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

import com.twitter.ads.dataservice\_account.snapshot.jobs.DbSnapshotsPromotedTweetsScalaDataset

import com.twitter.conversions.DurationOps.\_

import com.twitter.dal.client.dataset.TimePartitionedDALDataset

import com.twitter.scalding.\_

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

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

import com.twitter.scalding\_internal.dalv2.remote\_access.ExplicitLocation

import com.twitter.scalding\_internal.dalv2.remote\_access.ProcrevAtla

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

import com.twitter.simclusters\_v2.scalding.tweet\_similarity.TweetPairLabelCollectionUtil.MaxFavPerUser

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

import com.twitter.simclusters\_v2.thriftscala.{FeaturedTweet => FeaturedTweetThrift}

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

import java.util.TimeZone

/\*\*

\* Collect unhydrated training pairs for supervised tweet similarity.

\* Here're the steps for this job

\* 1) Consider non-promoted tweets that are created within the given #lookback days

\* 2) From the tweets in 1), get co-engaged pairs

\* 3) Take all tweets shown in 2), and get co-impressed pairs. Note that we take all tweets (not tweet pairs) in 2).

\* That is, a co-impressed pairs (t1,t2) will be considered iff t1 appears in 2) and t2 appears in 2).

\* But (t1, t2) doesn't need to appear as a pair in 2).

\* 4) Compute labels from co-engaged pairs and co-impressed pairs.

\* A pair is true if its user has co-engaged the pair, and is false if otherwise.

\*/

object UnhydratedPairsCollectionJob {

//tweets have to be created within dateRange - lookbackdays in order to be considered

val LookbackDays = 2

def getLabelledPairs(

dateRange: DateRange,

timeframe: Long,

maxSamplesPerClass: Int,

dalDataset: TimePartitionedDALDataset[LabelledTweetPairs],

outputPath: String

)(

implicit timeZone: TimeZone

): Execution[Unit] = {

val promotedTweets = DAL

.readMostRecentSnapshot(DbSnapshotsPromotedTweetsScalaDataset, dateRange)

.withRemoteReadPolicy(ExplicitLocation(ProcrevAtla))

.toTypedPipe

val tweetAuthorPairs =

TweetPairLabelCollectionUtil.getTweetAuthorPairs(dateRange.prepend(Days(LookbackDays)))

val tweets =

TweetPairLabelCollectionUtil.getNonPromotedTweets(promotedTweets, tweetAuthorPairs.keys)

val coengagedPairs = TweetPairLabelCollectionUtil.getCoengagedPairs(

TweetPairLabelCollectionUtil.getFavEvents(dateRange, MaxFavPerUser),

tweets,

timeframe)

val engagedTweets = coengagedPairs.map {

// Consider only query tweet b/c coengagedPairs contains both (t1,t2) and (t2,t1)

case (\_, queryFeaturedTweet, \_, \_) => queryFeaturedTweet.tweet

}.distinct

val coimpressedPairs = TweetPairLabelCollectionUtil

.getCoimpressedPairs(

TweetPairLabelCollectionUtil.getImpressionEvents(dateRange),

engagedTweets,

timeframe)

val rawLabelledPairs =

TweetPairLabelCollectionUtil.computeLabelledTweetPairs(coengagedPairs, coimpressedPairs)

val labelledPairs =

if (maxSamplesPerClass > 0)

TweetPairLabelCollectionUtil.getQueryTweetBalancedClassPairs(

rawLabelledPairs,

maxSamplesPerClass)

else

rawLabelledPairs

val perQueryStatsExec =

if (maxSamplesPerClass > 0) {

Execution

.zip(

TweetPairLabelCollectionUtil

.getPerQueryStatsExec(rawLabelledPairs, s"$outputPath/per\_query\_stats", "raw"),

TweetPairLabelCollectionUtil

.getPerQueryStatsExec(labelledPairs, s"$outputPath/per\_query\_stats", "final")

).unit

} else {

TweetPairLabelCollectionUtil.getPerQueryStatsExec(

labelledPairs,

s"$outputPath/per\_query\_stats",

"final")

}

Execution

.zip(

labelledPairs

.map {

case (queryFeaturedTweet, candidateFeaturedTweet, label) =>

LabelledTweetPairs(

FeaturedTweetThrift(

tweetId = queryFeaturedTweet.tweet,

timestamp = queryFeaturedTweet.timestamp),

FeaturedTweetThrift(

tweetId = candidateFeaturedTweet.tweet,

timestamp = candidateFeaturedTweet.timestamp),

label

)

}

.writeDALExecution(dalDataset, D.Daily, D.Suffix(outputPath), D.EBLzo())(dateRange),

perQueryStatsExec

).unit

}

}

/\*\* To run:

\* scalding remote run --target src/scala/com/twitter/simclusters\_v2/scalding/tweet\_similarity:unhydrated\_pair\_collection-adhoc \

--user cassowary \

--submitter hadoopnest2.atla.twitter.com \

--hadoop-properties "mapreduce.reduce.java.opts=-Xmx8000m mapreduce.reduce.memory.mb=8000 scalding.with.reducers.set.explicitly=true mapreduce.job.reduces=2000 mapreduce.task.timeout=0" \

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

--date 2020-03-04 \

--output\_path /user/cassowary/adhoc/unhydrated\_pairs/2020-03-04\_class\_balanced \

--samples\_per\_query\_tweet\_class 2000

\* \*/

object UnhydratedPairsCollectionAdhocApp extends TwitterExecutionApp {

implicit val timeZone: TimeZone = DateOps.UTC

implicit val dateParser: DateParser = DateParser.default

override def job: Execution[Unit] =

Execution.withId { implicit uniqueId =>

Execution.withArgs { args: Args =>

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

val maxSamplesPerClass: Int = args.int("samples\_per\_query\_tweet\_class", default = 2000)

val timeframe: Int = 30

val outputPath: String = s"${args("output\_path")}\_${timeframe}min"

UnhydratedPairsCollectionJob.getLabelledPairs(

dateRange,

timeframe.minute.inMilliseconds,

maxSamplesPerClass,

TweetSimilarityUnhydratedPairs30MinScalaDataset,

outputPath

)

}

}

}

/\*\*

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

unhydrated\_pair\_collection\_30min src/scala/com/twitter/simclusters\_v2/capesos\_config/atla\_proc3.yaml

\*/

object UnhydratedPairsCollection30MinScheduledApp extends ScheduledExecutionApp {

override def batchIncrement: Duration = Hours(24)

override def firstTime: RichDate = RichDate("2020-03-26")

override def runOnDateRange(

args: Args

)(

implicit dateRange: DateRange,

timeZone: TimeZone,

uniqueID: UniqueID

): Execution[Unit] = {

val maxSamplesPerClass: Int = args.int("samples\_per\_query\_tweet\_class", default = 2000)

val timeframe: Int = 30

val outputPath: String =

s"/user/cassowary/processed/tweet\_similarity/unhydrated\_pairs\_${timeframe}min"

UnhydratedPairsCollectionJob.getLabelledPairs(

dateRange,

timeframe.minute.inMilliseconds,

maxSamplesPerClass,

TweetSimilarityUnhydratedPairs30MinScalaDataset,

outputPath)

}

}

/\*\*

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

unhydrated\_pair\_collection\_120min src/scala/com/twitter/simclusters\_v2/capesos\_config/atla\_proc3.yaml

\*/

object UnhydratedPairsCollection120MinScheduledApp extends ScheduledExecutionApp {

override def batchIncrement: Duration = Hours(24)

override def firstTime: RichDate = RichDate("2020-03-26")

override def runOnDateRange(

args: Args

)(

implicit dateRange: DateRange,

timeZone: TimeZone,

uniqueID: UniqueID

): Execution[Unit] = {

val maxSamplesPerClass: Int = args.int("samples\_per\_query\_tweet\_class", default = 2000)

val timeframe: Int = 120

val outputPath: String =

s"/user/cassowary/processed/tweet\_similarity/unhydrated\_pairs\_${timeframe}min"

UnhydratedPairsCollectionJob.getLabelledPairs(

dateRange,

timeframe.minute.inMilliseconds,

maxSamplesPerClass,

TweetSimilarityUnhydratedPairs120MinScalaDataset,

outputPath)

}

}