package com.twitter.ann.brute\_force

import com.google.common.annotations.VisibleForTesting

import com.twitter.ann.common.{Distance, EntityEmbedding, Metric, QueryableDeserialization}

import com.twitter.ann.serialization.{PersistedEmbeddingInjection, ThriftIteratorIO}

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

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

import com.twitter.util.FuturePool

import java.io.File

/\*\*

\* @param factory creates a BruteForceIndex from the arguments. This is only exposed for testing.

\* If for some reason you pass this arg in make sure that it eagerly consumes the

\* iterator. If you don't you might close the input stream that the iterator is

\* using.

\* @tparam T the id of the embeddings

\*/

class BruteForceDeserialization[T, D <: Distance[D]] @VisibleForTesting private[brute\_force] (

metric: Metric[D],

embeddingInjection: PersistedEmbeddingInjection[T],

futurePool: FuturePool,

thriftIteratorIO: ThriftIteratorIO[PersistedEmbedding],

factory: (Metric[D], FuturePool, Iterator[EntityEmbedding[T]]) => BruteForceIndex[T, D])

extends QueryableDeserialization[T, BruteForceRuntimeParams.type, D, BruteForceIndex[T, D]] {

import BruteForceIndex.\_

def this(

metric: Metric[D],

embeddingInjection: PersistedEmbeddingInjection[T],

futurePool: FuturePool,

thriftIteratorIO: ThriftIteratorIO[PersistedEmbedding]

) = {

this(

metric,

embeddingInjection,

futurePool,

thriftIteratorIO,

factory = BruteForceIndex.apply[T, D]

)

}

override def fromDirectory(

serializationDirectory: AbstractFile

): BruteForceIndex[T, D] = {

val file = File.createTempFile(DataFileName, "tmp")

file.deleteOnExit()

val temp = new LocalFile(file)

val dataFile = serializationDirectory.getChild(DataFileName)

dataFile.copyTo(temp)

val inputStream = temp.getByteSource.openBufferedStream()

try {

val iterator: Iterator[PersistedEmbedding] = thriftIteratorIO.fromInputStream(inputStream)

val embeddings = iterator.map { thriftEmbedding =>

embeddingInjection.invert(thriftEmbedding).get

}

factory(metric, futurePool, embeddings)

} finally {

inputStream.close()

temp.delete()

}

}

}