package com.twitter.simclusters\_v2.scalding.embedding

import com.twitter.bijection.Injection

import com.twitter.bijection.scrooge.CompactScalaCodec

import com.twitter.hermit.candidate.thriftscala.Candidate

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

import com.twitter.scalding.\_

import com.twitter.scalding.commons.source.VersionedKeyValSource

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

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

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

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

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

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

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

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

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

import java.util.TimeZone

/\*\*

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

--start\_cron similar\_users\_by\_simclusters\_embeddings\_job \

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

\*/

object SimilarUsersBySimClustersEmbeddingBatchApp extends ScheduledExecutionApp {

override val firstTime: RichDate = RichDate("2019-07-10")

override val batchIncrement: Duration = Days(7)

private val outputByFav =

"/user/cassowary/manhattan\_sequence\_files/similar\_users\_by\_simclusters\_embeddings/by\_fav"

private val outputByFollow =

"/user/cassowary/manhattan\_sequence\_files/similar\_users\_by\_simclusters\_embeddings/by\_follow"

private implicit val valueInj: CompactScalaCodec[Candidates] = CompactScalaCodec(Candidates)

private val topClusterEmbeddingsByFavScore = DAL

.readMostRecentSnapshotNoOlderThan(

ProducerTopKSimclusterEmbeddingsByFavScoreUpdatedScalaDataset,

Days(14)

)

.withRemoteReadPolicy(AllowCrossClusterSameDC)

.toTypedPipe

.map { clusterScorePair => clusterScorePair.key -> clusterScorePair.value }

private val topProducersForClusterEmbeddingByFavScore = DAL

.readMostRecentSnapshotNoOlderThan(

SimclusterEmbeddingTopKProducersByFavScoreUpdatedScalaDataset,

Days(14)

)

.withRemoteReadPolicy(AllowCrossClusterSameDC)

.toTypedPipe

.map { producerScoresPair => producerScoresPair.key -> producerScoresPair.value }

private val topClusterEmbeddingsByFollowScore = DAL

.readMostRecentSnapshotNoOlderThan(

ProducerTopKSimclusterEmbeddingsByFollowScoreUpdatedScalaDataset,

Days(14)

)

.withRemoteReadPolicy(AllowCrossClusterSameDC)

.toTypedPipe

.map { clusterScorePair => clusterScorePair.key -> clusterScorePair.value }

private val topProducersForClusterEmbeddingByFollowScore = DAL

.readMostRecentSnapshotNoOlderThan(

SimclusterEmbeddingTopKProducersByFollowScoreUpdatedScalaDataset,

Days(14)

)

.withRemoteReadPolicy(AllowCrossClusterSameDC)

.toTypedPipe

.map { producerScoresPair => producerScoresPair.key -> producerScoresPair.value }

override def runOnDateRange(

args: Args

)(

implicit dateRange: DateRange,

timeZone: TimeZone,

uniqueID: UniqueID

): Execution[Unit] = {

Execution

.zip(

SimilarUsersBySimClustersEmbedding

.getTopUsersRelatedToUser(

topClusterEmbeddingsByFavScore,

topProducersForClusterEmbeddingByFavScore

)

.map { case (key, value) => KeyVal(key, value) }

.writeDALVersionedKeyValExecution(

SimilarUsersByFavBasedProducerEmbeddingScalaDataset,

D.Suffix(outputByFav)

),

SimilarUsersBySimClustersEmbedding

.getTopUsersRelatedToUser(

topClusterEmbeddingsByFollowScore,

topProducersForClusterEmbeddingByFollowScore

)

.map { case (key, value) => KeyVal(key, value) }

.writeDALVersionedKeyValExecution(

SimilarUsersByFollowBasedProducerEmbeddingScalaDataset,

D.Suffix(outputByFollow)

)

).unit

}

}

/\*\*

\* Adhoc job to calculate producer's simcluster embeddings, which essentially assigns interestedIn

\* SimClusters to each producer, regardless of whether the producer has a knownFor assignment.

\*

./bazel bundle src/scala/com/twitter/simclusters\_v2/scalding/embedding:similar\_users\_by\_simclusters\_embeddings-adhoc && \

oscar hdfs --user recos-platform --screen --tee similar\_users\_by\_simclusters\_embeddings --bundle similar\_users\_by\_simclusters\_embeddings-adhoc \

--tool com.twitter.simclusters\_v2.scalding.embedding.SimilarUsersBySimClustersEmbeddingAdhocApp \

-- --date 2019-07-10T00 2019-07-10T23

\*/

object SimilarUsersBySimClustersEmbeddingAdhocApp extends AdhocExecutionApp {

private val outputByFav =

"/user/recos-platform/adhoc/similar\_users\_by\_simclusters\_embeddings/by\_fav"

private val outputByFollow =

"/user/recos-platform/adhoc/similar\_users\_by\_simclusters\_embeddings/by\_follow"

private val topClusterEmbeddingsByFavScore = DAL

.readMostRecentSnapshotNoOlderThan(

ProducerTopKSimclusterEmbeddingsByFavScoreUpdatedScalaDataset,

Days(14)

)

.withRemoteReadPolicy(AllowCrossClusterSameDC)

.toTypedPipe

.map { clusterScorePair => clusterScorePair.key -> clusterScorePair.value }

private val topProducersForClusterEmbeddingByFavScore = DAL

.readMostRecentSnapshotNoOlderThan(

SimclusterEmbeddingTopKProducersByFavScoreUpdatedScalaDataset,

Days(14)

)

.withRemoteReadPolicy(AllowCrossClusterSameDC)

.toTypedPipe

.map { producerScoresPair => producerScoresPair.key -> producerScoresPair.value }

private val topClusterEmbeddingsByFollowScore = DAL

.readMostRecentSnapshotNoOlderThan(

ProducerTopKSimclusterEmbeddingsByFollowScoreUpdatedScalaDataset,

Days(14)

)

.withRemoteReadPolicy(AllowCrossClusterSameDC)

.toTypedPipe

.map { clusterScorePair => clusterScorePair.key -> clusterScorePair.value }

private val topProducersForClusterEmbeddingByFollowScore = DAL

.readMostRecentSnapshotNoOlderThan(

SimclusterEmbeddingTopKProducersByFollowScoreUpdatedScalaDataset,

Days(14)

)

.withRemoteReadPolicy(AllowCrossClusterSameDC)

.toTypedPipe

.map { producerScoresPair => producerScoresPair.key -> producerScoresPair.value }

implicit val candidatesInj: CompactScalaCodec[Candidates] = CompactScalaCodec(Candidates)

override def runOnDateRange(

args: Args

)(

implicit dateRange: DateRange,

timeZone: TimeZone,

uniqueID: UniqueID

): Execution[Unit] = {

Execution

.zip(

SimilarUsersBySimClustersEmbedding

.getTopUsersRelatedToUser(

topClusterEmbeddingsByFavScore,

topProducersForClusterEmbeddingByFavScore).writeExecution(

VersionedKeyValSource[Long, Candidates](outputByFav))

.getCounters

.flatMap {

case (\_, counters) =>

counters.toMap.toSeq

.sortBy(e => (e.\_1.group, e.\_1.counter))

.foreach {

case (statKey, value) =>

println(s"${statKey.group}\t${statKey.counter}\t$value")

}

Execution.unit

},

SimilarUsersBySimClustersEmbedding

.getTopUsersRelatedToUser(

topClusterEmbeddingsByFollowScore,

topProducersForClusterEmbeddingByFollowScore).writeExecution(

VersionedKeyValSource[Long, Candidates](outputByFollow))

.getCounters

.flatMap {

case (\_, counters) =>

counters.toMap.toSeq

.sortBy(e => (e.\_1.group, e.\_1.counter))

.foreach {

case (statKey, value) =>

println(s"${statKey.group}\t${statKey.counter}\t$value")

}

Execution.unit

}

).unit

}

}

object SimilarUsersBySimClustersEmbedding {

private val maxUsersPerCluster = 300

private val maxClustersPerUser = 50

private val topK = 100

def getTopUsersRelatedToUser(

clusterScores: TypedPipe[(Long, TopSimClustersWithScore)],

producerScores: TypedPipe[(PersistedFullClusterId, TopProducersWithScore)]

)(

implicit uniqueID: UniqueID

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

val numUserUserPair = Stat("num\_user\_producer\_pairs")

val numUserClusterPair = Stat("num\_user\_cluster\_pairs")

val numClusterProducerPair = Stat("num\_cluster\_producer\_pairs")

val clusterToUserMap =

clusterScores.flatMap {

case (userId, topSimClustersWithScore) =>

val targetUserClusters =

topSimClustersWithScore.topClusters.sortBy(-\_.score).take(maxClustersPerUser)

targetUserClusters.map { simClusterWithScore =>

numUserClusterPair.inc()

simClusterWithScore.clusterId -> userId

}

}

val clusterToProducerMap = producerScores.flatMap {

case (persistedFullClusterId, topProducersWithScore) =>

numClusterProducerPair.inc()

val targetProducers = topProducersWithScore.topProducers

.sortBy(-\_.score)

.take(maxUsersPerCluster)

targetProducers.map { topProducerWithScore =>

persistedFullClusterId.clusterId -> topProducerWithScore.userId

}

}

implicit val intInject: Int => Array[Byte] = Injection.int2BigEndian.toFunction

val userToProducerMap =

clusterToUserMap.group

.sketch(2000)

.join(clusterToProducerMap.group)

.values

.distinct

.collect({

//filter self-pair

case userPair if userPair.\_1 != userPair.\_2 =>

numUserUserPair.inc()

userPair

})

val userEmbeddingsAllGrouped = clusterScores.map {

case (userId, topSimClustersWithScore) =>

val targetUserClusters =

topSimClustersWithScore.topClusters.sortBy(-\_.score).take(maxClustersPerUser)

val embedding = targetUserClusters.map { simClustersWithScore =>

simClustersWithScore.clusterId -> simClustersWithScore.score

}.toMap

val embeddingNormalized = CosineSimilarityUtil.normalize(embedding)

userId -> embeddingNormalized

}.forceToDisk

val userToProducerMapJoinWithEmbedding =

userToProducerMap

.join(userEmbeddingsAllGrouped)

.map {

case (user, (producer, userEmbedding)) =>

producer -> (user, userEmbedding)

}

.join(userEmbeddingsAllGrouped)

.map {

case (producer, ((user, userEmbedding), producerEmbedding)) =>

user -> (producer, CosineSimilarityUtil.dotProduct(userEmbedding, producerEmbedding))

}

.group

.sortWithTake(topK)((a, b) => a.\_2 > b.\_2)

.map {

case (userId, candidatesList) =>

val candidatesSeq = candidatesList

.map {

case (candidateId, score) => Candidate(candidateId, score)

}

userId -> Candidates(userId, candidatesSeq)

}

userToProducerMapJoinWithEmbedding

}

}