package com.twitter.simclusters\_v2.scalding.mbcg

import com.twitter.ann.common.EntityEmbedding

import com.twitter.ann.common.Cosine

import com.twitter.ann.common.CosineDistance

import com.twitter.ann.common.InnerProduct

import com.twitter.ann.common.InnerProductDistance

import com.twitter.ann.common.ReadWriteFuturePool

import com.twitter.ann.hnsw.TypedHnswIndex

import com.twitter.ann.util.IndexBuilderUtils

import com.twitter.conversions.DurationOps.\_

import com.twitter.cortex.deepbird.runtime.prediction\_engine.TensorflowPredictionEngineConfig

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

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

import com.twitter.finagle.mtls.authentication.ServiceIdentifier

import com.twitter.finagle.stats.NullStatsReceiver

import com.twitter.iesource.common.util.InteractionEventUtils

import com.twitter.iesource.processing.events.batch.ServerEngagementsScalaDataset

import com.twitter.iesource.thriftscala.InteractionDetails

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

import com.twitter.ml.api.FeatureUtil

import com.twitter.ml.api.constant.SharedFeatures

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

import com.twitter.ml.api.thriftscala

import com.twitter.ml.api.thriftscala.{GeneralTensor => ThriftGeneralTensor}

import com.twitter.ml.api.util.FDsl.\_

import com.twitter.ml.api.util.ScalaToJavaDataRecordConversions

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

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

import com.twitter.scalding.Args

import com.twitter.scalding.DateParser

import com.twitter.scalding.DateRange

import com.twitter.scalding.Execution

import com.twitter.scalding.UniqueID

import com.twitter.scalding.\_

import com.twitter.scalding\_internal.dalv2.DAL

import com.twitter.scalding\_internal.dalv2.remote\_access.AllowCrossDC

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

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

import com.twitter.scalding\_internal.job.analytics\_batch.AnalyticsBatchExecution

import com.twitter.scalding\_internal.job.analytics\_batch.AnalyticsBatchExecutionArgs

import com.twitter.scalding\_internal.job.analytics\_batch.BatchDescription

import com.twitter.scalding\_internal.job.analytics\_batch.BatchFirstTime

import com.twitter.scalding\_internal.job.analytics\_batch.BatchIncrement

import com.twitter.scalding\_internal.job.analytics\_batch.BatchWidth

import com.twitter.scalding\_internal.job.analytics\_batch.TwitterScheduledExecutionApp

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

import com.twitter.simclusters\_v2.scalding.common.LogFavBasedPersistentTweetEmbeddingMhExportSource

import com.twitter.simclusters\_v2.thriftscala.PersistentSimClustersEmbedding

import com.twitter.tweetsource.common.thriftscala.MediaType

import com.twitter.tweetsource.public\_tweets.PublicTweetsScalaDataset

import com.twitter.tweetsource.public\_tweets.thriftscala.PublicTweet

import com.twitter.twml.runtime.scalding.TensorflowBatchPredictor

import com.twitter.twml.runtime.scalding.TensorflowBatchPredictor.ScaldingThreadingConfig

import com.twitter.util.FuturePool

import com.twitter.util.logging.Logger

import java.util.TimeZone

import java.util.concurrent.Executors

/\*

This class does the following:

1) Get tweet simcluster features from LogFavBasedPersistentTweetEmbeddingMhExportSource

2) Filter them down to English media tweets that aren't replies or quote tweets using TweetSource

3) Convert the remaining tweets into DataRecords using TweetSimclusterRecordAdapter

4) Run inference using a TF model exported with a DataRecord compatible serving signature

5) Create an ANN index from the generated tweet embeddings

\*/

trait TweetEmbeddingGenerationTrait {

implicit val tz: TimeZone = DateOps.UTC

implicit val dp: DateParser = DateParser.default

implicit val updateHours = 4

private val inputNodeName = "request:0"

private val outputNodeName = "response:0"

private val functionSignatureName = "serve"

private val predictionRequestTimeout = 5.seconds

private val SupportedLanguages = Set("en")

private val tweetSourceLookback = Days(2)

private val DEFAULT\_F2V\_VECTOR: Embedding[Float] = Embedding(Array.fill[Float](200)(0.0f))

def getPredictionEngine(modelName: String, modelPath: String): TensorflowBatchPredictor = {

val config = TensorflowPredictionEngineConfig(

modelName = modelName,

modelSource = modelPath,

threadingConfig = Some(ScaldingThreadingConfig),

defaultInputNode = inputNodeName,

defaultOutputNode = outputNodeName,

functionSignatureName = functionSignatureName,

statsReceiver = NullStatsReceiver

)

TensorflowBatchPredictor(config, predictionRequestTimeout)

}

def getEmbeddingWithEntity(tweetEmbeddingTensor: ThriftGeneralTensor, tweetId: Long) = {

tweetEmbeddingTensor match {

case ThriftGeneralTensor.RawTypedTensor(rawTensor) =>

val embedding = EmbeddingSerDe.floatEmbeddingSerDe.fromThrift(

thriftscala.Embedding(Some(rawTensor))

)

EmbeddingWithEntity[TweetId](TweetId(tweetId), embedding)

case \_ => throw new IllegalArgumentException("tensor is wrong type!")

}

}

def buildAnnIndex(

pipe: TypedPipe[EmbeddingWithEntity[TweetId]],

args: Args

): Execution[Unit] = {

def embeddingDimension: Int = args.int("embedding\_dimension", 128)

def efConstruction: Int = args.int("ef\_construction", 800)

def maxM: Int = args.int("max\_M", 40)

val log: Logger = Logger(getClass)

val annOutputPath: String = args("ann\_output\_path")

val embeddingWithEntity = pipe.map {

case EmbeddingWithEntity(tweetId, embedding) =>

EntityEmbedding[TweetId](tweetId, embedding)

}

val concurrencyLevel = args.int("concurrency\_level", 60)

val expectedElements = args.int("expected\_elements", 30000000)

val threadPool = Executors.newFixedThreadPool(concurrencyLevel)

val hnswIndex = TypedHnswIndex.serializableIndex[TweetId, InnerProductDistance](

embeddingDimension,

InnerProduct,

efConstruction,

maxM,

expectedElements,

TweetKind.byteInjection,

ReadWriteFuturePool(FuturePool.apply(threadPool))

)

// Create a timestamped directory to use for recovery in case of index corruption

val timeStampedAnnOutputPath: String = annOutputPath + "/" + (System.currentTimeMillis() / 1000)

val timeStampedAnnOutputDirectory = FileUtils.getFileHandle(timeStampedAnnOutputPath)

embeddingWithEntity.toIterableExecution

.flatMap { annEmbeddings =>

val future =

IndexBuilderUtils.addToIndex(hnswIndex, annEmbeddings.toStream, concurrencyLevel)

val result = future.map { numberUpdates =>

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

hnswIndex.toDirectory(timeStampedAnnOutputDirectory)

log.info(s"Finished writing to timestamped index directory - " +

s"$timeStampedAnnOutputDirectory")

}

FutureHelper.executionFrom(result).unit

}.onComplete { \_ =>

threadPool.shutdown()

Unit

}

}

def getTweetSimclusterFeatures(

args: Args

)(

implicit dateRange: DateRange

): TypedPipe[(Long, PersistentSimClustersEmbedding)] = {

val serviceIdEnv = args.getOrElse("sIdEnv", "prod")

val serviceIdRole = args.getOrElse("sIdRole", "cassowary")

val serviceIdZone = args.getOrElse("sIdZone", "atla")

val serviceIdName = args

.getOrElse("sIdName", "tweet-embedding-generation-batch-job")

val serviceId = ServiceIdentifier(

role = serviceIdRole,

service = serviceIdName,

environment = serviceIdEnv,

zone = serviceIdZone)

val logFavBasedPersistentTweetEmbeddingSource =

new LogFavBasedPersistentTweetEmbeddingMhExportSource(

range = dateRange.prepend(Hours(24)),

serviceIdentifier = serviceId)

val tweetSimclusterEmbeddingTypedPipe = TypedPipe

.from(logFavBasedPersistentTweetEmbeddingSource)

.collect {

case (

(tweetId, timestamp),

simclusterEmbedding: PersistentSimClustersEmbedding

) if timestamp == 1L => // 1L corresponds to the LongestL2Norm simcluster embedding

(tweetId.toLong, simclusterEmbedding)

}

tweetSimclusterEmbeddingTypedPipe

}

def getTweetSource()(implicit dateRange: DateRange): TypedPipe[PublicTweet] = {

val recentTweets = DAL

.read(PublicTweetsScalaDataset, dateRange.prepend(tweetSourceLookback))

.toTypedPipe

recentTweets

}

def isVideoTweet(tweet: PublicTweet): Boolean = {

tweet.media.exists { mediaSeq =>

mediaSeq.exists { e =>

e.mediaType.contains(MediaType.Video)

}

}

}

def getEngagementFilteredTweets(

minFavCount: Long

)(

implicit dateRange: DateRange

): TypedPipe[(Long, Int)] = {

val engagementFilteredTweetsPipe = DAL

.read(ServerEngagementsScalaDataset, dateRange.prepend(Days(2))).withRemoteReadPolicy(

AllowCrossDC).toTypedPipe

.collect {

case event if InteractionEventUtils.isTweetType(event) =>

val targetTweetId = event.targetId

event.details match {

case InteractionDetails.Favorite(\_) => (targetTweetId, 1)

case \_ => (targetTweetId, 0)

}

}

.sumByKey

.map {

case (tweetId, count) => (tweetId, count)

}

.filter(\_.\_2 >= minFavCount)

engagementFilteredTweetsPipe

}

def run(args: Args)(implicit dateRange: DateRange, idx: UniqueID) = {

val minFavCount = args.int("minFavCount", 32)

val indexAllTweets = args.boolean("indexAllTweets")

val tweetSimclusterDataset = getTweetSimclusterFeatures(args)

val tweetSourceDataset = getTweetSource()

val engagementFilteredTweetsPipe = getEngagementFilteredTweets(minFavCount)

val inputEmbeddingFormat = UserKind.parser

.getEmbeddingFormat(args, "f2v\_input", Some(dateRange.prepend(Days(14))))

val f2vProducerEmbeddings = inputEmbeddingFormat.getEmbeddings

.map {

case EmbeddingWithEntity(userId, embedding) => (userId.userId, embedding)

}

val engagementFilteredTweetInfoPipe = tweetSourceDataset

.groupBy(\_.tweetId)

.join(engagementFilteredTweetsPipe.groupBy(\_.\_1))

.map {

case (tweetId, (tweetInfo, tweetFavCount)) =>

(tweetId, tweetInfo)

}

val filteredSimclustersPipe = tweetSimclusterDataset

.groupBy(\_.\_1)

.join(engagementFilteredTweetInfoPipe.groupBy(\_.\_1))

.map {

case (tweetId, ((\_, simclusterEmbedding), (\_, tweetInfo))) =>

(tweetId, simclusterEmbedding, tweetInfo)

}

.filter {

case (\_, \_, tweetInfo) =>

tweetInfo.quotedTweetTweetId.isEmpty &&

tweetInfo.inReplyToTweetId.isEmpty &&

tweetInfo.language.exists(SupportedLanguages.contains) &&

(indexAllTweets || (!tweetInfo.media.exists(\_.isEmpty) && isVideoTweet(tweetInfo))) &&

!tweetInfo.nsfwAdmin &&

!tweetInfo.nsfwUser

}

.map {

case (tweetId, simclusterEmbedding, tweetInfo) =>

(tweetInfo.userId, tweetId, simclusterEmbedding)

}

val dataRecordsPipe = filteredSimclustersPipe

.groupBy(\_.\_1)

.leftJoin(f2vProducerEmbeddings.groupBy(\_.\_1))

.values

.map {

case ((authorId1, tweetId, simclusterEmbedding), Some((authorId2, f2vEmbedding))) =>

TweetSimclusterRecordAdapter.adaptToDataRecord(

(tweetId, simclusterEmbedding, f2vEmbedding))

case ((authorId, tweetId, simclusterEmbedding), None) =>

TweetSimclusterRecordAdapter.adaptToDataRecord(

(tweetId, simclusterEmbedding, DEFAULT\_F2V\_VECTOR))

}

val modelPath = args.getOrElse("model\_path", "")

val batchPredictor = getPredictionEngine(modelName = "tweet\_model", modelPath = modelPath)

val tweetIdFeature = SharedFeatures.TWEET\_ID

val tweetEmbeddingName = args.getOrElse("tweet\_embedding\_name", "output")

val outputPipe = batchPredictor.predict(dataRecordsPipe).map {

case (originalDataRecord, predictedDataRecord) =>

val tweetId = originalDataRecord.getFeatureValue(tweetIdFeature)

val scalaPredictedDataRecord =

ScalaToJavaDataRecordConversions.javaDataRecord2ScalaDataRecord(predictedDataRecord)

val tweetEmbeddingTensor =

scalaPredictedDataRecord.tensors.get(FeatureUtil.featureIdForName(tweetEmbeddingName))

val tweetEmbeddingWithEntity = getEmbeddingWithEntity(tweetEmbeddingTensor, tweetId)

tweetEmbeddingWithEntity

}

buildAnnIndex(outputPipe, args)

}

}

object TweetEmbeddingGenerationAdhocJob

extends TwitterExecutionApp

with TweetEmbeddingGenerationTrait {

override def job: Execution[Unit] =

Execution.withId { implicit uid =>

Execution.withArgs { args =>

implicit val dateRange: DateRange = DateRange.parse(args.list("dateRange"))

run(args)

}

}

}

object TweetEmbeddingGenerationBatchJob

extends TwitterScheduledExecutionApp

with TweetEmbeddingGenerationTrait {

override def scheduledJob: Execution[Unit] =

Execution.withId { implicit uid =>

Execution.withArgs { args =>

implicit val tz: TimeZone = DateOps.UTC

val batchFirstTime = BatchFirstTime(RichDate("2021-10-28")(tz, DateParser.default))

val analyticsArgs = AnalyticsBatchExecutionArgs(

batchDesc = BatchDescription(getClass.getName),

firstTime = batchFirstTime,

batchIncrement = BatchIncrement(Hours(updateHours)),

batchWidth = Some(BatchWidth(Hours(updateHours)))

)

AnalyticsBatchExecution(analyticsArgs) { implicit dateRange =>

run(args)

}

}

}

}

object TweetEmbeddingGenerationBatchJobAlternate

extends TwitterScheduledExecutionApp

with TweetEmbeddingGenerationTrait {

override def scheduledJob: Execution[Unit] =

Execution.withId { implicit uid =>

Execution.withArgs { args =>

implicit val tz: TimeZone = DateOps.UTC

val batchFirstTime = BatchFirstTime(RichDate("2022-03-28")(tz, DateParser.default))

val analyticsArgs = AnalyticsBatchExecutionArgs(

batchDesc = BatchDescription(getClass.getName),

firstTime = batchFirstTime,

batchIncrement = BatchIncrement(Hours(updateHours)),

batchWidth = Some(BatchWidth(Hours(updateHours)))

)

AnalyticsBatchExecution(analyticsArgs) { implicit dateRange =>

run(args)

}

}

}

}

object TweetEmbeddingGenerationBatchJobExperimental

extends TwitterScheduledExecutionApp

with TweetEmbeddingGenerationTrait {

override def scheduledJob: Execution[Unit] =

Execution.withId { implicit uid =>

Execution.withArgs { args =>

implicit val tz: TimeZone = DateOps.UTC

val batchFirstTime = BatchFirstTime(RichDate("2021-12-12")(tz, DateParser.default))

val analyticsArgs = AnalyticsBatchExecutionArgs(

batchDesc = BatchDescription(getClass.getName),

firstTime = batchFirstTime,

batchIncrement = BatchIncrement(Hours(updateHours)),

batchWidth = Some(BatchWidth(Hours(updateHours)))

)

AnalyticsBatchExecution(analyticsArgs) { implicit dateRange =>

run(args)

}

}

}

}