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

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

import com.twitter.dal.client.dataset.KeyValDALDataset

import com.twitter.scalding.\_

import com.twitter.scalding\_internal.dalv2.DALWrite.{D, \_}

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

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

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

import com.twitter.simclusters\_v2.scalding.common.matrix.{SparseMatrix, SparseRowMatrix}

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

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

EmbeddingUtil,

ExternalDataSources,

SimClustersEmbeddingBaseJob

}

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

EmbeddingType,

InternalId,

ModelVersion,

SimClustersEmbeddingId,

UserToInterestedInClusterScores,

SimClustersEmbedding => ThriftSimClustersEmbedding,

TopicId => ThriftTopicId

}

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

import java.util.TimeZone

/\*\*

\* Base app to generate Topic-Follow-Graph (TFG) topic embeddings from inferred languages.

\* In this app, topic embeddings are keyed by (topic, language, country).

\* Given a (topic t, country c, language l) tuple, the embedding is the sum of the

\* InterestedIn of the topic followers whose inferred language has l and account country is c

\* The language and the country fields in the keys are optional.

\* The app will generate 1) country-language-based 2) language-based 3) global embeddings in one dataset.

\* It's up to the clients to decide which embeddings to use

\*/

trait InferredLanguageTfgBasedTopicEmbeddingsBaseApp

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

with DateRangeExecutionApp {

val isAdhoc: Boolean

val embeddingType: EmbeddingType

val embeddingSource: KeyValDALDataset[KeyVal[SimClustersEmbeddingId, ThriftSimClustersEmbedding]]

val pathSuffix: String

val modelVersion: ModelVersion

def scoreExtractor: UserToInterestedInClusterScores => Double

override def numClustersPerNoun: Int = 50

override def numNounsPerClusters: Int = 1 // not used for now. Set to an arbitrary number

override def thresholdForEmbeddingScores: Double = 0.001

implicit val inj: Injection[(TopicId, Option[Language], Option[Country]), Array[Byte]] =

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

// Default to 10K, top 1% for (topic, country, language) follows

// Child classes may want to tune this number for their own use cases.

val minPerCountryFollowers = 10000

val minFollowers = 100

def getTopicUsers(

topicFollowGraph: TypedPipe[(TopicId, UserId)],

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

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

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

topicFollowGraph

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

.join(userSource)

.join(userLanguages)

.flatMap {

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

scoredLangs.flatMap {

case (lang, score) =>

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)) // non-language

}.forceToDisk

}

def getValidTopics(

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

)(

implicit uniqueID: UniqueID

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

val countryBasedTopics = Stat("country\_based\_topics")

val nonCountryBasedTopics = Stat("non\_country\_based\_topics")

val (countryBased, nonCountryBased) = topicUsers.partition {

case ((\_, lang, country), \_, \_) => lang.isDefined && country.isDefined

}

SparseMatrix(countryBased).rowL1Norms.collect {

case (key, l1Norm) if l1Norm >= minPerCountryFollowers =>

countryBasedTopics.inc()

key

} ++

SparseMatrix(nonCountryBased).rowL1Norms.collect {

case (key, l1Norm) if l1Norm >= minFollowers =>

nonCountryBasedTopics.inc()

key

}

}

override def prepareNounToUserMatrix(

implicit dateRange: DateRange,

timeZone: TimeZone,

uniqueID: UniqueID

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

val topicUsers = getTopicUsers(

ExternalDataSources.topicFollowGraphSource,

ExternalDataSources.userSource,

ExternalDataSources.inferredUserConsumedLanguageSource)

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

.filterRows(getValidTopics(topicUsers))

}

override def prepareUserToClusterMatrix(

implicit dateRange: DateRange,

timeZone: TimeZone,

uniqueID: UniqueID

): SparseRowMatrix[UserId, ClusterId, Double] =

SparseRowMatrix(

InterestedInSources

.simClustersInterestedInSource(modelVersion, dateRange, timeZone)

.map {

case (userId, clustersUserIsInterestedIn) =>

userId -> clustersUserIsInterestedIn.clusterIdToScores

.map {

case (clusterId, scores) =>

clusterId -> scoreExtractor(scores)

}

.filter(\_.\_2 > 0.0)

.toMap

},

isSkinnyMatrix = true

)

override def writeNounToClustersIndex(

output: TypedPipe[((TopicId, Option[Language], Option[Country]), Seq[(ClusterId, Double)])]

)(

implicit dateRange: DateRange,

timeZone: TimeZone,

uniqueID: UniqueID

): Execution[Unit] = {

val topicEmbeddingCount = Stat(s"topic\_embedding\_count")

val tsvExec =

output

.map {

case ((entityId, language, country), clustersWithScores) =>

(entityId, language, country, clustersWithScores.take(5).mkString(","))

}

.shard(5)

.writeExecution(TypedTsv[(TopicId, Option[Language], Option[Country], String)](

s"/user/recos-platform/adhoc/topic\_embedding/$pathSuffix/$ModelVersionPathMap($modelVersion)"))

val keyValExec = output

.map {

case ((entityId, lang, country), clustersWithScores) =>

topicEmbeddingCount.inc()

KeyVal(

SimClustersEmbeddingId(

embeddingType,

modelVersion,

InternalId.TopicId(ThriftTopicId(entityId, lang, country))

),

SimClustersEmbedding(clustersWithScores).toThrift

)

}

.writeDALVersionedKeyValExecution(

embeddingSource,

D.Suffix(

EmbeddingUtil

.getHdfsPath(isAdhoc = isAdhoc, isManhattanKeyVal = true, modelVersion, pathSuffix))

)

if (isAdhoc)

Execution.zip(tsvExec, keyValExec).unit

else

keyValExec

}

override def writeClusterToNounsIndex(

output: TypedPipe[(ClusterId, Seq[((TopicId, Option[Language], Option[Country]), Double)])]

)(

implicit dateRange: DateRange,

timeZone: TimeZone,

uniqueID: UniqueID

): Execution[Unit] = {

Execution.unit // do not need this

}

}