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

import com.twitter.data.proto.Flock

import com.twitter.scalding.{DateOps, DateRange, Days, RichDate, UniqueID}

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

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

import com.twitter.simclusters\_v2.scalding.common.matrix.SparseMatrix

import com.twitter.simclusters\_v2.scalding.embedding.common.EmbeddingUtil.{ClusterId, UserId}

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

import graphstore.common.FlockBlocksJavaDataset

import java.util.TimeZone

object DataSources {

private val ValidEdgeStateId = 0

val NumBlocksP95 = 49

/\*\*

\* Helper function to return Sparse Matrix of user's interestedIn clusters and fav scores

\* @param dateRange

\* @return

\*/

def getUserInterestedInSparseMatrix(

implicit dateRange: DateRange,

timeZone: TimeZone

): SparseMatrix[UserId, ClusterId, Double] = {

val simClusters = ExternalDataSources.simClustersInterestInSource

val simClusterMatrixEntries = simClusters

.flatMap { keyVal =>

keyVal.value.clusterIdToScores.flatMap {

case (clusterId, score) =>

score.favScore.map { favScore =>

(keyVal.key, clusterId, favScore)

}

}

}

SparseMatrix.apply[UserId, ClusterId, Double](simClusterMatrixEntries)

}

def getUserInterestedInTruncatedKMatrix(

topK: Int

)(

implicit dateRange: DateRange,

timeZone: TimeZone,

uniqueID: UniqueID

): SparseMatrix[UserId, ClusterId, Double] = {

SparseMatrix(

InterestedInSources

.simClustersInterestedInUpdatedSource(dateRange, timeZone)

.flatMap {

case (userId, clustersUserIsInterestedIn) =>

val sortedAndTruncatedList = clustersUserIsInterestedIn.clusterIdToScores

.mapValues(\_.favScore.getOrElse(0.0)).filter(\_.\_2 > 0.0).toList.sortBy(-\_.\_2).take(

topK)

sortedAndTruncatedList.map {

case (clusterId, score) =>

(userId, clusterId, score)

}

}

)

}

/\*\*

\* Helper function to return SparseMatrix of user block interactions from the FlockBlocks

\* dataset. All users with greater than numBlocks are filtered out

\* @param dateRange

\* @return

\*/

def getFlockBlocksSparseMatrix(

maxNumBlocks: Int,

rangeForData: DateRange

)(

implicit dateRange: DateRange

): SparseMatrix[UserId, UserId, Double] = {

implicit val tz: java.util.TimeZone = DateOps.UTC

val userGivingBlocks = SparseMatrix.apply[UserId, UserId, Double](

DAL

.readMostRecentSnapshotNoOlderThan(FlockBlocksJavaDataset, Days(30))

.toTypedPipe

.flatMap { data: Flock.Edge =>

// Consider edges that are valid and have been updated in the past 1 year

if (data.getStateId == ValidEdgeStateId &&

rangeForData.contains(RichDate(data.getUpdatedAt \* 1000L))) {

Some((data.getSourceId, data.getDestinationId, 1.0))

} else {

None

}

})

// Find all users who give less than numBlocksP95 blocks.

// This is to remove those who might be responsible for automatically blocking users

// on the twitter platform.

val usersWithLegitBlocks = userGivingBlocks.rowL1Norms.collect {

case (userId, l1Norm) if l1Norm <= maxNumBlocks =>

userId

}

// retain only those users who give legit blocks (i.e those users who give less than numBlocks95)

userGivingBlocks.filterRows(usersWithLegitBlocks)

}

}