package com.twitter.ann.brute\_force

import com.twitter.ann.common.Appendable

import com.twitter.ann.common.Distance

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

import com.twitter.ann.common.EntityEmbedding

import com.twitter.ann.common.IndexOutputFile

import com.twitter.ann.common.Metric

import com.twitter.ann.common.NeighborWithDistance

import com.twitter.ann.common.Queryable

import com.twitter.ann.common.RuntimeParams

import com.twitter.ann.common.Serialization

import com.twitter.ann.serialization.PersistedEmbeddingInjection

import com.twitter.ann.serialization.ThriftIteratorIO

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

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

import com.twitter.util.Future

import com.twitter.util.FuturePool

import java.util.concurrent.ConcurrentLinkedQueue

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

import scala.collection.JavaConverters.\_

import scala.collection.mutable

object BruteForceRuntimeParams extends RuntimeParams

object BruteForceIndex {

val DataFileName = "BruteForceFileData"

def apply[T, D <: Distance[D]](

metric: Metric[D],

futurePool: FuturePool,

initialEmbeddings: Iterator[EntityEmbedding[T]] = Iterator()

): BruteForceIndex[T, D] = {

val linkedQueue = new ConcurrentLinkedQueue[EntityEmbedding[T]]

initialEmbeddings.foreach(embedding => linkedQueue.add(embedding))

new BruteForceIndex(metric, futurePool, linkedQueue)

}

}

class BruteForceIndex[T, D <: Distance[D]] private (

metric: Metric[D],

futurePool: FuturePool,

// visible for serialization

private[brute\_force] val linkedQueue: ConcurrentLinkedQueue[EntityEmbedding[T]])

extends Appendable[T, BruteForceRuntimeParams.type, D]

with Queryable[T, BruteForceRuntimeParams.type, D] {

override def append(embedding: EntityEmbedding[T]): Future[Unit] = {

futurePool {

linkedQueue.add(embedding)

}

}

override def toQueryable: Queryable[T, BruteForceRuntimeParams.type, D] = this

override def query(

embedding: EmbeddingVector,

numOfNeighbours: Int,

runtimeParams: BruteForceRuntimeParams.type

): Future[List[T]] = {

queryWithDistance(embedding, numOfNeighbours, runtimeParams).map { neighborsWithDistance =>

neighborsWithDistance.map(\_.neighbor)

}

}

override def queryWithDistance(

embedding: EmbeddingVector,

numOfNeighbours: Int,

runtimeParams: BruteForceRuntimeParams.type

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

futurePool {

// Use the reverse ordering so that we can call dequeue to remove the largest element.

val ordering = Ordering.by[NeighborWithDistance[T, D], D](\_.distance)

val priorityQueue =

new mutable.PriorityQueue[NeighborWithDistance[T, D]]()(ordering)

linkedQueue

.iterator()

.asScala

.foreach { entity =>

val neighborWithDistance =

NeighborWithDistance(entity.id, metric.distance(entity.embedding, embedding))

priorityQueue.+=(neighborWithDistance)

if (priorityQueue.size > numOfNeighbours) {

priorityQueue.dequeue()

}

}

val reverseList: List[NeighborWithDistance[T, D]] =

priorityQueue.dequeueAll

reverseList.reverse

}

}

}

object SerializableBruteForceIndex {

def apply[T, D <: Distance[D]](

metric: Metric[D],

futurePool: FuturePool,

embeddingInjection: PersistedEmbeddingInjection[T],

thriftIteratorIO: ThriftIteratorIO[PersistedEmbedding]

): SerializableBruteForceIndex[T, D] = {

val bruteForceIndex = BruteForceIndex[T, D](metric, futurePool)

new SerializableBruteForceIndex(bruteForceIndex, embeddingInjection, thriftIteratorIO)

}

}

/\*\*

\* This is a class that wrapps a BruteForceIndex and provides a method for serialization.

\*

\* @param bruteForceIndex all queries and updates are sent to this index.

\* @param embeddingInjection injection that can convert embeddings to thrift embeddings.

\* @param thriftIteratorIO class that provides a way to write PersistedEmbeddings to disk

\*/

class SerializableBruteForceIndex[T, D <: Distance[D]](

bruteForceIndex: BruteForceIndex[T, D],

embeddingInjection: PersistedEmbeddingInjection[T],

thriftIteratorIO: ThriftIteratorIO[PersistedEmbedding])

extends Appendable[T, BruteForceRuntimeParams.type, D]

with Queryable[T, BruteForceRuntimeParams.type, D]

with Serialization {

import BruteForceIndex.\_

override def append(entity: EntityEmbedding[T]): Future[Unit] =

bruteForceIndex.append(entity)

override def toQueryable: Queryable[T, BruteForceRuntimeParams.type, D] = this

override def query(

embedding: EmbeddingVector,

numOfNeighbours: Int,

runtimeParams: BruteForceRuntimeParams.type

): Future[List[T]] =

bruteForceIndex.query(embedding, numOfNeighbours, runtimeParams)

override def queryWithDistance(

embedding: EmbeddingVector,

numOfNeighbours: Int,

runtimeParams: BruteForceRuntimeParams.type

): Future[List[NeighborWithDistance[T, D]]] =

bruteForceIndex.queryWithDistance(embedding, numOfNeighbours, runtimeParams)

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

toDirectory(new IndexOutputFile(serializationDirectory))

}

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

toDirectory(new IndexOutputFile(serializationDirectory))

}

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

val outputStream = serializationDirectory.createFile(DataFileName).getOutputStream()

val thriftEmbeddings =

bruteForceIndex.linkedQueue.iterator().asScala.map { embedding =>

embeddingInjection(embedding)

}

try {

thriftIteratorIO.toOutputStream(thriftEmbeddings, outputStream)

} finally {

outputStream.close()

}

}

}