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

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

import com.twitter.scalding.\_

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

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.embedding.common.EmbeddingUtil.\_

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

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

import com.twitter.wtf.scalding.jobs.common.{AdhocExecutionApp, ScheduledExecutionApp}

import java.util.TimeZone

object ProducerEmbeddingsFromInterestedInBatchAppUtil {

import ProducerEmbeddingsFromInterestedIn.\_

val user = System.getenv("USER")

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

// Helps speed up the multiplication step which can get very big

val numReducersForMatrixMultiplication: Int = 12000

/\*\*

\* Given the producer x cluster matrix, key by producer / cluster individually, and write output

\* to individual DAL datasets

\*/

def writeOutput(

producerClusterEmbedding: TypedPipe[((ClusterId, UserId), Double)],

producerTopKEmbeddingsDataset: KeyValDALDataset[KeyVal[Long, TopSimClustersWithScore]],

clusterTopKProducersDataset: KeyValDALDataset[

KeyVal[PersistedFullClusterId, TopProducersWithScore]

],

producerTopKEmbeddingsPath: String,

clusterTopKProducersPath: String,

modelVersion: ModelVersion

): Execution[Unit] = {

val keyedByProducer =

toSimClusterEmbedding(producerClusterEmbedding, topKClustersToKeep, modelVersion)

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

.writeDALVersionedKeyValExecution(

producerTopKEmbeddingsDataset,

D.Suffix(producerTopKEmbeddingsPath)

)

val keyedBySimCluster = fromSimClusterEmbedding(

producerClusterEmbedding,

topKUsersToKeep,

modelVersion

).map {

case (clusterId, topProducers) => KeyVal(clusterId, topProducersToThrift(topProducers))

}

.writeDALVersionedKeyValExecution(

clusterTopKProducersDataset,

D.Suffix(clusterTopKProducersPath)

)

Execution.zip(keyedByProducer, keyedBySimCluster).unit

}

}

/\*\*

\* Base class for Fav based producer embeddings. Helps reuse the code for different model versions

\*/

trait ProducerEmbeddingsFromInterestedInByFavScoreBase extends ScheduledExecutionApp {

import ProducerEmbeddingsFromInterestedIn.\_

import ProducerEmbeddingsFromInterestedInBatchAppUtil.\_

def modelVersion: ModelVersion

val producerTopKEmbeddingsByFavScorePathPrefix: String =

"/producer\_top\_k\_simcluster\_embeddings\_by\_fav\_score\_"

val clusterTopKProducersByFavScorePathPrefix: String =

"/simcluster\_embedding\_top\_k\_producers\_by\_fav\_score\_"

val minNumFavers: Int = minNumFaversForProducer

def producerTopKSimclusterEmbeddingsByFavScoreDataset: KeyValDALDataset[

KeyVal[Long, TopSimClustersWithScore]

]

def simclusterEmbeddingTopKProducersByFavScoreDataset: KeyValDALDataset[

KeyVal[PersistedFullClusterId, TopProducersWithScore]

]

def getInterestedInFn: (DateRange, TimeZone) => TypedPipe[(Long, ClustersUserIsInterestedIn)]

override def runOnDateRange(

args: Args

)(

implicit dateRange: DateRange,

timeZone: TimeZone,

uniqueID: UniqueID

): Execution[Unit] = {

val producerTopKEmbeddingsByFavScorePathUpdated: String =

rootPath + producerTopKEmbeddingsByFavScorePathPrefix + ModelVersions

.toKnownForModelVersion(modelVersion)

val clusterTopKProducersByFavScorePathUpdated: String =

rootPath + clusterTopKProducersByFavScorePathPrefix + ModelVersions

.toKnownForModelVersion(modelVersion)

val producerClusterEmbeddingByFavScore = getProducerClusterEmbedding(

getInterestedInFn(dateRange.embiggen(Days(5)), timeZone),

DataSources.userUserNormalizedGraphSource,

DataSources.userNormsAndCounts,

userToProducerFavScore,

userToClusterFavScore, // Fav score

\_.faverCount.exists(\_ > minNumFavers),

numReducersForMatrixMultiplication,

modelVersion,

cosineSimilarityThreshold

).forceToDisk

writeOutput(

producerClusterEmbeddingByFavScore,

producerTopKSimclusterEmbeddingsByFavScoreDataset,

simclusterEmbeddingTopKProducersByFavScoreDataset,

producerTopKEmbeddingsByFavScorePathUpdated,

clusterTopKProducersByFavScorePathUpdated,

modelVersion

)

}

}

/\*\*

\* Base class for Follow based producer embeddings. Helps reuse the code for different model versions

\*/

trait ProducerEmbeddingsFromInterestedInByFollowScoreBase extends ScheduledExecutionApp {

import ProducerEmbeddingsFromInterestedIn.\_

import ProducerEmbeddingsFromInterestedInBatchAppUtil.\_

def modelVersion: ModelVersion

val producerTopKEmbeddingsByFollowScorePathPrefix: String =

"/producer\_top\_k\_simcluster\_embeddings\_by\_follow\_score\_"

val clusterTopKProducersByFollowScorePathPrefix: String =

"/simcluster\_embedding\_top\_k\_producers\_by\_follow\_score\_"

def producerTopKSimclusterEmbeddingsByFollowScoreDataset: KeyValDALDataset[

KeyVal[Long, TopSimClustersWithScore]

]

def simclusterEmbeddingTopKProducersByFollowScoreDataset: KeyValDALDataset[

KeyVal[PersistedFullClusterId, TopProducersWithScore]

]

def getInterestedInFn: (DateRange, TimeZone) => TypedPipe[(Long, ClustersUserIsInterestedIn)]

val minNumFollowers: Int = minNumFollowersForProducer

override def runOnDateRange(

args: Args

)(

implicit dateRange: DateRange,

timeZone: TimeZone,

uniqueID: UniqueID

): Execution[Unit] = {

val producerTopKEmbeddingsByFollowScorePath: String =

rootPath + producerTopKEmbeddingsByFollowScorePathPrefix + ModelVersions

.toKnownForModelVersion(modelVersion)

val clusterTopKProducersByFollowScorePath: String =

rootPath + clusterTopKProducersByFollowScorePathPrefix + ModelVersions

.toKnownForModelVersion(modelVersion)

val producerClusterEmbeddingByFollowScore = getProducerClusterEmbedding(

getInterestedInFn(dateRange.embiggen(Days(5)), timeZone),

DataSources.userUserNormalizedGraphSource,

DataSources.userNormsAndCounts,

userToProducerFollowScore,

userToClusterFollowScore, // Follow score

\_.followerCount.exists(\_ > minNumFollowers),

numReducersForMatrixMultiplication,

modelVersion,

cosineSimilarityThreshold

).forceToDisk

writeOutput(

producerClusterEmbeddingByFollowScore,

producerTopKSimclusterEmbeddingsByFollowScoreDataset,

simclusterEmbeddingTopKProducersByFollowScoreDataset,

producerTopKEmbeddingsByFollowScorePath,

clusterTopKProducersByFollowScorePath,

modelVersion

)

}

}

/\*\*

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

--start\_cron producer\_embeddings\_from\_interested\_in\_by\_fav\_score \

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

\*/

object ProducerEmbeddingsFromInterestedInByFavScoreBatchApp

extends ProducerEmbeddingsFromInterestedInByFavScoreBase {

override def modelVersion: ModelVersion = ModelVersion.Model20m145kUpdated

override def getInterestedInFn: (

DateRange,

TimeZone

) => TypedPipe[(UserId, ClustersUserIsInterestedIn)] =

InterestedInSources.simClustersInterestedInUpdatedSource

override val firstTime: RichDate = RichDate("2019-09-10")

override val batchIncrement: Duration = Days(7)

override def producerTopKSimclusterEmbeddingsByFavScoreDataset: KeyValDALDataset[

KeyVal[Long, TopSimClustersWithScore]

] =

ProducerTopKSimclusterEmbeddingsByFavScoreUpdatedScalaDataset

override def simclusterEmbeddingTopKProducersByFavScoreDataset: KeyValDALDataset[

KeyVal[PersistedFullClusterId, TopProducersWithScore]

] =

SimclusterEmbeddingTopKProducersByFavScoreUpdatedScalaDataset

}

/\*\*

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

--start\_cron producer\_embeddings\_from\_interested\_in\_by\_fav\_score\_2020 \

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

\*/

object ProducerEmbeddingsFromInterestedInByFavScore2020BatchApp

extends ProducerEmbeddingsFromInterestedInByFavScoreBase {

override def modelVersion: ModelVersion = ModelVersion.Model20m145k2020

override def getInterestedInFn: (

DateRange,

TimeZone

) => TypedPipe[(UserId, ClustersUserIsInterestedIn)] =

InterestedInSources.simClustersInterestedIn2020Source

override val firstTime: RichDate = RichDate("2021-03-01")

override val batchIncrement: Duration = Days(7)

override def producerTopKSimclusterEmbeddingsByFavScoreDataset: KeyValDALDataset[

KeyVal[Long, TopSimClustersWithScore]

] =

ProducerTopKSimclusterEmbeddingsByFavScore2020ScalaDataset

override def simclusterEmbeddingTopKProducersByFavScoreDataset: KeyValDALDataset[

KeyVal[PersistedFullClusterId, TopProducersWithScore]

] =

SimclusterEmbeddingTopKProducersByFavScore2020ScalaDataset

}

/\*\*

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

--start\_cron producer\_embeddings\_from\_interested\_in\_by\_fav\_score\_dec11 \

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

\*/

object ProducerEmbeddingsFromInterestedInByFavScoreDec11BatchApp

extends ProducerEmbeddingsFromInterestedInByFavScoreBase {

override def modelVersion: ModelVersion = ModelVersion.Model20m145kDec11

override def getInterestedInFn: (

DateRange,

TimeZone

) => TypedPipe[(UserId, ClustersUserIsInterestedIn)] =

InterestedInSources.simClustersInterestedInDec11Source

override val firstTime: RichDate = RichDate("2019-11-18")

override val batchIncrement: Duration = Days(7)

override def producerTopKSimclusterEmbeddingsByFavScoreDataset: KeyValDALDataset[

KeyVal[Long, TopSimClustersWithScore]

] =

ProducerTopKSimclusterEmbeddingsByFavScoreScalaDataset

override def simclusterEmbeddingTopKProducersByFavScoreDataset: KeyValDALDataset[

KeyVal[PersistedFullClusterId, TopProducersWithScore]

] =

SimclusterEmbeddingTopKProducersByFavScoreScalaDataset

}

/\*\*

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

--start\_cron producer\_embeddings\_from\_interested\_in\_by\_follow\_score \

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

\*/

object ProducerEmbeddingsFromInterestedInByFollowScoreBatchApp

extends ProducerEmbeddingsFromInterestedInByFollowScoreBase {

override def modelVersion: ModelVersion = ModelVersion.Model20m145kUpdated

override def getInterestedInFn: (

DateRange,

TimeZone

) => TypedPipe[(UserId, ClustersUserIsInterestedIn)] =

InterestedInSources.simClustersInterestedInUpdatedSource

override val firstTime: RichDate = RichDate("2019-09-10")

override val batchIncrement: Duration = Days(7)

override def producerTopKSimclusterEmbeddingsByFollowScoreDataset: KeyValDALDataset[

KeyVal[Long, TopSimClustersWithScore]

] =

ProducerTopKSimclusterEmbeddingsByFollowScoreUpdatedScalaDataset

override def simclusterEmbeddingTopKProducersByFollowScoreDataset: KeyValDALDataset[

KeyVal[PersistedFullClusterId, TopProducersWithScore]

] =

SimclusterEmbeddingTopKProducersByFollowScoreUpdatedScalaDataset

}

/\*\*

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

--start\_cron producer\_embeddings\_from\_interested\_in\_by\_follow\_score\_2020 \

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

\*/

object ProducerEmbeddingsFromInterestedInByFollowScore2020BatchApp

extends ProducerEmbeddingsFromInterestedInByFollowScoreBase {

override def modelVersion: ModelVersion = ModelVersion.Model20m145k2020

override def getInterestedInFn: (

DateRange,

TimeZone

) => TypedPipe[(UserId, ClustersUserIsInterestedIn)] =

InterestedInSources.simClustersInterestedIn2020Source

override val firstTime: RichDate = RichDate("2021-03-01")

override val batchIncrement: Duration = Days(7)

override def producerTopKSimclusterEmbeddingsByFollowScoreDataset: KeyValDALDataset[

KeyVal[Long, TopSimClustersWithScore]

] =

ProducerTopKSimclusterEmbeddingsByFollowScore2020ScalaDataset

override def simclusterEmbeddingTopKProducersByFollowScoreDataset: KeyValDALDataset[

KeyVal[PersistedFullClusterId, TopProducersWithScore]

] =

SimclusterEmbeddingTopKProducersByFollowScore2020ScalaDataset

}

/\*\*

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

--start\_cron producer\_embeddings\_from\_interested\_in\_by\_follow\_score\_dec11 \

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

\*/

object ProducerEmbeddingsFromInterestedInByFollowScoreDec11BatchApp

extends ProducerEmbeddingsFromInterestedInByFollowScoreBase {

override def modelVersion: ModelVersion = ModelVersion.Model20m145kDec11

override def getInterestedInFn: (

DateRange,

TimeZone

) => TypedPipe[(UserId, ClustersUserIsInterestedIn)] =

InterestedInSources.simClustersInterestedInDec11Source

override val firstTime: RichDate = RichDate("2019-11-18")

override val batchIncrement: Duration = Days(7)

override def producerTopKSimclusterEmbeddingsByFollowScoreDataset: KeyValDALDataset[

KeyVal[Long, TopSimClustersWithScore]

] =

ProducerTopKSimclusterEmbeddingsByFollowScoreScalaDataset

override def simclusterEmbeddingTopKProducersByFollowScoreDataset: KeyValDALDataset[

KeyVal[PersistedFullClusterId, TopProducersWithScore]

] =

SimclusterEmbeddingTopKProducersByFollowScoreScalaDataset

}

/\*\*

\* 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:producer\_embeddings\_from\_interested\_in-adhoc

$ scalding remote run \

--main-class com.twitter.simclusters\_v2.scalding.embedding.ProducerEmbeddingsFromInterestedInAdhocApp \

--target src/scala/com/twitter/simclusters\_v2/scalding/embedding:producer\_embeddings\_from\_interested\_in-adhoc \

--user cassowary --cluster bluebird-qus1 \

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

--principal service\_acoount@TWITTER.BIZ \

-- --date 2020-08-25 --model\_version 20M\_145K\_updated \

--outputDir /gcs/user/cassowary/adhoc/producerEmbeddings/

\*/

object ProducerEmbeddingsFromInterestedInAdhocApp extends AdhocExecutionApp {

import ProducerEmbeddingsFromInterestedIn.\_

private val numReducersForMatrixMultiplication = 12000

/\*\*

\* Calculate the embedding and writes the results keyed by producers and clusters separately into

\* individual locations

\*/

private def runAdhocByScore(

interestedInClusters: TypedPipe[(Long, ClustersUserIsInterestedIn)],

userUserNormalGraph: TypedPipe[UserAndNeighbors],

userNormsAndCounts: TypedPipe[NormsAndCounts],

keyedByProducerSinkPath: String,

keyedByClusterSinkPath: String,

userToProducerScoringFn: NeighborWithWeights => Double,

userToClusterScoringFn: UserToInterestedInClusterScores => Double,

userFilter: NormsAndCounts => Boolean,

modelVersion: ModelVersion

)(

implicit uniqueID: UniqueID

): Execution[Unit] = {

val producerClusterEmbedding = getProducerClusterEmbedding(

interestedInClusters,

userUserNormalGraph,

userNormsAndCounts,

userToProducerScoringFn,

userToClusterScoringFn,

userFilter,

numReducersForMatrixMultiplication,

modelVersion,

cosineSimilarityThreshold

).forceToDisk

val keyByProducerExec =

toSimClusterEmbedding(producerClusterEmbedding, topKClustersToKeep, modelVersion)

.writeExecution(

AdhocKeyValSources.topProducerToClusterEmbeddingsSource(keyedByProducerSinkPath))

val keyByClusterExec =

fromSimClusterEmbedding(producerClusterEmbedding, topKUsersToKeep, modelVersion)

.map { case (clusterId, topProducers) => (clusterId, topProducersToThrift(topProducers)) }

.writeExecution(

AdhocKeyValSources.topClusterEmbeddingsToProducerSource(keyedByClusterSinkPath))

Execution.zip(keyByProducerExec, keyByClusterExec).unit

}

// Calculate the embeddings using follow scores

private def runFollowScore(

interestedInClusters: TypedPipe[(Long, ClustersUserIsInterestedIn)],

userUserNormalGraph: TypedPipe[UserAndNeighbors],

userNormsAndCounts: TypedPipe[NormsAndCounts],

modelVersion: ModelVersion,

outputDir: String

)(

implicit uniqueID: UniqueID

): Execution[Unit] = {

val keyByClusterSinkPath = outputDir + "keyedByCluster/byFollowScore\_" + modelVersion

val keyByProducerSinkPath = outputDir + "keyedByProducer/byFollowScore\_" + modelVersion

runAdhocByScore(

interestedInClusters,

userUserNormalGraph,

userNormsAndCounts,

keyedByProducerSinkPath = keyByProducerSinkPath,

keyedByClusterSinkPath = keyByClusterSinkPath,

userToProducerScoringFn = userToProducerFollowScore,

userToClusterScoringFn = userToClusterFollowScore,

\_.followerCount.exists(\_ > minNumFollowersForProducer),

modelVersion

)

}

// Calculate the embeddings using fav scores

private def runFavScore(

interestedInClusters: TypedPipe[(Long, ClustersUserIsInterestedIn)],

userUserNormalGraph: TypedPipe[UserAndNeighbors],

userNormsAndCounts: TypedPipe[NormsAndCounts],

modelVersion: ModelVersion,

outputDir: String

)(

implicit uniqueID: UniqueID

): Execution[Unit] = {

val keyByClusterSinkPath = outputDir + "keyedByCluster/byFavScore\_" + modelVersion

val keyByProducerSinkPath = outputDir + "keyedByProducer/byFavScore\_" + modelVersion

runAdhocByScore(

interestedInClusters,

userUserNormalGraph,

userNormsAndCounts,

keyedByProducerSinkPath = keyByProducerSinkPath,

keyedByClusterSinkPath = keyByClusterSinkPath,

userToProducerScoringFn = userToProducerFavScore,

userToClusterScoringFn = userToClusterFavScore,

\_.faverCount.exists(\_ > minNumFaversForProducer),

modelVersion

)

}

override def runOnDateRange(

args: Args

)(

implicit dateRange: DateRange,

timeZone: TimeZone,

uniqueID: UniqueID

): Execution[Unit] = {

val outputDir = args("outputDir")

val modelVersion =

ModelVersions.toModelVersion(args.required("model\_version"))

val interestedInClusters = modelVersion match {

case ModelVersion.Model20m145k2020 =>

InterestedInSources.simClustersInterestedIn2020Source(dateRange, timeZone).forceToDisk

case ModelVersion.Model20m145kUpdated =>

InterestedInSources.simClustersInterestedInUpdatedSource(dateRange, timeZone).forceToDisk

case \_ =>

InterestedInSources.simClustersInterestedInDec11Source(dateRange, timeZone).forceToDisk

}

Execution

.zip(

runFavScore(

interestedInClusters,

DataSources.userUserNormalizedGraphSource,

DataSources.userNormsAndCounts,

modelVersion,

outputDir

),

runFollowScore(

interestedInClusters,

DataSources.userUserNormalizedGraphSource,

DataSources.userNormsAndCounts,

modelVersion,

outputDir

)

).unit

}

}

/\*\*

\* Computes the producer's interestedIn cluster embedding. i.e. If a tweet author (producer) is not

\* associated with a KnownFor cluster, do a cross-product between

\* [user, interestedIn] and [user, producer] to find the similarity matrix [interestedIn, producer].

\*/

object ProducerEmbeddingsFromInterestedIn {

val minNumFollowersForProducer: Int = 100

val minNumFaversForProducer: Int = 100

val topKUsersToKeep: Int = 300

val topKClustersToKeep: Int = 60

val cosineSimilarityThreshold: Double = 0.01

type ClusterId = Int

def topProducersToThrift(producersWithScore: Seq[(UserId, Double)]): TopProducersWithScore = {

val thrift = producersWithScore.map { producer =>

TopProducerWithScore(producer.\_1, producer.\_2)

}

TopProducersWithScore(thrift)

}

def userToProducerFavScore(neighbor: NeighborWithWeights): Double = {

neighbor.favScoreHalfLife100DaysNormalizedByNeighborFaversL2.getOrElse(0.0)

}

def userToProducerFollowScore(neighbor: NeighborWithWeights): Double = {

neighbor.followScoreNormalizedByNeighborFollowersL2.getOrElse(0.0)

}

def userToClusterFavScore(clusterScore: UserToInterestedInClusterScores): Double = {

clusterScore.favScoreClusterNormalizedOnly.getOrElse(0.0)

}

def userToClusterFollowScore(clusterScore: UserToInterestedInClusterScores): Double = {

clusterScore.followScoreClusterNormalizedOnly.getOrElse(0.0)

}

def getUserSimClustersMatrix(

simClustersSource: TypedPipe[(UserId, ClustersUserIsInterestedIn)],

extractScore: UserToInterestedInClusterScores => Double,

modelVersion: ModelVersion

): TypedPipe[(UserId, Seq[(Int, Double)])] = {

simClustersSource.collect {

case (userId, clusters)

if ModelVersions.toModelVersion(clusters.knownForModelVersion).equals(modelVersion) =>

userId -> clusters.clusterIdToScores

.map {

case (clusterId, clusterScores) =>

(clusterId, extractScore(clusterScores))

}.toSeq.filter(\_.\_2 > 0)

}

}

/\*\*

\* Given a weighted user-producer engagement history matrix, as well as a

\* weighted user-interestedInCluster matrix, do the matrix multiplication to yield a weighted

\* producer-cluster embedding matrix

\*/

def getProducerClusterEmbedding(

interestedInClusters: TypedPipe[(UserId, ClustersUserIsInterestedIn)],

userProducerEngagementGraph: TypedPipe[UserAndNeighbors],

userNormsAndCounts: TypedPipe[NormsAndCounts],

userToProducerScoringFn: NeighborWithWeights => Double,

userToClusterScoringFn: UserToInterestedInClusterScores => Double,

userFilter: NormsAndCounts => Boolean, // function to decide whether to compute embeddings for the user or not

numReducersForMatrixMultiplication: Int,

modelVersion: ModelVersion,

threshold: Double

)(

implicit uid: UniqueID

): TypedPipe[((ClusterId, UserId), Double)] = {

val userSimClustersMatrix = getUserSimClustersMatrix(

interestedInClusters,

userToClusterScoringFn,

modelVersion

)

val userUserNormalizedGraph = getFilteredUserUserNormalizedGraph(

userProducerEngagementGraph,

userNormsAndCounts,

userToProducerScoringFn,

userFilter

)

SimClustersEmbeddingJob

.legacyMultiplyMatrices(

userUserNormalizedGraph,

userSimClustersMatrix,

numReducersForMatrixMultiplication

)

.filter(\_.\_2 >= threshold)

}

def getFilteredUserUserNormalizedGraph(

userProducerEngagementGraph: TypedPipe[UserAndNeighbors],

userNormsAndCounts: TypedPipe[NormsAndCounts],

userToProducerScoringFn: NeighborWithWeights => Double,

userFilter: NormsAndCounts => Boolean

)(

implicit uid: UniqueID

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

val numUsersCount = Stat("num\_users\_with\_engagements")

val userUserFilteredEdgeCount = Stat("num\_filtered\_user\_user\_engagements")

val validUsersCount = Stat("num\_valid\_users")

val validUsers = userNormsAndCounts.collect {

case user if userFilter(user) =>

validUsersCount.inc()

user.userId

}

userProducerEngagementGraph

.flatMap { userAndNeighbors =>

numUsersCount.inc()

userAndNeighbors.neighbors

.map { neighbor =>

userUserFilteredEdgeCount.inc()

(neighbor.neighborId, (userAndNeighbors.userId, userToProducerScoringFn(neighbor)))

}

.filter(\_.\_2.\_2 > 0.0)

}

.join(validUsers.asKeys)

.map {

case (neighborId, ((userId, score), \_)) =>

(userId, (neighborId, score))

}

}

def fromSimClusterEmbedding[T, E](

resultMatrix: TypedPipe[((ClusterId, T), Double)],

topK: Int,

modelVersion: ModelVersion

): TypedPipe[(PersistedFullClusterId, Seq[(T, Double)])] = {

resultMatrix

.map {

case ((clusterId, inputId), score) => (clusterId, (inputId, score))

}

.group

.sortedReverseTake(topK)(Ordering.by(\_.\_2))

.map {

case (clusterId, topEntitiesWithScore) =>

PersistedFullClusterId(modelVersion, clusterId) -> topEntitiesWithScore

}

}

def toSimClusterEmbedding[T](

resultMatrix: TypedPipe[((ClusterId, T), Double)],

topK: Int,

modelVersion: ModelVersion

)(

implicit ordering: Ordering[T]

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

resultMatrix

.map {

case ((clusterId, inputId), score) => (inputId, (clusterId, score))

}

.group

//.withReducers(3000) // uncomment for producer-simclusters job

.sortedReverseTake(topK)(Ordering.by(\_.\_2))

.map {

case (inputId, topSimClustersWithScore) =>

val topSimClusters = topSimClustersWithScore.map {

case (clusterId, score) => SimClusterWithScore(clusterId, score)

}

inputId -> TopSimClustersWithScore(topSimClusters, modelVersion)

}

}

}