package com.twitter.simclusters\_v2.scalding

import com.twitter.algebird.OptionMonoid

import com.twitter.algebird.QTree

import com.twitter.algebird.QTreeSemigroup

import com.twitter.algebird.Semigroup

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

import com.twitter.dal.client.dataset.SnapshotDALDataset

import com.twitter.hermit.candidate.thriftscala.Candidates

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

import com.twitter.pluck.source.cassowary.SimsCandidatesSource

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.\_

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.Util

import com.twitter.simclusters\_v2.scalding.embedding.common.ExternalDataSources

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

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

import com.twitter.usersource.snapshot.flat.thriftscala.FlatUser

object ClusterDetailsJob {

case class Scores(followScore: Double, favScore: Double, logFavScore: Double)

case class IntermediateDetails(

numUsersWithAnyNonZeroScore: Int,

numUsersWithNonZeroFollowScore: Int,

numUsersWithNonZeroFavScore: Int,

favQTree: Option[QTree[Double]],

followQTree: Option[QTree[Double]],

logFavQTree: Option[QTree[Double]],

sumOfSquares: Scores,

sum: Scores,

min: Scores,

max: Scores)

case class InfoFromUserSource(

fractionMarkedNSFWUser: Double,

languageToFractionDeviceLanguage: Map[String, Double],

countryCodeToFractionKnownForWithCountryCode: Map[String, Double],

languageToFractionInferredLanguage: Map[String, Double])

def positiveMin(a: Double, b: Double) = {

if (math.min(a, b) == 0.0) math.max(a, b) else math.min(a, b)

}

case class ClusterDetailsSemigroup(implicit qtreeSemigroup: Semigroup[QTree[Double]])

extends Semigroup[IntermediateDetails] {

val optionMonoid: OptionMonoid[QTree[Double]] = new OptionMonoid[QTree[Double]]()

override def plus(

left: IntermediateDetails,

right: IntermediateDetails

): IntermediateDetails = {

IntermediateDetails(

left.numUsersWithAnyNonZeroScore + right.numUsersWithAnyNonZeroScore,

left.numUsersWithNonZeroFollowScore + right.numUsersWithNonZeroFollowScore,

left.numUsersWithNonZeroFavScore + right.numUsersWithNonZeroFavScore,

optionMonoid.plus(left.favQTree, right.favQTree),

optionMonoid.plus(left.followQTree, right.followQTree),

optionMonoid.plus(left.logFavQTree, right.logFavQTree),

Scores(

left.sumOfSquares.followScore + right.sumOfSquares.followScore,

left.sumOfSquares.favScore + right.sumOfSquares.favScore,

left.sumOfSquares.logFavScore + right.sumOfSquares.logFavScore

),

Scores(

left.sum.followScore + right.sum.followScore,

left.sum.favScore + right.sum.favScore,

left.sum.logFavScore + right.sum.logFavScore

),

Scores(

positiveMin(left.min.followScore, right.min.followScore),

positiveMin(left.min.favScore, right.min.favScore),

positiveMin(left.min.logFavScore, right.min.logFavScore)

),

Scores(

math.max(left.max.followScore, right.max.followScore),

math.max(left.max.favScore, right.max.favScore),

math.max(left.max.logFavScore, right.max.logFavScore)

)

)

}

}

def intermediateDetailsPipe(

input: TypedPipe[(Long, ClustersUserIsInterestedIn)],

qtreeSemigroupKParameter: Int

): TypedPipe[(Int, IntermediateDetails)] = {

implicit val qtSg: Semigroup[QTree[Double]] =

new QTreeSemigroup[Double](qtreeSemigroupKParameter)

implicit val cdSg: Semigroup[IntermediateDetails] = ClusterDetailsSemigroup()

input

.flatMap {

case (userId, clusterScoresStruct) =>

val clusterScoresArray = clusterScoresStruct.clusterIdToScores.toArray

clusterScoresArray.map {

case (clusterId, scoresStruct) =>

val followScore = scoresStruct.followScore.getOrElse(0.0)

val favScore = scoresStruct.favScore.getOrElse(0.0)

val logFavScore = scoresStruct.logFavScore.getOrElse(0.0)

(

clusterId,

IntermediateDetails(

numUsersWithAnyNonZeroScore = 1,

numUsersWithNonZeroFollowScore = if (followScore > 0) 1 else 0,

numUsersWithNonZeroFavScore = if (favScore > 0) 1 else 0,

favQTree = if (favScore > 0) Some(QTree(favScore)) else None,

followQTree = if (followScore > 0) Some(QTree(followScore)) else None,

logFavQTree = if (logFavScore > 0) Some(QTree(logFavScore)) else None,

sumOfSquares = Scores(

followScore \* followScore,

favScore \* favScore,

logFavScore \* logFavScore),

sum = Scores(followScore, favScore, logFavScore),

min = Scores(followScore, favScore, logFavScore),

max = Scores(followScore, favScore, logFavScore)

)

)

}

}

.sumByKey

// Uncomment for adhoc job

//.withReducers(100)

.toTypedPipe

}

private def safeGetDoubleOpt(x: Option[Double]): Double = {

x.map { y => if (y.isNaN) 0 else y }.getOrElse(0)

}

private def getSimilaritiesForAllPairs(

input: TypedPipe[(Long, ClustersUserIsInterestedIn)]

)(

implicit uniqueID: UniqueID

): TypedPipe[((Int, Int), Scores)] = {

val allClusterPairsBeforeSumByKey = Stat("all\_cluster\_pairs\_before\_sum\_by\_key")

val clusterPairsWithin10Ratio = Stat("cluster\_pairs\_within\_10\_ratio")

val clusterPairsBeforeTopK = Stat("cluster\_pairs\_before\_thresholding")

input

.flatMap {

case (userId, clusterScoresStruct) =>

val clusterScoresArray = clusterScoresStruct.clusterIdToScores.toArray

(0 until clusterScoresArray.length).flatMap { i =>

(0 until clusterScoresArray.length).map { j =>

val (clusterI, scoresI) = clusterScoresArray(i)

val (clusterJ, scoresJ) = clusterScoresArray(j)

val ratioOfSizes =

scoresI.numUsersInterestedInThisClusterUpperBound.getOrElse(1).toDouble /

scoresJ.numUsersInterestedInThisClusterUpperBound.getOrElse(1).toDouble

allClusterPairsBeforeSumByKey.inc()

if (ratioOfSizes > 0.1 && ratioOfSizes < 10) {

clusterPairsWithin10Ratio.inc()

}

val followI = safeGetDoubleOpt(scoresI.followScoreClusterNormalizedOnly)

val followJ = safeGetDoubleOpt(scoresJ.followScoreClusterNormalizedOnly)

val follow = followI \* followJ

val favI = safeGetDoubleOpt(scoresI.favScoreClusterNormalizedOnly)

val favJ = safeGetDoubleOpt(scoresJ.favScoreClusterNormalizedOnly)

val fav = favI \* favJ

val logFavI = safeGetDoubleOpt(scoresI.logFavScoreClusterNormalizedOnly)

val logFavJ = safeGetDoubleOpt(scoresJ.logFavScoreClusterNormalizedOnly)

val logFav = logFavI \* logFavJ

((clusterI, clusterJ), (follow, fav, logFav))

}

}

}

.sumByKey

// Uncomment for adhoc job

//.withReducers(600)

.map {

case (key, (follow, fav, logFav)) =>

clusterPairsBeforeTopK.inc()

(key, Scores(follow, fav, logFav))

}

}

private def keepTopNeighbors(

allPairs: TypedPipe[((Int, Int), Scores)],

cosineThreshold: Double

)(

implicit uniqueID: UniqueID

): TypedPipe[(Int, List[ClusterNeighbor])] = {

val clusterPairsMoreThanThreshold = Stat("cluster\_pairs\_cosine\_gt\_" + cosineThreshold)

val clusterPairsAfterTopK = Stat("cluster\_pairs\_after\_topk")

val clustersWithFewNeighbors = Stat(s"clusters\_with\_fewer\_than\_100\_neighbors")

val clustersWithManyNeighbors = Stat(s"clusters\_with\_more\_than\_100\_neighbors")

allPairs

.flatMap {

case ((cI, cJ), Scores(followScore, favScore, logFavScore)) =>

if (followScore > cosineThreshold || logFavScore > cosineThreshold || favScore > cosineThreshold) {

clusterPairsMoreThanThreshold.inc()

Some((cI, ClusterNeighbor(cJ, Some(followScore), Some(favScore), Some(logFavScore))))

} else None

}

.group

.toList

// Uncomment for adhoc job

//.withReducers(40)

.map {

case (key, seq) =>

val finalSize = seq.size

clusterPairsAfterTopK.incBy(finalSize)

if (finalSize < 100) {

clustersWithFewNeighbors.inc()

} else {

clustersWithManyNeighbors.inc()

}

(

key,

seq.sortBy {

case cn: ClusterNeighbor =>

-(cn.followCosineSimilarity.getOrElse(0.0) + cn.logFavCosineSimilarity.getOrElse(

0.0)) / 2

})

}

}

def getTopSimilarClustersWithCosine(

input: TypedPipe[(Long, ClustersUserIsInterestedIn)],

cosineThreshold: Double

)(

implicit uniqueID: UniqueID

): TypedPipe[(Int, List[ClusterNeighbor])] = {

keepTopNeighbors(getSimilaritiesForAllPairs(input), cosineThreshold)

}

def getDistributionDetails(

qtree: QTree[Double],

sum: Double,

sumOfSquares: Double,

min: Double,

max: Double,

fullSize: Int

): DistributionDetails = {

val mean = sum / fullSize

// note that the below is the naive calculation, and not the sample standard dev formula

// that divides by n-1. I don't think it makes a difference at our scale whether we use n or n-1

// and I'd rather use the simpler one.

val stdDev = math.sqrt(sumOfSquares / fullSize - mean \* mean)

def getQB(percentile: Double): QuantileBounds = {

val (lb, ub) = qtree.quantileBounds(percentile)

QuantileBounds(lb, ub)

}

DistributionDetails(

mean = mean,

standardDeviation = Some(stdDev),

min = Some(min),

p25 = Some(getQB(0.25)),

p50 = Some(getQB(0.5)),

p75 = Some(getQB(0.75)),

p95 = Some(getQB(0.95)),

max = Some(max)

)

}

def keepCorrectModel(

input: TypedPipe[(Long, ClustersUserIsInterestedIn)],

modelVersionToKeep: String

)(

implicit uniqId: UniqueID

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

val allRecords = Stat("all\_input\_records")

val withCorrectVersion = Stat("with\_correct\_version")

input.filter {

case (\_, clusterScoresStruct) =>

// allRecords.inc()

val result = clusterScoresStruct.knownForModelVersion == modelVersionToKeep

// if (result) withCorrectVersion.inc()

result

}

}

def getInfoFromUserSource(

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

usersource: TypedPipe[FlatUser],

inferredLanguages: TypedPipe[(Long, Seq[(String, Double)])]

)(

implicit uniqId: UniqueID

): TypedPipe[(Int, InfoFromUserSource)] = {

val knownForUsers = knownFor.flatMap {

case (clusterId, userScoreList) =>

userScoreList.map {

case (userId, \_) =>

(userId, clusterId)

}

}

usersource

.collect {

case fuser: FlatUser if fuser.id.isDefined =>

(

fuser.id.get,

(

fuser.accountCountryCode.getOrElse(""),

fuser.language.getOrElse(""),

fuser.nsfwUser.getOrElse(false)

))

}

.join(knownForUsers)

.leftJoin(inferredLanguages)

.map {

case (\_, (((countryCode, language, nsfw), clusterId), inferredLangsOpt)) =>

val nsfwInt = if (nsfw) 1 else 0

(

clusterId,

(

1,

nsfwInt,

Map(language -> 1),

Map(countryCode -> 1),

inferredLangsOpt.getOrElse(Seq(("", 1.0))).toMap

)

)

}

.sumByKey

.mapValues {

case (

denominator,

nsfwNumerator,

languageNumeratorsMap,

countryNumeratorsMap,

inferredLangsNumeratorsMap) =>

InfoFromUserSource(

nsfwNumerator \* 1.0 / denominator,

languageNumeratorsMap.mapValues { x => x \* 1.0 / denominator },

countryNumeratorsMap.mapValues { x => x \* 1.0 / denominator },

inferredLangsNumeratorsMap.mapValues { x => x \* 1.0 / denominator }

)

}

}

/\*\*

\* Run the cluster details job and return the details for each cluster

\* @param input interestedIn data

\* @param qtreeSemigroupKParameter parameter for calculating percentiles using qtree monoid (set to a small number, usually < 7)

\* @param modelVersionToKeep which modelVersion to use from interestedIn dataset

\* @param knownFor clusterId -> users known for this cluster and their scores

\* @param knownForTranspose userId -> clusters this user is known for and their scores

\* @param usersource -> user source

\* @param simsGraph -> sims graph in the form of userId -> adjacency list

\* @param cosineThreshold -> cosine threshold to include a cluster in the list of similar clusters for a given cluster

\* @param uniqId

\* @return pipe with (modelVersion, clusterId) as the key and ClusterDetails struct as the value.

\*/

def run(

input: TypedPipe[(Long, ClustersUserIsInterestedIn)],

qtreeSemigroupKParameter: Int,

modelVersionToKeep: String,

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

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

usersource: Option[TypedPipe[FlatUser]],

inferredLanguageSource: Option[TypedPipe[(Long, Seq[(String, Double)])]],

simsGraph: Option[TypedPipe[(Long, Map[Long, Float])]],

cosineThreshold: Double

)(

implicit uniqId: UniqueID

): Execution[TypedPipe[((String, Int), ClusterDetails)]] = {

val topSimilarClusters = getTopSimilarClustersWithCosine(input, cosineThreshold)

val infoFromUserSource: TypedPipe[(Int, InfoFromUserSource)] = (for {

us <- usersource

inferredLanguages <- inferredLanguageSource

} yield getInfoFromUserSource(knownFor, us, inferredLanguages)).getOrElse(TypedPipe.empty)

val clusterEvaluationExec = simsGraph match {

case Some(sg) =>

ClusterEvaluation.clusterLevelEvaluation(sg, knownForTranspose, "eval")

case None =>

val dummyPipe: TypedPipe[(Int, (Int, ClusterQuality))] = TypedPipe.empty

Execution.from(dummyPipe)

}

clusterEvaluationExec

.map { clusterIdToSizesAndQualities =>

val clusterQualities: TypedPipe[(Int, ClusterQuality)] =

clusterIdToSizesAndQualities.mapValues(\_.\_2)

intermediateDetailsPipe(

keepCorrectModel(input, modelVersionToKeep),

qtreeSemigroupKParameter)

.leftJoin(topSimilarClusters)

.leftJoin(infoFromUserSource)

.leftJoin(clusterQualities)

.join(knownFor)

.map {

case (

clusterId,

(

(

((intermediateDetails, topSimilarNeighborsOpt), userSourceInfoOpt),

qualityOpt),

knownForUsers)

) =>

val knownForSorted = knownForUsers.sortBy(-\_.\_2).map {

case (userId, score) =>

UserWithScore(userId, score)

}

(modelVersionToKeep, clusterId) ->

ClusterDetails(

numUsersWithAnyNonZeroScore = intermediateDetails.numUsersWithAnyNonZeroScore,

numUsersWithNonZeroFavScore = intermediateDetails.numUsersWithNonZeroFavScore,

numUsersWithNonZeroFollowScore =

intermediateDetails.numUsersWithNonZeroFollowScore,

favScoreDistributionDetails = intermediateDetails.favQTree.map { qt =>

getDistributionDetails(

qtree = qt,

sum = intermediateDetails.sum.favScore,

sumOfSquares = intermediateDetails.sumOfSquares.favScore,

min = intermediateDetails.min.favScore,

max = intermediateDetails.max.favScore,

fullSize = intermediateDetails.numUsersWithNonZeroFavScore

)

},

followScoreDistributionDetails = intermediateDetails.followQTree.map { qt =>

getDistributionDetails(

qtree = qt,

sum = intermediateDetails.sum.followScore,

sumOfSquares = intermediateDetails.sumOfSquares.followScore,

min = intermediateDetails.min.followScore,

max = intermediateDetails.max.followScore,

fullSize = intermediateDetails.numUsersWithNonZeroFollowScore

)

},

logFavScoreDistributionDetails = intermediateDetails.logFavQTree.map { qt =>

getDistributionDetails(

qtree = qt,

sum = intermediateDetails.sum.logFavScore,

sumOfSquares = intermediateDetails.sumOfSquares.logFavScore,

min = intermediateDetails.min.logFavScore,

max = intermediateDetails.max.logFavScore,

// note: user has non-zero fav score iff a user has non-zero log-fav score

fullSize = intermediateDetails.numUsersWithNonZeroFavScore

)

},

knownForUsersAndScores = Some(knownForSorted),

neighborClusters = topSimilarNeighborsOpt,

fractionKnownForMarkedNSFWUser = userSourceInfoOpt.map(\_.fractionMarkedNSFWUser),

languageToFractionDeviceLanguage =

userSourceInfoOpt.map(\_.languageToFractionDeviceLanguage),

countryCodeToFractionKnownForWithCountryCode =

userSourceInfoOpt.map(\_.countryCodeToFractionKnownForWithCountryCode),

qualityMeasuredOnSimsGraph = qualityOpt,

languageToFractionInferredLanguage =

userSourceInfoOpt.map(\_.languageToFractionInferredLanguage),

)

}

}

}

def getTruncatedSims(

sims: TypedPipe[Candidates],

maxNeighbors: Int

): TypedPipe[(Long, Map[Long, Float])] = {

sims.map { cands =>

(

cands.userId,

// These candidates are already sorted, but leaving it in just in case the behavior changes upstream

cands.candidates

.map { c => (c.userId, c.score.toFloat) }.sortBy(-\_.\_2).take(maxNeighbors).toMap

)

}

}

}

/\*\*

scalding remote run --main-class com.twitter.simclusters\_v2.scalding.ClusterDetailsAdhoc \

--target src/scala/com/twitter/simclusters\_v2/scalding:cluster\_details-adhoc \

--hadoop-properties "scalding.with.reducers.set.explicitly=true mapreduce.job.reduces=4000" \

--user recos-platform -- \

--date 2020-06-25 \

--dateForUserSource 2020-06-25 \

--includeUserSource \

--outputDir /user/recos-platform/adhoc/your\_ldap/cluster\_details\_inferred\_lang

\*/

object ClusterDetailsAdhoc 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 date = DateRange.parse(args("dateForUserSource"))

val (knownFor, knownForTranspose) =

args

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

(

KnownForSources.transpose(KnownForSources.readKnownFor(location)),

KnownForSources.readKnownFor(location)

)

}.getOrElse(

(

KnownForSources.clusterToKnownFor\_20M\_145K\_updated,

KnownForSources.knownFor\_20M\_145K\_updated

)

)

val interestedIn = args

.optional("inputDir").map { interestedInInputDir =>

TypedPipe.from(AdhocKeyValSources.interestedInSource(interestedInInputDir))

}.getOrElse(

DAL

.readMostRecentSnapshotNoOlderThan(

SimclustersV2InterestedIn20M145KUpdatedScalaDataset,

Days(14))

.withRemoteReadPolicy(ExplicitLocation(ProcAtla))

.toTypedPipe

.map {

case KeyVal(userId, clustersUserIsInterestedIn) =>

(userId, clustersUserIsInterestedIn)

}

)

val userSourceOpt = if (args.boolean("includeUserSource")) {

Some(DAL.readMostRecentSnapshot(UsersourceFlatScalaDataset, date).toTypedPipe)

} else None

val inferredLanguagesOpt = if (args.boolean("includeUserSource")) {

Some(ExternalDataSources.inferredUserProducedLanguageSource)

} else None

val simsGraphOpt = args.optional("simsForEvalInputDir").map { sgDir =>

ClusterDetailsJob.getTruncatedSims(

TypedPipe.from(WTFCandidatesSource(sgDir)),

args.int("maxSimsNeighborsForEval", 20)

)

}

Util.printCounters(

ClusterDetailsJob

.run(

interestedIn,

args.int("qtreeSemigroupKParameter", 3),

args.getOrElse("modelVersion", "20M\_145K\_updated"),

knownFor,

knownForTranspose,

userSourceOpt,

inferredLanguagesOpt,

simsGraphOpt,

cosineThreshold = args.double("cosineThreshold", 0.01)

).flatMap(

\_.writeExecution(AdhocKeyValSources.clusterDetailsSource(args("outputDir"))))

)

}

}

}

trait ClusterDetailsBatchTrait extends TwitterScheduledExecutionApp {

implicit val tz = DateOps.UTC

implicit val parser = DateParser.default

def firstTime: String

def batchIncrement: Duration

def manhattanOutputPath: String

def clusterDetailsLiteOutputPath: String

def modelVersion: String

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

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

def outputDataset: KeyValDALDataset[KeyVal[(String, Int), ClusterDetails]]

def clusterDetailsLiteOutputDataset: SnapshotDALDataset[ClusterDetailsLite]

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 qtreeSemigroupKParameter = args.int("qtreeSemigroupKParameter", 5)

val maxSimsNeighborsForEval = args.int("maxSimsNeighborsForEval", 20)

val knownForTranspose =

KnownForSources.fromKeyVal(

DAL.readMostRecentSnapshot(knownForDataset, dateRange.extend(Days(7))).toTypedPipe,

modelVersion)

val knownFor = KnownForSources.transpose(knownForTranspose)

val cosineThreshold = args.double("cosineThreshold", 0.01)

val interestedIn =

DAL

.readMostRecentSnapshot(interestedInDataset, dateRange.extend(Days(7)))

.toTypedPipe

.map {

case KeyVal(userId, clustersUserIsInterestedIn) =>

(userId, clustersUserIsInterestedIn)

}

val sims = if (modelVersion == ModelVersions.Model20M145K2020) {

// The model version 20m\_145k\_2020 uses approximate\_cosine\_follow as the input sims graph

// to cluster users. The same graph is used to evaluate the clusters

TypedPipe

.from(FollowingsCosineSimilaritiesManhattanSource())

.map(\_.\_2)

} else {

TypedPipe.from(

SimsCandidatesSource()(

dateRange = dateRange,

suffixPath = "/classified\_candidates\_rollup"

))

}

val resultExec = ClusterDetailsJob

.run(

interestedIn,

qtreeSemigroupKParameter,

modelVersion,

knownFor,

knownForTranspose,

Some(DAL.readMostRecentSnapshot(UsersourceFlatScalaDataset, dateRange).toTypedPipe),

Some(ExternalDataSources.inferredUserProducedLanguageSource),

Some(

ClusterDetailsJob.getTruncatedSims(sims, maxNeighbors = maxSimsNeighborsForEval)),

cosineThreshold

).flatMap { resultUnmapped =>

val clusterDetailsExec = resultUnmapped

.map {

case (clusterKey, details) =>

KeyVal(clusterKey, details)

}.writeDALVersionedKeyValExecution(

outputDataset,

D.Suffix(manhattanOutputPath)

)

val clusterDetailsLiteExec =

resultUnmapped

.map {

case ((\_, clusterId), details)

if modelVersion == ModelVersions.Model20M145KDec11 =>

ClusterDetailsLite(

FullClusterId(ModelVersion.Model20m145kDec11, clusterId),

details.numUsersWithAnyNonZeroScore,

details.numUsersWithNonZeroFollowScore,

details.numUsersWithNonZeroFavScore,

details.knownForUsersAndScores.getOrElse(Nil)

)

case ((\_, clusterId), details)

if modelVersion == ModelVersions.Model20M145KUpdated =>

ClusterDetailsLite(

FullClusterId(ModelVersion.Model20m145kUpdated, clusterId),

details.numUsersWithAnyNonZeroScore,

details.numUsersWithNonZeroFollowScore,

details.numUsersWithNonZeroFavScore,

details.knownForUsersAndScores.getOrElse(Nil)

)

case ((\_, clusterId), details)

if modelVersion == ModelVersions.Model20M145K2020 =>

ClusterDetailsLite(

FullClusterId(ModelVersion.Model20m145k2020, clusterId),

details.numUsersWithAnyNonZeroScore,

details.numUsersWithNonZeroFollowScore,

details.numUsersWithNonZeroFavScore,

details.knownForUsersAndScores.getOrElse(Nil)

)

}.writeDALSnapshotExecution(

clusterDetailsLiteOutputDataset,

D.Daily,

D.Suffix(clusterDetailsLiteOutputPath),

D.EBLzo(),

dateRange.end)

Execution.zip(clusterDetailsExec, clusterDetailsLiteExec)

}

Util.printCounters(resultExec)

}

}

}

}

object ClusterDetailsBatch extends ClusterDetailsBatchTrait {

override val firstTime: String = "2018-07-28"

override val batchIncrement: Duration = Days(7)

override val manhattanOutputPath: String =

"/user/cassowary/manhattan\_sequence\_files/simclusters\_v2\_cluster\_details"

override val clusterDetailsLiteOutputPath: String =

"/user/cassowary/processed/simclusters\_v2\_cluster\_details\_lite"

override val modelVersion: String = ModelVersions.Model20M145KDec11

override val knownForDataset = SimclustersV2KnownFor20M145KDec11ScalaDataset

override val interestedInDataset = SimclustersV2InterestedInScalaDataset

override val outputDataset = SimclustersV2ClusterDetailsScalaDataset

override val clusterDetailsLiteOutputDataset =

SimclustersV2ClusterDetailsLiteScalaDataset

}

object ClusterDetails20M145KUpdated extends ClusterDetailsBatchTrait {

override val firstTime: String = "2019-06-16"

override val batchIncrement: Duration = Days(7)

override val manhattanOutputPath: String =

"/user/cassowary/manhattan\_sequence\_files/simclusters\_v2\_cluster\_details\_20m\_145k\_updated"

override val clusterDetailsLiteOutputPath: String =

"/user/cassowary/processed/simclusters\_v2\_cluster\_details\_lite\_20m\_145k\_updated"

override val modelVersion: String = ModelVersions.Model20M145KUpdated

override val knownForDataset = SimclustersV2KnownFor20M145KUpdatedScalaDataset

override val interestedInDataset = SimclustersV2InterestedIn20M145KUpdatedScalaDataset

override val outputDataset = SimclustersV2ClusterDetails20M145KUpdatedScalaDataset

override val clusterDetailsLiteOutputDataset =

SimclustersV2ClusterDetailsLite20M145KUpdatedScalaDataset

}

/\*\*

\* capesospy-v2 update --build\_locally --start\_cron cluster\_details\_20m\_145k\_2020 \

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

\*/

object ClusterDetails20M145K2020 extends ClusterDetailsBatchTrait {

override val firstTime: String = "2020-10-15"

override val batchIncrement: Duration = Days(7)

override val manhattanOutputPath: String =

"/user/cassowary/manhattan\_sequence\_files/simclusters\_v2\_cluster\_details\_20m\_145k\_2020"

override val clusterDetailsLiteOutputPath: String =

"/user/cassowary/processed/simclusters\_v2\_cluster\_details\_lite\_20m\_145k\_2020"

override val modelVersion: String = ModelVersions.Model20M145K2020

override val knownForDataset = SimclustersV2KnownFor20M145K2020ScalaDataset

override val interestedInDataset = SimclustersV2InterestedIn20M145K2020ScalaDataset

override val outputDataset = SimclustersV2ClusterDetails20M145K2020ScalaDataset

override val clusterDetailsLiteOutputDataset =

SimclustersV2ClusterDetailsLite20M145K2020ScalaDataset

}

/\*\*

scalding remote run --main-class com.twitter.simclusters\_v2.scalding.DumpClusterDetailsAdhoc \

--target src/scala/com/twitter/simclusters\_v2/scalding:cluster\_details-dump \

--user recos-platform -- \

--date 2020-06-25 \

--clusterIds 5542 129677 48645 \

--inputDir /user/recos-platform/adhoc/your\_ldap/cluster\_details\_inferred\_lang

\*/

object DumpClusterDetailsAdhoc extends TwitterExecutionApp {

def job: Execution[Unit] =

Execution.getConfigMode.flatMap {

case (config, mode) =>

Execution.withId { implicit uniqueId =>

val args = config.getArgs

val clusters = args.list("clusterIds").map(\_.toInt).toSet //(1 to 2500).toSet //

TypedPipe

.from(AdhocKeyValSources.clusterDetailsSource(args("inputDir")))

.filter { case ((modelVersion, clusterId), details) => clusters.contains(clusterId) }

.toIterableExecution

.map { iter =>

iter.foreach { x => println(Util.prettyJsonMapper.writeValueAsString(x)) }

}

}

}

}

/\*\*

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

\* oscar hdfs --user cassowary --host hadoopnest2.atla.twitter.com --bundle cluster\_details \

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

\* --tee your\_ldap/dumpClusterSimilarities\_20200103 -- \

\* --inputDir /user/cassowary/manhattan\_sequence\_files/simclusters\_v2\_cluster\_details\_20m\_145k\_updated/ \

\* --outputDir adhoc/your\_ldap

\*/

object DumpClusterSimilaritiesAdhoc extends TwitterExecutionApp {

def job: Execution[Unit] =

Execution.getConfigMode.flatMap {

case (config, mode) =>

Execution.withId { implicit uniqueId =>

val args = config.getArgs

TypedPipe

.from(AdhocKeyValSources.clusterDetailsSource(args("inputDir")))

.flatMap {

case ((\_, clusterId), details) =>

details.neighborClusters.getOrElse(Nil).map { neighbor =>

val compositeScore = (neighbor.followCosineSimilarity

.getOrElse(0.0) + neighbor.favCosineSimilarity.getOrElse(0.0)) / 2

(

clusterId,

neighbor.clusterId,

"%.4f".format(compositeScore)

)

}

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

}

}

}