package com.twitter.ann.scalding.offline

import com.twitter.core\_workflows.user\_model.thriftscala.CondensedUserState

import com.twitter.cortex.ml.embeddings.common.{DataSourceManager, GraphEdge, Helpers, UserKind}

import com.twitter.ml.featurestore.lib.UserId

import com.twitter.entityembeddings.neighbors.thriftscala.{EntityKey, NearestNeighbors}

import com.twitter.pluck.source.core\_workflows.user\_model.CondensedUserStateScalaDataset

import com.twitter.scalding.\_

import com.twitter.scalding.typed.TypedPipe

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

import com.twitter.usersource.snapshot.flat.UsersourceFlatScalaDataset

import com.twitter.usersource.snapshot.flat.thriftscala.FlatUser

case class ConsumerAssoc(consumerId: UserId, assoc: List[String])

object KnnDebug {

def getConsumerAssociations(

graph: TypedPipe[GraphEdge[UserId, UserId]],

usernames: TypedPipe[(UserId, String)],

reducers: Int

): TypedPipe[ConsumerAssoc] = {

graph

.groupBy(\_.itemId)

.join(usernames).withReducers(reducers)

.values

.map {

case (edge: GraphEdge[UserId, UserId], producerScreenName: String) =>

ConsumerAssoc(consumerId = edge.consumerId, assoc = List(producerScreenName))

}

.groupBy(\_.consumerId).withReducers(reducers)

.reduce[ConsumerAssoc] {

case (uFollow1: ConsumerAssoc, uFollow2: ConsumerAssoc) =>

ConsumerAssoc(consumerId = uFollow1.consumerId, assoc = uFollow1.assoc ++ uFollow2.assoc)

}

.values

}

/\*\*

\* Write the neighbors and a set of follows to a tsv for easier analysis during debugging

\* We take the set of users with between 25-50 follows and grab only those users

\*

\* This returns 4 strings of the form:

\* consumerId, state, followUserName<f>followUserName<f>followUserName, neighborName<n>neighborName<n>neighborName

\*/

def getDebugTable(

neighborsPipe: TypedPipe[(EntityKey, NearestNeighbors)],

shards: Int,

reducers: Int,

limit: Int = 10000,

userDataset: Option[TypedPipe[FlatUser]] = None,

followDataset: Option[TypedPipe[GraphEdge[UserId, UserId]]] = None,

consumerStatesDataset: Option[TypedPipe[CondensedUserState]] = None,

minFollows: Int = 25,

maxFollows: Int = 50

)(

implicit dateRange: DateRange

): TypedPipe[(String, String, String, String)] = {

val usersourcePipe: TypedPipe[FlatUser] = userDataset

.getOrElse(DAL.readMostRecentSnapshot(UsersourceFlatScalaDataset, dateRange).toTypedPipe)

val followGraph: TypedPipe[GraphEdge[UserId, UserId]] = followDataset

.getOrElse(new DataSourceManager().getFollowGraph())

val consumerStates: TypedPipe[CondensedUserState] = consumerStatesDataset

.getOrElse(DAL.read(CondensedUserStateScalaDataset).toTypedPipe)

val usernames: TypedPipe[(UserId, String)] = usersourcePipe.flatMap { flatUser =>

(flatUser.screenName, flatUser.id) match {

case (Some(name: String), Some(userId: Long)) => Some((UserId(userId), name))

case \_ => None

}

}.fork

val consumerFollows: TypedPipe[ConsumerAssoc] =

getConsumerAssociations(followGraph, usernames, reducers)

.filter { uFollow => (uFollow.assoc.size > minFollows && uFollow.assoc.size < maxFollows) }

val neighborGraph: TypedPipe[GraphEdge[UserId, UserId]] = neighborsPipe

.limit(limit)

.flatMap {

case (entityKey: EntityKey, neighbors: NearestNeighbors) =>

Helpers.optionalToLong(entityKey.id) match {

case Some(entityId: Long) =>

neighbors.neighbors.flatMap { neighbor =>

Helpers

.optionalToLong(neighbor.neighbor.id)

.map { neighborId =>

GraphEdge[UserId, UserId](

consumerId = UserId(entityId),

itemId = UserId(neighborId),

weight = 1.0F)

}

}

case None => List()

}

}

val consumerNeighbors: TypedPipe[ConsumerAssoc] =

getConsumerAssociations(neighborGraph, usernames, reducers)

consumerFollows

.groupBy(\_.consumerId)

.join(consumerStates.groupBy { consumer => UserId(consumer.uid) }).withReducers(reducers)

.join(consumerNeighbors.groupBy(\_.consumerId)).withReducers(reducers)

.values

.map {

case ((uFollow: ConsumerAssoc, state: CondensedUserState), uNeighbors: ConsumerAssoc) =>

(

UserKind.stringInjection(uFollow.consumerId),

state.state.toString,

uFollow.assoc mkString "<f>",

uNeighbors.assoc mkString "<n>")

}

.shard(shards)

}

}