package com.twitter.ann.scalding.offline

import com.twitter.ann.common.Distance

import com.twitter.ann.common.Metric

import com.twitter.cortex.ml.embeddings.common.EntityKind

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

import com.twitter.scalding.typed.TypedPipe

import com.twitter.scalding.\_

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

/\*\*

\* This job do an exhaustive search for nearest neighbours helpful for debugging recommendations

\* for a given list of sample queryIds and entity embeddings for the recos to be made.

\* Sample job script:

./bazel bundle ann/src/main/scala/com/twitter/ann/scalding/offline:ann-offline-deploy

oscar hdfs \

--screen --tee log.txt \

--hadoop-client-memory 6000 \

--hadoop-properties "yarn.app.mapreduce.am.resource.mb=6000;yarn.app.mapreduce.am.command-opts='-Xmx7500m';mapreduce.map.memory.mb=7500;mapreduce.reduce.java.opts='-Xmx6000m';mapreduce.reduce.memory.mb=7500;mapred.task.timeout=36000000;" \

--bundle ann-offline-deploy \

--min-split-size 284217728 \

--host hadoopnest1.smf1.twitter.com \

--tool com.twitter.ann.scalding.offline.KnnEntityRecoDebugJob -- \

--neighbors 10 \

--metric InnerProduct \

--query\_entity\_kind user \

--search\_space\_entity\_kind user \

--query.embedding\_path /user/apoorvs/sample\_embeddings \

--query.embedding\_format tab \

--search\_space.embedding\_path /user/apoorvs/sample\_embeddings \

--search\_space.embedding\_format tab \

--query\_ids 974308319300149248 988871266244464640 2719685122 2489777564 \

--output\_path /user/apoorvs/adhochadoop/test \

--reducers 100

\*/

object KnnEntityRecoDebugJob extends TwitterExecutionApp {

override def job: Execution[Unit] = Execution.withId { implicit uniqueId =>

Execution.getArgs.flatMap { args: Args =>

val queryEntityKind = EntityKind.getEntityKind(args("query\_entity\_kind"))

val searchSpaceEntityKind = EntityKind.getEntityKind(args("search\_space\_entity\_kind"))

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

run(queryEntityKind, searchSpaceEntityKind, metric, args)

}

}

private[this] def run[A <: EntityId, B <: EntityId, D <: Distance[D]](

uncastQueryEntityKind: EntityKind[\_],

uncastSearchSpaceEntityKind: EntityKind[\_],

uncastMetric: Metric[\_],

args: Args

)(

implicit uniqueID: UniqueID

): Execution[Unit] = {

import KnnHelper.\_

val numNeighbors = args.int("neighbors")

val reducers = args.getOrElse("reducers", "100").toInt

val queryEntityKind = uncastQueryEntityKind.asInstanceOf[EntityKind[A]]

val searchSpaceEntityKind = uncastSearchSpaceEntityKind.asInstanceOf[EntityKind[B]]

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

// Filter the query entity embeddings with the queryIds

val queryIds = args.list("query\_ids")

assert(queryIds.nonEmpty)

val filterQueryIds: TypedPipe[A] = TypedPipe

.from(queryIds)

.map(queryEntityKind.stringInjection.invert(\_).get)

val queryEmbeddings = queryEntityKind.parser.getEmbeddingFormat(args, "query").getEmbeddings

// Get the neighbour embeddings

val searchSpaceEmbeddings =

searchSpaceEntityKind.parser.getEmbeddingFormat(args, "search\_space").getEmbeddings

val nearestNeighborString = findNearestNeighbours(

queryEmbeddings,

searchSpaceEmbeddings,

metric,

numNeighbors,

Some(filterQueryIds),

reducers

)(queryEntityKind.ordering, uniqueID).map(

nearestNeighborsToString(\_, queryEntityKind, searchSpaceEntityKind)

)

// Write the nearest neighbor string to one part file.

nearestNeighborString

.shard(1)

.writeExecution(TypedTsv(args("output\_path")))

}

}