package com.twitter.simclusters\_v2.scalding

import com.twitter.algebird.Monoid

import com.twitter.algebird.mutable.PriorityQueueMonoid

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

import com.twitter.pluck.source.cassowary.FollowingsCosineSimilaritiesManhattanSource

import com.twitter.scalding.\_

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

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

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

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

import com.twitter.simclusters\_v2.common.ModelVersions

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.scalding.common.Util.Distribution

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

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

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

import java.util.PriorityQueue

import scala.collection.JavaConverters.\_

object ClusterEvaluation {

val samplerMonoid: PriorityQueueMonoid[((Long, Long), (Double, Double))] =

Util.reservoirSamplerMonoidForPairs[(Long, Long), (Double, Double)](5000)(Util.edgeOrdering)

case class ClusterResults(

numEdgesInsideCluster: Int,

wtOfEdgesInsideCluster: Double,

numEdgesOutsideCluster: Int,

wtOfEdgesOutsideCluster: Double,

originalWtAndProductOfNodeScoresSample: PriorityQueue[((Long, Long), (Double, Double))]) {

def clusterQuality(clusterSize: Int, averagePrecisionWholeGraph: Double): ClusterQuality = {

val unweightedRecallDenominator = numEdgesInsideCluster + numEdgesOutsideCluster

val unweightedRecall = if (unweightedRecallDenominator > 0) {

numEdgesInsideCluster.toDouble / unweightedRecallDenominator.toDouble

} else 0.0

val weightedRecallDenominator = wtOfEdgesInsideCluster + wtOfEdgesOutsideCluster

val weightedRecall = if (weightedRecallDenominator > 0) {

wtOfEdgesInsideCluster / weightedRecallDenominator

} else 0.0

val precision = if (clusterSize > 1) {

Some(wtOfEdgesInsideCluster / (clusterSize \* (clusterSize - 1)))

} else Some(0.0)

val relativePrecision = if (averagePrecisionWholeGraph > 0) {

precision.flatMap { p => Some(p / averagePrecisionWholeGraph) }

} else Some(0.0)

ClusterQuality(

unweightedRecall = Some(unweightedRecall),

weightedRecall = Some(weightedRecall),

unweightedRecallDenominator = Some(unweightedRecallDenominator),

weightedRecallDenominator = Some(weightedRecallDenominator),

relativePrecisionNumerator = precision,

relativePrecision = relativePrecision,

weightAndProductOfNodeScoresCorrelation = Some(

Util.computeCorrelation(

originalWtAndProductOfNodeScoresSample.iterator.asScala.map(\_.\_2)))

)

}

}

object ClusterResultsMonoid extends Monoid[ClusterResults] {

override def zero = ClusterResults(0, 0, 0, 0, samplerMonoid.zero)

override def plus(l: ClusterResults, r: ClusterResults) = ClusterResults(

l.numEdgesInsideCluster + r.numEdgesInsideCluster,

l.wtOfEdgesInsideCluster + r.wtOfEdgesInsideCluster,

l.numEdgesOutsideCluster + r.numEdgesOutsideCluster,

l.wtOfEdgesOutsideCluster + r.wtOfEdgesOutsideCluster,

samplerMonoid

.plus(l.originalWtAndProductOfNodeScoresSample, r.originalWtAndProductOfNodeScoresSample)

)

}

/\*\*

\* Evaluate the quality of a cluster.

\* @param memberScores A map with the members of the cluster as the keys and their scores

\* inside the cluster as values. The more central a member is inside the score,

\* the higher it's score is.

\* @param membersAdjLists A map that gives the weighted neighbors of each member in the cluster.

\*/

def evaluateCluster(

memberScores: Map[Long, Double],

membersAdjLists: Map[Long, Map[Long, Float]]

): ClusterResults = {

val resultsIter = membersAdjLists.flatMap {

case (fromNodeId, adjList) =>

val fromNodeWt = memberScores.getOrElse(fromNodeId, 0.0)

adjList.map {

case (toNodeId, edgeWt) =>

if (memberScores.contains(toNodeId)) {

val productOfMembershipScores = fromNodeWt \* memberScores(toNodeId)

ClusterResults(

1,

edgeWt,

0,

0,

samplerMonoid.build(

((fromNodeId, toNodeId), (edgeWt.toDouble, productOfMembershipScores))))

} else {

ClusterResults(0, 0, 1, edgeWt, samplerMonoid.zero)

}

}

}

Monoid.sum(resultsIter)(ClusterResultsMonoid)

}

/\*\*

\* Evaluate each cluster with respect to the provided graph.

\* @param graph graph represented via the adjacency lists of each node, needs to be symmetrized i.e. if u is in v's adjlist, then v needs to be in u's adjlist as well

\* @param clusters cluster memberships of each node.

\* @param statsPrefix convenience argument to act as prefix for stats counters

\* @return key-value pipe with clusterId as key and (size of the cluster, quality struct) as value

\*/

def clusterLevelEvaluation(

graph: TypedPipe[(Long, Map[Long, Float])],

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

statsPrefix: String = ""

)(

implicit uniqueId: UniqueID

): Execution[TypedPipe[(Int, (Int, ClusterQuality))]] = {

val numRealClusters = Stat(s"${statsPrefix}/numRealClusters")

val numFakeClusters = Stat(s"${statsPrefix}/numFakeClusters")

val numNodesAndEdgesExec = graph

.map {

case (nId, nbrMap) =>

(1L, nbrMap.size.toLong, nbrMap.values.sum.toDouble)

}.sum.getExecution

numNodesAndEdgesExec.map {

case (numNodes, numEdges, sumOfAllEdgeWts) =>

println("numNodes " + numNodes)

println("numEdges " + numEdges)

println("sumOfAllEdgeWts " + sumOfAllEdgeWts)

val numFakeClustersForUnassignedNodes = numNodes / 1e4

val averagePrecisionWholeGraph = sumOfAllEdgeWts / (numNodes \* (numNodes - 1))

graph

.leftJoin(clusters)

// uncomment for adhoc job

.withReducers(200)

.flatMap {

case (nodeId, (adjList, assignedClustersOpt)) =>

val nodeDegree = adjList.size.toLong

val nodeWeightedDegree = adjList.values.sum

assignedClustersOpt match {

case Some(assignedClusters) if assignedClusters.nonEmpty =>

assignedClusters.toList.map {

case (clusterId, scoreOfNodeInCluster) =>

(

clusterId,

(

Map(nodeId -> (scoreOfNodeInCluster.toDouble, adjList)),

1,

nodeDegree,

nodeWeightedDegree))

}

case \_ =>

// For nodes that don't belong to any cluster, create a fake clusterId (0 or lesser)

// and add the node's statistics to that clusterId. We don't need the adjacency lists for

// unassigned nodes, we'll simply track how many edges are incident on those nodes and their weighted sum etc

val fakeClusterId =

(-1 \* (math.abs(

Util.hashToLong(nodeId)) % numFakeClustersForUnassignedNodes)).toInt

List(

(

fakeClusterId,

(

Map.empty[Long, (Double, Map[Long, Float])],

1,

nodeDegree,

nodeWeightedDegree)))

}

}

.sumByKey

// uncomment for adhoc job

.withReducers(60)

.map {

case (clusterId, (membersMap, clusterSize, volumeOfCluster, weightedVolumeOfCluster)) =>

if (clusterId > 0) {

numRealClusters.inc()

val scoresMap =

if (clusterId > 0) membersMap.mapValues(\_.\_1) else Map.empty[Long, Double]

val adjListsMap = membersMap.mapValues(\_.\_2)

val quality = evaluateCluster(scoresMap, adjListsMap)

.clusterQuality(clusterSize, averagePrecisionWholeGraph)

(clusterId, (clusterSize, quality))

} else {

// clusterId <= 0 means that this is a fake cluster.

numFakeClusters.inc()

(

clusterId,

(

clusterSize,

ClusterQuality(

unweightedRecallDenominator = Some(volumeOfCluster),

weightedRecallDenominator = Some(weightedVolumeOfCluster)

)

)

)

}

}

}

}

case class OverallResults(

unweightedRecall: Double,

edgesInsideClusters: Long,

allEdges: Long,

allNodes: Int,

weightedRecall: Double,

wtOnEdgesInsideClusters: Double,

wtOnAllEdges: Double,

weightCorrelation: Double,

relativePrecision: Double,

numUnassignedNodes: Int,

numAssignedNodes: Int,

sizeDist: Distribution,

recallDist: Distribution,

weightedRecallDist: Distribution,

relativePrecisionDist: Distribution,

weightCorrelationDist: Distribution,

numClustersWithNegativeCorrelation: Double,

numClustersWithZeroRecall: Double,

numClustersWithLessThanOneRelativePrecision: Double,

numSingletonClusters: Int)

def summarizePerClusterResults(

perClusterResults: TypedPipe[(Int, (Int, ClusterQuality))]

): Execution[Option[OverallResults]] = {

perClusterResults

.map {

case (clusterId, (size, quality)) =>

val unweightedRecallDen = quality.unweightedRecallDenominator.getOrElse(0.0)

val unweightedRecallNum = quality.unweightedRecall.getOrElse(0.0) \* unweightedRecallDen

val weightedRecallDen = quality.weightedRecallDenominator.getOrElse(0.0)

val weightedRecallNum = quality.weightedRecall.getOrElse(0.0) \* weightedRecallDen

val weightCorrelationDen = size

val weightCorrelationNum =

weightCorrelationDen \* quality.weightAndProductOfNodeScoresCorrelation

.getOrElse(0.0)

val relativePrecisionDen = size

val relativePrecisionNum = relativePrecisionDen \* quality.relativePrecision.getOrElse(0.0)

val numClustersWithNegativeCorrelation =

if (weightCorrelationNum < 0 && clusterId > 0) 1 else 0

val numClustersWithLessThanOneRelativePrecision =

if (quality.relativePrecision.getOrElse(0.0) < 1 && clusterId > 0) 1 else 0

val numClustersWithZeroRecall = if (weightedRecallNum < 1e-5 && clusterId > 0) 1 else 0

val numUnassignedNodes = if (clusterId < 1) size else 0

val numAssignedNodes = if (clusterId > 0) size else 0

val numSingletonClusters = if (clusterId > 0 && size == 1) 1 else 0

(

unweightedRecallDen,

unweightedRecallNum,

weightedRecallDen,

weightedRecallNum,

weightCorrelationDen,

weightCorrelationNum,

relativePrecisionDen,

relativePrecisionNum,

numClustersWithNegativeCorrelation,

numClustersWithLessThanOneRelativePrecision,

numClustersWithZeroRecall,

List(size.toDouble),

List(quality.unweightedRecall.getOrElse(0.0)),

List(quality.weightedRecall.getOrElse(0.0)),

List(quality.relativePrecision.getOrElse(0.0)),

List(quality.weightAndProductOfNodeScoresCorrelation.getOrElse(0.0)),

numUnassignedNodes,

numAssignedNodes,

numSingletonClusters

)

}

.sum

.toOptionExecution

.map { opt =>

opt.map {

case (

unweightedRecallDen,

unweightedRecallNum,

weightedRecallDen,

weightedRecallNum,

weightCorrelationDen,

weightCorrelationNum,

relativePrecisionDen,

relativePrecisionNum,

numClustersWithNegativeCorrelation,

numClustersWithLessThanOneRelativePrecision,

numClustersWithZeroRecall,

sizeList,

unweightedRecallList,

weightedRecallList,

relativePrecisionList,

weightCorrelationList,

numUnassignedNodes,

numAssignedNodes,

numSingletonClusters) =>

OverallResults(

unweightedRecall = unweightedRecallNum / unweightedRecallDen,

edgesInsideClusters = unweightedRecallNum.toLong,

allEdges = unweightedRecallDen.toLong,

allNodes = numAssignedNodes + numUnassignedNodes,

weightedRecall = weightedRecallNum / weightedRecallDen,

wtOnEdgesInsideClusters = weightedRecallNum,

wtOnAllEdges = weightedRecallDen,

weightCorrelation = weightCorrelationNum / weightCorrelationDen,

relativePrecision = relativePrecisionNum / relativePrecisionDen,

numAssignedNodes = numAssignedNodes,

numUnassignedNodes = numUnassignedNodes,

sizeDist = Util.distributionFromArray(sizeList.toArray),

recallDist = Util.distributionFromArray(unweightedRecallList.toArray),

weightedRecallDist = Util.distributionFromArray(weightedRecallList.toArray),

weightCorrelationDist = Util.distributionFromArray(weightCorrelationList.toArray),

relativePrecisionDist = Util.distributionFromArray(relativePrecisionList.toArray),

numClustersWithNegativeCorrelation = numClustersWithNegativeCorrelation,

numClustersWithLessThanOneRelativePrecision =

numClustersWithLessThanOneRelativePrecision,

numClustersWithZeroRecall = numClustersWithZeroRecall,

numSingletonClusters = numSingletonClusters

)

}

}

}

/\*\*

\* @param graph Input similarity graph, needs to be symmetrized i.e. if u is in v's adjlist, then v needs to be in u's adjlist as well

\* @param clusters cluster assignments to be evaluated

\* @return summary of results

\*/

def overallEvaluation(

graph: TypedPipe[(Long, Map[Long, Float])],

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

statsPrefix: String

)(

implicit uniqueId: UniqueID

): Execution[Option[OverallResults]] = {

clusterLevelEvaluation(graph, clusters, statsPrefix).flatMap(summarizePerClusterResults)

}

}

/\*\*

\* ./bazel bundle src/scala/com/twitter/simclusters\_v2/scalding:cluster\_evaluation && \

\* oscar hdfs --user frigate --host hadoopnest1.atla.twitter.com --bundle cluster\_evaluation \

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

\* --tee logs/clusterQualityFor\_updatedUnnormalizedInputScores\_usingSims20190318 -- \

\* --simsInputDir /user/frigate/your\_ldap/commonDirForClusterEvaluation/classifiedSims\_20190314\_copiedFromAtlaProc \

\* --topK 20000000 --date 2019-03-18 --minActiveFollowers 400 \

\* --topUsersDir /user/frigate/your\_ldap/commonDirForClusterEvaluation/top20MUsers\_minActiveFollowers400\_20190215 \

\* --maxSimsNeighborsForEval 40 \

\* --preparedSimsGraph /user/frigate/your\_ldap/commonDirForClusterEvaluation/symmetrized\_classifiedSims20190318\_top20MUsers \

\* --outputDir /user/frigate/your\_ldap/dirFor\_updatedKnownFor20M\_145K\_dec11\_usingSims20190127\_unnormalizedInputScores/knownForClusterEvaluation \

\* --knownForDir /user/frigate/your\_ldap/dirFor\_updatedKnownFor20M\_145K\_dec11\_usingSims20190127\_unnormalizedInputScores/knownFor

\*/

object ClusterEvaluationAdhoc 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

val knownFor = args

.optional("knownForDir").map { location =>

KnownForSources.readKnownFor(location)

}.getOrElse(KnownForSources.knownFor\_20M\_Dec11\_145K)

val minActiveFollowers = args.int("minActiveFollowers", 400)

val topK = args.int("topK")

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

val topUsersExec =

TopUsersSimilarityGraph

.topUsers(

DAL.readMostRecentSnapshot(UsersourceFlatScalaDataset, date).toTypedPipe,

minActiveFollowers,

topK

)

.map(\_.id)

.count("num\_top\_users")

.make(TypedTsv(args("topUsersDir")))

val simsGraphExec = topUsersExec.flatMap { topUsers =>

TopUsersSimilarityGraph.makeGraph(

TopUsersSimilarityGraph.getSubgraphFromUserGroupedInput(

TypedPipe.from(WTFCandidatesSource(args("simsInputDir"))),

topUsers,

args.int("maxSimsNeighborsForEval", 40),

degreeThresholdForStat = 5

),

args("preparedSimsGraph")

)

}

val fullExec = simsGraphExec.flatMap { sims =>

ClusterEvaluation

.clusterLevelEvaluation(sims, knownFor, "eval")

.flatMap { clusterResultsPipe =>

val clusterResults = clusterResultsPipe.forceToDiskExecution

val outputExec = clusterResults.flatMap { pipe =>

pipe

.map {

case (clusterId, (clusterSize, quality)) =>

"%d\t%d\t%.2g\t%.2g\t%.1f\t%.2g\t%.2f\t%.2g\t%.2g"

.format(

clusterId,

clusterSize,

quality.unweightedRecall.getOrElse(0.0),

quality.weightedRecall.getOrElse(0.0),

quality.unweightedRecallDenominator.getOrElse(0.0),

quality.weightedRecallDenominator.getOrElse(0.0),

quality.relativePrecision.getOrElse(0.0),

quality.relativePrecisionNumerator.getOrElse(0.0),

quality.weightAndProductOfNodeScoresCorrelation.getOrElse(0.0)

)

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

}

val printExec = clusterResults.flatMap { pipe =>

ClusterEvaluation.summarizePerClusterResults(pipe).map {

case Some(res) =>

println("Overall results: " + Util.prettyJsonMapper.writeValueAsString(res))

case None =>

println("No overall results!!! Probably cluster results pipe is empty.")

}

}

Execution.zip(outputExec, printExec)

}

}

Util.printCounters(fullExec)

}

}

}

trait ClusterEvaluationBatch extends TwitterScheduledExecutionApp {

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

implicit val dp = DateParser.default

def firstTime: String

def batchDescription: String

def batchIncrement: Duration

private lazy val execArgs = AnalyticsBatchExecutionArgs(

batchDesc = BatchDescription(batchDescription),

firstTime = BatchFirstTime(RichDate(firstTime)),

lastTime = None,

batchIncrement = BatchIncrement(batchIncrement)

)

val emailAddress: String = "no-reply@twitter.com"

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

def knownForModelVersion: String

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

def baselineKnownForModelVersion: String

override def scheduledJob: Execution[Unit] =

AnalyticsBatchExecution(execArgs) { implicit dateRange =>

Execution.withId { implicit uniqueId =>

Execution.withArgs { args =>

val baselineKnownFor =

KnownForSources.fromKeyVal(

DAL

.readMostRecentSnapshot(baselineKnownForDALDataset, dateRange.prepend(Days(7)))

.toTypedPipe,

baselineKnownForModelVersion

)

val knownFor =

KnownForSources.fromKeyVal(

DAL

.readMostRecentSnapshot(knownForDALDataset, dateRange.prepend(Days(7)))

.toTypedPipe,

knownForModelVersion

)

val inputSimsGraph = TypedPipe

.from(FollowingsCosineSimilaritiesManhattanSource())

.map(\_.\_2)

val minActiveFollowers = args.int("minActiveFollowers")

val topK = args.int("topK")

val maxSimsNeighborsForEval =

args.int("maxSimsNeighborsForEval", 40)

val topUsers = TopUsersSimilarityGraph

.topUsers(

DAL

.readMostRecentSnapshot(UsersourceFlatScalaDataset, dateRange)

.toTypedPipe,

minActiveFollowers,

topK

)

.map(\_.id)

.count("num\_top\_users")

TopUsersSimilarityGraph

.getSubgraphFromUserGroupedInput(

fullGraph = inputSimsGraph,

usersToInclude = topUsers,

maxNeighborsPerNode = maxSimsNeighborsForEval,

degreeThresholdForStat = 2

)

.forceToDiskExecution

.flatMap { symmetrizedSims =>

val baselineResultsExec = ClusterEvaluation

.overallEvaluation(symmetrizedSims, baselineKnownFor, "baselineKnownForEval")

val newResultsExec = ClusterEvaluation

.overallEvaluation(symmetrizedSims, knownFor, "newKnownForEval")

val minSizeOfBiggerClusterForComparison = 10

val compareExec = CompareClusters.summarize(

CompareClusters.compare(

KnownForSources.transpose(baselineKnownFor),

KnownForSources.transpose(knownFor),

minSizeOfBiggerCluster = minSizeOfBiggerClusterForComparison

))

Execution

.zip(baselineResultsExec, newResultsExec, compareExec)

.map {

case (oldResults, newResults, compareResults) =>

val emailText =

s"Evaluation Results for baseline knownFor: $baselineKnownForModelVersion \n" +

Util.prettyJsonMapper.writeValueAsString(oldResults) +

"\n\n-------------------\n\n" +

s"Evaluation Results for new knownFor:$knownForModelVersion\n" +

Util.prettyJsonMapper.writeValueAsString(newResults) +

"\n\n-------------------\n\n" +

s"Cosine similarity distribution between $baselineKnownForModelVersion and " +

s"$knownForModelVersion cluster membership vectors for " +

s"clusters with at least $minSizeOfBiggerClusterForComparison members:\n" +

Util.prettyJsonMapper

.writeValueAsString(compareResults)

Util

.sendEmail(

emailText,

s"Evaluation results comparing $knownForModelVersion with baseline $baselineKnownForModelVersion",

emailAddress)

()

}

}

}

}

}

}

/\*\*

\* capesospy-v2 update --build\_locally --start\_cron cluster\_evaluation\_for\_20M\_145k \

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

\*/

object ClusterEvaluationFor20M145K extends ClusterEvaluationBatch {

override val firstTime: String = "2019-06-11"

override val batchIncrement: Duration = Days(7)

override val batchDescription = "com.twitter.simclusters\_v2.scalding.ClusterEvaluationFor20M145K"

override val knownForDALDataset = SimclustersV2KnownFor20M145KUpdatedScalaDataset

override val knownForModelVersion = ModelVersions.Model20M145KUpdated

override val baselineKnownForDALDataset = SimclustersV2KnownFor20M145KDec11ScalaDataset

override val baselineKnownForModelVersion = ModelVersions.Model20M145KDec11

}

/\*\*

\* capesospy-v2 update --build\_locally --start\_cron cluster\_evaluation\_for\_20M\_145k\_2020 \

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

\*/

object ClusterEvaluationFor20M145K2020 extends ClusterEvaluationBatch {

override val firstTime: String = "2021-01-25"

override val batchIncrement: Duration = Days(7)

override val batchDescription =

"com.twitter.simclusters\_v2.scalding.ClusterEvaluationFor20M145K2020"

override val knownForDALDataset = SimclustersV2KnownFor20M145K2020ScalaDataset

override val knownForModelVersion = ModelVersions.Model20M145K2020

override val baselineKnownForDALDataset = SimclustersV2KnownFor20M145KUpdatedScalaDataset

override val baselineKnownForModelVersion = ModelVersions.Model20M145KUpdated

}