package com.twitter.ann.hnsw

import com.google.common.annotations.VisibleForTesting

import com.twitter.ann.common.EmbeddingType.EmbeddingVector

import com.twitter.ann.common.thriftscala.HnswIndexMetadata

import com.twitter.ann.common.Distance

import com.twitter.ann.common.EntityEmbedding

import com.twitter.ann.common.Metric

import com.twitter.ann.hnsw.HnswCommon.\_

import com.twitter.ann.serialization.PersistedEmbeddingInjection

import com.twitter.ann.serialization.ThriftIteratorIO

import com.twitter.ann.serialization.thriftscala.PersistedEmbedding

import com.twitter.bijection.Injection

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

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

import java.io.BufferedInputStream

import java.io.BufferedOutputStream

import java.io.OutputStream

private[hnsw] object HnswIOUtil {

private val BufferSize = 64 \* 1024 // Default 64Kb

@VisibleForTesting

private[hnsw] def loadEmbeddings[T](

embeddingFile: AbstractFile,

injection: Injection[T, Array[Byte]],

idEmbeddingMap: IdEmbeddingMap[T],

): IdEmbeddingMap[T] = {

val inputStream = {

val stream = embeddingFile.getByteSource.openStream()

if (stream.isInstanceOf[BufferedInputStream]) {

stream

} else {

new BufferedInputStream(stream, BufferSize)

}

}

val thriftIteratorIO =

new ThriftIteratorIO[PersistedEmbedding](PersistedEmbedding)

val iterator = thriftIteratorIO.fromInputStream(inputStream)

val embeddingInjection = new PersistedEmbeddingInjection(injection)

try {

iterator.foreach { persistedEmbedding =>

val embedding = embeddingInjection.invert(persistedEmbedding).get

idEmbeddingMap.putIfAbsent(embedding.id, embedding.embedding)

Unit

}

} finally {

inputStream.close()

}

idEmbeddingMap

}

@VisibleForTesting

private[hnsw] def saveEmbeddings[T](

stream: OutputStream,

injection: Injection[T, Array[Byte]],

iter: Iterator[(T, EmbeddingVector)]

): Unit = {

val thriftIteratorIO =

new ThriftIteratorIO[PersistedEmbedding](PersistedEmbedding)

val embeddingInjection = new PersistedEmbeddingInjection(injection)

val iterator = iter.map {

case (id, emb) =>

embeddingInjection(EntityEmbedding(id, emb))

}

val outputStream = {

if (stream.isInstanceOf[BufferedOutputStream]) {

stream

} else {

new BufferedOutputStream(stream, BufferSize)

}

}

try {

thriftIteratorIO.toOutputStream(iterator, outputStream)

} finally {

outputStream.close()

}

}

@VisibleForTesting

private[hnsw] def saveIndexMetadata(

dimension: Int,

metric: Metric[\_ <: Distance[\_]],

numElements: Int,

metadataStream: OutputStream

): Unit = {

val metadata = HnswIndexMetadata(

dimension,

Metric.toThrift(metric),

numElements

)

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

metadataStream.write(bytes)

metadataStream.close()

}

@VisibleForTesting

private[hnsw] def loadIndexMetadata(

metadataFile: AbstractFile

): HnswIndexMetadata = {

MetadataCodec.decode(

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

)

}

}