package com.twitter.ann.annoy

import com.spotify.annoy.jni.base.{Annoy => AnnoyLib}

import com.twitter.ann.annoy.AnnoyCommon.IndexFileName

import com.twitter.ann.annoy.AnnoyCommon.MetaDataFileName

import com.twitter.ann.annoy.AnnoyCommon.MetadataCodec

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

import com.twitter.ann.common.\_

import com.twitter.ann.common.thriftscala.AnnoyIndexMetadata

import com.twitter.concurrent.AsyncSemaphore

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

import com.twitter.search.common.file.AbstractFile

import com.twitter.search.common.file.LocalFile

import com.twitter.util.Future

import com.twitter.util.FuturePool

import java.io.File

import java.nio.file.Files

import org.apache.beam.sdk.io.fs.ResourceId

import scala.collection.JavaConverters.\_

private[annoy] object RawAnnoyIndexBuilder {

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

dimension: Int,

numOfTrees: Int,

metric: Metric[D],

futurePool: FuturePool

): RawAppendable[AnnoyRuntimeParams, D] with Serialization = {

val indexBuilder = AnnoyLib.newIndex(dimension, annoyMetric(metric))

new RawAnnoyIndexBuilder(dimension, numOfTrees, metric, indexBuilder, futurePool)

}

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

metric match {

case L2 => AnnoyLib.Metric.EUCLIDEAN

case Cosine => AnnoyLib.Metric.ANGULAR

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

}

}

}

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

dimension: Int,

numOfTrees: Int,

metric: Metric[D],

indexBuilder: AnnoyLib.Builder,

futurePool: FuturePool)

extends RawAppendable[AnnoyRuntimeParams, D]

with Serialization {

private[this] var counter = 0

// Note: Only one thread can access the underlying index, multithreaded index building not supported

private[this] val semaphore = new AsyncSemaphore(1)

override def append(embedding: EmbeddingVector): Future[Long] =

semaphore.acquireAndRun({

counter += 1

indexBuilder.addItem(

counter,

embedding.toArray

.map(float => float2Float(float))

.toList

.asJava

)

Future.value(counter)

})

override def toQueryable: Queryable[Long, AnnoyRuntimeParams, D] = {

val tempDirParent = Files.createTempDirectory("raw\_annoy\_index").toFile

tempDirParent.deleteOnExit

val tempDir = new LocalFile(tempDirParent)

this.toDirectory(tempDir)

RawAnnoyQueryIndex(

dimension,

metric,

futurePool,

tempDir

)

}

override def toDirectory(directory: ResourceId): Unit = {

toDirectory(new IndexOutputFile(directory))

}

/\*\*

\* Serialize the annoy index in a directory.

\* @param directory: Directory to save to.

\*/

override def toDirectory(directory: AbstractFile): Unit = {

toDirectory(new IndexOutputFile(directory))

}

private def toDirectory(directory: IndexOutputFile): Unit = {

val indexFile = directory.createFile(IndexFileName)

saveIndex(indexFile)

val metaDataFile = directory.createFile(MetaDataFileName)

saveMetadata(metaDataFile)

}

private[this] def saveIndex(indexFile: IndexOutputFile): Unit = {

val index = indexBuilder

.build(numOfTrees)

val temp = new LocalFile(File.createTempFile(IndexFileName, null))

index.save(temp.getPath)

indexFile.copyFrom(temp.getByteSource.openStream())

temp.delete()

}

private[this] def saveMetadata(metadataFile: IndexOutputFile): Unit = {

val numberOfVectorsIndexed = counter

val metadata = AnnoyIndexMetadata(

dimension,

Metric.toThrift(metric),

numOfTrees,

numberOfVectorsIndexed

)

val bytes = ArrayByteBufferCodec.decode(MetadataCodec.encode(metadata))

val temp = new LocalFile(File.createTempFile(MetaDataFileName, null))

temp.getByteSink.write(bytes)

metadataFile.copyFrom(temp.getByteSource.openStream())

temp.delete()

}

}