package com.twitter.simclusters\_v2.scalding

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

import com.twitter.scalding.Execution

import com.twitter.scalding.TypedTsv

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.multiformat.format.keyval.KeyVal

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

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

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

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

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

import com.twitter.simclusters\_v2.hdfs\_sources.SimclustersV2InterestedInFromProducerEmbeddings20M145KUpdatedScalaDataset

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

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

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

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

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

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

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

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

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

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

import java.util.TimeZone

import scala.util.Random

/\*\*

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

\*

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

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

\* interestedIn a cluster. To compute the engagement and determine the clusters for the user, we reuse

\* the functions defined in InterestedInKnownFor.

\*

\* Using producerEmbeddings instead of knownFor to obtain interestedIn increases the coverage (especially

\* for medium and light users) and also the density of the cluster embeddings for the user.

\*/

/\*\*

\* Adhoc job to generate the interestedIn from producer embeddings for the model version 20M145KUpdated

\*

scalding remote run \

--target src/scala/com/twitter/simclusters\_v2/scalding:interested\_in\_from\_producer\_embeddings \

--main-class com.twitter.simclusters\_v2.scalding.InterestedInFromProducerEmbeddingsAdhocApp \

--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\_prod\_embeddings/ \

--date 2020-08-25 --typedTsv true

\*/

object InterestedInFromProducerEmbeddingsAdhocApp extends AdhocExecutionApp {

override def runOnDateRange(

args: Args

)(

implicit dateRange: DateRange,

timeZone: TimeZone,

uniqueID: UniqueID

): Execution[Unit] = {

val outputDir = args("outputDir")

val inputGraph = args.optional("graphInputDir") match {

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

case None =>

DAL

.readMostRecentSnapshotNoOlderThan(UserUserNormalizedGraphScalaDataset, Days(30))

.toTypedPipe

}

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

val maxClustersPerUserFinalResult = args.int("maxInterestedInClustersPerUser", 50)

val maxClustersFromProducer = args.int("maxClustersPerProducer", 25)

val typedTsvTag = args.boolean("typedTsv")

val embeddingType =

EmbeddingType.ProducerFavBasedSemanticCoreEntity

val modelVersion = ModelVersions.Model20M145KUpdated

val producerEmbeddings = ProducerEmbeddingSources

.producerEmbeddingSourceLegacy(embeddingType, ModelVersions.toModelVersion(modelVersion))(

dateRange.embiggen(Days(7)))

import InterestedInFromProducerEmbeddingsBatchApp.\_

val numProducerMappings = Stat("num\_producer\_embeddings\_total")

val numProducersWithLargeClusterMappings = Stat(

"num\_producers\_with\_more\_clusters\_than\_threshold")

val numProducersWithSmallClusterMappings = Stat(

"num\_producers\_with\_clusters\_less\_than\_threshold")

val totalClustersCoverageProducerEmbeddings = Stat("num\_clusters\_total\_producer\_embeddings")

val producerEmbeddingsWithScore = producerEmbeddings.map {

case (userId: Long, topSimClusters: TopSimClustersWithScore) =>

(

userId,

topSimClusters.topClusters.toArray

.map {

case (simCluster: SimClusterWithScore) =>

(simCluster.clusterId, simCluster.score.toFloat)

}

)

}

val producerEmbeddingsPruned = producerEmbeddingsWithScore.map {

case (producerId, clusterArray) =>

numProducerMappings.inc()

val clusterSize = clusterArray.size

totalClustersCoverageProducerEmbeddings.incBy(clusterSize)

val prunedList = if (clusterSize > maxClustersFromProducer) {

numProducersWithLargeClusterMappings.inc()

clusterArray

.sortBy {

case (\_, knownForScore) => -knownForScore

}.take(maxClustersFromProducer)

} else {

numProducersWithSmallClusterMappings.inc()

clusterArray

}

(producerId, prunedList)

}

val result = InterestedInFromKnownFor

.run(

inputGraph,

producerEmbeddingsPruned,

socialProofThreshold,

maxClustersPerUserFinalResult,

modelVersion

)

val resultWithoutSocial = getInterestedInDiscardSocial(result)

if (typedTsvTag) {

Util.printCounters(

resultWithoutSocial

.map {

case (userId: Long, clusters: ClustersUserIsInterestedIn) =>

(

userId,

clusters.clusterIdToScores.keys.toString()

)

}

.writeExecution(

TypedTsv(outputDir)

)

)

} else {

Util.printCounters(

resultWithoutSocial

.writeExecution(

AdhocKeyValSources.interestedInSource(outputDir)

)

)

}

}

}

/\*\*

\* Production job for computing interestedIn data set from the producer embeddings for the model version 20M145KUpdated.

\* It writes the data set in KeyVal format to produce a MH DAL data set.

\*

\* To deploy the job:

\*

\* capesospy-v2 update --build\_locally --start\_cron

\* --start\_cron interested\_in\_from\_producer\_embeddings

\* src/scala/com/twitter/simclusters\_v2/capesos\_config/atla\_proc3.yaml

\*/

object InterestedInFromProducerEmbeddingsBatchApp extends ScheduledExecutionApp {

override val firstTime: RichDate = RichDate("2019-11-01")

override val batchIncrement: Duration = Days(7)

def getPrunedEmbeddings(

producerEmbeddings: TypedPipe[(Long, TopSimClustersWithScore)],

maxClustersFromProducer: Int

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

producerEmbeddings.map {

case (producerId, producerClusters) =>

val prunedProducerClusters =

producerClusters.topClusters

.sortBy {

case simCluster => -simCluster.score.toFloat

}.take(maxClustersFromProducer)

(producerId, TopSimClustersWithScore(prunedProducerClusters, producerClusters.modelVersion))

}

}

def getInterestedInDiscardSocial(

interestedInFromProducersResult: TypedPipe[(UserId, ClustersUserIsInterestedIn)]

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

interestedInFromProducersResult.map {

case (srcId, fullClusterList) =>

val fullClusterListWithoutSocial = fullClusterList.clusterIdToScores.map {

case (clusterId, clusterDetails) =>

val clusterDetailsWithoutSocial = UserToInterestedInClusterScores(

followScore = clusterDetails.followScore,

followScoreClusterNormalizedOnly = clusterDetails.followScoreClusterNormalizedOnly,

followScoreProducerNormalizedOnly = clusterDetails.followScoreProducerNormalizedOnly,

followScoreClusterAndProducerNormalized =

clusterDetails.followScoreClusterAndProducerNormalized,

favScore = clusterDetails.favScore,

favScoreClusterNormalizedOnly = clusterDetails.favScoreClusterNormalizedOnly,

favScoreProducerNormalizedOnly = clusterDetails.favScoreProducerNormalizedOnly,

favScoreClusterAndProducerNormalized =

clusterDetails.favScoreClusterAndProducerNormalized,

// Social proof is currently not being used anywhere else, hence being discarded to reduce space for this dataset

usersBeingFollowed = None,

usersThatWereFaved = None,

numUsersInterestedInThisClusterUpperBound =

clusterDetails.numUsersInterestedInThisClusterUpperBound,

logFavScore = clusterDetails.logFavScore,

logFavScoreClusterNormalizedOnly = clusterDetails.logFavScoreClusterNormalizedOnly,

// Counts of the social proof are maintained

numUsersBeingFollowed = Some(clusterDetails.usersBeingFollowed.getOrElse(Nil).size),

numUsersThatWereFaved = Some(clusterDetails.usersThatWereFaved.getOrElse(Nil).size)

)

(clusterId, clusterDetailsWithoutSocial)

}

(

srcId,

ClustersUserIsInterestedIn(

fullClusterList.knownForModelVersion,

fullClusterListWithoutSocial))

}

}

override def runOnDateRange(

args: Args

)(

implicit dateRange: DateRange,

timeZone: TimeZone,

uniqueID: UniqueID

): Execution[Unit] = {

//Input args for the run

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

val maxClustersFromProducer = args.int("maxClustersPerProducer", 25)

val maxClustersPerUserFinalResult = args.int("maxInterestedInClustersPerUser", 50)

//Path variables

val modelVersionUpdated = ModelVersions.toModelVersion(ModelVersions.Model20M145KUpdated)

val rootPath: String = s"/user/cassowary/manhattan\_sequence\_files"

val interestedInFromProducersPath =

rootPath + "/interested\_in\_from\_producer\_embeddings/" + modelVersionUpdated

//Input adjacency list and producer embeddings

val userUserNormalGraph =

DataSources.userUserNormalizedGraphSource(dateRange.prepend(Days(7))).forceToDisk

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

SimclustersV2InterestedInFromProducerEmbeddings20M145KUpdatedScalaDataset

val producerEmbeddings = ProducerEmbeddingSources

.producerEmbeddingSourceLegacy(

EmbeddingType.ProducerFavBasedSemanticCoreEntity,

modelVersionUpdated)(dateRange.embiggen(Days(7)))

val producerEmbeddingsPruned = getPrunedEmbeddings(producerEmbeddings, maxClustersFromProducer)

val producerEmbeddingsWithScore = producerEmbeddingsPruned.map {

case (userId: Long, topSimClusters: TopSimClustersWithScore) =>

(

userId,

topSimClusters.topClusters.toArray

.map {

case (simCluster: SimClusterWithScore) =>

(simCluster.clusterId, simCluster.score.toFloat)

}

)

}

val interestedInFromProducersResult =

InterestedInFromKnownFor.run(

userUserNormalGraph,

producerEmbeddingsWithScore,

socialProofThreshold,

maxClustersPerUserFinalResult,

modelVersionUpdated.toString

)

val interestedInFromProducersWithoutSocial =

getInterestedInDiscardSocial(interestedInFromProducersResult)

val writeKeyValResultExec = interestedInFromProducersWithoutSocial

.map { case (userId, clusters) => KeyVal(userId, clusters) }

.writeDALVersionedKeyValExecution(

outputKVDataset,

D.Suffix(interestedInFromProducersPath)

)

writeKeyValResultExec

}

}