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

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

import com.twitter.dal.client.dataset.{KeyValDALDataset, SnapshotDALDatasetBase}

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.{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.{

ClustersScore,

EmbeddingType,

TfgTopicEmbeddings,

InternalId,

LocaleEntityId,

ModelVersion,

SimClustersEmbeddingId,

UserToInterestedInClusterScores,

SimClustersEmbedding => ThriftSimClustersEmbedding,

TopicId => TID

}

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

import java.util.TimeZone

/\*\*

\* Base app for the Topic-Follow-Graph (TFG) topic embeddings

\* A topic's TFG embedding is represented by the sum of all the users who followed the topic

\*/

trait TfgBasedTopicEmbeddingsBaseApp

extends SimClustersEmbeddingBaseJob[(TopicId, Language)]

with DateRangeExecutionApp {

val isAdhoc: Boolean

val embeddingType: EmbeddingType

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

val pathSuffix: String

val modelVersion: ModelVersion

val parquetDataSource: SnapshotDALDatasetBase[TfgTopicEmbeddings]

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

val minNumFollowers = 100

override def prepareNounToUserMatrix(

implicit dateRange: DateRange,

timeZone: TimeZone,

uniqueID: UniqueID

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

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

Bufferable.injectionOf[(TopicId, Language)]

val topicLangUsers = ExternalDataSources.topicFollowGraphSource

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

.join(ExternalDataSources.userSource)

.map {

case (user, (topic, (\_, language))) =>

((topic, language), user, 1.0)

}

.forceToDisk

val validTopicLang =

SparseMatrix(topicLangUsers).rowNnz.filter {

case (\_, nzCount) => nzCount >= minNumFollowers

}.keys

SparseMatrix[(TopicId, Language), UserId, Double](topicLangUsers).filterRows(validTopicLang)

}

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, Language), Seq[(ClusterId, Double)])]

)(

implicit dateRange: DateRange,

timeZone: TimeZone,

uniqueID: UniqueID

): Execution[Unit] = {

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

val user = System.getenv("USER")

val parquetExec = output

.map {

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

TfgTopicEmbeddings(

TID(

entityId = entityId,

language = Some(language),

),

clusterScore = clustersWithScores.map {

case (clusterId, score) => ClustersScore(clusterId, score)

}

)

}

.writeDALSnapshotExecution(

parquetDataSource,

D.Daily,

D.Suffix(

EmbeddingUtil.getHdfsPath(

isAdhoc = isAdhoc,

isManhattanKeyVal = false,

modelVersion,

pathSuffix + "/snapshot")),

D.Parquet,

dateRange.end

)

val tsvExec =

output

.map {

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

(entityId, language, clustersWithScores.mkString(";"))

}

.shard(10)

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

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

val keyValExec = output

.flatMap {

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

topicEmbeddingCount.inc()

val embedding = SimClustersEmbedding(clustersWithScores).toThrift

Seq(

KeyVal(

SimClustersEmbeddingId(

embeddingType,

modelVersion,

InternalId.LocaleEntityId(LocaleEntityId(entityId, lang))

),

embedding

),

KeyVal(

SimClustersEmbeddingId(

embeddingType,

modelVersion,

InternalId.TopicId(TID(entityId, Some(lang), country = None))

),

embedding

),

)

}

.writeDALVersionedKeyValExecution(

embeddingSource,

D.Suffix(

EmbeddingUtil

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

)

if (isAdhoc)

Execution.zip(tsvExec, keyValExec, parquetExec).unit

else

Execution.zip(keyValExec, parquetExec).unit

}

override def writeClusterToNounsIndex(

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

)(

implicit dateRange: DateRange,

timeZone: TimeZone,

uniqueID: UniqueID

): Execution[Unit] = {

Execution.unit // do not need this

}

}