt) =>

val candidates = candidatesOpt.getOrElse(Nil)

val tweetIds = candidates.map(\_.tweetId).toSet

val numTweetsPreFilter = tweetIds.size

Future.collect(tweetInfoStore.multiGet(tweetIds)).map { tweetInfos =>

/\*\* \*

\* If tweetInfo does not exist, we will filter out this tweet candidate.

\*/

val tweetyPieFilteredInitialCandidates = candidates.collect {

case candidate if tweetInfos.getOrElse(candidate.tweetId, None).isDefined =>

val tweetInfo = tweetInfos(candidate.tweetId)

.getOrElse(throw new IllegalStateException("Check previous line's condition"))

InitialCandidate(

tweetId = candidate.tweetId,

tweetInfo = tweetInfo,

CandidateGenerationInfo(

None,

SimilarityEngineInfo(

similarityEngineType = candidate.similarityEngineType,

modelId = None,

score = Some(candidate.score)),

Seq.empty

)

)

}

val numTweetsPostFilter = tweetyPieFilteredInitialCandidates.size

tweetyPieFilteredStats.add(numTweetsPreFilter - numTweetsPostFilter)

topicId -> tweetyPieFilteredInitialCandidates

}

}

Future.collect(initialCandidates.toSeq).map(\_.toMap)

}

private def filterCandidates(

topicTweetMap: Map[TopicId, Seq[InitialCandidate]],

maxTweetAge: Duration,

isVideoOnly: Boolean,

excludeTweetIds: Set[TweetId]

): Future[Map[TopicId, Seq[InitialCandidate]]] = {

val earliestTweetId = SnowflakeId.firstIdFor(Time.now - maxTweetAge)

val filteredResults = topicTweetMap.map {

case (topicId, tweetsWithScore) =>

topicId -> StatsUtil.trackItemsStats(filterCandidatesStats) {

val timeFilteredTweets =

tweetsWithScore.filter { tweetWithScore =>

tweetWithScore.tweetId >= earliestTweetId && !excludeTweetIds.contains(

tweetWithScore.tweetId)

}

filterCandidatesStats

.stat("exclude\_and\_time\_filtered").add(tweetsWithScore.size - timeFilteredTweets.size)

val tweetNudityFilteredTweets =

timeFilteredTweets.collect {

case tweet if tweet.tweetInfo.isPassTweetMediaNudityTag.contains(true) => tweet

}

filterCandidatesStats

.stat("tweet\_nudity\_filtered").add(

timeFilteredTweets.size - tweetNudityFilteredTweets.size)

val userNudityFilteredTweets =

tweetNudityFilteredTweets.collect {

case tweet if tweet.tweetInfo.isPassUserNudityRateStrict.contains(true) => tweet

}

filterCandidatesStats

.stat("user\_nudity\_filtered").add(

tweetNudityFilteredTweets.size - userNudityFilteredTweets.size)

val videoFilteredTweets = {

if (isVideoOnly) {

userNudityFilteredTweets.collect {

case tweet if tweet.tweetInfo.hasVideo.contains(true) => tweet

}

} else {

userNudityFilteredTweets

}

}

Future.value(videoFilteredTweets)

}

}

Future.collect(filteredResults)

}

private def rankCandidates(

tweetCandidatesMap: Map[TopicId, Seq[InitialCandidate]]

): Map[TopicId, Seq[InitialCandidate]] = {

tweetCandidatesMap.mapValues { tweetCandidates =>

tweetCandidates.sortBy { candidate =>

-candidate.tweetInfo.favCount

}

}

}

private def hydrateCandidates(

topicCandidatesMap: Map[TopicId, Seq[InitialCandidate]]

): Map[Long, Seq[TopicTweet]] = {

topicCandidatesMap.map {

case (topicId, tweetsWithScore) =>

topicId.entityId ->

tweetsWithScore.map { tweetWithScore =>

val similarityEngineType: SimilarityEngineType =

tweetWithScore.candidateGenerationInfo.similarityEngineInfo.similarityEngineType

TopicTweet(

tweetId = tweetWithScore.tweetId,

score = tweetWithScore.getSimilarityScore,

similarityEngineType = similarityEngineType

)

}

}

}

}