package com.twitter.ann.common

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

import com.twitter.ml.api.embedding.Embedding

import com.twitter.ml.api.embedding.EmbeddingMath

import com.twitter.ml.api.embedding.EmbeddingSerDe

import com.twitter.util.Future

object EmbeddingType {

type EmbeddingVector = Embedding[Float]

val embeddingSerDe = EmbeddingSerDe.apply[Float]

private[common] val math = EmbeddingMath.Float

}

/\*\*

\* Typed entity with an embedding associated with it.

\* @param id : Unique Id for an entity.

\* @param embedding : Embedding/Vector of an entity.

\* @tparam T: Type of id.

\*/

case class EntityEmbedding[T](id: T, embedding: EmbeddingVector)

// Query interface for ANN

trait Queryable[T, P <: RuntimeParams, D <: Distance[D]] {

/\*\*

\* ANN query for ids.

\* @param embedding: Embedding/Vector to be queried with.

\* @param numOfNeighbors: Number of neighbours to be queried for.

\* @param runtimeParams: Runtime params associated with index to control accuracy/latency etc.

\* @return List of approximate nearest neighbour ids.

\*/

def query(

embedding: EmbeddingVector,

numOfNeighbors: Int,

runtimeParams: P

): Future[List[T]]

/\*\*

\* ANN query for ids with distance.

\* @param embedding: Embedding/Vector to be queried with.

\* @param numOfNeighbors: Number of neighbours to be queried for.

\* @param runtimeParams: Runtime params associated with index to control accuracy/latency etc.

\* @return List of approximate nearest neighbour ids with distance from the query embedding.

\*/

def queryWithDistance(

embedding: EmbeddingVector,

numOfNeighbors: Int,

runtimeParams: P

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

}

// Query interface for ANN over indexes that are grouped

trait QueryableGrouped[T, P <: RuntimeParams, D <: Distance[D]] extends Queryable[T, P, D] {

/\*\*

\* ANN query for ids.

\* @param embedding: Embedding/Vector to be queried with.

\* @param numOfNeighbors: Number of neighbours to be queried for.

\* @param runtimeParams: Runtime params associated with index to control accuracy/latency etc.

\* @param key: Optional key to lookup specific ANN index and perform query there

\* @return List of approximate nearest neighbour ids.

\*/

def query(

embedding: EmbeddingVector,

numOfNeighbors: Int,

runtimeParams: P,

key: Option[String]

): Future[List[T]]

/\*\*

\* ANN query for ids with distance.

\* @param embedding: Embedding/Vector to be queried with.

\* @param numOfNeighbors: Number of neighbours to be queried for.

\* @param runtimeParams: Runtime params associated with index to control accuracy/latency etc.

\* @param key: Optional key to lookup specific ANN index and perform query there

\* @return List of approximate nearest neighbour ids with distance from the query embedding.

\*/

def queryWithDistance(

embedding: EmbeddingVector,

numOfNeighbors: Int,

runtimeParams: P,

key: Option[String]

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

}

/\*\*

\* Runtime params associated with index to control accuracy/latency etc while querying.

\*/

trait RuntimeParams {}

/\*\*

\* ANN query result with distance.

\* @param neighbor : Id of the neighbours

\* @param distance: Distance of neighbour from query ex: D: CosineDistance, L2Distance, InnerProductDistance

\*/

case class NeighborWithDistance[T, D <: Distance[D]](neighbor: T, distance: D)

/\*\*

\* ANN query result with seed entity for which this neighbor was provided.

\* @param seed: Seed Id for which ann query was called

\* @param neighbor : Id of the neighbours

\*/

case class NeighborWithSeed[T1, T2](seed: T1, neighbor: T2)

/\*\*

\* ANN query result with distance with seed entity for which this neighbor was provided.

\* @param seed: Seed Id for which ann query was called

\* @param neighbor : Id of the neighbours

\* @param distance: Distance of neighbour from query ex: D: CosineDistance, L2Distance, InnerProductDistance

\*/

case class NeighborWithDistanceWithSeed[T1, T2, D <: Distance[D]](

seed: T1,

neighbor: T2,

distance: D)

trait RawAppendable[P <: RuntimeParams, D <: Distance[D]] {

/\*\*

\* Append an embedding in an index.

\* @param embedding: Embedding/Vector

\* @return Future of long id associated with embedding autogenerated.

\*/

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

/\*\*

\* Convert an Appendable to Queryable interface to query an index.

\*/

def toQueryable: Queryable[Long, P, D]

}

// Index building interface for ANN.

trait Appendable[T, P <: RuntimeParams, D <: Distance[D]] {

/\*\*

\* Append an entity with embedding in an index.

\* @param entity: Entity with its embedding

\*/

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

/\*\*

\* Convert an Appendable to Queryable interface to query an index.

\*/

def toQueryable: Queryable[T, P, D]

}

// Updatable index interface for ANN.

trait Updatable[T] {

def update(entity: EntityEmbedding[T]): Future[Unit]

}