package com.twitter.ann.scalding.offline.indexbuilderfrombq

import com.twitter.ann.common.Appendable

import com.twitter.ann.common.Distance

import com.twitter.ann.common.EntityEmbedding

import com.twitter.ann.common.Serialization

import com.twitter.ann.util.IndexBuilderUtils

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

import com.twitter.ml.featurestore.lib.embedding.EmbeddingWithEntity

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

import com.twitter.scalding.Execution

import com.twitter.scalding.TypedPipe

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

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

import com.twitter.util.logging.Logger

object IndexBuilder {

private[this] val Log = Logger.apply[IndexBuilder.type]

def run[T <: EntityId, \_, D <: Distance[D]](

embeddingsPipe: TypedPipe[EmbeddingWithEntity[T]],

embeddingLimit: Option[Int],

index: Appendable[T, \_, D] with Serialization,

concurrencyLevel: Int,

outputDirectory: AbstractFile,

numDimensions: Int

): Execution[Unit] = {

val limitedEmbeddingsPipe = embeddingLimit

.map { limit =>

embeddingsPipe.limit(limit)

}.getOrElse(embeddingsPipe)

val annEmbeddingPipe = limitedEmbeddingsPipe.map { embedding =>

val embeddingSize = embedding.embedding.length

assert(

embeddingSize == numDimensions,

s"Specified number of dimensions $numDimensions does not match the dimensions of the " +

s"embedding $embeddingSize"

)

EntityEmbedding[T](embedding.entityId, Embedding(embedding.embedding.toArray))

}

annEmbeddingPipe.toIterableExecution.flatMap { annEmbeddings =>

val future = IndexBuilderUtils.addToIndex(index, annEmbeddings.toStream, concurrencyLevel)

val result = future.map { numberUpdates =>

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

index.toDirectory(outputDirectory)

Log.info(s"Finished writing to $outputDirectory")

}

FutureHelper.executionFrom(result).unit

}

}

}