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.TweetBasedUserTweetGraphParams

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\_tweet\_graph.thriftscala.RelatedTweetResponse

import com.twitter.recos.user\_tweet\_graph.thriftscala.TweetBasedRelatedTweetRequest

import com.twitter.recos.user\_tweet\_graph.thriftscala.ConsumersBasedRelatedTweetRequest

import com.twitter.recos.user\_tweet\_graph.thriftscala.UserTweetGraph

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

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

import com.twitter.storehaus.ReadableStore

import com.twitter.twistly.thriftscala.TweetRecentEngagedUsers

import com.twitter.util.Future

import javax.inject.Singleton

import com.twitter.snowflake.id.SnowflakeId

import com.twitter.timelines.configapi

import com.twitter.util.Duration

import com.twitter.util.Time

import scala.concurrent.duration.HOURS

/\*\*

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

\* For a query tweet,User Tweet Graph (UTG),

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

\* one-pager: go/UTG

\*/

@Singleton

case class TweetBasedUserTweetGraphSimilarityEngine(

userTweetGraphService: UserTweetGraph.MethodPerEndpoint,

tweetEngagedUsersStore: ReadableStore[TweetId, TweetRecentEngagedUsers],

statsReceiver: StatsReceiver)

extends ReadableStore[

TweetBasedUserTweetGraphSimilarityEngine.Query,

Seq[TweetWithScore]

] {

import TweetBasedUserTweetGraphSimilarityEngine.\_

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

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

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

override def get(

query: TweetBasedUserTweetGraphSimilarityEngine.Query

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

query.sourceId match {

case InternalId.TweetId(tweetId) if query.enableCoverageExpansionAllTweet =>

getCoverageExpansionCandidates(tweetId, query)

case InternalId.TweetId(tweetId) if query.enableCoverageExpansionOldTweet => // For Home

if (isOldTweet(tweetId)) getCoverageExpansionCandidates(tweetId, query)

else getCandidates(tweetId, query)

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

case \_ =>

Future.value(None)

}

}

// This is the main candidate source

private def getCandidates(

tweetId: TweetId,

query: TweetBasedUserTweetGraphSimilarityEngine.Query

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

StatsUtil.trackOptionItemsStats(fetchCandidatesStat) {

val tweetBasedRelatedTweetRequest = {

TweetBasedRelatedTweetRequest(

tweetId,

maxResults = Some(query.maxResults),

minCooccurrence = Some(query.minCooccurrence),

excludeTweetIds = Some(Seq(tweetId)),

minScore = Some(query.tweetBasedMinScore),

maxTweetAgeInHours = Some(query.maxTweetAgeInHours)

)

}

toTweetWithScore(

userTweetGraphService.tweetBasedRelatedTweets(tweetBasedRelatedTweetRequest).map {

Some(\_)

})

}

}

// function for DDGs, for coverage expansion algo, we first fetch tweet's recent engaged users as consumeSeedSet from MH store,

// and query consumersBasedUTG using the consumeSeedSet

private def getCoverageExpansionCandidates(

tweetId: TweetId,

query: TweetBasedUserTweetGraphSimilarityEngine.Query

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

StatsUtil

.trackOptionItemsStats(fetchCoverageExpansionCandidatesStat) {

tweetEngagedUsersStore

.get(tweetId).flatMap {

\_.map { tweetRecentEngagedUsers =>

val consumerSeedSet =

tweetRecentEngagedUsers.recentEngagedUsers

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

val consumersBasedRelatedTweetRequest =

ConsumersBasedRelatedTweetRequest(

consumerSeedSet = consumerSeedSet,

maxResults = Some(query.maxResults),

minCooccurrence = Some(query.minCooccurrence),

excludeTweetIds = Some(Seq(tweetId)),

minScore = Some(query.consumersBasedMinScore),

maxTweetAgeInHours = Some(query.maxTweetAgeInHours)

)

toTweetWithScore(userTweetGraphService

.consumersBasedRelatedTweets(consumersBasedRelatedTweetRequest).map { Some(\_) })

}.getOrElse(Future.value(None))

}

}

}

}

object TweetBasedUserTweetGraphSimilarityEngine {

def toSimilarityEngineInfo(score: Double): SimilarityEngineInfo = {

SimilarityEngineInfo(

similarityEngineType = SimilarityEngineType.TweetBasedUserTweetGraph,

modelId = None,

score = Some(score))

}

private val oldTweetCap: Duration = Duration(48, HOURS)

private def toTweetWithScore(

relatedTweetResponseFut: Future[Option[RelatedTweetResponse]]

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

relatedTweetResponseFut.map { relatedTweetResponseOpt =>

relatedTweetResponseOpt.map { relatedTweetResponse =>

val candidates =

relatedTweetResponse.tweets.map(tweet => TweetWithScore(tweet.tweetId, tweet.score))

candidates

}

}

}

private def isOldTweet(tweetId: TweetId): Boolean = {

SnowflakeId

.timeFromIdOpt(tweetId).forall { tweetTime => tweetTime < Time.now - oldTweetCap }

// If there's no snowflake timestamp, we have no idea when this tweet happened.

}

case class Query(

sourceId: InternalId,

maxResults: Int,

minCooccurrence: Int,

tweetBasedMinScore: Double,

consumersBasedMinScore: Double,

maxTweetAgeInHours: Int,

maxConsumerSeedsNum: Int,

enableCoverageExpansionOldTweet: Boolean,

enableCoverageExpansionAllTweet: Boolean,

)

def fromParams(

sourceId: InternalId,

params: configapi.Params,

): EngineQuery[Query] = {

EngineQuery(

Query(

sourceId = sourceId,

maxResults = params(GlobalParams.MaxCandidateNumPerSourceKeyParam),

minCooccurrence = params(TweetBasedUserTweetGraphParams.MinCoOccurrenceParam),

tweetBasedMinScore = params(TweetBasedUserTweetGraphParams.TweetBasedMinScoreParam),

consumersBasedMinScore = params(TweetBasedUserTweetGraphParams.ConsumersBasedMinScoreParam),

maxTweetAgeInHours = params(GlobalParams.MaxTweetAgeHoursParam).inHours,

maxConsumerSeedsNum = params(TweetBasedUserTweetGraphParams.MaxConsumerSeedsNumParam),

enableCoverageExpansionOldTweet =

params(TweetBasedUserTweetGraphParams.EnableCoverageExpansionOldTweetParam),

enableCoverageExpansionAllTweet =

params(TweetBasedUserTweetGraphParams.EnableCoverageExpansionAllTweetParam),

),

params

)

}

}