package com.twitter.ann.annoy

import com.spotify.annoy.{ANNIndex, IndexType}

import com.twitter.ann.annoy.AnnoyCommon.\_

import com.twitter.ann.common.\_

import com.twitter.ann.common.EmbeddingType.\_

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

import com.twitter.search.common.file.{AbstractFile, LocalFile}

import com.twitter.util.{Future, FuturePool}

import java.io.File

import scala.collection.JavaConverters.\_

private[annoy] object RawAnnoyQueryIndex {

private[annoy] def apply[D <: Distance[D]](

dimension: Int,

metric: Metric[D],

futurePool: FuturePool,

directory: AbstractFile

): Queryable[Long, AnnoyRuntimeParams, D] = {

val metadataFile = directory.getChild(MetaDataFileName)

val indexFile = directory.getChild(IndexFileName)

val metadata = MetadataCodec.decode(

ArrayByteBufferCodec.encode(metadataFile.getByteSource.read())

)

val existingDimension = metadata.dimension

assert(

existingDimension == dimension,

s"Dimensions do not match. requested: $dimension existing: $existingDimension"

)

val existingMetric = Metric.fromThrift(metadata.distanceMetric)

assert(

existingMetric == metric,

s"DistanceMetric do not match. requested: $metric existing: $existingMetric"

)

val index = loadIndex(indexFile, dimension, annoyMetric(metric))

new RawAnnoyQueryIndex[D](

dimension,

metric,

metadata.numOfTrees,

index,

futurePool

)

}

private[this] def annoyMetric(metric: Metric[\_]): IndexType = {

metric match {

case L2 => IndexType.EUCLIDEAN

case Cosine => IndexType.ANGULAR

case \_ => throw new RuntimeException("Not supported: " + metric)

}

}

private[this] def loadIndex(

indexFile: AbstractFile,

dimension: Int,

indexType: IndexType

): ANNIndex = {

var localIndexFile = indexFile

// If not a local file copy to local, so that it can be memory mapped.

if (!indexFile.isInstanceOf[LocalFile]) {

val tempFile = File.createTempFile(IndexFileName, null)

tempFile.deleteOnExit()

val temp = new LocalFile(tempFile)

indexFile.copyTo(temp)

localIndexFile = temp

}

new ANNIndex(

dimension,

localIndexFile.getPath(),

indexType

)

}

}

private[this] class RawAnnoyQueryIndex[D <: Distance[D]](

dimension: Int,

metric: Metric[D],

numOfTrees: Int,

index: ANNIndex,

futurePool: FuturePool)

extends Queryable[Long, AnnoyRuntimeParams, D]

with AutoCloseable {

override def query(

embedding: EmbeddingVector,

numOfNeighbours: Int,

runtimeParams: AnnoyRuntimeParams

): Future[List[Long]] = {

queryWithDistance(embedding, numOfNeighbours, runtimeParams)

.map(\_.map(\_.neighbor))

}

override def queryWithDistance(

embedding: EmbeddingVector,

numOfNeighbours: Int,

runtimeParams: AnnoyRuntimeParams

): Future[List[NeighborWithDistance[Long, D]]] = {

futurePool {

val queryVector = embedding.toArray

val neigboursToRequest = neighboursToRequest(numOfNeighbours, runtimeParams)

val neigbours = index

.getNearestWithDistance(queryVector, neigboursToRequest)

.asScala

.take(numOfNeighbours)

.map { nn =>

val id = nn.getFirst.toLong

val distance = metric.fromAbsoluteDistance(nn.getSecond)

NeighborWithDistance(id, distance)

}

.toList

neigbours

}

}

// Annoy java lib do not expose param for numOfNodesToExplore.

// Default number is numOfTrees\*numOfNeigbours.

// Simple hack is to artificially increase the numOfNeighbours to be requested and then just cap it before returning.

private[this] def neighboursToRequest(

numOfNeighbours: Int,

annoyParams: AnnoyRuntimeParams

): Int = {

annoyParams.nodesToExplore match {

case Some(nodesToExplore) => {

val neigboursToRequest = nodesToExplore / numOfTrees

if (neigboursToRequest < numOfNeighbours)

numOfNeighbours

else

neigboursToRequest

}

case \_ => numOfNeighbours

}

}

// To close the memory map based file resource.

override def close(): Unit = index.close()

}