package com.twitter.simclusters\_v2.scalding.topic\_recommendations

import com.twitter.bijection.{Bufferable, Injection}

import com.twitter.scalding.\_

import com.twitter.simclusters\_v2.common.{Country, Language, SemanticCoreEntityId, TopicId, UserId}

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

import com.twitter.simclusters\_v2.scalding.embedding.common.EmbeddingUtil.ProducerId

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

object TopicsForProducersUtils {

implicit val sparseMatrixInj: Injection[

(SemanticCoreEntityId, Option[Language], Option[Country]),

Array[Byte]

] =

Bufferable.injectionOf[(SemanticCoreEntityId, Option[Language], Option[Country])]

// This function provides the set of 'valid' topics, i.e topics with atleast a certain number of

// follows. This helps remove some noisy topic associations to producers in the dataset.

def getValidTopics(

topicUsers: TypedPipe[((TopicId, Option[Language], Option[Country]), UserId, Double)],

minTopicFollowsThreshold: Int

)(

implicit uniqueID: UniqueID

): TypedPipe[(TopicId, Option[Language], Option[Country])] = {

val numValidTopics = Stat("num\_valid\_topics")

SparseMatrix(topicUsers).rowNnz.collect {

case (topicsWithLocaleKey, numFollows) if numFollows >= minTopicFollowsThreshold =>

numValidTopics.inc()

topicsWithLocaleKey

}

}

// Get the users with atleast minNumUserFollowers following

def getValidProducers(

userToFollowersEdges: TypedPipe[(UserId, UserId, Double)],

minNumUserFollowers: Int

)(

implicit uniqueID: UniqueID

): TypedPipe[ProducerId] = {

val numProducersForTopics = Stat("num\_producers\_for\_topics")

SparseMatrix(userToFollowersEdges).rowL1Norms.collect {

case (userId, l1Norm) if l1Norm >= minNumUserFollowers =>

numProducersForTopics.inc()

userId

}

}

// This function returns the User to Followed Topics Matrix

def getFollowedTopicsToUserSparseMatrix(

followedTopicsToUsers: TypedPipe[(TopicId, UserId)],

userCountryAndLanguage: TypedPipe[(UserId, (Country, Language))],

userLanguages: TypedPipe[(UserId, Seq[(Language, Double)])],

minTopicFollowsThreshold: Int

)(

implicit uniqueID: UniqueID

): SparseMatrix[(TopicId, Option[Language], Option[Country]), UserId, Double] = {

val localeTopicsWithUsers: TypedPipe[

((TopicId, Option[Language], Option[Country]), UserId, Double)

] =

followedTopicsToUsers

.map { case (topic, user) => (user, topic) }

.join(userCountryAndLanguage)

.join(userLanguages)

.withDescription("joining user locale information")

.flatMap {

case (user, ((topic, (country, \_)), scoredLangs)) =>

scoredLangs.flatMap {

case (lang, score) =>

// To compute the top topics with/without language and country level personalization

// So the same dataset has 3 keys for each topicId (unless it gets filtered after):

// (TopicId, Language, Country), (TopicId, Language, None), (TopicId, None, None)

Seq(

((topic, Some(lang), Some(country)), user, score), // with language and country

((topic, Some(lang), None), user, score) // with language

)

} ++ Seq(((topic, None, None), user, 1.0)) // no locale

}

SparseMatrix(localeTopicsWithUsers).filterRowsByMinSum(minTopicFollowsThreshold)

}

// This function returns the Producers To User Followers Matrix

def getProducersToFollowedByUsersSparseMatrix(

userUserGraph: TypedPipe[UserAndNeighbors],

minActiveFollowers: Int,

)(

implicit uniqueID: UniqueID

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

val numEdgesFromUsersToFollowers = Stat("num\_edges\_from\_users\_to\_followers")

val userToFollowersEdges: TypedPipe[(UserId, UserId, Double)] =

userUserGraph

.flatMap { userAndNeighbors =>

userAndNeighbors.neighbors

.collect {

case neighbor if neighbor.isFollowed.getOrElse(false) =>

numEdgesFromUsersToFollowers.inc()

(neighbor.neighborId, userAndNeighbors.userId, 1.0)

}

}

SparseMatrix(userToFollowersEdges).filterRowsByMinSum(minActiveFollowers)

}

}