package com.twitter.simclusters\_v2.scalding

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

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

import com.twitter.scalding.\_

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

import com.twitter.scalding\_internal.dalv2.DALWrite.D

import com.twitter.scalding\_internal.dalv2.DALWrite.WriteExtension

import com.twitter.scalding\_internal.dalv2.remote\_access.AllowCrossClusterSameDC

import com.twitter.scalding\_internal.dalv2.remote\_access.AllowCrossDC

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

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

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

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

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

import com.twitter.simclusters\_v2.hdfs\_sources.AggregatableProducerSimclustersEmbeddingsByLogFavScore2020ScalaDataset

import com.twitter.simclusters\_v2.hdfs\_sources.SimclustersV2InterestedInFromAggregatableProducerEmbeddings20M145K2020ScalaDataset

import com.twitter.simclusters\_v2.hdfs\_sources.SimclustersV2UserToInterestedInFromAggregatableProducerEmbeddings20M145K2020ScalaDataset

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

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

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

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

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

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

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

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

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

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

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

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

import java.util.TimeZone

/\*\*

\* Production job for computing interestedIn data set from the aggregatable producer embeddings for the model version 20M145K2020.

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

\*

\* A high level description of this job:

\* - Read the APE dataset

\* - Apply log1p to the scores from the above dataset as the scores for producers is high

\* - Normalize the scores for each producer (offline benchmarking has shown better results from this step.)

\* - Truncate the number of clusters for each producer from the APE dataset to reduce noise

\* - Compute interestedIn

\*

\* To deploy the job:

\*

\* capesospy-v2 update --build\_locally --start\_cron interested\_in\_from\_ape\_2020 \

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

\*/

object InterestedInFromAPE2020BatchApp extends InterestedInFromAggregatableProducerEmbeddingsBase {

override val firstTime: RichDate = RichDate("2021-03-03")

override val batchIncrement: Duration = Days(7)

override def modelVersion: ModelVersion = ModelVersion.Model20m145k2020

override def producerEmbeddingsInputKVDataset: KeyValDALDataset[

KeyVal[SimClustersEmbeddingId, SimClustersEmbedding]

] = AggregatableProducerSimclustersEmbeddingsByLogFavScore2020ScalaDataset

override def interestedInFromAPEOutputKVDataset: KeyValDALDataset[

KeyVal[UserId, ClustersUserIsInterestedIn]

] = SimclustersV2InterestedInFromAggregatableProducerEmbeddings20M145K2020ScalaDataset

override def interestedInFromAPEOutputThriftDatset: SnapshotDALDataset[

UserToInterestedInClusters

] = SimclustersV2UserToInterestedInFromAggregatableProducerEmbeddings20M145K2020ScalaDataset

}

trait InterestedInFromAggregatableProducerEmbeddingsBase extends ScheduledExecutionApp {

def modelVersion: ModelVersion

def interestedInFromAPEOutputKVDataset: KeyValDALDataset[

KeyVal[UserId, ClustersUserIsInterestedIn]

]

def producerEmbeddingsInputKVDataset: KeyValDALDataset[

KeyVal[SimClustersEmbeddingId, SimClustersEmbedding]

]

def interestedInFromAPEOutputThriftDatset: SnapshotDALDataset[UserToInterestedInClusters]

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", 5)

val maxClustersPerUserFinalResult = args.int("maxInterestedInClustersPerUser", 200)

//Path variables

val interestedInFromProducersPath =

s"/user/cassowary/manhattan\_sequence\_files/interested\_in\_from\_ape/" + modelVersion

val interestedInFromProducersThriftPath =

s"/user/cassowary/manhattan\_sequence\_files/interested\_in\_from\_ape\_thrift/" + modelVersion

val userUserGraph: TypedPipe[UserAndNeighbors] =

DAL

.readMostRecentSnapshotNoOlderThan(UserUserNormalizedGraphScalaDataset, Days(30))

.withRemoteReadPolicy(AllowCrossDC)

.toTypedPipe

val producerEmbeddings = DAL

.readMostRecentSnapshotNoOlderThan(

producerEmbeddingsInputKVDataset,

Days(30)).withRemoteReadPolicy(AllowCrossClusterSameDC).toTypedPipe.map {

case KeyVal(producer, embeddings) => (producer, embeddings)

}

val result = InterestedInFromAggregatableProducerEmbeddingsBase.run(

userUserGraph,

producerEmbeddings,

maxClustersFromProducer,

socialProofThreshold,

maxClustersPerUserFinalResult,

modelVersion)

val keyValExec =

result

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

.writeDALVersionedKeyValExecution(

interestedInFromAPEOutputKVDataset,

D.Suffix(interestedInFromProducersPath)

)

val thriftExec =

result

.map {

case (userId, clusters) =>

UserToInterestedInClusters(

userId,

ModelVersions.toKnownForModelVersion(modelVersion),

clusters.clusterIdToScores)

}

.writeDALSnapshotExecution(

interestedInFromAPEOutputThriftDatset,

D.Daily,

D.Suffix(interestedInFromProducersThriftPath),

D.EBLzo(),

dateRange.end

)

Execution.zip(keyValExec, thriftExec).unit

}

}

/\*\*

\* Adhoc job to generate the interestedIn from aggregatable producer embeddings for the model version 20M145K2020

\*

\* scalding remote run \

\* --user cassowary \

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

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

\* --cluster bluebird-qus1 \

\* --main-class com.twitter.simclusters\_v2.scalding.InterestedInFromAPE2020AdhocApp \

\* --target src/scala/com/twitter/simclusters\_v2/scalding:interested\_in\_from\_ape\_2020-adhoc \

\* --hadoop-properties "mapreduce.map.memory.mb=8192 mapreduce.map.java.opts='-Xmx7618M' mapreduce.reduce.memory.mb=8192 mapreduce.reduce.java.opts='-Xmx7618M'" \

\* -- --outputDir /gcs/user/cassowary/adhoc/your\_ldap/interested\_in\_from\_ape\_2020\_keyval --date 2021-03-05

\*/

object InterestedInFromAPE2020AdhocApp extends AdhocExecutionApp {

override def runOnDateRange(

args: Args

)(

implicit dateRange: DateRange,

timeZone: TimeZone,

uniqueID: UniqueID

): Execution[Unit] = {

val outputDir = args("outputDir")

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

val maxClustersPerUserFinalResult = args.int("maxInterestedInClustersPerUser", 200)

val maxClustersFromProducer = args.int("maxClustersFromProducer", 5)

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

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

case None =>

DAL

.readMostRecentSnapshotNoOlderThan(UserUserNormalizedGraphScalaDataset, Days(30))

.withRemoteReadPolicy(AllowCrossClusterSameDC)

.toTypedPipe

}

val producerEmbeddings = DAL

.readMostRecentSnapshotNoOlderThan(

AggregatableProducerSimclustersEmbeddingsByLogFavScore2020ScalaDataset,

Days(30)).withRemoteReadPolicy(AllowCrossClusterSameDC).toTypedPipe.map {

case KeyVal(producer, embeddings) => (producer, embeddings)

}

val result = InterestedInFromAggregatableProducerEmbeddingsBase.run(

inputGraph,

producerEmbeddings,

maxClustersFromProducer,

socialProofThreshold,

maxClustersPerUserFinalResult,

ModelVersion.Model20m145k2020)

result

.writeExecution(AdhocKeyValSources.interestedInSource(outputDir))

}

}

/\*\*

\* Helper functions

\*/

object InterestedInFromAggregatableProducerEmbeddingsBase {

/\*\*

\* Helper function to prune the embeddings

\* @param embeddingsWithScore embeddings

\* @param maxClusters number of clusters to keep, per userId

\* @param uniqueId for stats

\* @return

\*/

def getPrunedEmbeddings(

embeddingsWithScore: TypedPipe[(UserId, Seq[(ClusterId, Float)])],

maxClusters: Int

)(

implicit uniqueId: UniqueID

): TypedPipe[(UserId, Array[(ClusterId, Float)])] = {

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")

embeddingsWithScore.map {

case (producerId, clusterArray) =>

numProducerMappings.inc()

val clusterSize = clusterArray.size

totalClustersCoverageProducerEmbeddings.incBy(clusterSize)

val prunedList = if (clusterSize > maxClusters) {

numProducersWithLargeClusterMappings.inc()

clusterArray

.sortBy {

case (\_, knownForScore) => -knownForScore

}.take(maxClusters)

} else {

numProducersWithSmallClusterMappings.inc()

clusterArray

}

(producerId, prunedList.toArray)

}

}

/\*\*

\* helper function to remove all scores except follow and logFav

\* @param interestedInResult interestedIn clusters for a user

\* @return

\*/

def getInterestedInDiscardScores(

interestedInResult: TypedPipe[(UserId, List[(ClusterId, UserToInterestedInClusterScores)])]

): TypedPipe[(UserId, List[(ClusterId, UserToInterestedInClusterScores)])] = {

interestedInResult.map {

case (srcId, fullClusterList) =>

val fullClusterListWithDiscardedScores = fullClusterList.map {

case (clusterId, clusterDetails) =>

val clusterDetailsWithoutSocial = UserToInterestedInClusterScores(

// We are not planning to use the other scores except for logFav and Follow.

// Hence, setting others as None for now, we can add them back when needed

followScore = clusterDetails.followScore,

logFavScore = clusterDetails.logFavScore,

logFavScoreClusterNormalizedOnly = clusterDetails.logFavScoreClusterNormalizedOnly

)

(clusterId, clusterDetailsWithoutSocial)

}

(srcId, fullClusterListWithDiscardedScores)

}

}

/\*\*

\* Helper function to normalize the embeddings

\* @param embeddings cluster embeddings

\* @return

\*/

def getNormalizedEmbeddings(

embeddings: TypedPipe[(UserId, Seq[(ClusterId, Float)])]

): TypedPipe[(UserId, Seq[(ClusterId, Float)])] = {

embeddings.map {

case (userId, clustersWithScores) =>

val l2norm = math.sqrt(clustersWithScores.map(\_.\_2).map(score => score \* score).sum)

(

userId,

clustersWithScores.map {

case (clusterId, score) => (clusterId, (score / l2norm).toFloat)

})

}

}

def run(

userUserGraph: TypedPipe[UserAndNeighbors],

producerEmbeddings: TypedPipe[(SimClustersEmbeddingId, SimClustersEmbedding)],

maxClustersFromProducer: Int,

socialProofThreshold: Int,

maxClustersPerUserFinalResult: Int,

modelVersion: ModelVersion

)(

implicit uniqueId: UniqueID

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

import InterestedInFromKnownFor.\_

val producerEmbeddingsWithScore: TypedPipe[(UserId, Seq[(ClusterId, Float)])] =

producerEmbeddings.map {

case (

SimClustersEmbeddingId(embeddingType, modelVersion, InternalId.UserId(producerId)),

simclusterEmbedding) =>

(

producerId,

simclusterEmbedding.embedding.map { simclusterWithScore =>

// APE dataset has very high producer scores, hence applying log to smoothen them out before

// computing interestedIn

(simclusterWithScore.clusterId, math.log(1.0 + simclusterWithScore.score).toFloat)

})

}

val result = keepOnlyTopClusters(

getInterestedInDiscardScores(

attachNormalizedScores(

userClusterPairsWithoutNormalization(

userUserGraph,

getPrunedEmbeddings(

getNormalizedEmbeddings(producerEmbeddingsWithScore),

maxClustersFromProducer),

socialProofThreshold,

))),

maxClustersPerUserFinalResult,

ModelVersions.toKnownForModelVersion(modelVersion)

)

result

}

}