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

import com.google.auth.oauth2.ServiceAccountCredentials

import com.google.cloud.bigquery.BigQueryOptions

import com.google.cloud.bigquery.QueryJobConfiguration

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.PersistedEmbeddingInjection

import com.twitter.ann.serialization.ThriftIteratorIO

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

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

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

import com.twitter.ml.featurestore.lib.\_

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

import com.twitter.scalding.Args

import com.twitter.scalding.Execution

import com.twitter.scalding.typed.TypedPipe

import com.twitter.scalding\_internal.bigquery.BigQueryConfig

import com.twitter.scalding\_internal.bigquery.BigQuerySource

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

import com.twitter.scalding\_internal.multiformat.format.keyval.KeyVal

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

import com.twitter.util.FuturePool

import java.io.FileInputStream

import java.time.LocalDateTime

import java.time.ZoneOffset

import java.util.concurrent.Executors

import org.apache.avro.generic.GenericRecord

import scala.collection.JavaConverters.\_

/\*\*

\* Scalding execution app for building ANN index from embeddings present in BigQuery table.

\* The output index is written to a GCS file.

\*

\* Note:

\* - Assumes input data has the fields entityId

\* - Assumes input data has the fields embedding

\*

\* Command for running the app (from source repo root):

\* scalding remote run \

\* --target ann/src/main/scala/com/twitter/ann/scalding/offline/indexbuilderfrombq:ann-index-builder-binary

\*/

trait IndexBuilderFromBQExecutable {

// 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 injection = entityKind.byteInjection

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

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

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

val bigQuery =

BigQueryOptions

.newBuilder().setProjectId(args.required("bq\_gcp\_job\_project")).setCredentials(

ServiceAccountCredentials.fromStream(

new FileInputStream(args.required("gcp\_service\_account\_key\_json")))).build().getService

// Query to get the latest partition of the BigQuery table.

val query =

s"SELECT MAX(ts) AS RecentPartition FROM ${args.required("bq\_gcp\_table\_project")}.${args

.required("bq\_dataset")}.${args.required("bq\_table")}"

val queryConfig = QueryJobConfiguration

.newBuilder(query)

.setUseLegacySql(false)

.build

val recentPartition =

bigQuery

.query(queryConfig).iterateAll().asScala.map(field => {

field.get(0).getStringValue

}).toArray.apply(0)

// Query to extract the embeddings from the latest partition of the BigQuery table

val bigQueryConfig = BigQueryConfig(

args.required("bq\_gcp\_table\_project"),

args

.required("bq\_dataset"),

args.required("bq\_table"))

.withServiceAccountKey(args.required("gcp\_service\_account\_key\_json"))

val bqFilter = Some(

s"ts >= '${recentPartition}' AND DATE(TIMESTAMP\_MILLIS(createdAt)) >= DATE\_SUB(DATE('${recentPartition}'), INTERVAL 1 DAY) AND DATE(TIMESTAMP\_MILLIS(createdAt)) <= DATE('${recentPartition}')")

val withFilterBigQueryConfig = bqFilter

.map { filter: String =>

bigQueryConfig.withFilter(filter)

}.getOrElse(bigQueryConfig)

val source = new BigQuerySource(withFilterBigQueryConfig)

.andThen(avroMapper)

val sourcePipe = TypedPipe

.from(source)

.map(transform[T](entityKind))

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))

)

}

// Output directory for the ANN index. We place the index under a timestamped directory which

// will be used by the ANN service to read the latest index

val timestamp = LocalDateTime.now().toEpochSecond(ZoneOffset.UTC)

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

IndexBuilder

.run(

sourcePipe,

embeddingLimit,

serialization,

concurrencyLevel,

outputDirectory,

numDimensions

).onComplete { \_ =>

threadPool.shutdown()

Unit

}

}

def avroMapper(row: GenericRecord): KeyVal[Long, java.util.List[Double]] = {

val entityId = row.get("entityId")

val embedding = row.get("embedding")

KeyVal(

entityId.toString.toLong,

embedding.asInstanceOf[java.util.List[Double]]

)

}

def transform[T <: EntityId](

entityKind: EntityKind[T]

)(

bqRecord: KeyVal[Long, java.util.List[Double]]

): EmbeddingWithEntity[T] = {

val embeddingArray = bqRecord.value.asScala.map(\_.floatValue()).toArray

val entity\_id = entityKind match {

case UserKind => UserId(bqRecord.key).toThrift

case TweetKind => TweetId(bqRecord.key).toThrift

case TfwKind => TfwId(bqRecord.key).toThrift

case SemanticCoreKind => SemanticCoreId(bqRecord.key).toThrift

case \_ => throw new IllegalArgumentException(s"Unsupported embedding kind: $entityKind")

}

EmbeddingWithEntity[T](

EntityId.fromThrift(entity\_id).asInstanceOf[T],

Embedding(embeddingArray))

}

}

/\*

scalding remote run \

--target ann/src/main/scala/com/twitter/ann/scalding/offline/indexbuilderfrombq:ann-index-builder-binary

\*/

object IndexBuilderFromBQApp extends TwitterExecutionApp with IndexBuilderFromBQExecutable {

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

indexBuilderExecution(args)

}

}