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.ProducerBasedUserTweetGraphParams

import com.twitter.finagle.stats.StatsReceiver

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

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

import com.twitter.storehaus.ReadableStore

import com.twitter.util.Future

import javax.inject.Singleton

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

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

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

import com.twitter.timelines.configapi

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

/\*\*

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

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

\* lets us find out which tweets the query producer's followers co-engaged

\*/

@Singleton

case class ProducerBasedUserTweetGraphSimilarityEngine(

userTweetGraphService: UserTweetGraph.MethodPerEndpoint,

statsReceiver: StatsReceiver)

extends ReadableStore[ProducerBasedUserTweetGraphSimilarityEngine.Query, Seq[

TweetWithScore

]] {

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

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

override def get(

query: ProducerBasedUserTweetGraphSimilarityEngine.Query

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

query.sourceId match {

case InternalId.UserId(producerId) =>

StatsUtil.trackOptionItemsStats(fetchCandidatesStat) {

val relatedTweetRequest =

ProducerBasedRelatedTweetRequest(

producerId,

maxResults = Some(query.maxResults),

minCooccurrence = Some(query.minCooccurrence),

minScore = Some(query.minScore),

maxNumFollowers = Some(query.maxNumFollowers),

maxTweetAgeInHours = Some(query.maxTweetAgeInHours),

)

userTweetGraphService.producerBasedRelatedTweets(relatedTweetRequest).map {

relatedTweetResponse =>

val candidates =

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

Some(candidates)

}

}

case \_ =>

Future.value(None)

}

}

}

object ProducerBasedUserTweetGraphSimilarityEngine {

def toSimilarityEngineInfo(score: Double): SimilarityEngineInfo = {

SimilarityEngineInfo(

similarityEngineType = SimilarityEngineType.ProducerBasedUserTweetGraph,

modelId = None,

score = Some(score))

}

case class Query(

sourceId: InternalId,

maxResults: Int,

minCooccurrence: Int, // require at least {minCooccurrence} lhs user engaged with returned tweet

minScore: Double,

maxNumFollowers: Int, // max number of lhs users

maxTweetAgeInHours: Int)

def fromParams(

sourceId: InternalId,

params: configapi.Params,

): EngineQuery[Query] = {

EngineQuery(

Query(

sourceId = sourceId,

maxResults = params(GlobalParams.MaxCandidateNumPerSourceKeyParam),

minCooccurrence = params(ProducerBasedUserTweetGraphParams.MinCoOccurrenceParam),

maxNumFollowers = params(ProducerBasedUserTweetGraphParams.MaxNumFollowersParam),

maxTweetAgeInHours = params(GlobalParams.MaxTweetAgeHoursParam).inHours,

minScore = params(ProducerBasedUserTweetGraphParams.MinScoreParam)

),

params

)

}

}