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

import com.twitter.conversions.DurationOps.\_

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

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

import com.twitter.finagle.stats.NullStatsReceiver

import com.twitter.ml.api.FeatureUtil

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

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

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.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.DALWrite.D

import com.twitter.scalding\_internal.dalv2.DALWrite.\_

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

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.scalding\_internal.multiformat.format.keyval.KeyVal

import com.twitter.simclusters\_v2.hdfs\_sources.AdhocKeyValSources

import com.twitter.simclusters\_v2.hdfs\_sources.ExploreMbcgUserEmbeddingsKvScalaDataset

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

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

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

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

import com.twitter.usersource.snapshot.flat.UsersourceFlatScalaDataset

import com.twitter.usersource.snapshot.flat.thriftscala.FlatUser

import java.util.TimeZone

/\*

This class does the following:

1) Get user IIAPE Simcluster features that use LogFav scores

2) Filter them down to users whose accounts are not deactivated or suspended

3) Convert the remaining user Simclusters into DataRecords using UserSimclusterRecordAdapter

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

5) Write to MH using a KeyVal format

\*/

trait UserEmbeddingGenerationTrait {

implicit val tz: TimeZone = DateOps.UTC

implicit val dp: DateParser = DateParser.default

implicit val updateHours = 12

private val inputNodeName = "request:0"

private val outputNodeName = "response:0"

private val functionSignatureName = "serve"

private val predictionRequestTimeout = 5.seconds

private val IIAPEHdfsPath: String =

"/atla/proc3/user/cassowary/manhattan\_sequence\_files/interested\_in\_from\_ape/Model20m145k2020"

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(userEmbeddingTensor: ThriftGeneralTensor, userId: Long) = {

userEmbeddingTensor match {

case ThriftGeneralTensor.RawTypedTensor(rawTensor) =>

val embedding =

thriftscala.Embedding(Some(rawTensor))

KeyVal(userId, embedding)

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

}

}

def writeUserEmbedding(

result: TypedPipe[KeyVal[Long, thriftscala.Embedding]],

args: Args

): Execution[Unit] = {

result.writeDALVersionedKeyValExecution(

ExploreMbcgUserEmbeddingsKvScalaDataset,

D.Suffix(

args.getOrElse("kvs\_output\_path", "/user/cassowary/explore\_mbcg/user\_kvs\_store/test")

)

)

}

def getUserSimclusterFeatures(

args: Args

)(

implicit dateRange: DateRange

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

val userSimclusterEmbeddingTypedPipe = TypedPipe

.from(AdhocKeyValSources.interestedInSource(IIAPEHdfsPath))

.collect {

case (

userId,

iIAPE: ClustersUserIsInterestedIn

) =>

(userId.toLong, iIAPE)

}

userSimclusterEmbeddingTypedPipe

}

def getUserSource()(implicit dateRange: DateRange): TypedPipe[FlatUser] = {

val userSource =

DAL

.readMostRecentSnapshotNoOlderThan(UsersourceFlatScalaDataset, Days(7))

.withRemoteReadPolicy(AllowCrossDC)

.toTypedPipe

userSource

}

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

val userSimclusterDataset = getUserSimclusterFeatures(args)

val userSourceDataset = getUserSource()

val inputEmbeddingFormat = UserKind.parser

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

val f2vConsumerEmbeddings = inputEmbeddingFormat.getEmbeddings

.map {

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

}

val filteredUserPipe = userSimclusterDataset

.groupBy(\_.\_1)

.join(userSourceDataset.groupBy(\_.id.getOrElse(-1L)))

.map {

case (userId, ((\_, simclusterEmbedding), userInfo)) =>

(userId, simclusterEmbedding, userInfo)

}

.filter {

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

!userInfo.deactivated.contains(true) && !userInfo.suspended

.contains(true)

}

.map {

case (userId, simclusterEmbedding, \_) =>

(userId, simclusterEmbedding)

}

val dataRecordsPipe = filteredUserPipe

.groupBy(\_.\_1)

.leftJoin(f2vConsumerEmbeddings.groupBy(\_.\_1))

.values

.map {

case ((userId1, simclusterEmbedding), Some((userId2, f2vEmbedding))) =>

UserSimclusterRecordAdapter.adaptToDataRecord(

(userId1, simclusterEmbedding, f2vEmbedding))

case ((userId, simclusterEmbedding), None) =>

UserSimclusterRecordAdapter.adaptToDataRecord(

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

}

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

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

val userIdFeature = SharedFeatures.USER\_ID

val userEmbeddingName = args.getOrElse("user\_embedding\_name", "output")

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

case (originalDataRecord, predictedDataRecord) =>

val userId = originalDataRecord.getFeatureValue(userIdFeature)

val scalaPredictedDataRecord =

ScalaToJavaDataRecordConversions.javaDataRecord2ScalaDataRecord(predictedDataRecord)

val userEmbeddingTensor =

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

val userEmbeddingWithEntity = getEmbeddingWithEntity(userEmbeddingTensor, userId)

userEmbeddingWithEntity

}

Util.printCounters(writeUserEmbedding(outputPipe, args))

}

}

object UserEmbeddingGenerationAdhocJob

extends TwitterExecutionApp

with UserEmbeddingGenerationTrait {

override def job: Execution[Unit] =

Execution.withId { implicit uid =>

Execution.withArgs { args =>

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

run(args)

}

}

}

object UserEmbeddingGenerationBatchJob

extends TwitterScheduledExecutionApp

with UserEmbeddingGenerationTrait {

override def scheduledJob: Execution[Unit] =

Execution.withId { implicit uid =>

Execution.withArgs { args =>

implicit val tz: TimeZone = DateOps.UTC

val batchFirstTime = BatchFirstTime(RichDate("2021-12-04")(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 UserEmbeddingGenerationBatchJobAlternate

extends TwitterScheduledExecutionApp

with UserEmbeddingGenerationTrait {

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 UserEmbeddingGenerationBatchJobExperimental

extends TwitterScheduledExecutionApp

with UserEmbeddingGenerationTrait {

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)

}

}

}

}