package com.twitter.cr\_mixer.similarity\_engine

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

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

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

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

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

import com.twitter.finagle.stats.StatsReceiver

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

import com.twitter.recos.user\_ad\_graph.thriftscala.ConsumersBasedRelatedAdRequest

import com.twitter.recos.user\_ad\_graph.thriftscala.RelatedAdResponse

import com.twitter.recos.user\_ad\_graph.thriftscala.UserAdGraph

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

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

import com.twitter.storehaus.ReadableStore

import com.twitter.timelines.configapi

import com.twitter.twistly.thriftscala.TweetRecentEngagedUsers

import com.twitter.util.Future

import javax.inject.Singleton

/\*\*

\* This store looks for similar tweets from UserAdGraph for a Source TweetId

\* For a query tweet,User Ad Graph (UAG)

\* lets us find out which other tweets share a lot of the same engagers with the query tweet

\*/

@Singleton

case class TweetBasedUserAdGraphSimilarityEngine(

userAdGraphService: UserAdGraph.MethodPerEndpoint,

tweetEngagedUsersStore: ReadableStore[TweetId, TweetRecentEngagedUsers],

statsReceiver: StatsReceiver)

extends ReadableStore[

TweetBasedUserAdGraphSimilarityEngine.Query,

Seq[TweetWithScore]

] {

import TweetBasedUserAdGraphSimilarityEngine.\_

private val stats = statsReceiver.scope(this.getClass.getSimpleName)

private val fetchCoverageExpansionCandidatesStat = stats.scope("fetchCoverageExpansionCandidates")

override def get(

query: TweetBasedUserAdGraphSimilarityEngine.Query

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

query.sourceId match {

case InternalId.TweetId(tweetId) => getCandidates(tweetId, query)

case \_ =>

Future.value(None)

}

}

// We first fetch tweet's recent engaged users as consumeSeedSet from MH store,

// then query consumersBasedUTG using the consumerSeedSet

private def getCandidates(

tweetId: TweetId,

query: TweetBasedUserAdGraphSimilarityEngine.Query

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

StatsUtil

.trackOptionItemsStats(fetchCoverageExpansionCandidatesStat) {

tweetEngagedUsersStore

.get(tweetId).flatMap {

\_.map { tweetRecentEngagedUsers =>

val consumerSeedSet =

tweetRecentEngagedUsers.recentEngagedUsers

.map { \_.userId }.take(query.maxConsumerSeedsNum)

val consumersBasedRelatedAdRequest =

ConsumersBasedRelatedAdRequest(

consumerSeedSet = consumerSeedSet,

maxResults = Some(query.maxResults),

minCooccurrence = Some(query.minCooccurrence),

excludeTweetIds = Some(Seq(tweetId)),

minScore = Some(query.consumersBasedMinScore),

maxTweetAgeInHours = Some(query.maxTweetAgeInHours)

)

toTweetWithScore(userAdGraphService

.consumersBasedRelatedAds(consumersBasedRelatedAdRequest).map { Some(\_) })

}.getOrElse(Future.value(None))

}

}

}

}

object TweetBasedUserAdGraphSimilarityEngine {

def toSimilarityEngineInfo(score: Double): SimilarityEngineInfo = {

SimilarityEngineInfo(

similarityEngineType = SimilarityEngineType.TweetBasedUserAdGraph,

modelId = None,

score = Some(score))

}

private def toTweetWithScore(

relatedAdResponseFut: Future[Option[RelatedAdResponse]]

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

relatedAdResponseFut.map { relatedAdResponseOpt =>

relatedAdResponseOpt.map { relatedAdResponse =>

val candidates =

relatedAdResponse.adTweets.map(tweet => TweetWithScore(tweet.adTweetId, tweet.score))

candidates

}

}

}

case class Query(

sourceId: InternalId,

maxResults: Int,

minCooccurrence: Int,

consumersBasedMinScore: Double,

maxTweetAgeInHours: Int,

maxConsumerSeedsNum: Int,

)

def fromParams(

sourceId: InternalId,

params: configapi.Params,

): EngineQuery[Query] = {

EngineQuery(

Query(

sourceId = sourceId,

maxResults = params(GlobalParams.MaxCandidateNumPerSourceKeyParam),

minCooccurrence = params(TweetBasedUserAdGraphParams.MinCoOccurrenceParam),

consumersBasedMinScore = params(TweetBasedUserAdGraphParams.ConsumersBasedMinScoreParam),

maxTweetAgeInHours = params(GlobalParams.MaxTweetAgeHoursParam).inHours,

maxConsumerSeedsNum = params(TweetBasedUserAdGraphParams.MaxConsumerSeedsNumParam),

),

params

)

}

}