package com.twitter.ann.experimental

import com.twitter.ann.annoy.{AnnoyRuntimeParams, TypedAnnoyIndex}

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

import com.twitter.ann.common.{Cosine, CosineDistance, EntityEmbedding, ReadWriteFuturePool}

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

import com.twitter.bijection.Injection

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

import com.twitter.search.common.file.LocalFile

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

import java.nio.file.Files

import java.util

import java.util.concurrent.Executors

import java.util.{Collections, Random}

import scala.collection.JavaConverters.\_

import scala.collection.mutable

object Runner {

def main(args: Array[String]): Unit = {

val rng = new Random()

val dimen = 300

val neighbours = 20

val trainDataSetSize = 2000

val testDataSetSize = 30

// Hnsw (ef -> (time, recall))

val hnswEfConfig = new mutable.HashMap[Int, (Float, Float)]

val efConstruction = 200

val maxM = 16

val threads = 24

val efSearch =

Seq(20, 30, 50, 70, 100, 120)

efSearch.foreach(hnswEfConfig.put(\_, (0.0f, 0.0f)))

// Annoy (nodes to explore -> (time, recall))

val numOfTrees = 80

val annoyConfig = new mutable.HashMap[Int, (Float, Float)]

val nodesToExplore = Seq(0, 2000, 3000, 5000, 7000, 10000, 15000, 20000,

30000, 35000, 40000, 50000)

nodesToExplore.foreach(annoyConfig.put(\_, (0.0f, 0.0f)))

val injection = Injection.int2BigEndian

val distance = Cosine

val exec = Executors.newFixedThreadPool(threads)

val pool = FuturePool.apply(exec)

val hnswMultiThread =

TypedHnswIndex.index[Int, CosineDistance](

dimen,

distance,

efConstruction = efConstruction,

maxM = maxM,

trainDataSetSize,

ReadWriteFuturePool(pool)

)

val bruteforce = BruteForceIndex[Int, CosineDistance](distance, pool)

val annoyBuilder =

TypedAnnoyIndex.indexBuilder(dimen, numOfTrees, distance, injection, FuturePool.immediatePool)

val temp = new LocalFile(Files.createTempDirectory("test").toFile)

println("Creating bruteforce.........")

val data =

Collections.synchronizedList(new util.ArrayList[EntityEmbedding[Int]]())

val bruteforceFutures = 1 to trainDataSetSize map { id =>

val vec = Array.fill(dimen)(rng.nextFloat() \* 50)

val emb = EntityEmbedding[Int](id, Embedding(vec))

data.add(emb)

bruteforce.append(emb)

}

Await.result(Future.collect(bruteforceFutures))

println("Creating hnsw multithread test.........")

val (\_, multiThreadInsertion) = time {

Await.result(Future.collect(data.asScala.toList.map { emb =>

hnswMultiThread.append(emb)

}))

}

println("Creating annoy.........")

val (\_, annoyTime) = time {

Await.result(Future.collect(data.asScala.toList.map(emb =>

annoyBuilder.append(emb))))

annoyBuilder.toDirectory(temp)

}

val annoyQuery = TypedAnnoyIndex.loadQueryableIndex(

dimen,

Cosine,

injection,

FuturePool.immediatePool,

temp

)

val hnswQueryable = hnswMultiThread.toQueryable

println(s"Total train size : $trainDataSetSize")

println(s"Total querySize : $testDataSetSize")

println(s"Dimension : $dimen")

println(s"Distance type : $distance")

println(s"Annoy index creation time trees: $numOfTrees => $annoyTime ms")

println(

s"Hnsw multi thread creation time : $multiThreadInsertion ms efCons: $efConstruction maxM $maxM thread : $threads")

println("Querying.........")

var bruteForceTime = 0.0f

1 to testDataSetSize foreach { id =>

println("Querying id " + id)

val embedding = Embedding(Array.fill(dimen)(rng.nextFloat()))

val (list, timeTakenB) =

time(

Await

.result(

bruteforce.query(embedding, neighbours, BruteForceRuntimeParams))

.toSet)

bruteForceTime += timeTakenB

val annoyConfigCopy = annoyConfig.toMap

val hnswEfConfigCopy = hnswEfConfig.toMap

hnswEfConfigCopy.keys.foreach { ef =>

val (nn, timeTaken) =

time(Await

.result(hnswQueryable.query(embedding, neighbours, HnswParams(ef)))

.toSet)

val recall = (list.intersect(nn).size) \* 1.0f / neighbours

val (oldTime, oldRecall) = hnswEfConfig(ef)

hnswEfConfig.put(ef, (oldTime + timeTaken, oldRecall + recall))

}

annoyConfigCopy.keys.foreach { nodes =>

val (nn, timeTaken) =

time(

Await.result(

annoyQuery

.query(embedding,

neighbours,

AnnoyRuntimeParams(nodesToExplore = Some(nodes)))

.map(\_.toSet)))

val recall = (list.intersect(nn).size) \* 1.0f / neighbours

val (oldTime, oldRecall) = annoyConfig(nodes)

annoyConfig.put(nodes, (oldTime + timeTaken, oldRecall + recall))

}

}

println(

s"Bruteforce avg query time : ${bruteForceTime / testDataSetSize} ms")

efSearch.foreach { ef =>

val data = hnswEfConfig(ef)

println(

s"Hnsw avg recall and time with query ef : $ef => ${data.\_2 / testDataSetSize} ${data.\_1 / testDataSetSize} ms"

)

}

nodesToExplore.foreach { n =>

val data = annoyConfig(n)

println(

s"Annoy avg recall and time with nodes\_to\_explore : $n => ${data.\_2 / testDataSetSize} ${data.\_1 / testDataSetSize} ms"

)

}

exec.shutdown()

}

def time[T](fn: => T): (T, Long) = {

val start = System.currentTimeMillis()

val result = fn

val end = System.currentTimeMillis()

(result, (end - start))

}

}