package com.twitter.representationscorer.scorestore

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

import com.twitter.simclusters\_v2.thriftscala.ScoreInternalId.GenericPairScoreId

import com.twitter.simclusters\_v2.thriftscala.ScoringAlgorithm.CertoNormalizedDotProductScore

import com.twitter.simclusters\_v2.thriftscala.ScoringAlgorithm.CertoNormalizedCosineScore

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

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

import com.twitter.simclusters\_v2.thriftscala.{Score => ThriftScore}

import com.twitter.simclusters\_v2.thriftscala.{ScoreId => ThriftScoreId}

import com.twitter.storehaus.FutureOps

import com.twitter.storehaus.ReadableStore

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

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

import com.twitter.util.Future

/\*\*

\* Score store to get Certo <topic, tweet> scores.

\* Currently, the store supports two Scoring Algorithms (i.e., two types of Certo scores):

\* 1. NormalizedDotProduct

\* 2. NormalizedCosine

\* Querying with corresponding scoring algorithms results in different Certo scores.

\*/

case class TopicTweetCertoScoreStore(certoStratoStore: ReadableStore[TweetId, TopicToScores])

extends ReadableStore[ThriftScoreId, ThriftScore] {

override def multiGet[K1 <: ThriftScoreId](ks: Set[K1]): Map[K1, Future[Option[ThriftScore]]] = {

val tweetIds =

ks.map(\_.internalId).collect {

case GenericPairScoreId(scoreId) =>

((scoreId.id1, scoreId.id2): @annotation.nowarn(

"msg=may not be exhaustive|max recursion depth")) match {

case (InternalId.TweetId(tweetId), \_) => tweetId

case (\_, InternalId.TweetId(tweetId)) => tweetId

}

}

val result = for {

certoScores <- Future.collect(certoStratoStore.multiGet(tweetIds))

} yield {

ks.map { k =>

(k.algorithm, k.internalId) match {

case (CertoNormalizedDotProductScore, GenericPairScoreId(scoreId)) =>

(scoreId.id1, scoreId.id2) match {

case (InternalId.TweetId(tweetId), InternalId.TopicId(topicId)) =>

(

k,

extractScore(

tweetId,

topicId,

certoScores,

\_.followerL2NormalizedDotProduct8HrHalfLife))

case (InternalId.TopicId(topicId), InternalId.TweetId(tweetId)) =>

(

k,

extractScore(

tweetId,

topicId,

certoScores,

\_.followerL2NormalizedDotProduct8HrHalfLife))

case \_ => (k, None)

}

case (CertoNormalizedCosineScore, GenericPairScoreId(scoreId)) =>

(scoreId.id1, scoreId.id2) match {

case (InternalId.TweetId(tweetId), InternalId.TopicId(topicId)) =>

(

k,

extractScore(

tweetId,

topicId,

certoScores,

\_.followerL2NormalizedCosineSimilarity8HrHalfLife))

case (InternalId.TopicId(topicId), InternalId.TweetId(tweetId)) =>

(

k,

extractScore(

tweetId,

topicId,

certoScores,

\_.followerL2NormalizedCosineSimilarity8HrHalfLife))

case \_ => (k, None)

}

case \_ => (k, None)

}

}.toMap

}

FutureOps.liftValues(ks, result)

}

/\*\*

\* Given tweetToCertoScores, extract certain Certo score between the given tweetId and topicId.

\* The Certo score of interest is specified using scoreExtractor.

\*/

def extractScore(

tweetId: TweetId,

topicId: TopicId,

tweetToCertoScores: Map[TweetId, Option[TopicToScores]],

scoreExtractor: Scores => Double

): Option[ThriftScore] = {

tweetToCertoScores.get(tweetId).flatMap {

case Some(topicToScores) =>

topicToScores.topicToScores.flatMap(\_.get(topicId).map(scoreExtractor).map(ThriftScore(\_)))

case \_ => Some(ThriftScore(0.0))

}

}

}