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

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

import com.twitter.dal.client.dataset.SnapshotDALDataset

import com.twitter.scalding.DateRange

import com.twitter.scalding.Days

import com.twitter.scalding.UniqueID

import com.twitter.scalding.\_

import com.twitter.scalding.typed.TypedPipe

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

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

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

import com.twitter.scalding\_internal.job.RequiredBinaryComparators.ordSer

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.Timestamp

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

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

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

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

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

import com.twitter.simclusters\_v2.thriftscala.InternalId.ClusterId

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

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

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

import com.twitter.simclusters\_v2.hdfs\_sources.SimclustersV2GlobalLanguageEmbeddingScalaDataset

import com.twitter.simclusters\_v2.hdfs\_sources.SimclustersV2GlobalLanguageEmbeddingThriftScalaDataset

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

import java.util.TimeZone

/\*\*

capesospy-v2 update --build\_locally --start\_cron \

--start\_cron global\_simclusters\_language\_embedding\_job \

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

\*/

object GlobalSimClustersLanguageEmbeddingBatchApp extends ScheduledExecutionApp {

override val firstTime: RichDate = RichDate("2023-03-07")

override val batchIncrement: Duration = Days(1)

val outputHdfsDirectory =

"/user/cassowary/manhattan\_sequence\_files/global\_simclusters\_language\_embeddings"

val outputThriftHdfsDirectory =

"/user/cassowary/processed/global\_simclusters\_language\_embeddings"

val globalLanguageEmbeddingsKeyValDataset: KeyValDALDataset[

KeyVal[String, ClustersUserIsInterestedIn]

] = SimclustersV2GlobalLanguageEmbeddingScalaDataset

val globalLanguageEmbeddingsThriftDataset: SnapshotDALDataset[LanguageToClusters] =

SimclustersV2GlobalLanguageEmbeddingThriftScalaDataset

val numOfClustersPerLanguage: Int = 400

def getInterestedInFn: (

DateRange,

TimeZone

) => TypedPipe[(UserId, ClustersUserIsInterestedIn)] =

InterestedInSources.simClustersInterestedIn2020Source

def flattenAndFilterUserInterestedIn(

interestedIn: TypedPipe[(UserId, ClustersUserIsInterestedIn)]

): TypedPipe[(UserId, (Int, Double))] = {

interestedIn

// Get (userId, Seq[(clusterId, scores)]

.map {

case (user, clusterUserIsInterestedIn) => {

(user, clusterUserIsInterestedIn.clusterIdToScores)

}

}

// Flatten it into (UserId, ClusterId, LogFavScore)

.flatMap {

case (userId, clusterUserIsInterestedIn) => {

clusterUserIsInterestedIn.toSeq.map {

case (clusterId, scores) => {

(userId, (clusterId, scores.logFavScore.getOrElse(0.0)))

}

}

}

}.filter(\_.\_2.\_2 > 0.0) // Filter out zero scores

}

def getGlobalSimClustersEmbeddingPerLanguage(

interestedIn: TypedPipe[(UserId, (Int, Double))],

favEdges: TypedPipe[(UserId, TweetId, Timestamp)],

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

): TypedPipe[(Language, ClustersUserIsInterestedIn)] = {

// Engagement fav edges

val edges = favEdges.map { case (userId, tweetId, ts) => (userId, (tweetId, ts)) }

// Language information for users

val userLanguage = language.map {

case (userId, (country, lang)) => (userId, lang)

}

val numUsersPerLanguage = userLanguage.map {

case (\_, lang) => (lang, 1L)

}.sumByKey

val embeddings =

interestedIn

.join(edges) // Join InterestedIn and user-tweet engagements

.map {

case (userId, ((clusterId, score), (\_, \_))) => {

(userId, (clusterId, score))

}

}

.join(userLanguage) // Join and get cluster scores per language

.map {

case (userId, ((clusterId, score), lang)) => {

((lang, clusterId), score)

}

}

.sumByKey // Sum the user embeddings per language based on the engagements

.map { case ((lang, clusterId), score) => (lang, (clusterId, score)) }

.join(numUsersPerLanguage)

// We compute the average cluster scores per language

.map {

case (lang, ((clusterId, score), count)) => (lang, (clusterId -> score / count))

}

.group

.sortedReverseTake(numOfClustersPerLanguage)(Ordering

.by(\_.\_2)) // Take top 400 clusters per language

.flatMap {

case (lang, clusterScores) => {

clusterScores.map {

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

}

}

}.mapValues { case (clusterId, score) => Map(clusterId -> score) }

// Build the final SimClusters embeddings per language

embeddings.sumByKey.map {

case (lang, clusterToScore) => {

val clusterScores = clusterToScore.map {

case (clusterId, score) =>

clusterId -> UserToInterestedInClusterScores(logFavScore = Some(score))

}

(lang, ClustersUserIsInterestedIn(ModelVersion.Model20m145k2020.name, clusterScores))

}

}

}

override def runOnDateRange(

args: Args

)(

implicit dateRange: DateRange,

timeZone: TimeZone,

uniqueID: UniqueID

): Execution[Unit] = {

// Read the most recent InterestedIn snapshot from the past 21 days

val interestedIn =

InterestedInSources

.simClustersInterestedIn2020Source(dateRange.prepend(Days(21)), timeZone).forceToDisk

// Get the user tweet fav engagement history from the past 2 days

val userTweetFavEdges = ExternalDataSources.userTweetFavoritesSource

// Read user language from UserSource

val userLanguages = ExternalDataSources.userSource

val globalEmbeddings = getGlobalSimClustersEmbeddingPerLanguage(

flattenAndFilterUserInterestedIn(interestedIn),

userTweetFavEdges,

userLanguages)

// Write results as a key-val dataset

globalEmbeddings

.map {

case (lang, embeddings) =>

KeyVal(lang, embeddings)

}

.writeDALVersionedKeyValExecution(

globalLanguageEmbeddingsKeyValDataset,

D.Suffix(outputHdfsDirectory)

)

// Write results as a thrift dataset

globalEmbeddings

.map {

case (lang, clusterUserIsInterestedIn) =>

LanguageToClusters(

lang,

clusterUserIsInterestedIn.knownForModelVersion,

clusterUserIsInterestedIn.clusterIdToScores

)

}

.writeDALSnapshotExecution(

globalLanguageEmbeddingsThriftDataset,

D.Daily,

D.Suffix(outputThriftHdfsDirectory),

D.Parquet,

dateRange.`end`

)

}

}