package com.twitter.representationscorer.scorestore

import com.twitter.finagle.stats.StatsReceiver

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

import com.twitter.representationscorer.scorestore.TopicTweetsCosineSimilarityAggregateStore.ScoreKey

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

import com.twitter.simclusters\_v2.score.AggregatedScoreStore

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

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

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

EmbeddingType,

InternalId,

ModelVersion,

ScoreInternalId,

ScoringAlgorithm,

SimClustersEmbeddingId,

TopicId,

Score => ThriftScore,

ScoreId => ThriftScoreId,

SimClustersEmbeddingPairScoreId => ThriftSimClustersEmbeddingPairScoreId

}

import com.twitter.storehaus.ReadableStore

import com.twitter.topic\_recos.common.Configs.{DefaultModelVersion, MinCosineSimilarityScore}

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

import com.twitter.util.Future

/\*\*

\* Calculates the cosine similarity scores of arbitrary combinations of TopicEmbeddings and

\* TweetEmbeddings.

\* The class has 2 uses:

\* 1. For internal uses. TSP will call this store to fetch the raw scores for (topic, tweet) with

\* all available embedding types. We calculate all the scores here, so the caller can do filtering

\* & score caching on their side. This will make it possible to DDG different embedding scores.

\*

\* 2. For external calls from Cortex. We return true (or 1.0) for any given (topic, tweet) if their

\* cosine similarity passes the threshold for any of the embedding types.

\* The expected input type is

\* ScoreId(

\* PairEmbeddingCosineSimilarity,

\* GenericPairScoreId(TopicId, TweetId)

\* )

\*/

case class TopicTweetsCosineSimilarityAggregateStore(

scoreKeys: Seq[ScoreKey],

statsReceiver: StatsReceiver)

extends AggregatedScoreStore {

def toCortexScore(scoresMap: Map[ScoreKey, Double]): Double = {

val passThreshold = scoresMap.exists {

case (\_, score) => score >= MinCosineSimilarityScore

}

if (passThreshold) 1.0 else 0.0

}

/\*\*

\* To be called by Cortex through Unified Score API ONLY. Calculates all possible (topic, tweet),

\* return 1.0 if any of the embedding scores passes the minimum threshold.

\*

\* Expect a GenericPairScoreId(PairEmbeddingCosineSimilarity, (TopicId, TweetId)) as input

\*/

override def get(k: ThriftScoreId): Future[Option[ThriftScore]] = {

StatsUtil.trackOptionStats(statsReceiver) {

(k.algorithm, k.internalId) match {

case (CortexTopicTweetLabel, GenericPairScoreId(genericPairScoreId)) =>

(genericPairScoreId.id1, genericPairScoreId.id2) match {

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

TopicTweetsCosineSimilarityAggregateStore

.getRawScoresMap(topicId, tweetId, scoreKeys, scoreFacadeStore)

.map { scoresMap => Some(ThriftScore(toCortexScore(scoresMap))) }

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

TopicTweetsCosineSimilarityAggregateStore

.getRawScoresMap(topicId, tweetId, scoreKeys, scoreFacadeStore)

.map { scoresMap => Some(ThriftScore(toCortexScore(scoresMap))) }

case \_ =>

Future.None

// Do not accept other InternalId combinations

}

case \_ =>

// Do not accept other Id types for now

Future.None

}

}

}

}

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],

uniformScoringStore: 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 = uniformScoringStore

.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)

})

}

}