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

import com.twitter.bijection.Bufferable

import com.twitter.bijection.Injection

import com.twitter.recos.entities.thriftscala.\_

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.Country

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

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

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

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

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

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

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

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

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

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

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

import java.util.TimeZone

/\*\*

\* In this file, we compute the top producers for a topic from the Topic Follow Graph

\*

\* It works as follows:

\*

\* 1. Producer embedding: List of users who follow the producer's profile and follow atleast one topic

\*

\* 2. Topic embedding: List of users who follow the topic

\*

\* 3. Score(producer, topic) = cosine similarity of the producer and topic embedding as defined above

\*

\* 4. Please note that we compute the top producers for each topic locale.

\*/

/\*\*

scalding remote run --user cassowary \

--target src/scala/com/twitter/simclusters\_v2/scalding/topic\_recommendations:top\_producers\_for\_topics\_from\_topic\_follow\_graph-adhoc \

--main-class com.twitter.simclusters\_v2.scalding.topic\_recommendations.ProducersForTopicsFromTopicFollowGraphAdhocApp \

--submitter hadoopnest1.atla.twitter.com \

-- --date 2021-01-06 --minActiveFollowers 400 --maxProducersPerTopic 50 \

--output\_dir\_producers\_per\_topic /user/cassowary/adhoc/ldap/ttf\_profile\_pages\_topics\_to\_producers

\*/

object ProducersForTopicsFromTopicFollowGraphAdhocApp extends AdhocExecutionApp {

override def runOnDateRange(

args: Args

)(

implicit dateRange: DateRange,

timeZone: TimeZone,

uniqueID: UniqueID

): Execution[Unit] = {

import ProducersForTopicsFromTopicFollowGraph.\_

val outputDirProducersPerTopic = args("output\_dir\_producers\_per\_topic")

val minActiveFollowersForProducer = args.int("minActiveFollowers", 400)

val maxProducersPerTopicPerLocale = args.int("maxProducersPerTopic", 50)

val minTopicFollows = args.int("minTopicFollows", 100)

val topicsFollowedByProducersFollowers = getTopicsFromProducersFollowers(

DataSources

.userUserNormalizedGraphSource(dateRange.prepend(Days(7))),

ExternalDataSources.topicFollowGraphSource,

ExternalDataSources.userSource,

ExternalDataSources.inferredUserConsumedLanguageSource,

minActiveFollowersForProducer,

minTopicFollows

)

sortAndGetTopProducersPerLocaleTopic(

topicsFollowedByProducersFollowers,

maxProducersPerTopicPerLocale).writeExecution(TypedTsv(outputDirProducersPerTopic))

}

}

/\*\*

capesospy-v2 update --build\_locally \

--start\_cron top\_producers\_for\_topics\_from\_topic\_follow\_graph \

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

\*/

object ProducersForTopicsFromTopicFollowGraphBatchApp extends ScheduledExecutionApp {

override val firstTime: RichDate = RichDate("2020-10-01")

override val batchIncrement: Duration = Days(1)

private val topProducersForLocaleTopicsPath: String =

"/user/cassowary/manhattan\_sequence\_files/top\_producers\_for\_topics\_from\_topic\_follow\_graph"

override def runOnDateRange(

args: Args

)(

implicit dateRange: DateRange,

timeZone: TimeZone,

uniqueID: UniqueID

): Execution[Unit] = {

import ProducersForTopicsFromTopicFollowGraph.\_

val minActiveFollowersForProducer = args.int("minActiveFollowers", 400)

val maxProducersPerTopicPerLocale = args.int("maxProducersPerTopic", 50)

val minTopicFollows = args.int("minTopicFollows", 100)

val topicsFollowedByProducersFollowers = getTopicsFromProducersFollowers(

DataSources

.userUserNormalizedGraphSource(dateRange.prepend(Days(7))),

ExternalDataSources.topicFollowGraphSource,

ExternalDataSources.userSource,

ExternalDataSources.inferredUserConsumedLanguageSource,

minActiveFollowersForProducer,

minTopicFollows

)

sortAndGetTopProducersPerLocaleTopic(

topicsFollowedByProducersFollowers,

maxProducersPerTopicPerLocale)

.map {

case ((topicId, languageOpt, countryOpt), producersWithScores) =>

KeyVal(

SemanticCoreEntityWithLocale(

entityId = topicId,

context = Locale(language = languageOpt, country = countryOpt)),

UserScoreList(producersWithScores.map {

case (producerId, producerScore) =>

UserWithScore(userId = producerId, score = producerScore)

})

)

}.writeDALVersionedKeyValExecution(

TopProducersForLocaleTopicsFromTopicFollowGraphScalaDataset,

D.Suffix(topProducersForLocaleTopicsPath),

version = ExplicitEndTime(dateRange.end)

)

}

}

object ProducersForTopicsFromTopicFollowGraph {

implicit val sparseMatrixInj: Injection[

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

Array[Byte]

] =

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

// This function takes the producer to topics map and generates the sorted and

// truncated top producers ranked list for each locale topic

def sortAndGetTopProducersPerLocaleTopic(

producerToTopics: TypedPipe[(ProducerId, (TopicId, Option[Language], Option[Country]), Double)],

maxProducersPerLocaleTopic: Int

)(

implicit uniqueID: UniqueID

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

val numTopicsWithLocales = Stat("num\_topics\_with\_locales")

producerToTopics

.map {

case (producerId, (topicId, languageOpt, countryOpt), score) =>

((topicId, languageOpt, countryOpt), Seq((producerId, score)))

}

.sumByKey.mapValues { producersList =>

numTopicsWithLocales.inc()

producersList.sortBy(-\_.\_2).take(maxProducersPerLocaleTopic).toList

}.toTypedPipe

}

def getTopicsFromProducersFollowers(

userUserGraph: TypedPipe[UserAndNeighbors],

followedTopicsToUsers: TypedPipe[(TopicId, UserId)],

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

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

minActiveFollowersForProducer: Int,

minTopicFollows: Int

)(

implicit dateRange: DateRange,

timeZone: TimeZone,

uniqueID: UniqueID

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

val usersFollowingTopics: TypedPipe[UserId] = followedTopicsToUsers.map(\_.\_2).distinct

val producerToUsersSparseMatrix: SparseMatrix[ProducerId, UserId, Double] =

TopicsForProducersUtils

.getProducersToFollowedByUsersSparseMatrix(

userUserGraph,

minActiveFollowersForProducer).filterCols(usersFollowingTopics).rowL2Normalize

val userToTopicsSparseSkinnyMatrix: SparseMatrix[

UserId,

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

Double

] =

TopicsForProducersUtils

.getFollowedTopicsToUserSparseMatrix(

followedTopicsToUsers,

userSource,

userLanguages,

minTopicFollows).rowL2Normalize.transpose

// Obtain the Producer to Locale Topics Matrix

val producersToLocaleTopicsMatrix: SparseMatrix[

ProducerId,

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

Double

] =

producerToUsersSparseMatrix.multiplySparseMatrix(userToTopicsSparseSkinnyMatrix)

producersToLocaleTopicsMatrix.toTypedPipe

}

}