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

import com.twitter.ann.common.thriftscala.AnnQueryService

import com.twitter.ann.common.thriftscala.Distance

import com.twitter.ann.common.thriftscala.NearestNeighborQuery

import com.twitter.ann.common.thriftscala.NearestNeighborResult

import com.twitter.ann.hnsw.HnswCommon

import com.twitter.ann.hnsw.HnswParams

import com.twitter.bijection.Injection

import com.twitter.conversions.DurationOps.\_

import com.twitter.cortex.ml.embeddings.common.TweetKind

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

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

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

import com.twitter.finagle.stats.StatsReceiver

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

import com.twitter.mediaservices.commons.codec.ArrayByteBufferCodec

import com.twitter.ml.api.thriftscala.{Embedding => ThriftEmbedding}

import com.twitter.ml.featurestore.lib

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

import com.twitter.storehaus.ReadableStore

import com.twitter.util.Duration

import com.twitter.util.Future

import javax.inject.Singleton

/\*\*

\* This store looks for tweets whose similarity is close to a Source Dense Embedding.

\* Only support Long based embedding lookup. UserId or TweetId

\*/

@Singleton

class ModelBasedANNStore(

embeddingStoreLookUpMap: Map[String, ReadableStore[InternalId, ThriftEmbedding]],

annServiceLookUpMap: Map[String, AnnQueryService.MethodPerEndpoint],

globalStats: StatsReceiver)

extends ReadableStore[

ModelBasedANNStore.Query,

Seq[TweetWithScore]

] {

import ModelBasedANNStore.\_

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

private val fetchEmbeddingStat = stats.scope("fetchEmbedding")

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

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

for {

maybeEmbedding <- StatsUtil.trackOptionStats(fetchEmbeddingStat.scope(query.modelId)) {

fetchEmbedding(query)

}

maybeCandidates <- StatsUtil.trackOptionStats(fetchCandidatesStat.scope(query.modelId)) {

maybeEmbedding match {

case Some(embedding) =>

fetchCandidates(query, embedding)

case None =>

Future.None

}

}

} yield {

maybeCandidates.map(

\_.nearestNeighbors

.map { nearestNeighbor =>

val candidateId = TweetIdByteInjection

.invert(ArrayByteBufferCodec.decode(nearestNeighbor.id))

.toOption

.map(\_.tweetId)

(candidateId, nearestNeighbor.distance)

}.collect {

case (Some(candidateId), Some(distance)) =>

TweetWithScore(candidateId, toScore(distance))

})

}

}

private def fetchEmbedding(query: Query): Future[Option[ThriftEmbedding]] = {

embeddingStoreLookUpMap.get(query.modelId) match {

case Some(embeddingStore) =>

embeddingStore.get(query.sourceId)

case \_ =>

Future.None

}

}

private def fetchCandidates(

query: Query,

embedding: ThriftEmbedding

): Future[Option[NearestNeighborResult]] = {

val hnswParams = HnswCommon.RuntimeParamsInjection.apply(HnswParams(query.ef))

annServiceLookUpMap.get(query.modelId) match {

case Some(annService) =>

val annQuery =

NearestNeighborQuery(embedding, withDistance = true, hnswParams, MaxNumResults)

annService.query(annQuery).map(v => Some(v))

case \_ =>

Future.None

}

}

}

object ModelBasedANNStore {

val MaxNumResults: Int = 200

val MaxTweetCandidateAge: Duration = 1.day

val TweetIdByteInjection: Injection[lib.TweetId, Array[Byte]] = TweetKind.byteInjection

// For more information about HNSW algorithm: https://docbird.twitter.biz/ann/hnsw.html

case class Query(

sourceId: InternalId,

modelId: String,

similarityEngineType: SimilarityEngineType,

ef: Int = 800)

def toScore(distance: Distance): Double = {

distance match {

case Distance.L2Distance(l2Distance) =>

// (-Infinite, 0.0]

0.0 - l2Distance.distance

case Distance.CosineDistance(cosineDistance) =>

// [0.0 - 1.0]

1.0 - cosineDistance.distance

case Distance.InnerProductDistance(innerProductDistance) =>

// (-Infinite, Infinite)

1.0 - innerProductDistance.distance

case \_ =>

0.0

}

}

def toSimilarityEngineInfo(query: Query, score: Double): SimilarityEngineInfo = {

SimilarityEngineInfo(

similarityEngineType = query.similarityEngineType,

modelId = Some(query.modelId),

score = Some(score))

}

}