package com.twitter.simclusters\_v2.scalding

import com.twitter.algebird.{Monoid, OptionMonoid, Semigroup}

import com.twitter.algebird.mutable.PriorityQueueMonoid

import com.twitter.scalding.Execution

import com.twitter.scalding.typed.TypedPipe

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

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

import com.twitter.simclusters\_v2.thriftscala.{BipartiteClusterQuality, SampledEdge}

import java.util.PriorityQueue

import scala.collection.JavaConverters.\_

object BipartiteClusterEvaluationClasses {

case class Weights(

isFollowEdge: Double,

isFavEdge: Double,

favWtIfFollowEdge: Double,

favWtIfFavEdge: Double)

object WeightsMonoid extends Monoid[Weights] {

override def zero = Weights(0.0, 0.0, 0.0, 0.0)

override def plus(l: Weights, r: Weights): Weights = {

Weights(

l.isFollowEdge + r.isFollowEdge,

l.isFavEdge + r.isFavEdge,

l.favWtIfFollowEdge + r.favWtIfFollowEdge,

l.favWtIfFavEdge + r.favWtIfFavEdge

)

}

}

implicit val wm: Monoid[Weights] = WeightsMonoid

case class SampledEdgeData(

favWtIfFollowEdge: Double,

favWtIfFavEdge: Double,

followScoreToCluster: Double,

favScoreToCluster: Double)

implicit val samplerMonoid: PriorityQueueMonoid[((Long, Long), SampledEdgeData)] =

Util.reservoirSamplerMonoidForPairs[(Long, Long), SampledEdgeData](2000)(Util.edgeOrdering)

implicit val sampledEdgesMonoid: PriorityQueueMonoid[SampledEdge] =

Util.reservoirSamplerMonoid(

10000,

{ sampledEdge: SampledEdge => (sampledEdge.followerId, sampledEdge.followeeId) }

)(Util.edgeOrdering)

case class BipartiteIntermediateResults(

inClusterWeights: Weights,

totalOutgoingVolumes: Weights,

interestedInSize: Int,

edgeSample: PriorityQueue[((Long, Long), SampledEdgeData)]) {

override def toString: String = {

"BCR(%s, %s, %d, %s)".format(

inClusterWeights,

totalOutgoingVolumes,

interestedInSize,

edgeSample.iterator().asScala.toSeq.toString()

)

}

}

object BIRMonoid extends Monoid[BipartiteIntermediateResults] {

override def zero =

BipartiteIntermediateResults(WeightsMonoid.zero, WeightsMonoid.zero, 0, samplerMonoid.zero)

override def plus(

l: BipartiteIntermediateResults,

r: BipartiteIntermediateResults

): BipartiteIntermediateResults = {

BipartiteIntermediateResults(

WeightsMonoid.plus(l.inClusterWeights, r.inClusterWeights),

WeightsMonoid.plus(l.totalOutgoingVolumes, r.totalOutgoingVolumes),

l.interestedInSize + r.interestedInSize,

samplerMonoid.plus(l.edgeSample, r.edgeSample)

)

}

}

implicit val bIRMonoid: Monoid[BipartiteIntermediateResults] = BIRMonoid

def makeThriftSampledEdge(edge: (Long, Long), data: SampledEdgeData): SampledEdge = {

val (followerId, followeeId) = edge

SampledEdge(

followerId = followerId,

followeeId = followeeId,

favWtIfFollowEdge = Some(data.favWtIfFollowEdge),

favWtIfFavEdge = Some(data.favWtIfFavEdge),

followScoreToCluster = Some(data.followScoreToCluster),

favScoreToCluster = Some(data.favScoreToCluster)

)

}

object ClusterQualitySemigroup extends Semigroup[BipartiteClusterQuality] {

val doubleOM: Monoid[Option[Double]] = new OptionMonoid[Double]

val intOM: Monoid[Option[Int]] = new OptionMonoid[Int]

val longOM: Monoid[Option[Long]] = new OptionMonoid[Long]

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

BipartiteClusterQuality(

inClusterFollowEdges = doubleOM.plus(l.inClusterFollowEdges, r.inClusterFollowEdges),

inClusterFavEdges = doubleOM.plus(l.inClusterFavEdges, r.inClusterFavEdges),

favWtSumOfInClusterFollowEdges = doubleOM

.plus(l.favWtSumOfInClusterFollowEdges, r.favWtSumOfInClusterFollowEdges),

favWtSumOfInClusterFavEdges = doubleOM

.plus(l.favWtSumOfInClusterFavEdges, r.favWtSumOfInClusterFavEdges),

outgoingFollowEdges = doubleOM.plus(l.outgoingFollowEdges, r.outgoingFollowEdges),

outgoingFavEdges = doubleOM.plus(l.outgoingFavEdges, r.outgoingFavEdges),

favWtSumOfOutgoingFollowEdges = doubleOM

.plus(l.favWtSumOfOutgoingFollowEdges, r.favWtSumOfOutgoingFollowEdges),

favWtSumOfOutgoingFavEdges = doubleOM

.plus(l.favWtSumOfOutgoingFavEdges, r.favWtSumOfOutgoingFavEdges),

incomingFollowEdges = doubleOM.plus(l.incomingFollowEdges, r.incomingFollowEdges),

incomingFavEdges = doubleOM.plus(l.incomingFavEdges, r.incomingFavEdges),

favWtSumOfIncomingFollowEdges = doubleOM

.plus(l.favWtSumOfIncomingFollowEdges, r.favWtSumOfIncomingFollowEdges),

favWtSumOfIncomingFavEdges = doubleOM

.plus(l.favWtSumOfIncomingFavEdges, r.favWtSumOfIncomingFavEdges),

interestedInSize = None,

sampledEdges = Some(

sampledEdgesMonoid

.plus(

sampledEdgesMonoid.build(l.sampledEdges.getOrElse(Nil)),

sampledEdgesMonoid.build(r.sampledEdges.getOrElse(Nil))

)

.iterator()

.asScala

.toSeq),

knownForSize = intOM.plus(l.knownForSize, r.knownForSize),

correlationOfFavWtIfFollowWithPredictedFollow = None,

correlationOfFavWtIfFavWithPredictedFav = None,

relativePrecisionUsingFavWtIfFav = None,

averagePrecisionOfWholeGraphUsingFavWtIfFav = l.averagePrecisionOfWholeGraphUsingFavWtIfFav

)

}

implicit val bcqSemigroup: Semigroup[BipartiteClusterQuality] =

ClusterQualitySemigroup

case class PrintableBipartiteQuality(

incomingFollowUnweightedRecall: String,

incomingFavUnweightedRecall: String,

incomingFollowWeightedRecall: String,

incomingFavWeightedRecall: String,

outgoingFollowUnweightedRecall: String,

outgoingFavUnweightedRecall: String,

outgoingFollowWeightedRecall: String,

outgoingFavWeightedRecall: String,

incomingFollowEdges: String,

incomingFavEdges: String,

favWtSumOfIncomingFollowEdges: String,

favWtSumOfIncomingFavEdges: String,

outgoingFollowEdges: String,

outgoingFavEdges: String,

favWtSumOfOutgoingFollowEdges: String,

favWtSumOfOutgoingFavEdges: String,

correlationOfFavWtIfFollow: String,

correlationOfFavWtIfFav: String,

relativePrecisionUsingFavWt: String,

averagePrecisionOfWholeGraphUsingFavWt: String,

interestedInSize: String,

knownForSize: String)

def printableBipartiteQuality(in: BipartiteClusterQuality): PrintableBipartiteQuality = {

def getRatio(numOpt: Option[Double], denOpt: Option[Double]): String = {

val r = if (denOpt.exists(\_ > 0)) {

numOpt.getOrElse(0.0) / denOpt.get

} else 0.0

"%.3f".format(r)

}

val formatter = new java.text.DecimalFormat("###,###.#")

def denString(denOpt: Option[Double]): String =

formatter.format(denOpt.getOrElse(0.0))

val correlationOfFavWtIfFollow =

in.correlationOfFavWtIfFollowWithPredictedFollow match {

case None =>

in.sampledEdges.map { samples =>

val pairs = samples.map { s =>

(s.predictedFollowScore.getOrElse(0.0), s.favWtIfFollowEdge.getOrElse(0.0))

}

Util.computeCorrelation(pairs.iterator)

}

case x @ \_ => x

}

val correlationOfFavWtIfFav =

in.correlationOfFavWtIfFavWithPredictedFav match {

case None =>

in.sampledEdges.map { samples =>

val pairs = samples.map { s =>

(s.predictedFavScore.getOrElse(0.0), s.favWtIfFavEdge.getOrElse(0.0))

}

Util.computeCorrelation(pairs.iterator)

}

case x @ \_ => x

}

PrintableBipartiteQuality(

incomingFollowUnweightedRecall = getRatio(in.inClusterFollowEdges, in.incomingFollowEdges),

incomingFavUnweightedRecall = getRatio(in.inClusterFavEdges, in.incomingFavEdges),

incomingFollowWeightedRecall =

getRatio(in.favWtSumOfInClusterFollowEdges, in.favWtSumOfIncomingFollowEdges),

incomingFavWeightedRecall =

getRatio(in.favWtSumOfInClusterFavEdges, in.favWtSumOfIncomingFavEdges),

outgoingFollowUnweightedRecall = getRatio(in.inClusterFollowEdges, in.outgoingFollowEdges),

outgoingFavUnweightedRecall = getRatio(in.inClusterFavEdges, in.outgoingFavEdges),

outgoingFollowWeightedRecall =

getRatio(in.favWtSumOfInClusterFollowEdges, in.favWtSumOfOutgoingFollowEdges),

outgoingFavWeightedRecall =

getRatio(in.favWtSumOfInClusterFavEdges, in.favWtSumOfOutgoingFavEdges),

incomingFollowEdges = denString(in.incomingFollowEdges),

incomingFavEdges = denString(in.incomingFavEdges),

favWtSumOfIncomingFollowEdges = denString(in.favWtSumOfIncomingFollowEdges),

favWtSumOfIncomingFavEdges = denString(in.favWtSumOfIncomingFavEdges),

outgoingFollowEdges = denString(in.outgoingFollowEdges),

outgoingFavEdges = denString(in.outgoingFavEdges),

favWtSumOfOutgoingFollowEdges = denString(in.favWtSumOfOutgoingFollowEdges),

favWtSumOfOutgoingFavEdges = denString(in.favWtSumOfOutgoingFavEdges),

correlationOfFavWtIfFollow = "%.3f"

.format(correlationOfFavWtIfFollow.getOrElse(0.0)),

correlationOfFavWtIfFav = "%.3f"

.format(correlationOfFavWtIfFav.getOrElse(0.0)),

relativePrecisionUsingFavWt =

"%.2g".format(in.relativePrecisionUsingFavWtIfFav.getOrElse(0.0)),

averagePrecisionOfWholeGraphUsingFavWt =

"%.2g".format(in.averagePrecisionOfWholeGraphUsingFavWtIfFav.getOrElse(0.0)),

interestedInSize = in.interestedInSize.getOrElse(0).toString,

knownForSize = in.knownForSize.getOrElse(0).toString

)

}

case class ClusterResultsSummary(

numClustersWithZeroInterestedIn: Int,

numClustersWithZeroFollowWtRecall: Int,

numClustersWithZeroFavWtRecall: Int,

numClustersWithZeroFollowAndFavWtRecall: Int,

interestedInSizeDist: Distribution,

outgoingFollowWtRecallDist: Distribution,

outgoingFavWtRecallDist: Distribution,

incomingFollowWtRecallDist: Distribution,

incomingFavWtRecallDist: Distribution,

followCorrelationDist: Distribution,

favCorrelationDist: Distribution,

relativePrecisionDist: Distribution)

def getClusterResultsSummary(

perClusterResults: TypedPipe[BipartiteClusterQuality]

): Execution[Option[ClusterResultsSummary]] = {

perClusterResults

.map { clusterQuality =>

val printableQuality = printableBipartiteQuality(clusterQuality)

val isFollowRecallZero =

if (!clusterQuality.favWtSumOfInClusterFollowEdges

.exists(\_ > 0)) 1

else 0

val isFavRecallZero =

if (!clusterQuality.favWtSumOfInClusterFavEdges.exists(\_ > 0)) 1

else 0

(

if (!clusterQuality.interestedInSize.exists(\_ > 0)) 1 else 0,

isFollowRecallZero,

isFavRecallZero,

isFavRecallZero \* isFollowRecallZero,

clusterQuality.interestedInSize.toList.map(\_.toDouble),

List(printableQuality.outgoingFollowWeightedRecall.toDouble),

List(printableQuality.outgoingFavWeightedRecall.toDouble),

List(printableQuality.incomingFollowWeightedRecall.toDouble),

List(printableQuality.incomingFavWeightedRecall.toDouble),

List(printableQuality.correlationOfFavWtIfFollow.toDouble),

List(printableQuality.correlationOfFavWtIfFav.toDouble),

List(printableQuality.relativePrecisionUsingFavWt.toDouble)

)

}

.sum

.toOptionExecution

.map { opt =>

opt.map {

case (

zeroInterestedIn,

zeroFollowRecall,

zeroFavRecall,

zeroFollowAndFavRecall,

interestedInSizeList,

outgoingFollowWtRecallList,

outgoingFavWtRecallList,

incomingFollowWtRecallList,

incomingFavWtRecallList,

followCorrelationList,

favCorrelationList,

relativePrecisionList

) =>

ClusterResultsSummary(

numClustersWithZeroInterestedIn = zeroInterestedIn,

numClustersWithZeroFollowWtRecall = zeroFollowRecall,

numClustersWithZeroFavWtRecall = zeroFavRecall,

numClustersWithZeroFollowAndFavWtRecall = zeroFollowAndFavRecall,

interestedInSizeDist = Util.distributionFromArray(interestedInSizeList.toArray),

outgoingFollowWtRecallDist = Util

.distributionFromArray(outgoingFollowWtRecallList.toArray),

outgoingFavWtRecallDist = Util.distributionFromArray(outgoingFavWtRecallList.toArray),

incomingFollowWtRecallDist = Util

.distributionFromArray(incomingFollowWtRecallList.toArray),

incomingFavWtRecallDist = Util.distributionFromArray(incomingFavWtRecallList.toArray),

followCorrelationDist = Util.distributionFromArray(followCorrelationList.toArray),

favCorrelationDist = Util.distributionFromArray(favCorrelationList.toArray),

relativePrecisionDist = Util.distributionFromArray(relativePrecisionList.toArray)

)

}

}

}

}