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

import com.twitter.contentrecommender.thriftscala.TweetInfo

import com.twitter.cr\_mixer.logging.UtegTweetScribeLogger

import com.twitter.cr\_mixer.filter.UtegFilterRunner

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

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

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

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

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

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

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

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

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

import com.twitter.cr\_mixer.source\_signal.RealGraphInSourceGraphFetcher

import com.twitter.cr\_mixer.source\_signal.SourceFetcher.FetcherQuery

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

import com.twitter.finagle.stats.StatsReceiver

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

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

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

import com.twitter.storehaus.ReadableStore

import com.twitter.util.Future

import javax.inject.Inject

import javax.inject.Named

import javax.inject.Singleton

@Singleton

class UtegTweetCandidateGenerator @Inject() (

@Named(ModuleNames.UserTweetEntityGraphSimilarityEngine) userTweetEntityGraphSimilarityEngine: StandardSimilarityEngine[

UserTweetEntityGraphSimilarityEngine.Query,

TweetWithScoreAndSocialProof

],

utegTweetScribeLogger: UtegTweetScribeLogger,

tweetInfoStore: ReadableStore[TweetId, TweetInfo],

realGraphInSourceGraphFetcher: RealGraphInSourceGraphFetcher,

utegFilterRunner: UtegFilterRunner,

globalStats: StatsReceiver) {

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

private val fetchSeedsStats = stats.scope("fetchSeeds")

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

private val utegFilterStats = stats.scope("utegFilter")

private val rankStats = stats.scope("rank")

def get(

query: UtegTweetCandidateGeneratorQuery

): Future[Seq[TweetWithScoreAndSocialProof]] = {

val allStats = stats.scope("all")

val perProductStats = stats.scope("perProduct", query.product.toString)

StatsUtil.trackItemsStats(allStats) {

StatsUtil.trackItemsStats(perProductStats) {

/\*\*

\* The candidate we return in the end needs a social proof field, which isn't

\* supported by the any existing Candidate type, so we created TweetWithScoreAndSocialProof

\* instead.

\*

\* However, filters and light ranker expect Candidate-typed param to work. In order to minimise the

\* changes to them, we are doing conversions from/to TweetWithScoreAndSocialProof to/from Candidate

\* in this method.

\*/

for {

realGraphSeeds <- StatsUtil.trackItemMapStats(fetchSeedsStats) {

fetchSeeds(query)

}

initialTweets <- StatsUtil.trackItemsStats(fetchCandidatesStats) {

fetchCandidates(query, realGraphSeeds)

}

initialCandidates <- convertToInitialCandidates(initialTweets)

filteredCandidates <- StatsUtil.trackItemsStats(utegFilterStats) {

utegFilter(query, initialCandidates)

}

rankedCandidates <- StatsUtil.trackItemsStats(rankStats) {

rankCandidates(query, filteredCandidates)

}

} yield {

val topTweets = rankedCandidates.take(query.maxNumResults)

convertToTweets(topTweets, initialTweets.map(tweet => tweet.tweetId -> tweet).toMap)

}

}

}

}

private def utegFilter(

query: UtegTweetCandidateGeneratorQuery,

candidates: Seq[InitialCandidate]

): Future[Seq[InitialCandidate]] = {

utegFilterRunner.runSequentialFilters(query, Seq(candidates)).map(\_.flatten)

}

private def fetchSeeds(

query: UtegTweetCandidateGeneratorQuery

): Future[Map[UserId, Double]] = {

realGraphInSourceGraphFetcher

.get(FetcherQuery(query.userId, query.product, query.userState, query.params))

.map(\_.map(\_.seedWithScores).getOrElse(Map.empty))

}

private[candidate\_generation] def rankCandidates(

query: UtegTweetCandidateGeneratorQuery,

filteredCandidates: Seq[InitialCandidate],

): Future[Seq[RankedCandidate]] = {

val blendedCandidates = filteredCandidates.map(candidate =>

candidate.toBlendedCandidate(Seq(candidate.candidateGenerationInfo)))

Future(

blendedCandidates.map { candidate =>

val score = candidate.getSimilarityScore

candidate.toRankedCandidate(score)

}

)

}

def fetchCandidates(

query: UtegTweetCandidateGeneratorQuery,

realGraphSeeds: Map[UserId, Double],

): Future[Seq[TweetWithScoreAndSocialProof]] = {

val engineQuery = UserTweetEntityGraphSimilarityEngine.fromParams(

query.userId,

realGraphSeeds,

Some(query.impressedTweetList.toSeq),

query.params

)

utegTweetScribeLogger.scribeInitialCandidates(

query,

userTweetEntityGraphSimilarityEngine.getCandidates(engineQuery).map(\_.toSeq.flatten)

)

}

private[candidate\_generation] def convertToInitialCandidates(

candidates: Seq[TweetWithScoreAndSocialProof],

): Future[Seq[InitialCandidate]] = {

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

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

/\*\* \*

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

\*/

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 = SimilarityEngineType.Uteg,

modelId = None,

score = Some(candidate.score)),

Seq.empty

)

)

}

}

}

private[candidate\_generation] def convertToTweets(

candidates: Seq[RankedCandidate],

tweetMap: Map[TweetId, TweetWithScoreAndSocialProof]

): Seq[TweetWithScoreAndSocialProof] = {

candidates.map { candidate =>

tweetMap

.get(candidate.tweetId).map { tweet =>

TweetWithScoreAndSocialProof(

tweet.tweetId,

candidate.predictionScore,

tweet.socialProofByType

)

// The exception should never be thrown

}.getOrElse(throw new Exception("Cannot find ranked candidate in original UTEG tweets"))

}

}

}