package com.twitter.simclusters\_v2.scalding

import com.twitter.algebird.Semigroup

import com.twitter.bijection.Injection

import com.twitter.dal.client.dataset.KeyValDALDataset

import com.twitter.scalding.\_

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

import com.twitter.scalding\_internal.dalv2.DALWrite.{D, WriteExtension}

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

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

AnalyticsBatchExecution,

AnalyticsBatchExecutionArgs,

BatchDescription,

BatchFirstTime,

BatchIncrement,

TwitterScheduledExecutionApp

}

import com.twitter.scalding\_internal.multiformat.format.keyval.KeyVal

import com.twitter.simclusters\_v2.common.{ClusterId, ModelVersions, UserId}

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

AdhocKeyValSources,

InternalDataPaths,

SimclustersV2KnownFor20M145K2020ScalaDataset,

SimclustersV2RawInterestedInLite20M145K2020ScalaDataset,

SimclustersV2RawInterestedIn20M145KUpdatedScalaDataset,

UserAndNeighborsFixedPathSource,

UserUserGraphScalaDataset

}

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

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

ClustersUserIsInterestedIn,

ClustersUserIsKnownFor,

UserAndNeighbors,

UserToInterestedInClusterScores

}

import com.twitter.wtf.scalding.jobs.common.AdhocExecutionApp

import java.util.TimeZone

/\*\*

\* This file implements the job for computing users' interestedIn vector from KnownFor data set.

\*

\* It reads the UserUserGraphScalaDataset to get user-user follow + fav graph, and then

\* based on the known-for clusters of each followed/faved user, we calculate how much a user is

\* interestedIn a cluster.

\*

\* The main differences of the InterestedInFromKnownForLite compared to InterestedInFromKnownFor are

\* the following:

\* - We read the UserUserGraph dataset that doesnot contain the producer normalized scores

\* - We donot compute the cluster normalized scores for the clusters per user

\* - For social proof thresholding, we donot keep track of the entire list of follow and

\* fav social proofs but rather make use of numFollowSocial and numFavSocial (this introduces

\* some noise if follow and fav social proof contain the same users)

\* - Store 200 clusters per user compared to 50 in IIKF

\* - Runs more frequently compared to weekly in IIKF

\*/

/\*\*

\* Production job for computing interestedIn data set for the model version 20M145K2020.

\*

\* To deploy the job:

\*

\* capesospy-v2 update --build\_locally --start\_cron interested\_in\_lite\_for\_20M\_145k\_2020 \

src/scala/com/twitter/simclusters\_v2/capesos\_config/atla\_proc.yaml

\*/

object InterestedInFromKnownForLite20M145K2020 extends InterestedInFromKnownForLite {

override val firstTime: String = "2021-04-24"

override val outputKVDataset: KeyValDALDataset[KeyVal[Long, ClustersUserIsInterestedIn]] =

SimclustersV2RawInterestedInLite20M145K2020ScalaDataset

override val outputPath: String = InternalDataPaths.RawInterestedInLite2020Path

override val knownForModelVersion: String = ModelVersions.Model20M145K2020

override val knownForDALDataset: KeyValDALDataset[KeyVal[Long, ClustersUserIsKnownFor]] =

SimclustersV2KnownFor20M145K2020ScalaDataset

}

trait InterestedInFromKnownForLite extends TwitterScheduledExecutionApp {

implicit val tz = DateOps.UTC

implicit val parser = DateParser.default

def firstTime: String

val batchIncrement: Duration = Days(2)

val lookBackDays: Duration = Days(30)

def outputKVDataset: KeyValDALDataset[KeyVal[Long, ClustersUserIsInterestedIn]]

def outputPath: String

def knownForModelVersion: String

def knownForDALDataset: KeyValDALDataset[KeyVal[Long, ClustersUserIsKnownFor]]

private lazy 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 userUserGraph =

DAL.readMostRecentSnapshot(UserUserGraphScalaDataset).toTypedPipe

val knownFor = KnownForSources.fromKeyVal(

DAL.readMostRecentSnapshot(knownForDALDataset, dateRange.extend(Days(30))).toTypedPipe,

knownForModelVersion

)

val socialProofThreshold = args.int("socialProofThreshold", 2)

val maxClustersPerUser = args.int("maxClustersPerUser", 200)

val result = InterestedInFromKnownForLite

.run(

userUserGraph,

knownFor,

socialProofThreshold,

maxClustersPerUser,

knownForModelVersion

)

val writeKeyValResultExec = result

.map {

case (userId, clusters) => KeyVal(userId, clusters)

}.writeDALVersionedKeyValExecution(

outputKVDataset,

D.Suffix(outputPath)

)

Util.printCounters(writeKeyValResultExec)

}

}

}

}

/\*\*

\* Adhoc job to compute user interestedIn.

\*

\* scalding remote run \

\* --target src/scala/com/twitter/simclusters\_v2/scalding:interested\_in\_lite\_20m\_145k\_2020-adhoc \

\* --main-class com.twitter.simclusters\_v2.scalding.InterestedInFromKnownForLite20M145K2020Adhoc \

\* --user cassowary --cluster bluebird-qus1 \

\* --keytab /var/lib/tss/keys/fluffy/keytabs/client/cassowary.keytab \

\* --principal service\_acoount@TWITTER.BIZ \

\* -- \

\* --outputDir /gcs/user/cassowary/adhoc/interested\_in\_from\_knownfor\_lite/ \

\* --date 2020-08-25

\*/

object InterestedInFromKnownForLite20M145K2020Adhoc extends AdhocExecutionApp {

override def runOnDateRange(

args: Args

)(

implicit dateRange: DateRange,

timeZone: TimeZone,

uniqueID: UniqueID

): Execution[Unit] = {

val userUserGraph = DAL.readMostRecentSnapshot(UserUserGraphScalaDataset).toTypedPipe

val socialProofThreshold = args.int("socialProofThreshold", 2)

val maxClustersPerUser = args.int("maxClustersPerUser", 200)

val knownForModelVersion = ModelVersions.Model20M145K2020

val knownFor = KnownForSources.fromKeyVal(

DAL

.readMostRecentSnapshotNoOlderThan(

SimclustersV2KnownFor20M145K2020ScalaDataset,

Days(30)).toTypedPipe,

knownForModelVersion

)

val outputSink = AdhocKeyValSources.interestedInSource(args("outputDir"))

Util.printCounters(

InterestedInFromKnownForLite

.run(

userUserGraph,

knownFor,

socialProofThreshold,

maxClustersPerUser,

knownForModelVersion

).writeExecution(outputSink)

)

}

}

object InterestedInFromKnownForLite {

private def ifNanMake0(x: Double): Double = if (x.isNaN) 0.0 else x

case class SrcClusterIntermediateInfo(

followScore: Double,

favScore: Double,

logFavScore: Double,

numFollowed: Int,

numFaved: Int) {

// helper function used for test cases

override def equals(obj: scala.Any): Boolean = {

obj match {

case that: SrcClusterIntermediateInfo =>

math.abs(followScore - that.followScore) < 1e-5 &&

math.abs(favScore - that.favScore) < 1e-5 &&

math.abs(logFavScore - that.logFavScore) < 1e-5 &&

numFollowed == that.numFollowed &&

numFaved == that.numFaved

case \_ => false

}

}

}

implicit object SrcClusterIntermediateInfoSemigroup

extends Semigroup[SrcClusterIntermediateInfo] {

override def plus(

left: SrcClusterIntermediateInfo,

right: SrcClusterIntermediateInfo

): SrcClusterIntermediateInfo = {

SrcClusterIntermediateInfo(

followScore = left.followScore + right.followScore,

favScore = left.favScore + right.favScore,

logFavScore = left.logFavScore + right.logFavScore,

numFollowed = left.numFollowed + right.numFollowed,

numFaved = left.numFaved + right.numFaved

)

}

}

def run(

adjacencyLists: TypedPipe[UserAndNeighbors],

knownFor: TypedPipe[(UserId, Array[(ClusterId, Float)])],

socialProofThreshold: Int,

maxClustersPerUser: Int,

knownForModelVersion: String

)(

implicit uniqueId: UniqueID

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

InterestedInFromKnownFor.keepOnlyTopClusters(

groupClusterScores(

userClusterPairs(

adjacencyLists,

knownFor,

socialProofThreshold

)

),

maxClustersPerUser,

knownForModelVersion

)

}

def userClusterPairs(

adjacencyLists: TypedPipe[UserAndNeighbors],

knownFor: TypedPipe[(Long, Array[(Int, Float)])],

socialProofThreshold: Int

)(

implicit uniqueId: UniqueID

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

val edgesToUsersWithKnownFor = Stat("num\_edges\_to\_users\_with\_known\_for")

val srcDestClusterTriples = Stat("num\_src\_dest\_cluster\_triples")

val srcClusterPairsBeforeSocialProofThresholding =

Stat("num\_src\_cluster\_pairs\_before\_social\_proof\_thresholding")

val srcClusterPairsAfterSocialProofThresholding =

Stat("num\_src\_cluster\_pairs\_after\_social\_proof\_thresholding")

val edges = adjacencyLists.flatMap {

case UserAndNeighbors(srcId, neighborsWithWeights) =>

neighborsWithWeights.map { neighborWithWeights =>

(

neighborWithWeights.neighborId,

neighborWithWeights.copy(neighborId = srcId)

)

}

}

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

edges

.sketch(4000)

.join(knownFor)

.flatMap {

case (destId, (srcWithWeights, clusterArray)) =>

edgesToUsersWithKnownFor.inc()

clusterArray.toList.map {

case (clusterId, knownForScoreF) =>

val knownForScore = math.max(0.0, knownForScoreF.toDouble)

srcDestClusterTriples.inc()

val followScore =

if (srcWithWeights.isFollowed.contains(true)) knownForScore else 0.0

val favScore =

srcWithWeights.favScoreHalfLife100Days.getOrElse(0.0) \* knownForScore

val logFavScore = srcWithWeights.logFavScore.getOrElse(0.0) \* knownForScore

val numFollowed = if (srcWithWeights.isFollowed.contains(true)) {

1

} else 0

val numFaved = if (srcWithWeights.favScoreHalfLife100Days.exists(\_ > 0)) {

1

} else 0

(

(srcWithWeights.neighborId, clusterId),

SrcClusterIntermediateInfo(

followScore,

favScore,

logFavScore,

numFollowed,

numFaved

)

)

}

}

.sumByKey

.withReducers(10000)

.filter {

case ((\_, \_), SrcClusterIntermediateInfo(\_, \_, \_, numFollowed, numFaved)) =>

srcClusterPairsBeforeSocialProofThresholding.inc()

// we donot remove duplicates

val socialProofSize = numFollowed + numFaved

val result = socialProofSize >= socialProofThreshold

if (result) {

srcClusterPairsAfterSocialProofThresholding.inc()

}

result

}

}

def groupClusterScores(

intermediate: TypedPipe[((Long, Int), SrcClusterIntermediateInfo)]

)(

implicit uniqueId: UniqueID

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

implicit val i2b: Int => Array[Byte] = Injection.int2BigEndian

intermediate

.map {

case (

(srcId, clusterId),

SrcClusterIntermediateInfo(

followScore,

favScore,

logFavScore,

numFollowed,

numFaved

)) =>

(

srcId,

List(

(

clusterId,

UserToInterestedInClusterScores(

followScore = Some(ifNanMake0(followScore)),

favScore = Some(ifNanMake0(favScore)),

logFavScore = Some(ifNanMake0(logFavScore)),

numUsersBeingFollowed = Some(numFollowed),

numUsersThatWereFaved = Some(numFaved)

))

)

)

}

.sumByKey

// .withReducers(1000)

.toTypedPipe

}

}