package com.twitter.ann.dataflow.offline

import com.spotify.scio.ScioContext

import com.spotify.scio.ScioMetrics

import com.twitter.ann.annoy.TypedAnnoyIndex

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

import com.twitter.ann.common.thriftscala.AnnIndexMetadata

import com.twitter.ann.common.Distance

import com.twitter.ann.common.Cosine

import com.twitter.ann.common.EntityEmbedding

import com.twitter.ann.common.IndexOutputFile

import com.twitter.ann.common.Metric

import com.twitter.ann.common.ReadWriteFuturePool

import com.twitter.ann.faiss.FaissIndexer

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.ann.util.IndexBuilderUtils

import com.twitter.beam.io.bigquery.BigQueryIO

import com.twitter.beam.io.dal.DalObservedDatasetRegistration

import com.twitter.beam.job.DateRange

import com.twitter.beam.job.DateRangeOptions

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

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

import com.twitter.ml.api.embedding.EmbeddingMath

import com.twitter.ml.api.embedding.EmbeddingSerDe

import com.twitter.ml.api.thriftscala.{Embedding => TEmbedding}

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

import com.twitter.ml.featurestore.lib.SemanticCoreId

import com.twitter.ml.featurestore.lib.TfwId

import com.twitter.ml.featurestore.lib.TweetId

import com.twitter.ml.featurestore.lib.UserId

import com.twitter.scalding.DateOps

import com.twitter.scalding.RichDate

import com.twitter.scio\_internal.job.ScioBeamJob

import com.twitter.statebird.v2.thriftscala.{Environment => StatebirdEnvironment}

import com.twitter.util.Await

import com.twitter.util.FuturePool

import com.twitter.wtf.beam.bq\_embedding\_export.BQQueryUtils

import java.time.Instant

import java.util.TimeZone

import java.util.concurrent.Executors

import org.apache.beam.sdk.io.FileSystems

import org.apache.beam.sdk.io.fs.ResolveOptions

import org.apache.beam.sdk.io.fs.ResourceId

import org.apache.beam.sdk.io.gcp.bigquery.BigQueryIO.TypedRead

import org.apache.beam.sdk.options.Default

import org.apache.beam.sdk.options.Description

import org.apache.beam.sdk.transforms.DoFn

import org.apache.beam.sdk.transforms.DoFn.\_

import org.apache.beam.sdk.transforms.PTransform

import org.apache.beam.sdk.transforms.ParDo

import org.apache.beam.sdk.values.KV

import org.apache.beam.sdk.values.PCollection

import org.apache.beam.sdk.values.PDone

import org.slf4j.Logger

import org.slf4j.LoggerFactory

trait ANNOptions extends DateRangeOptions {

@Description("Output GCS path for the generated index")

def getOutputPath(): String

def setOutputPath(value: String): Unit

@Description("If set, the index is grouped")

@Default.Boolean(false)

def getGrouped: Boolean

def setGrouped(value: Boolean): Unit

@Description(

"If set, a segment will be registered for the provided DAL dataset module which will trigger " +

"DAL registration.")

@Default.Boolean(false)

def getEnableDalRegistration: Boolean

def setEnableDalRegistration(value: Boolean): Unit

@Description(

"Output GCS path for the generated index. The OutputPath should be of the format " +

"'gs://user.{{user\_name}}.dp.gcp.twttr.net/subDir/outputDir' and OutputDALPath will be " +

"'subDir/outputDir' for this to work")

def getOutputDALPath: String

def setOutputDALPath(value: String): Unit

@Description("Get ANN index dataset name")

def getDatasetModuleName: String

def setDatasetModuleName(value: String): Unit

@Description("Get ANN index dataset owner role")

def getDatasetOwnerRole: String

def setDatasetOwnerRole(value: String): Unit

@Description("If set, index is written in <output>/<timestamp>")

@Default.Boolean(false)

def getOutputWithTimestamp: Boolean

def setOutputWithTimestamp(value: Boolean): Unit

@Description("File which contains a SQL query to retrieve embeddings from BQ")

def getDatasetSqlPath: String

def setDatasetSqlPath(value: String): Unit

@Description("Dimension of embedding in the input data. See go/ann")

def getDimension: Int

def setDimension(value: Int): Unit

@Description("The type of entity ID that is used with the embeddings. See go/ann")

def getEntityKind: String

def setEntityKind(value: String): Unit

@Description("The kind of index you want to generate (HNSW/Annoy/Brute Force/faiss). See go/ann")

def getAlgo: String

def setAlgo(value: String): Unit

@Description("Distance metric (InnerProduct/Cosine/L2). See go/ann")

def getMetric: String

def setMetric(value: String): Unit

@Description("Specifies how many parallel inserts happen to the index. See go/ann")

def getConcurrencyLevel: Int

def setConcurrencyLevel(value: Int): Unit

@Description(

"Used by HNSW algo. Larger value increases build time but will give better recall. See go/ann")

def getEfConstruction: Int

def setEfConstruction(value: Int): Unit

@Description(

"Used by HNSW algo. Larger value increases the index size but will give better recall. " +

"See go/ann")

def getMaxM: Int

def setMaxM(value: Int): Unit

@Description("Used by HNSW algo. Approximate number of elements that will be indexed. See go/ann")

def getExpectedElements: Int

def setExpectedElements(value: Int): Unit

@Description(

"Used by Annoy. num\_trees is provided during build time and affects the build time and the " +

"index size. A larger value will give more accurate results, but larger indexes. See go/ann")

def getAnnoyNumTrees: Int

def setAnnoyNumTrees(value: Int): Unit

@Description(

"FAISS factory string determines the ANN algorithm and compression. " +

"See https://github.com/facebookresearch/faiss/wiki/The-index-factory")

def getFAISSFactoryString: String

def setFAISSFactoryString(value: String): Unit

@Description("Sample rate for training during creation of FAISS index. Default is 0.05f")

@Default.Float(0.05f)

def getTrainingSampleRate: Float

def setTrainingSampleRate(value: Float): Unit

}

/\*\*

\* Builds ANN index.

\*

\* The input embeddings are read from BigQuery using the input SQL query. The output from this SQL

\* query needs to have two columns, "entityID" [Long] and "embedding" [List[Double]]

\*

\* Output directory supported is GCS bucket

\*/

object ANNIndexBuilderBeamJob extends ScioBeamJob[ANNOptions] {

val counterNameSpace = "ANNIndexBuilderBeamJob"

val LOG: Logger = LoggerFactory.getLogger(this.getClass)

implicit val timeZone: TimeZone = DateOps.UTC

def configurePipeline(sc: ScioContext, opts: ANNOptions): Unit = {

val startDate: RichDate = RichDate(opts.interval.getStart.toDate)

val endDate: RichDate = RichDate(opts.interval.getEnd.toDate)

val instant = Instant.now()

val out = {

val base = FileSystems.matchNewResource(opts.getOutputPath, /\*isDirectory=\*/ true)

if (opts.getOutputWithTimestamp) {

base.resolve(

instant.toEpochMilli.toString,

ResolveOptions.StandardResolveOptions.RESOLVE\_DIRECTORY)

} else {

base

}

}

// Define template variables which we would like to be replaced in the corresponding sql file

val templateVariables =

Map(

"START\_DATE" -> startDate.toString(DateOps.DATETIME\_HMS\_WITH\_DASH),

"END\_DATE" -> endDate.toString(DateOps.DATETIME\_HMS\_WITH\_DASH)

)

val embeddingFetchQuery =

BQQueryUtils.getBQQueryFromSqlFile(opts.getDatasetSqlPath, templateVariables)

val sCollection = if (opts.getGrouped) {

sc.customInput(

"Read grouped data from BQ",

BigQueryIO

.readClass[GroupedEmbeddingData]()

.fromQuery(embeddingFetchQuery).usingStandardSql()

.withMethod(TypedRead.Method.DIRECT\_READ)

)

} else {

sc.customInput(

"Read flat data from BQ",

BigQueryIO

.readClass[FlatEmbeddingData]().fromQuery(embeddingFetchQuery).usingStandardSql()

.withMethod(TypedRead.Method.DIRECT\_READ)

)

}

val processedCollection =

sCollection

.flatMap(transformTableRowToKeyVal)

.groupBy(\_.getKey)

.map {

case (groupName, groupValue) =>

Map(groupName -> groupValue.map(\_.getValue))

}

val annIndexMetadata =

AnnIndexMetadata(timestamp = Some(instant.getEpochSecond), withGroups = Some(opts.getGrouped))

// Count the number of groups and output the ANN index metadata

processedCollection.count.map(count => {

val annGroupedIndexMetadata = annIndexMetadata.copy(

numGroups = Some(count.intValue())

)

val indexOutDir = new IndexOutputFile(out)

indexOutDir.writeIndexMetadata(annGroupedIndexMetadata)

})

// Generate Index

processedCollection.saveAsCustomOutput(

"Serialise to Disk",

OutputSink(

out,

opts.getAlgo.equals("faiss"),

opts.getOutputDALPath,

opts.getEnableDalRegistration,

opts.getDatasetModuleName,

opts.getDatasetOwnerRole,

instant,

opts.getDate(),

counterNameSpace

)

)

}

def transformTableRowToKeyVal(

data: BaseEmbeddingData

): Option[KV[String, KV[Long, TEmbedding]]] = {

val transformTable = ScioMetrics.counter(counterNameSpace, "transform\_table\_row\_to\_kv")

for {

id <- data.entityId

} yield {

transformTable.inc()

val groupName: String = if (data.isInstanceOf[GroupedEmbeddingData]) {

(data.asInstanceOf[GroupedEmbeddingData]).groupId.get

} else {

""

}

KV.of[String, KV[Long, TEmbedding]](

groupName,

KV.of[Long, TEmbedding](

id,

EmbeddingSerDe.toThrift(Embedding(data.embedding.map(\_.toFloat).toArray)))

)

}

}

case class OutputSink(

outDir: ResourceId,

isFaiss: Boolean,

outputDALPath: String,

enableDalRegistration: Boolean,

datasetModuleName: String,

datasetOwnerRole: String,

instant: Instant,

date: DateRange,

counterNameSpace: String)

extends PTransform[PCollection[Map[String, Iterable[KV[Long, TEmbedding]]]], PDone] {

override def expand(input: PCollection[Map[String, Iterable[KV[Long, TEmbedding]]]]): PDone = {

PDone.in {

val dummyOutput = {

if (isFaiss) {

input

.apply(

"Build&WriteFaissANNIndex",

ParDo.of(new BuildFaissANNIndex(outDir, counterNameSpace))

)

} else {

input

.apply(

"Build&WriteANNIndex",

ParDo.of(new BuildANNIndex(outDir, counterNameSpace))

)

}

}

if (enableDalRegistration) {

input

.apply(

"Register DAL Dataset",

DalObservedDatasetRegistration(

datasetModuleName,

datasetOwnerRole,

outputDALPath,

instant,

Some(StatebirdEnvironment.Prod),

Some("ANN Index Data Files"))

)

.getPipeline

} else {

dummyOutput.getPipeline

}

}

}

}

class BuildANNIndex(outDir: ResourceId, counterNameSpace: String)

extends DoFn[Map[String, Iterable[KV[Long, TEmbedding]]], Unit] {

def transformKeyValToEmbeddingWithEntity[T <: EntityId](

entityKind: EntityKind[T]

)(

keyVal: KV[Long, TEmbedding]

): EntityEmbedding[T] = {

val entityId = entityKind match {

case UserKind => UserId(keyVal.getKey).toThrift

case TweetKind => TweetId(keyVal.getKey).toThrift

case TfwKind => TfwId(keyVal.getKey).toThrift

case SemanticCoreKind => SemanticCoreId(keyVal.getKey).toThrift

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

}

EntityEmbedding[T](

EntityId.fromThrift(entityId).asInstanceOf[T],

EmbeddingSerDe.fromThrift(keyVal.getValue))

}

@ProcessElement

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

@Element dataGrouped: Map[String, Iterable[KV[Long, TEmbedding]]],

context: ProcessContext

): Unit = {

val opts = context.getPipelineOptions.as(classOf[ANNOptions])

val uncastEntityKind = EntityKind.getEntityKind(opts.getEntityKind)

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

val transformKVtoEmbeddings =

ScioMetrics.counter(counterNameSpace, "transform\_kv\_to\_embeddings")

val \_ = dataGrouped.map {

case (groupName, data) =>

val annEmbeddings = data.map { kv =>

transformKVtoEmbeddings.inc()

transformKeyValToEmbeddingWithEntity(entityKind)(kv)

}

val out = {

if (opts.getGrouped && groupName != "") {

outDir.resolve(groupName, ResolveOptions.StandardResolveOptions.RESOLVE\_DIRECTORY)

} else {

outDir

}

}

LOG.info(s"Writing output to ${out}")

val metric = Metric.fromString(opts.getMetric).asInstanceOf[Metric[D]]

val concurrencyLevel = opts.getConcurrencyLevel

val dimension = opts.getDimension

val threadPool = Executors.newFixedThreadPool(concurrencyLevel)

LOG.info(s"Building ANN index of type ${opts.getAlgo}")

val serialization = opts.getAlgo match {

case "brute\_force" =>

val PersistedEmbeddingIO =

new ThriftIteratorIO[PersistedEmbedding](PersistedEmbedding)

SerializableBruteForceIndex(

metric,

FuturePool.apply(threadPool),

new PersistedEmbeddingInjection(entityKind.byteInjection),

PersistedEmbeddingIO

)

case "annoy" =>

TypedAnnoyIndex.indexBuilder(

dimension,

opts.getAnnoyNumTrees,

metric,

entityKind.byteInjection,

FuturePool.apply(threadPool)

)

case "hnsw" =>

val efConstruction = opts.getEfConstruction

val maxM = opts.getMaxM

val expectedElements = opts.getExpectedElements

TypedHnswIndex.serializableIndex(

dimension,

metric,

efConstruction,

maxM,

expectedElements,

entityKind.byteInjection,

ReadWriteFuturePool(FuturePool.apply(threadPool))

)

}

val future =

IndexBuilderUtils.addToIndex(serialization, annEmbeddings.toSeq, concurrencyLevel)

Await.result(future.map { \_ =>

serialization.toDirectory(out)

})

}

}

}

class BuildFaissANNIndex(outDir: ResourceId, counterNameSpace: String)

extends DoFn[Map[String, Iterable[KV[Long, TEmbedding]]], Unit] {

@ProcessElement

def processElement[D <: Distance[D]](

@Element dataGrouped: Map[String, Iterable[KV[Long, TEmbedding]]],

context: ProcessContext

): Unit = {

val opts = context.getPipelineOptions.as(classOf[ANNOptions])

val transformKVtoEmbeddings =

ScioMetrics.counter(counterNameSpace, "transform\_kv\_to\_embeddings")

val \_ = dataGrouped.map {

case (groupName, data) =>

val out = {

if (opts.getGrouped && groupName != "") {

outDir.resolve(groupName, ResolveOptions.StandardResolveOptions.RESOLVE\_DIRECTORY)

} else {

outDir

}

}

LOG.info(s"Writing output to ${out}")

val metric = Metric.fromString(opts.getMetric).asInstanceOf[Metric[D]]

val maybeNormalizedPipe = data.map { kv =>

transformKVtoEmbeddings.inc()

val embedding = EmbeddingSerDe.floatEmbeddingSerDe.fromThrift(kv.getValue)

EntityEmbedding[Long](

kv.getKey,

if (metric == Cosine) {

EmbeddingMath.Float.normalize(embedding)

} else {

embedding

}

)

}

// Generate Index

FaissIndexer.buildAndWriteFaissIndex(

maybeNormalizedPipe,

opts.getTrainingSampleRate,

opts.getFAISSFactoryString,

metric,

new IndexOutputFile(out))

}

}

}

}