package com.twitter.ann.hnsw

import com.twitter.ann.common.\_

import com.twitter.bijection.Injection

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

// Class to provide HNSW based approximate nearest neighbour index

object TypedHnswIndex {

/\*\*

\* Creates in-memory HNSW based index which supports querying/addition/updates of the entity embeddings.

\* See https://docbird.twitter.biz/ann/hnsw.html to check information about arguments.

\*

\* @param dimension Dimension of the embedding to be indexed

\* @param metric Distance metric (InnerProduct/Cosine/L2)

\* @param efConstruction The parameter has the same meaning as ef, but controls the

\* index\_time/index\_accuracy ratio. Bigger ef\_construction leads to longer

\* construction, but better index quality. At some point, increasing

\* ef\_construction does not improve the quality of the index. One way to

\* check if the selection of ef\_construction was ok is to measure a recall

\* for M nearest neighbor search when ef = ef\_constuction: if the recall is

\* lower than 0.9, than there is room for improvement.

\* @param maxM The number of bi-directional links created for every new element during construction.

\* Reasonable range for M is 2-100. Higher M work better on datasets with high

\* intrinsic dimensionality and/or high recall, while low M work better for datasets

\* with low intrinsic dimensionality and/or low recalls. The parameter also determines

\* the algorithm's memory consumption, bigger the param more the memory requirement.

\* For high dimensional datasets (word embeddings, good face descriptors), higher M

\* are required (e.g. M=48, 64) for optimal performance at high recall.

\* The range M=12-48 is ok for the most of the use cases.

\* @param expectedElements Approximate number of elements to be indexed

\* @param readWriteFuturePool Future pool for performing read (query) and write operation (addition/updates).

\* @tparam T Type of item to index

\* @tparam D Type of distance

\*/

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

dimension: Int,

metric: Metric[D],

efConstruction: Int,

maxM: Int,

expectedElements: Int,

readWriteFuturePool: ReadWriteFuturePool

): Appendable[T, HnswParams, D] with Queryable[T, HnswParams, D] with Updatable[T] = {

Hnsw[T, D](

dimension,

metric,

efConstruction,

maxM,

expectedElements,

readWriteFuturePool,

JMapBasedIdEmbeddingMap.applyInMemory[T](expectedElements)

)

}

/\*\*

\* Creates in-memory HNSW based index which supports querying/addition/updates of the entity embeddings.

\* It can be serialized to a directory (HDFS/Local file system)

\* See https://docbird.twitter.biz/ann/hnsw.html to check information about arguments.

\*

\* @param dimension Dimension of the embedding to be indexed

\* @param metric Distance metric (InnerProduct/Cosine/L2)

\* @param efConstruction The parameter has the same meaning as ef, but controls the

\* index\_time/index\_accuracy ratio. Bigger ef\_construction leads to longer

\* construction, but better index quality. At some point, increasing

\* ef\_construction does not improve the quality of the index. One way to

\* check if the selection of ef\_construction was ok is to measure a recall

\* for M nearest neighbor search when ef = ef\_constuction: if the recall is

\* lower than 0.9, than there is room for improvement.

\* @param maxM The number of bi-directional links created for every new element during construction.

\* Reasonable range for M is 2-100. Higher M work better on datasets with high

\* intrinsic dimensionality and/or high recall, while low M work better for datasets

\* with low intrinsic dimensionality and/or low recalls. The parameter also determines

\* the algorithm's memory consumption, bigger the param more the memory requirement.

\* For high dimensional datasets (word embeddings, good face descriptors), higher M

\* are required (e.g. M=48, 64) for optimal performance at high recall.

\* The range M=12-48 is ok for the most of the use cases.

\* @param expectedElements Approximate number of elements to be indexed

\* @param injection Injection for typed Id T to Array[Byte]

\* @param readWriteFuturePool Future pool for performing read (query) and write operation (addition/updates).

\* @tparam T Type of item to index

\* @tparam D Type of distance

\*/

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

dimension: Int,

metric: Metric[D],

efConstruction: Int,

maxM: Int,

expectedElements: Int,

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

readWriteFuturePool: ReadWriteFuturePool

): Appendable[T, HnswParams, D]

with Queryable[T, HnswParams, D]

with Updatable[T]

with Serialization = {

val index = Hnsw[T, D](

dimension,

metric,

efConstruction,

maxM,

expectedElements,

readWriteFuturePool,

JMapBasedIdEmbeddingMap

.applyInMemoryWithSerialization[T](expectedElements, injection)

)

SerializableHnsw[T, D](

index,

injection

)

}

/\*\*

\* Loads HNSW index from a directory to in-memory

\* @param dimension dimension of the embedding to be indexed

\* @param metric Distance metric

\* @param readWriteFuturePool Future pool for performing read (query) and write operation (addition/updates).

\* @param injection : Injection for typed Id T to Array[Byte]

\* @param directory : Directory(HDFS/Local file system) where hnsw index is stored

\* @tparam T : Type of item to index

\* @tparam D : Type of distance

\*/

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

dimension: Int,

metric: Metric[D],

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

readWriteFuturePool: ReadWriteFuturePool,

directory: AbstractFile

): Appendable[T, HnswParams, D]

with Queryable[T, HnswParams, D]

with Updatable[T]

with Serialization = {

SerializableHnsw.loadMapBasedQueryableIndex[T, D](

dimension,

metric,

injection,

readWriteFuturePool,

directory

)

}

/\*\*

\* Loads a HNSW index from a directory and memory map it.

\* It will take less memory but rely more on disk as it leverages memory mapped file backed by disk.

\* Latency will go up considerably (Could be by factor of > 10x) if used on instance with low

\* memory since lot of page faults may occur. Best use case to use would with scalding jobs

\* where mapper/reducers instance are limited by 8gb memory.

\* @param dimension dimension of the embedding to be indexed

\* @param metric Distance metric

\* @param readWriteFuturePool Future pool for performing read (query) and write operation (addition/updates).

\* @param injection Injection for typed Id T to Array[Byte]

\* @param directory Directory(HDFS/Local file system) where hnsw index is stored

\* @tparam T Type of item to index

\* @tparam D Type of distance

\*/

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

dimension: Int,

metric: Metric[D],

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

readWriteFuturePool: ReadWriteFuturePool,

directory: AbstractFile

): Appendable[T, HnswParams, D]

with Queryable[T, HnswParams, D]

with Updatable[T]

with Serialization = {

SerializableHnsw.loadMMappedBasedQueryableIndex[T, D](

dimension,

metric,

injection,

readWriteFuturePool,

directory

)

}

}