package com.twitter.simclusters\_v2.scalding

import com.twitter.logging.Logger

import com.twitter.scalding.\_

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

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

import com.twitter.scalding\_internal.dalv2.remote\_access.{ExplicitLocation, ProcAtla}

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

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

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

NormsAndCountsFixedPathSource,

ProducerNormsAndCountsScalaDataset

}

import com.twitter.simclusters\_v2.scalding.common.TypedRichPipe.\_

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

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

object ProducerNormsAndCounts {

def getNormsAndCounts(

input: TypedPipe[Edge]

)(

implicit uniqueID: UniqueID

): TypedPipe[NormsAndCounts] = {

val numRecordsInNormsAndCounts = Stat("num\_records\_in\_norms\_and\_counts")

input

.map {

case Edge(srcId, destId, isFollowEdge, favWt) =>

val followOrNot = if (isFollowEdge) 1 else 0

((srcId, destId), (followOrNot, favWt))

}

.sumByKey

// Uncomment for adhoc job

//.withReducers(2500)

.map {

case ((srcId, destId), (followOrNot, favWt)) =>

val favOrNot = if (favWt > 0) 1 else 0

val logFavScore = if (favWt > 0) UserUserNormalizedGraph.logTransformation(favWt) else 0.0

(

destId,

(

followOrNot,

favWt \* favWt,

favOrNot,

favWt,

favWt \* followOrNot.toDouble,

logFavScore \* logFavScore,

logFavScore,

logFavScore \* followOrNot.toDouble))

}

.sumByKey

// Uncomment for adhoc job

//.withReducers(500)

.map {

case (

id,

(

followCount,

favSumSquare,

favCount,

favSumOnFavEdges,

favSumOnFollowEdges,

logFavSumSquare,

logFavSumOnFavEdges,

logFavSumOnFollowEdges)) =>

val followerNorm = math.sqrt(followCount)

val faverNorm = math.sqrt(favSumSquare)

numRecordsInNormsAndCounts.inc()

NormsAndCounts(

userId = id,

followerL2Norm = Some(followerNorm),

faverL2Norm = Some(faverNorm),

followerCount = Some(followCount),

faverCount = Some(favCount),

favWeightsOnFavEdgesSum = Some(favSumOnFavEdges),

favWeightsOnFollowEdgesSum = Some(favSumOnFollowEdges),

logFavL2Norm = Some(math.sqrt(logFavSumSquare)),

logFavWeightsOnFavEdgesSum = Some(logFavSumOnFavEdges),

logFavWeightsOnFollowEdgesSum = Some(logFavSumOnFollowEdges)

)

}

}

def run(

halfLifeInDaysForFavScore: Int

)(

implicit uniqueID: UniqueID,

date: DateRange

): TypedPipe[NormsAndCounts] = {

val input =

UserUserNormalizedGraph.getFollowEdges.map {

case (src, dest) =>

Edge(src, dest, isFollowEdge = true, 0.0)

} ++ UserUserNormalizedGraph.getFavEdges(halfLifeInDaysForFavScore).map {

case (src, dest, wt) =>

Edge(src, dest, isFollowEdge = false, wt)

}

getNormsAndCounts(input)

}

}

object ProducerNormsAndCountsBatch extends TwitterScheduledExecutionApp {

private val firstTime: String = "2018-06-16"

implicit val tz = DateOps.UTC

implicit val parser = DateParser.default

private val batchIncrement: Duration = Days(7)

private val firstStartDate = DateRange.parse(firstTime).start

private val halfLifeInDaysForFavScore = 100

private val outputPath: String = "/user/cassowary/processed/producer\_norms\_and\_counts"

private val log = Logger()

private val execArgs = AnalyticsBatchExecutionArgs(

batchDesc = BatchDescription(this.getClass.getName.replace("$", "")),

firstTime = BatchFirstTime(RichDate(firstTime)),

lastTime = None,

batchIncrement = BatchIncrement(batchIncrement)

)

override def scheduledJob: Execution[Unit] = AnalyticsBatchExecution(execArgs) {

implicit dateRange =>

Execution.withId { implicit uniqueId =>

Execution.withArgs { args =>

Util.printCounters(

ProducerNormsAndCounts

.run(halfLifeInDaysForFavScore)

.writeDALSnapshotExecution(

ProducerNormsAndCountsScalaDataset,

D.Daily,

D.Suffix(outputPath),

D.EBLzo(),

dateRange.end)

)

}

}

}

}

object ProducerNormsAndCountsAdhoc extends TwitterExecutionApp {

implicit val tz: java.util.TimeZone = DateOps.UTC

implicit val dp = DateParser.default

def job: Execution[Unit] =

Execution.getConfigMode.flatMap {

case (config, mode) =>

Execution.withId { implicit uniqueId =>

val args = config.getArgs

implicit val date = DateRange.parse(args.list("date"))

Util.printCounters(

ProducerNormsAndCounts

.run(halfLifeInDaysForFavScore = 100)

.forceToDiskExecution.flatMap { result =>

Execution.zip(

result.writeExecution(NormsAndCountsFixedPathSource(args("outputDir"))),

result.printSummary("Producer norms and counts")

)

}

)

}

}

}

object DumpNormsAndCountsAdhoc extends TwitterExecutionApp {

implicit val tz: java.util.TimeZone = DateOps.UTC

def job: Execution[Unit] =

Execution.getConfigMode.flatMap {

case (config, mode) =>

Execution.withId { implicit uniqueId =>

val args = config.getArgs

val users = args.list("users").map(\_.toLong).toSet

val input = args.optional("inputDir") match {

case Some(inputDir) => TypedPipe.from(NormsAndCountsFixedPathSource(inputDir))

case None =>

DAL

.readMostRecentSnapshotNoOlderThan(ProducerNormsAndCountsScalaDataset, Days(30))

.withRemoteReadPolicy(ExplicitLocation(ProcAtla))

.toTypedPipe

}

if (users.isEmpty) {

input.printSummary("Producer norms and counts")

} else {

input

.collect {

case rec if users.contains(rec.userId) =>

Util.prettyJsonMapper.writeValueAsString(rec).replaceAll("\n", " ")

}

.toIterableExecution

.map { strings => println(strings.mkString("\n")) }

}

}

}

}