package com.twitter.tsp.stores

import com.twitter.finagle.stats.StatsReceiver

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

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

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

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

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

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

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

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

SimClustersEmbeddingPairScoreId => ThriftSimClustersEmbeddingPairScoreId

}

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.ReadableStore

import com.twitter.topic\_recos.common.\_

import com.twitter.topic\_recos.common.Configs.DefaultModelVersion

import com.twitter.tsp.stores.TopicTweetsCosineSimilarityAggregateStore.ScoreKey

import com.twitter.util.Future

object TopicTweetsCosineSimilarityAggregateStore {

val TopicEmbeddingTypes: Seq[EmbeddingType] =

Seq(

EmbeddingType.FavTfgTopic,

EmbeddingType.LogFavBasedKgoApeTopic

)

// Add the new embedding types if want to test the new Tweet embedding performance.

val TweetEmbeddingTypes: Seq[EmbeddingType] = Seq(EmbeddingType.LogFavBasedTweet)

val ModelVersions: Seq[ModelVersion] =

Seq(DefaultModelVersion)

val DefaultScoreKeys: Seq[ScoreKey] = {

for {

modelVersion <- ModelVersions

topicEmbeddingType <- TopicEmbeddingTypes

tweetEmbeddingType <- TweetEmbeddingTypes

} yield {

ScoreKey(

topicEmbeddingType = topicEmbeddingType,

tweetEmbeddingType = tweetEmbeddingType,

modelVersion = modelVersion

)

}

}

case class ScoreKey(

topicEmbeddingType: EmbeddingType,

tweetEmbeddingType: EmbeddingType,

modelVersion: ModelVersion)

def getRawScoresMap(

topicId: TopicId,

tweetId: TweetId,

scoreKeys: Seq[ScoreKey],

representationScorerStore: ReadableStore[ThriftScoreId, ThriftScore]

): Future[Map[ScoreKey, Double]] = {

val scoresMapFut = scoreKeys.map { key =>

val scoreInternalId = ScoreInternalId.SimClustersEmbeddingPairScoreId(

ThriftSimClustersEmbeddingPairScoreId(

buildTopicEmbedding(topicId, key.topicEmbeddingType, key.modelVersion),

SimClustersEmbeddingId(

key.tweetEmbeddingType,

key.modelVersion,

InternalId.TweetId(tweetId))

))

val scoreFut = representationScorerStore

.get(

ThriftScoreId(

algorithm = ScoringAlgorithm.PairEmbeddingCosineSimilarity, // Hard code as cosine sim

internalId = scoreInternalId

))

key -> scoreFut

}.toMap

Future

.collect(scoresMapFut).map(\_.collect {

case (key, Some(ThriftScore(score))) =>

(key, score)

})

}

}

case class TopicTweetsCosineSimilarityAggregateStore(

representationScorerStore: ReadableStore[ThriftScoreId, ThriftScore]

)(

statsReceiver: StatsReceiver)

extends ReadableStore[(TopicId, TweetId, Seq[ScoreKey]), Map[ScoreKey, Double]] {

import TopicTweetsCosineSimilarityAggregateStore.\_

override def get(k: (TopicId, TweetId, Seq[ScoreKey])): Future[Option[Map[ScoreKey, Double]]] = {

statsReceiver.counter("topicTweetsCosineSimilariltyAggregateStore").incr()

getRawScoresMap(k.\_1, k.\_2, k.\_3, representationScorerStore).map(Some(\_))

}

}