package com.twitter.ann.scalding.offline

import com.twitter.ann.brute\_force.{BruteForceIndex, BruteForceRuntimeParams}

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

import com.twitter.ann.hnsw.{HnswParams, TypedHnswIndex}

import com.twitter.ann.util.IndexBuilderUtils

import com.twitter.scalding.Args

import com.twitter.util.logging.Logger

import com.twitter.util.{Await, FuturePool}

/\*\*

\* IndexingStrategy is used for determining how we will build the index when doing a KNN in

\* scalding. Right now there are 2 strategies a BruteForce and HNSW strategy.

\* @tparam D distance that the index uses.

\*/

sealed trait IndexingStrategy[D <: Distance[D]] {

private[offline] def buildIndex[T](

indexItems: TraversableOnce[EntityEmbedding[T]]

): ParameterlessQueryable[T, \_, D]

}

object IndexingStrategy {

/\*\*

\* Parse an indexing strategy from scalding args.

\* ${argumentName}.type Is hsnw or brute\_force

\* ${argumentName}.type is the metric to use. See Metric.fromString for options.

\*

\* hsnw has these additional parameters:

\* ${argumentName}.dimension the number of dimension for the embeddings.

\* ${argumentName}.ef\_construction, ${argumentName}.ef\_construction and ${argumentName}.ef\_query.

\* See TypedHnswIndex for more details on these parameters.

\* @param args scalding arguments to parse.

\* @param argumentName A specifier to use in case you want to parse more than one indexing

\* strategy. indexing\_strategy by default.

\* @return parse indexing strategy

\*/

def parse(

args: Args,

argumentName: String = "indexing\_strategy"

): IndexingStrategy[\_] = {

def metricArg[D <: Distance[D]] =

Metric.fromString(args(s"$argumentName.metric")).asInstanceOf[Metric[D]]

args(s"$argumentName.type") match {

case "brute\_force" =>

BruteForceIndexingStrategy(metricArg)

case "hnsw" =>

val dimensionArg = args.int(s"$argumentName.dimension")

val efConstructionArg = args.int(s"$argumentName.ef\_construction")

val maxMArg = args.int(s"$argumentName.max\_m")

val efQuery = args.int(s"$argumentName.ef\_query")

HnswIndexingStrategy(

dimension = dimensionArg,

metric = metricArg,

efConstruction = efConstructionArg,

maxM = maxMArg,

hnswParams = HnswParams(efQuery)

)

}

}

}

case class BruteForceIndexingStrategy[D <: Distance[D]](metric: Metric[D])

extends IndexingStrategy[D] {

private[offline] def buildIndex[T](

indexItems: TraversableOnce[EntityEmbedding[T]]

): ParameterlessQueryable[T, \_, D] = {

val appendable = BruteForceIndex[T, D](metric, FuturePool.immediatePool)

indexItems.foreach { item =>

Await.result(appendable.append(item))

}

val queryable = appendable.toQueryable

ParameterlessQueryable[T, BruteForceRuntimeParams.type, D](

queryable,

BruteForceRuntimeParams

)

}

}

case class HnswIndexingStrategy[D <: Distance[D]](

dimension: Int,

metric: Metric[D],

efConstruction: Int,

maxM: Int,

hnswParams: HnswParams,

concurrencyLevel: Int = 1)

extends IndexingStrategy[D] {

private[offline] def buildIndex[T](

indexItems: TraversableOnce[EntityEmbedding[T]]

): ParameterlessQueryable[T, \_, D] = {

val log: Logger = Logger(getClass)

val appendable = TypedHnswIndex.index[T, D](

dimension = dimension,

metric = metric,

efConstruction = efConstruction,

maxM = maxM,

// This is not really that important.

expectedElements = 1000,

readWriteFuturePool = ReadWriteFuturePool(FuturePool.immediatePool)

)

val future =

IndexBuilderUtils

.addToIndex(appendable, indexItems.toStream, concurrencyLevel)

.map { numberUpdates =>

log.info(s"Performed $numberUpdates updates")

}

Await.result(future)

val queryable = appendable.toQueryable

ParameterlessQueryable(

queryable,

hnswParams

)

}

}