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

import com.twitter.escherbird.scalding.source.FullMetadataSource

import com.twitter.interests\_ds.jobs.interests\_service.UserTopicRelationSnapshotScalaDataset

import com.twitter.interests.thriftscala.InterestRelationType

import com.twitter.interests.thriftscala.UserInterestsRelationSnapshot

import com.twitter.recos.entities.thriftscala.SemanticCoreEntity

import com.twitter.recos.entities.thriftscala.SemanticCoreEntityScoreList

import com.twitter.recos.entities.thriftscala.SemanticEntityScore

import com.twitter.scalding.\_

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

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

import com.twitter.scalding\_internal.dalv2.remote\_access.ExplicitLocation

import com.twitter.scalding\_internal.dalv2.remote\_access.ProcAtla

import com.twitter.scalding\_internal.multiformat.format.keyval.KeyVal

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

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

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

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

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 similarities between topics based on how often they are co-followed

\* by users.

\*

\* Similarity(i, j) = #co-follow(i,j) / sqrt(#follow(i) \* #follow(j))

\*

\* It works as follows:

\*

\* 1. it first reads the data set of user to topics follow graph, and construct a sparse matrix M with

\* N rows and K columns, where N is the number of users, and K is the number of topics.

\* In the matrix, M(u,i) = 1 if user u follows topic i; otherwise it is 0. In the sparse matrix,

\* we only save non-zero elements.

\*

\* 2. we do l2-normalization for each column of the matrix M, to get a normalized version M'.

\*

\* 3. we get topic-topic similarity matrix S = M'.transpose.multiply(M'). The resulting matrix will

\* contain the similarities between all topics, i.e., S(i,j) is the similarity we mentioned above.

\*

\* 4. for each topic, we only keep its K similar topics with largest similarity scores, while not

\* including those with scores lower than a threshold.

\*

\*/

/\*\*

\* capesospy-v2 update --build\_locally \

\* --start\_cron similar\_topics\_from\_topic\_follow\_graph \

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

\*/

object SimilarTopicsFromTopicFollowGraphScheduledApp extends ScheduledExecutionApp {

import SimilarTopics.\_

private val outputPath: String =

"/user/cassowary/manhattan\_sequence\_files/similar\_topics\_from\_topics\_follow\_graph"

override def firstTime: RichDate = RichDate("2020-05-07")

override def batchIncrement: Duration = Days(7)

override def runOnDateRange(

args: Args

)(

implicit dateRange: DateRange,

timeZone: TimeZone,

uniqueID: UniqueID

): Execution[Unit] = {

val numSimilarTopics = args.int("numSimilarTopics", default = 100)

val scoreThreshold = args.double("scoreThreshold", default = 0.01)

val numOutputTopics = Stat("NumOutputTopics")

computeSimilarTopics(

getExplicitFollowedTopics,

getFollowableTopics,

numSimilarTopics,

scoreThreshold)

.map {

case (topicId, similarTopics) =>

numOutputTopics.inc()

KeyVal(

topicId,

SemanticCoreEntityScoreList(similarTopics.map {

case (similarTopicId, score) =>

SemanticEntityScore(SemanticCoreEntity(similarTopicId), score)

}))

}

.writeDALVersionedKeyValExecution(

SimilarTopicsFromTopicFollowGraphScalaDataset,

D.Suffix(outputPath),

version = ExplicitEndTime(dateRange.end)

)

}

}

/\*\*

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

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

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

--submitter hadoopnest1.atla.twitter.com \

-- --date 2020-04-28

\*/

object SimilarTopicsFromTopicFollowGraphAdhocApp extends AdhocExecutionApp {

import SimilarTopics.\_

override def runOnDateRange(

args: Args

)(

implicit dateRange: DateRange,

timeZone: TimeZone,

uniqueID: UniqueID

): Execution[Unit] = {

val numSimilarTopics = args.int("numSimilarTopics", default = 100)

val scoreThreshold = args.double("scoreThreshold", default = 0.01)

val numOutputTopics = Stat("NumOutputTopics")

computeSimilarTopics(

getExplicitFollowedTopics,

getFollowableTopics,

numSimilarTopics,

scoreThreshold)

.map {

case (topicId, similarTopics) =>

numOutputTopics.inc()

topicId -> similarTopics

.collect {

case (similarTopic, score) if similarTopic != topicId =>

s"$similarTopic:$score"

}

.mkString(",")

}

.writeExecution(

TypedTsv("/user/cassowary/adhoc/topic\_recos/similar\_topics")

)

}

}

object SimilarTopics {

val UTTDomain: Long = 131L

val FollowableTag: String = "utt:followable\_topic"

def getFollowableTopics(

implicit dateRange: DateRange,

timeZone: TimeZone,

uniqueID: UniqueID

): TypedPipe[SemanticCoreEntityId] = {

val NumFollowableTopics = Stat("NumFollowableTopics")

TypedPipe

.from(

new FullMetadataSource("/atla/proc" + FullMetadataSource.DefaultHdfsPath)()(

dateRange.embiggen(Days(7))))

.flatMap {

case fullMetadata if fullMetadata.domainId == UTTDomain =>

for {

basicMetadata <- fullMetadata.basicMetadata

indexableFields <- basicMetadata.indexableFields

tags <- indexableFields.tags

if tags.contains(FollowableTag)

} yield {

NumFollowableTopics.inc()

fullMetadata.entityId

}

case \_ => None

}

.forceToDisk

}

def getExplicitFollowedTopics(

implicit dateRange: DateRange,

timeZone: TimeZone,

uniqueID: UniqueID

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

DAL

.readMostRecentSnapshotNoOlderThan(UserTopicRelationSnapshotScalaDataset, Days(7))

.withRemoteReadPolicy(ExplicitLocation(ProcAtla))

.toTypedPipe

.collect {

case userInterestsRelationSnapshot: UserInterestsRelationSnapshot

if userInterestsRelationSnapshot.interestType == "UTT" &&

userInterestsRelationSnapshot.relation == InterestRelationType.Followed =>

(

userInterestsRelationSnapshot.userId,

Map(userInterestsRelationSnapshot.interestId -> 1.0))

}

.sumByKey

}

def computeSimilarTopics(

userTopicsFollowGraph: TypedPipe[(UserId, Map[SemanticCoreEntityId, Double])],

followableTopics: TypedPipe[SemanticCoreEntityId],

numSimilarTopics: Int,

scoreThreshold: Double

): TypedPipe[(SemanticCoreEntityId, Seq[(SemanticCoreEntityId, Double)])] = {

val userTopicFollowGraph =

SparseRowMatrix[UserId, SemanticCoreEntityId, Double](

userTopicsFollowGraph,

isSkinnyMatrix = true)

.filterCols(followableTopics) // filter out unfollowable topics

.colL2Normalize // normalization

// due to the small number of the topics,

// Scalding only allocates 1-2 mappers for the next step which makes it take unnecessarily long time.

// Changing it to 10 to make it a bit faster

.forceToDisk(numShardsOpt = Some(10))

userTopicFollowGraph

.transposeAndMultiplySkinnySparseRowMatrix(userTopicFollowGraph)

.filter { (i, j, v) =>

// exclude topic itself from being considered as similar; also the similarity score should

// be larger than a threshold

i != j && v > scoreThreshold

}

.sortWithTakePerRow(numSimilarTopics)(Ordering.by(-\_.\_2))

}

}