package com.twitter.simclusters\_v2.scalding.tweet\_similarity.evaluation

import com.twitter.rux.landing\_page.data\_pipeline.LabeledRuxServiceScribeScalaDataset

import com.twitter.rux.landing\_page.data\_pipeline.thriftscala.LandingPageLabel

import com.twitter.rux.service.thriftscala.FocalObject

import com.twitter.rux.service.thriftscala.UserContext

import com.twitter.scalding.\_

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

import com.twitter.scalding\_internal.job.TwitterExecutionApp

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

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

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

import java.util.TimeZone

/\*\* To run:

scalding remote run --target src/scala/com/twitter/simclusters\_v2/scalding/tweet\_similarity/evaluation:rux\_landing\_ddg\_analysis-adhoc \

--user cassowary \

--submitter hadoopnest2.atla.twitter.com \

--main-class com.twitter.simclusters\_v2.scalding.tweet\_similarity.evaluation.RUXLandingDdgAnalysisAdhocApp -- \

--date 2020-04-06 2020-04-13 \

--ddg model\_based\_tweet\_similarity\_10254 \

--version 1 \

--output\_path /user/cassowary/adhoc/ddg10254

\* \*/

object RUXLandingDdgAnalysisAdhocApp extends TwitterExecutionApp {

override def job: Execution[Unit] =

Execution.withId { implicit uniqueId =>

Execution.withArgs { args: Args =>

implicit val timeZone: TimeZone = DateOps.UTC

implicit val dateParser: DateParser = DateParser.default

implicit val dateRange: DateRange = DateRange.parse(args.list("date"))

val ddgName: String = args("ddg")

val ddgVersion: String = args("version")

val outputPath: String = args("output\_path")

val now = RichDate.now

val ruxLabels = getLabeledRuxServiceScribe(dateRange).map {

case (userId, focalTweet, candidateTweet, impression, fav) =>

userId -> (focalTweet, candidateTweet, impression, fav)

}

// getUsersInDDG reads from a snapshot dataset.

// Just prepend dateRange so that we can look back far enough to make sure there is data.

DDGUtil

.getUsersInDDG(ddgName, ddgVersion.toInt)(DateRange(now - Days(7), now)).map { ddgUser =>

ddgUser.userId -> (ddgUser.bucket, ddgUser.enterUserState.getOrElse("no\_user\_state"))

}.join(ruxLabels)

.map {

case (userId, ((bucket, state), (focalTweet, candidateTweet, impression, fav))) =>

(userId, bucket, state, focalTweet, candidateTweet, impression, fav)

}

.writeExecution(

TypedTsv[(UserId, String, String, TweetId, TweetId, Int, Int)](s"$outputPath"))

}

}

def getLabeledRuxServiceScribe(

dateRange: DateRange

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

DAL

.read(LabeledRuxServiceScribeScalaDataset, dateRange)

.toTypedPipe.map { record =>

(

record.ruxServiceScribe.userContext,

record.ruxServiceScribe.focalObject,

record.landingPageLabel)

}.flatMap {

case (

Some(UserContext(Some(userId), \_, \_, \_, \_, \_, \_, \_)),

Some(FocalObject.TweetId(tweet)),

Some(labels)) =>

labels.map {

case LandingPageLabel.LandingPageFavoriteEvent(favEvent) =>

//(focal tweet, impressioned tweet, impression, fav)

(userId, tweet, favEvent.tweetId, 0, 1)

case LandingPageLabel.LandingPageImpressionEvent(impressionEvent) =>

(userId, tweet, impressionEvent.tweetId, 1, 0)

}

case \_ => Nil

}

}

}