package com.twitter.ann.scalding.offline.indexbuilder

import com.twitter.ann.annoy.TypedAnnoyIndex

import com.twitter.ann.brute\_force.SerializableBruteForceIndex

import com.twitter.ann.common.Distance

import com.twitter.ann.common.Metric

import com.twitter.ann.common.ReadWriteFuturePool

import com.twitter.ann.hnsw.TypedHnswIndex

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

import com.twitter.ann.serialization.PersistedEmbeddingInjection

import com.twitter.ann.serialization.ThriftIteratorIO

import com.twitter.cortex.ml.embeddings.common.\_

import com.twitter.ml.featurestore.lib.EntityId

import com.twitter.scalding.Args

import com.twitter.scalding.Execution

import com.twitter.scalding\_internal.job.TwitterExecutionApp

import com.twitter.search.common.file.FileUtils

import com.twitter.util.FuturePool

import java.util.concurrent.Executors

trait IndexBuilderExecutable {

// This method is used to cast the entityKind and the metric to have parameters.

def indexBuilderExecution[T <: EntityId, D <: Distance[D]](

args: Args

): Execution[Unit] = {

// parse the arguments for this job

val uncastEntityKind = EntityKind.getEntityKind(args("entity\_kind"))

val uncastMetric = Metric.fromString(args("metric"))

val entityKind = uncastEntityKind.asInstanceOf[EntityKind[T]]

val metric = uncastMetric.asInstanceOf[Metric[D]]

val embeddingFormat = entityKind.parser.getEmbeddingFormat(args, "input")

val injection = entityKind.byteInjection

val numDimensions = args.int("num\_dimensions")

val embeddingLimit = args.optional("embedding\_limit").map(\_.toInt)

val concurrencyLevel = args.int("concurrency\_level")

val outputDirectory = FileUtils.getFileHandle(args("output\_dir"))

println(s"Job args: ${args.toString}")

val threadPool = Executors.newFixedThreadPool(concurrencyLevel)

val serialization = args("algo") match {

case "brute\_force" =>

val PersistedEmbeddingIO = new ThriftIteratorIO[PersistedEmbedding](PersistedEmbedding)

SerializableBruteForceIndex[T, D](

metric,

FuturePool.apply(threadPool),

new PersistedEmbeddingInjection[T](injection),

PersistedEmbeddingIO

)

case "annoy" =>

TypedAnnoyIndex.indexBuilder[T, D](

numDimensions,

args.int("annoy\_num\_trees"),

metric,

injection,

FuturePool.apply(threadPool)

)

case "hnsw" =>

val efConstruction = args.int("ef\_construction")

val maxM = args.int("max\_m")

val expectedElements = args.int("expected\_elements")

TypedHnswIndex.serializableIndex[T, D](

numDimensions,

metric,

efConstruction,

maxM,

expectedElements,

injection,

ReadWriteFuturePool(FuturePool.apply(threadPool))

)

}

IndexBuilder

.run(

embeddingFormat,

embeddingLimit,

serialization,

concurrencyLevel,

outputDirectory,

numDimensions

).onComplete { \_ =>

threadPool.shutdown()

Unit

}

}

}

object IndexBuilderApp extends TwitterExecutionApp with IndexBuilderExecutable {

override def job: Execution[Unit] = Execution.getArgs.flatMap { args: Args =>

indexBuilderExecution(args)

}

}