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.SemanticCoreEntityId

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.TopLocaleTopicsForProducerFromEmScalaDataset

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 com.twitter.wtf.scalding.jobs.common.EMRunner

import java.util.TimeZone

/\*\*

\* In this file, we compute the top topics for a producer to be shown on the Topics To Follow Module on Profile Pages

\*

\* The top topics for a producer are computed using the Expectation-Maximization (EM) approach

\*

\* It works as follows:

\*

\* 1. Obtain the background model distribution of number of followers for a topic

\*

\* 2. Obtain the domain model distribution of the number of producer's followers who follow a topic

\*

\* 4. Iteratively, use the Expectation-Maximization approach to get the best estimate of the domain model's topic distribution for a producer

\*

\* 5. for each producer, we only keep its top K topics with highest weights in the domain model's topic distribution after the EM step

\*

\* 6. Please note that we also store the locale info for each producer along with the topics

\*/

/\*\*

scalding remote run --user cassowary --reducers 2000 \

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

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

--submitter hadoopnest1.atla.twitter.com \

-- --date 2020-07-05 --minActiveFollowers 10000 --minTopicFollowsThreshold 100 --maxTopicsPerProducerPerLocale 50 \

--output\_dir\_topics\_per\_producer /user/cassowary/adhoc/your\_ldap/ttf\_profile\_pages\_producers\_to\_topics

\*/

object TopicsForProducersFromEMAdhocApp extends AdhocExecutionApp {

override def runOnDateRange(

args: Args

)(

implicit dateRange: DateRange,

timeZone: TimeZone,

uniqueID: UniqueID

): Execution[Unit] = {

import TopicsForProducersFromEM.\_

val outputDirTopicsPerProducer = args("output\_dir\_topics\_per\_producer")

val minActiveFollowersForProducer = args.int("minActiveFollowers", 100)

val minTopicFollowsThreshold = args.int("minNumTopicFollows", 100)

val maxTopicsPerProducerPerLocale = args.int("maxTopicsPerProducer", 100)

val lambda = args.double("lambda", 0.95)

val numEMSteps = args.int("numEM", 100)

val topicsFollowedByProducersFollowers: TypedPipe[

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

] = getTopLocaleTopicsForProducersFromEM(

DataSources

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

ExternalDataSources.topicFollowGraphSource,

ExternalDataSources.userSource,

ExternalDataSources.inferredUserConsumedLanguageSource,

minActiveFollowersForProducer,

minTopicFollowsThreshold,

lambda,

numEMSteps

)

val topTopicsPerLocaleProducerTsvExec = sortAndGetTopLocaleTopicsPerProducer(

topicsFollowedByProducersFollowers,

maxTopicsPerProducerPerLocale

).writeExecution(

TypedTsv(outputDirTopicsPerProducer)

)

topTopicsPerLocaleProducerTsvExec

}

}

/\*\*

capesospy-v2 update --build\_locally \

--start\_cron top\_topics\_for\_producers\_from\_em \

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

\*/

object TopicsForProducersFromEMBatchApp extends ScheduledExecutionApp {

override val firstTime: RichDate = RichDate("2020-07-26")

override val batchIncrement: Duration = Days(7)

private val topTopicsPerProducerFromEMPath: String =

"/user/cassowary/manhattan\_sequence\_files/top\_topics\_for\_producers\_from\_em"

override def runOnDateRange(

args: Args

)(

implicit dateRange: DateRange,

timeZone: TimeZone,

uniqueID: UniqueID

): Execution[Unit] = {

import TopicsForProducersFromEM.\_

// threshold of the minimum number of active followers needed for a user to be considered as a producer

val minActiveFollowersForProducer = args.int("minActiveFollowers", 100)

// threshold of the topic locale follows score needed for a topic to be considered as valid

val minTopicFollowsThreshold = args.int("minNumTopicFollows", 100)

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

// lambda parameter for the EM algorithm

val lambda = args.double("lambda", 0.95)

// number of EM iterations

val numEMSteps = args.int("numEM", 100)

// (producer, locale) -> List<(topics, scores)> from Expectation Maximization approach

val topicsFollowedByProducersFollowers = getTopLocaleTopicsForProducersFromEM(

DataSources

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

ExternalDataSources.topicFollowGraphSource,

ExternalDataSources.userSource,

ExternalDataSources.inferredUserConsumedLanguageSource,

minActiveFollowersForProducer,

minTopicFollowsThreshold,

lambda,

numEMSteps

)

val topLocaleTopicsForProducersFromEMKeyValExec =

sortAndGetTopLocaleTopicsPerProducer(

topicsFollowedByProducersFollowers,

maxTopicsPerProducer

).map {

case ((producerId, languageOpt, countryOpt), topicsWithScores) =>

KeyVal(

UserIdWithLocale(

userId = producerId,

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

SemanticCoreEntityScoreList(topicsWithScores.map {

case (topicid, topicScore) =>

SemanticEntityScore(SemanticCoreEntity(entityId = topicid), score = topicScore)

})

)

}.writeDALVersionedKeyValExecution(

TopLocaleTopicsForProducerFromEmScalaDataset,

D.Suffix(topTopicsPerProducerFromEMPath),

version = ExplicitEndTime(dateRange.end)

)

topLocaleTopicsForProducersFromEMKeyValExec

}

}

object TopicsForProducersFromEM {

private val MinProducerTopicScoreThreshold = 0.0

implicit val sparseMatrixInj: Injection[

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

Array[Byte]

] =

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

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

// truncated top locale topics ranked list for each producer

def sortAndGetTopLocaleTopicsPerProducer(

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

maxTopicsPerProducerPerLocale: Int

)(

implicit uniqueID: UniqueID

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

val numProducersWithLocales = Stat("num\_producers\_with\_locales")

producerToTopics

.map {

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

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

}.sumByKey.mapValues { topicsList: Seq[(TopicId, Double)] =>

numProducersWithLocales.inc()

topicsList

.filter(\_.\_2 >= MinProducerTopicScoreThreshold).sortBy(-\_.\_2).take(

maxTopicsPerProducerPerLocale).toList

}.toTypedPipe

}

def getTopLocaleTopicsForProducersFromEM(

userUserGraph: TypedPipe[UserAndNeighbors],

followedTopicsToUsers: TypedPipe[(TopicId, UserId)],

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

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

minActiveFollowersForProducer: Int,

minTopicFollowsThreshold: Int,

lambda: Double,

numEMSteps: Int

)(

implicit dateRange: DateRange,

timeZone: TimeZone,

uniqueID: UniqueID

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

// Obtain Producer To Users Matrix

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

TopicsForProducersUtils.getProducersToFollowedByUsersSparseMatrix(

userUserGraph,

minActiveFollowersForProducer)

// Obtain Users to TopicsWithLocales Matrix

val topicToUsersMatrix: SparseMatrix[

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

UserId,

Double

] = TopicsForProducersUtils.getFollowedTopicsToUserSparseMatrix(

followedTopicsToUsers,

userSource,

userLanguages,

minTopicFollowsThreshold)

// Domain input probability distribution is the Map(topics->followers) per producer locale

val domainInputModel = producersToUsersMatrix

.multiplySparseMatrix(topicToUsersMatrix.transpose).toTypedPipe.map {

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

((producerId, languageOpt, countryOpt), Map(topicId -> dotProduct))

}.sumByKey.toTypedPipe.map {

case ((producerId, languageOpt, countryOpt), topicsDomainInputMap) =>

((languageOpt, countryOpt), (producerId, topicsDomainInputMap))

}

// BackgroundModel is the Map(topics -> Expected value of the number of users who follow the topic)

val backgroundModel = topicToUsersMatrix.rowL1Norms.map {

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

((languageOpt, countryOpt), Map(topicId -> numFollowersOfTopic))

}.sumByKey

val resultsFromEMForEachLocale = domainInputModel.hashJoin(backgroundModel).flatMap {

case (

(languageOpt, countryOpt),

((producerId, domainInputTopicFollowersMap), backgroundModelTopicFollowersMap)) =>

val emScoredTopicsForEachProducerPerLocale = EMRunner.estimateDomainModel(

domainInputTopicFollowersMap,

backgroundModelTopicFollowersMap,

lambda,

numEMSteps)

emScoredTopicsForEachProducerPerLocale.map {

case (topicId, topicScore) =>

(producerId, (topicId, languageOpt, countryOpt), topicScore)

}

}

resultsFromEMForEachLocale

}

}