package com.twitter.simclusters\_v2.scalding.offline\_job.adhoc

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

import com.twitter.scalding.\_

import com.twitter.scalding.commons.source.VersionedKeyValSource

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

import com.twitter.scalding\_internal.dalv2.remote\_access.{ExplicitLocation, ProcAtla}

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

import com.twitter.simclusters\_v2.common.{ClusterId, TweetId, UserId}

import com.twitter.simclusters\_v2.hdfs\_sources.SimclustersV2InterestedIn20M145KUpdatedScalaDataset

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

import com.twitter.simclusters\_v2.scalding.offline\_job.SimClustersOfflineJobUtil

import com.twitter.simclusters\_v2.summingbird.common.{Configs, SimClustersInterestedInUtil}

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

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

import java.util.TimeZone

/\*\*

\* Adhoc job for computing Tweet SimClusters embeddings.

\* The output of this job includes two data sets: tweet -> top clusters (or Tweet Embedding), and cluster -> top tweets.

\* These data sets are supposed to be the snapshot of the two index at the end of the dataRange you run.

\*

\* Note that you can also use the output from SimClustersOfflineJobScheduledApp for analysis purpose.

\* The outputs from that job might be more close to the data we use in production.

\* The benefit of having this job is to keep the flexibility of experiment different ideas.

\*

\* It is recommended to put at least 2 days in the --date (dataRange in the code) in order to make sure

\* we have enough engagement data for tweets have more engagements in the last 1+ days.

\*

\*

\* There are several parameters to tune in the job. They are explained in the inline comments.

\*

\*

\* To run the job:

scalding remote run \

--target src/scala/com/twitter/simclusters\_v2/scalding/offline\_job/adhoc:tweet\_embedding-adhoc \

--user recos-platform \

--reducers 1000 \

--main-class com.twitter.simclusters\_v2.scalding.offline\_job.adhoc.SimClustersTweetEmbeddingAdhocApp -- \

--date 2021-01-27 2021-01-28 \

--score\_type logFav \

--output\_dir /user/recos-platform/adhoc/tweet\_embedding\_01\_27\_28\_unnormalized\_t9

\*/

object SimClustersTweetEmbeddingAdhocApp extends AdhocExecutionApp {

import SimClustersOfflineJobUtil.\_

override def runOnDateRange(

args: Args

)(

implicit dateRange: DateRange,

timeZone: TimeZone,

uniqueID: UniqueID

): Execution[Unit] = {

val outputDir = args("output\_dir")

// what interestedIn score to use. logFav is what we use in production

val scoringMethod = args.getOrElse("score\_type", "logFav")

// whether to use normalized score in the cluster -> top tweets.

// Currently, we do not do this in production. DONOT turn it on unless you know what you are doing.

// NOTE that for scalding args, "--run\_normalized" will just set the arg to be true, and

// even you use "--run\_normalized false", it will still be true.

val usingNormalizedScoringFunction = args.boolean("run\_normalized")

// filter out tweets that has less than X favs in the dateRange.

val tweetFavThreshold = args.long("tweet\_fav\_threshold", 0L)

// tweet -> top clusters will be saved in this subfolder

val tweetTopKClustersOutputPath: String = outputDir + "/tweet\_top\_k\_clusters"

// cluster -> top tweets will be saved in this subfolder

val clusterTopKTweetsOutputPath: String = outputDir + "/cluster\_top\_k\_tweets"

val interestedInData: TypedPipe[(Long, ClustersUserIsInterestedIn)] =

DAL

.readMostRecentSnapshot(

SimclustersV2InterestedIn20M145KUpdatedScalaDataset,

dateRange.embiggen(Days(14))

)

.withRemoteReadPolicy(ExplicitLocation(ProcAtla))

.toTypedPipe

.map {

case KeyVal(key, value) => (key, value)

}

// read user-tweet fav data. set the weight to be a decayed value. they will be decayed to the dateRang.end

val userTweetFavData: SparseMatrix[UserId, TweetId, Double] =

SparseMatrix(readTimelineFavoriteData(dateRange)).tripleApply {

case (userId, tweetId, timestamp) =>

(

userId,

tweetId,

thriftDecayedValueMonoid

.plus(

thriftDecayedValueMonoid.build(1.0, timestamp),

thriftDecayedValueMonoid.build(0.0, dateRange.end.timestamp)

)

.value)

}

// filter out tweets without x favs

val tweetSubset =

userTweetFavData.colNnz.filter(

\_.\_2 > tweetFavThreshold.toDouble

) // keep tweets with at least x favs

val userTweetFavDataSubset = userTweetFavData.filterCols(tweetSubset.keys)

// construct user-simclusters matrix

val userSimClustersInterestedInData: SparseRowMatrix[UserId, ClusterId, Double] =

SparseRowMatrix(

interestedInData.map {

case (userId, clusters) =>

val topClustersWithScores =

SimClustersInterestedInUtil

.topClustersWithScores(clusters)

.collect {

case (clusterId, scores)

if scores.favScore > Configs

.favScoreThresholdForUserInterest(

clusters.knownForModelVersion

) => // this is the same threshold used in the summingbird job

scoringMethod match {

case "fav" =>

clusterId -> scores.clusterNormalizedFavScore

case "follow" =>

clusterId -> scores.clusterNormalizedFollowScore

case "logFav" =>

clusterId -> scores.clusterNormalizedLogFavScore

case \_ =>

throw new IllegalArgumentException(

"score\_type can only be fav, follow or logFav")

}

}

.filter(\_.\_2 > 0.0)

.toMap

userId -> topClustersWithScores

},

isSkinnyMatrix = true

)

// multiply tweet -> user matrix with user -> cluster matrix to get tweet -> cluster matrix

val tweetClusterScoreMatrix = if (usingNormalizedScoringFunction) {

userTweetFavDataSubset.transpose.rowL2Normalize

.multiplySkinnySparseRowMatrix(userSimClustersInterestedInData)

} else {

userTweetFavDataSubset.transpose.multiplySkinnySparseRowMatrix(

userSimClustersInterestedInData)

}

// get the tweet -> top clusters by taking top K in each row

val tweetTopClusters = tweetClusterScoreMatrix

.sortWithTakePerRow(Configs.topKClustersPerTweet)(Ordering.by(-\_.\_2))

.fork

// get the cluster -> top tweets by taking top K in each colum

val clusterTopTweets = tweetClusterScoreMatrix

.sortWithTakePerCol(Configs.topKTweetsPerCluster)(Ordering.by(-\_.\_2))

.fork

// injections for saving a list

implicit val inj1: Injection[List[(Int, Double)], Array[Byte]] =

Bufferable.injectionOf[List[(Int, Double)]]

implicit val inj2: Injection[List[(Long, Double)], Array[Byte]] =

Bufferable.injectionOf[List[(Long, Double)]]

// save the data sets and also output to some tsv files for eyeballing the results

Execution

.zip(

tweetTopClusters

.mapValues(\_.toList)

.writeExecution(

VersionedKeyValSource[TweetId, List[(ClusterId, Double)]](tweetTopKClustersOutputPath)

),

tweetTopClusters

.map {

case (tweetId, topKClusters) =>

tweetId -> topKClusters

.map {

case (clusterId, score) =>

s"$clusterId:" + "%.3g".format(score)

}

.mkString(",")

}

.writeExecution(

TypedTsv(tweetTopKClustersOutputPath + "\_tsv")

),

tweetSubset.writeExecution(TypedTsv(tweetTopKClustersOutputPath + "\_tweet\_favs")),

clusterTopTweets

.mapValues(\_.toList)

.writeExecution(

VersionedKeyValSource[ClusterId, List[(TweetId, Double)]](clusterTopKTweetsOutputPath)

),

clusterTopTweets

.map {

case (clusterId, topKTweets) =>

clusterId -> topKTweets

.map {

case (tweetId, score) => s"$tweetId:" + "%.3g".format(score)

}

.mkString(",")

}

.writeExecution(

TypedTsv(clusterTopKTweetsOutputPath + "\_tsv")

)

)

.unit

}

}