package com.twitter.simclusters\_v2.scalding

import com.twitter.bijection.Injection

import com.twitter.frigate.user\_sampler.common.EmployeeIds

import com.twitter.hashing.KeyHasher

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

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

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.TwitterScheduledExecutionApp

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

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

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

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

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

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

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

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

import flockdb\_tools.datasets.flock.FlockFollowsEdgesScalaDataset

case class Edge(srcId: Long, destId: Long, isFollowEdge: Boolean, favWeight: Double)

object UserUserNormalizedGraph {

// The common function for applying logarithmic transformation

def logTransformation(weight: Double): Double = {

math.max(math.log10(1.0 + weight), 0.0)

}

def getFollowEdges(implicit dateRange: DateRange, uniqueID: UniqueID): TypedPipe[(Long, Long)] = {

val numInputFollowEdges = Stat("num\_input\_follow\_edges")

DAL

.readMostRecentSnapshot(FlockFollowsEdgesScalaDataset)

.toTypedPipe

.collect {

case edge if edge.state == 0 =>

numInputFollowEdges.inc()

(edge.sourceId, edge.destinationId)

}

}

def transformFavEdges(

input: TypedPipe[EdgeWithDecayedWeights],

halfLifeInDaysForFavScore: Int

)(

implicit uniqueID: UniqueID

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

val numEdgesWithSpecifiedHalfLife = Stat(

s"num\_edges\_with\_specified\_half\_life\_${halfLifeInDaysForFavScore}\_days")

val numEdgesWithoutSpecifiedHalfLife = Stat(

s"num\_edges\_without\_specified\_half\_life\_${halfLifeInDaysForFavScore}\_days")

input

.flatMap { edge =>

if (edge.weights.halfLifeInDaysToDecayedSums.contains(halfLifeInDaysForFavScore)) {

numEdgesWithSpecifiedHalfLife.inc()

Some((edge.sourceId, edge.destinationId, edge.weights.halfLifeInDaysToDecayedSums(100)))

} else {

numEdgesWithoutSpecifiedHalfLife.inc()

None

}

}

}

def getFavEdges(

halfLifeInDaysForFavScore: Int

)(

implicit dateRange: DateRange,

uniqueID: UniqueID

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

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

transformFavEdges(

DAL

.readMostRecentSnapshot(UserUserFavGraphScalaDataset)

.withRemoteReadPolicy(ExplicitLocation(ProcAtla))

.toTypedPipe,

halfLifeInDaysForFavScore

)

}

def getNeighborWithWeights(

inputEdge: Edge,

followerL2NormOfDest: Double,

faverL2NormOfDest: Double,

logFavL2Norm: Double

): NeighborWithWeights = {

val normalizedFollowScore = {

val numerator = if (inputEdge.isFollowEdge) 1.0 else 0.0

if (followerL2NormOfDest > 0) numerator / followerL2NormOfDest else 0.0

}

val normalizedFavScore =

if (faverL2NormOfDest > 0) inputEdge.favWeight / faverL2NormOfDest else 0.0

val logFavScore = if (inputEdge.favWeight > 0) logTransformation(inputEdge.favWeight) else 0.0

val logFavScoreL2Normalized = if (logFavL2Norm > 0) logFavScore / logFavL2Norm else 0.0

NeighborWithWeights(

inputEdge.destId,

Some(inputEdge.isFollowEdge),

Some(normalizedFollowScore),

Some(inputEdge.favWeight),

Some(normalizedFavScore),

logFavScore = Some(logFavScore),

logFavScoreL2Normalized = Some(logFavScoreL2Normalized)

)

}

def addNormalizedWeightsAndAdjListify(

input: TypedPipe[Edge],

maxNeighborsPerUser: Int,

normsAndCountsFull: TypedPipe[NormsAndCounts]

)(

implicit uniqueId: UniqueID

): TypedPipe[UserAndNeighbors] = {

val numUsersNeedingNeighborTruncation = Stat("num\_users\_needing\_neighbor\_truncation")

val numEdgesAfterTruncation = Stat("num\_edges\_after\_truncation")

val numEdgesBeforeTruncation = Stat("num\_edges\_before\_truncation")

val numFollowEdgesBeforeTruncation = Stat("num\_follow\_edges\_before\_truncation")

val numFavEdgesBeforeTruncation = Stat("num\_fav\_edges\_before\_truncation")

val numFollowEdgesAfterTruncation = Stat("num\_follow\_edges\_after\_truncation")

val numFavEdgesAfterTruncation = Stat("num\_fav\_edges\_after\_truncation")

val numRecordsInOutputGraph = Stat("num\_records\_in\_output\_graph")

val norms = normsAndCountsFull.map { record =>

(

record.userId,

(

record.followerL2Norm.getOrElse(0.0),

record.faverL2Norm.getOrElse(0.0),

record.logFavL2Norm.getOrElse(0.0)))

}

implicit val l2b: Long => Array[Byte] = Injection.long2BigEndian

input

.map { edge => (edge.destId, edge) }

.sketch(reducers = 2000)

.join(norms)

.map {

case (destId, (edge, (followNorm, favNorm, logFavNorm))) =>

numEdgesBeforeTruncation.inc()

if (edge.isFollowEdge) numFollowEdgesBeforeTruncation.inc()

if (edge.favWeight > 0) numFavEdgesBeforeTruncation.inc()

(edge.srcId, getNeighborWithWeights(edge, followNorm, favNorm, logFavNorm))

}

.group

//.withReducers(1000)

.sortedReverseTake(maxNeighborsPerUser)(Ordering.by { x: NeighborWithWeights =>

(

x.favScoreHalfLife100Days.getOrElse(0.0),

x.followScoreNormalizedByNeighborFollowersL2.getOrElse(0.0)

)

})

.map {

case (srcId, neighborList) =>

if (neighborList.size >= maxNeighborsPerUser) numUsersNeedingNeighborTruncation.inc()

neighborList.foreach { neighbor =>

numEdgesAfterTruncation.inc()

if (neighbor.favScoreHalfLife100Days.exists(\_ > 0)) numFavEdgesAfterTruncation.inc()

if (neighbor.isFollowed.contains(true)) numFollowEdgesAfterTruncation.inc()

}

numRecordsInOutputGraph.inc()

UserAndNeighbors(srcId, neighborList)

}

}

def combineFollowAndFav(

followEdges: TypedPipe[(Long, Long)],

favEdges: TypedPipe[(Long, Long, Double)]

): TypedPipe[Edge] = {

(

followEdges.map { case (src, dest) => ((src, dest), (1, 0.0)) } ++

favEdges.map { case (src, dest, wt) => ((src, dest), (0, wt)) }

).sumByKey

//.withReducers(2500)

.map {

case ((src, dest), (follow, favWt)) =>

Edge(src, dest, isFollowEdge = follow > 0, favWt)

}

}

def run(

followEdges: TypedPipe[(Long, Long)],

favEdges: TypedPipe[(Long, Long, Double)],

normsAndCounts: TypedPipe[NormsAndCounts],

maxNeighborsPerUser: Int

)(

implicit uniqueID: UniqueID

): TypedPipe[UserAndNeighbors] = {

val combined = combineFollowAndFav(followEdges, favEdges)

addNormalizedWeightsAndAdjListify(

combined,

maxNeighborsPerUser,

normsAndCounts

)

}

}

object UserUserNormalizedGraphBatch 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 halfLifeInDaysForFavScore = 100

private val outputPath: String = "/user/cassowary/processed/user\_user\_normalized\_graph"

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 =>

val maxNeighborsPerUser = args.int("maxNeighborsPerUser", 2000)

val producerNormsAndCounts =

DAL.readMostRecentSnapshot(ProducerNormsAndCountsScalaDataset).toTypedPipe

Util.printCounters(

UserUserNormalizedGraph

.run(

UserUserNormalizedGraph.getFollowEdges,

UserUserNormalizedGraph.getFavEdges(halfLifeInDaysForFavScore),

producerNormsAndCounts,

maxNeighborsPerUser

)

.writeDALSnapshotExecution(

UserUserNormalizedGraphScalaDataset,

D.Daily,

D.Suffix(outputPath),

D.EBLzo(),

dateRange.end)

)

}

}

}

}

object UserUserNormalizedGraphAdhoc extends TwitterExecutionApp {

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

implicit val dp = DateParser.default

val log = Logger()

def hashToLong(input: Long): Long = {

val bb = java.nio.ByteBuffer.allocate(8)

bb.putLong(input)

Math.abs(KeyHasher.KETAMA.hashKey(bb.array()))

}

def job: Execution[Unit] =

Execution.getConfigMode.flatMap {

case (config, mode) =>

Execution.withId { implicit uniqueId =>

val args = config.getArgs

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

val halfLifeInDaysForFavScore = 100

val maxNeighborsPerUser = args.int("maxNeighborsPerUser", 2000)

val producerNormsAndCounts = TypedPipe.from(

NormsAndCountsFixedPathSource(args("normsInputDir"))

)

val favEdges = args.optional("favGraphInputDir") match {

case Some(favGraphInputDir) =>

UserUserNormalizedGraph.transformFavEdges(

TypedPipe.from(

EdgeWithDecayedWtsFixedPathSource(favGraphInputDir)

),

halfLifeInDaysForFavScore

)

case None =>

UserUserNormalizedGraph.getFavEdges(halfLifeInDaysForFavScore)

}

val followEdges = UserUserNormalizedGraph.getFollowEdges

Util.printCounters(

UserUserNormalizedGraph

.run(

followEdges,

favEdges,

producerNormsAndCounts,

maxNeighborsPerUser

).writeExecution(UserAndNeighborsFixedPathSource(args("outputDir")))

)

}

}

}

object DumpUserUserGraphAdhoc 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 input = args.optional("inputDir") match {

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

case None =>

DAL

.readMostRecentSnapshotNoOlderThan(UserUserNormalizedGraphScalaDataset, Days(30))

.withRemoteReadPolicy(ExplicitLocation(ProcAtla))

.toTypedPipe

}

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

if (users.isEmpty) {

input.printSummary("Producer norms and counts")

} else {

input

.collect {

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

(Seq(rec.userId.toString) ++ rec.neighbors.map { n =>

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

}).mkString("\n")

}

.toIterableExecution

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

}

}

}

}

/\*

\* ./bazel bundle src/scala/com/twitter/simclusters\_v2/scalding:user\_user\_normalized\_graph && \

\* oscar hdfs --host hadoopnest2.atla.twitter.com --bundle user\_user\_normalized\_graph \

\* --tool com.twitter.simclusters\_v2.scalding.EmployeeGraph --screen --screen-detached \

\* --tee your\_ldap/employeeGraph20190809 -- --outputDir adhoc/employeeGraph20190809

\*/

object EmployeeGraph 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 input = args.optional("inputDir") match {

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

case None =>

DAL

.readMostRecentSnapshotNoOlderThan(UserUserNormalizedGraphScalaDataset, Days(30))

.withRemoteReadPolicy(ExplicitLocation(ProcAtla))

.toTypedPipe

}

val employeeIds = EmployeeIds.buildMerlinClientAndGetEmployees("frigate-scalding.dev")

input

.collect {

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

rec.neighbors.collect {

case n if employeeIds.contains(n.neighborId) =>

(

rec.userId,

n.neighborId,

n.favScoreHalfLife100Days.getOrElse(0),

n.isFollowed.getOrElse(false))

}

}

.flatten

.writeExecution(TypedTsv(args("outputDir")))

}

}

}

/\*

\* scalding remote run --target src/scala/com/twitter/simclusters\_v2/scalding:employee\_graph\_from\_user\_user

\* --main-class com.twitter.simclusters\_v2.scalding.EmployeeGraphFromUserUser

\* --submitter hadoopnest2.atla.twitter.com --user recos-platform -- --graphOutputDir "/user/recos-platform/adhoc/employee\_graph\_from\_user\_user/"

\*/

object EmployeeGraphFromUserUser 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 graphOutputDir = args("graphOutputDir")

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

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

case None =>

DAL

.readMostRecentSnapshotNoOlderThan(UserUserNormalizedGraphScalaDataset, Days(30))

.withRemoteReadPolicy(ExplicitLocation(ProcAtla))

.toTypedPipe

}

val employeeIds = EmployeeIds.buildMerlinClientAndGetEmployees("frigate-scalding.dev")

input

.collect {

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

rec

}

.writeExecution(UserAndNeighborsFixedPathSource(graphOutputDir))

}

}

}

/\*

\* ./bazel bundle src/scala/com/twitter/simclusters\_v2/scalding:user\_user\_normalized\_graph && \

\* oscar hdfs --host hadoopnest2.atla.twitter.com --bundle user\_user\_normalized\_graph \

\* --tool com.twitter.simclusters\_v2.scalding.VitGraph --screen --screen-detached \

\* --tee your\_ldap/vitGraph20190809 -- --outputDir adhoc/vitGraph20190809

\*/

object VitGraph 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 minActiveFollowers = args.int("minActiveFollowers")

val topK = args.int("topK")

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

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

case None =>

DAL

.readMostRecentSnapshotNoOlderThan(UserUserNormalizedGraphScalaDataset, Days(30))

.withRemoteReadPolicy(ExplicitLocation(ProcAtla))

.toTypedPipe

}

val userSource =

DAL.readMostRecentSnapshotNoOlderThan(UsersourceFlatScalaDataset, Days(30)).toTypedPipe

TopUsersSimilarityGraph

.vits(userSource, minActiveFollowers, topK).toIterableExecution.flatMap { vitsIter =>

val vits = vitsIter.toSet

println(s"Found ${vits.size} many vits. First few: " + vits.take(5).mkString(","))

input

.collect {

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

rec.neighbors.collect {

case n if vits.contains(n.neighborId) =>

(

rec.userId,

n.neighborId,

n.favScoreHalfLife100Days.getOrElse(0),

n.isFollowed.getOrElse(false))

}

}

.flatten

.writeExecution(TypedTsv(args("outputDir")))

}

}

}

}