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

import com.google.inject.Inject

import com.google.inject.Singleton

import com.google.inject.name.Named

import com.twitter.contentrecommender.thriftscala.AlgorithmType

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

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

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

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

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.thriftscala.EmbeddingType

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

import com.twitter.storehaus.ReadableStore

import com.twitter.timelines.configapi

import com.twitter.topic\_recos.thriftscala.TopicTweet

import com.twitter.topic\_recos.thriftscala.TopicTweetPartitionFlatKey

import com.twitter.util.Future

@Singleton

case class SkitHighPrecisionTopicTweetSimilarityEngine @Inject() (

@Named(ModuleNames.SkitStratoStoreName) skitStratoStore: ReadableStore[

TopicTweetPartitionFlatKey,

Seq[TopicTweet]

],

statsReceiver: StatsReceiver)

extends ReadableStore[EngineQuery[Query], Seq[TopicTweetWithScore]] {

private val name: String = this.getClass.getSimpleName

private val stats = statsReceiver.scope(name)

override def get(query: EngineQuery[Query]): Future[Option[Seq[TopicTweetWithScore]]] = {

StatsUtil.trackOptionItemsStats(stats) {

fetch(query).map { tweets =>

val topTweets =

tweets

.sortBy(-\_.favCount)

.take(query.storeQuery.maxCandidates)

.map { tweet =>

TopicTweetWithScore(

tweetId = tweet.tweetId,

score = tweet.favCount,

similarityEngineType = SimilarityEngineType.SkitHighPrecisionTopicTweet

)

}

Some(topTweets)

}

}

}

private def fetch(query: EngineQuery[Query]): Future[Seq[SkitTopicTweet]] = {

val latestTweetTimeInHour = System.currentTimeMillis() / 1000 / 60 / 60

val earliestTweetTimeInHour = latestTweetTimeInHour -

math.min(MaxTweetAgeInHours, query.storeQuery.maxTweetAge.inHours)

val timedKeys = for (timePartition <- earliestTweetTimeInHour to latestTweetTimeInHour) yield {

TopicTweetPartitionFlatKey(

entityId = query.storeQuery.topicId.entityId,

timePartition = timePartition,

algorithmType = Some(AlgorithmType.SemanticCoreTweet),

tweetEmbeddingType = Some(EmbeddingType.LogFavBasedTweet),

language = query.storeQuery.topicId.language.getOrElse("").toLowerCase,

country = None, // Disable country. It is not used.

semanticCoreAnnotationVersionId = Some(query.storeQuery.semanticCoreVersionId)

)

}

getTweetsForKeys(

timedKeys,

query.storeQuery.topicId

)

}

/\*\*

\* Given a set of keys, multiget the underlying Strato store, combine and flatten the results.

\*/

private def getTweetsForKeys(

keys: Seq[TopicTweetPartitionFlatKey],

sourceTopic: TopicId

): Future[Seq[SkitTopicTweet]] = {

Future

.collect { skitStratoStore.multiGet(keys.toSet).values.toSeq }

.map { combinedResults =>

val topTweets = combinedResults.flatten.flatten

topTweets.map { tweet =>

SkitTopicTweet(

tweetId = tweet.tweetId,

favCount = tweet.scores.favCount.getOrElse(0L),

cosineSimilarityScore = tweet.scores.cosineSimilarity.getOrElse(0.0),

sourceTopic = sourceTopic

)

}

}

}

}

object SkitHighPrecisionTopicTweetSimilarityEngine {

def fromParams(

topicId: TopicId,

isVideoOnly: Boolean,

params: configapi.Params,

): EngineQuery[Query] = {

val maxCandidates = if (isVideoOnly) {

params(TopicTweetParams.MaxSkitHighPrecisionCandidatesParam) \* 2

} else {

params(TopicTweetParams.MaxSkitHighPrecisionCandidatesParam)

}

EngineQuery(

Query(

topicId = topicId,

maxCandidates = maxCandidates,

maxTweetAge = params(TopicTweetParams.MaxTweetAge),

semanticCoreVersionId = params(TopicTweetParams.SemanticCoreVersionIdParam)

),

params

)

}

}