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

import com.twitter.scalding.\_

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

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

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

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

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

import com.twitter.simclusters\_v2.candidate\_source.ClusterRanker

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

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

import com.twitter.simclusters\_v2.hdfs\_sources.SimclustersV2InterestedInScalaDataset

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

import com.twitter.simclusters\_v2.scalding.common.Util

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

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

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

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

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

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

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

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

import java.util.TimeZone

/\*\*

\* Do evaluations for SimClusters' tweet recommendations by using offline datasets.

\* The job does the following:

\* 1. Take in a test date range, for which the offline simclusters rec will be evaluated

\* 2. For all users that had tweet impressions in timelines during the period, generate offline

\* SimClusters candidate tweets for these users

\* 3. Run offline evaluation and return metrics

./bazel bundle src/scala/com/twitter/simclusters\_v2/scalding/evaluation:simcluster\_offline\_eval\_adhoc

Note: Never specify reference date range across more than 1 day!

oscar hdfs --user cassowary --screen --screen-detached --tee your\_ldap/prod\_percentile \

--bundle simcluster\_offline\_eval\_adhoc \

--tool com.twitter.simclusters\_v2.scalding.evaluation.SimClustersEvaluationAdhocApp \

-- --cand\_tweet\_date 2019-03-04T00 2019-03-04T23 \

--ref\_tweet\_date 2019-03-05T00 2019-03-05T01 \

--timeline\_tweet rectweet \

--sample\_rate 0.05 \

--max\_cand\_tweets 16000000 \

--min\_tweet\_score 0.0 \

--user\_interested\_in\_dir /user/frigate/your\_ldap/interested\_in\_copiedFromAtlaProc\_20190228 \

--cluster\_top\_k\_dir /user/cassowary/your\_ldap/offline\_simcluster\_20190304/cluster\_top\_k\_tweets \

--output\_dir /user/cassowary/your\_ldap/prod\_percentile \

--toEmailAddress your\_ldap@twitter.com \

--testRunName TestingProdOn0305Data

\*/

object SimClustersEvaluationAdhocApp extends TwitterExecutionApp {

private val maxTweetResults = 40

private val maxClustersToQuery = 20

@Override

def job: Execution[Unit] = {

Execution.withArgs { args =>

Execution.withId { implicit uniqueId =>

implicit val tz: TimeZone = DateOps.UTC

implicit val dateParser: DateParser = DateParser.default

val candTweetDateRange = DateRange.parse(args.list("cand\_tweet\_date"))

val refTweetDateRange = DateRange.parse(args.list("ref\_tweet\_date"))

val toEmailAddressOpt = args.optional("toEmailAddress")

val testRunName = args.optional("testRunName")

println(

s"Using SimClusters tweets from ${candTweetDateRange.start} to ${candTweetDateRange.end}")

println(s"Using Timelines tweets on the day of ${refTweetDateRange.start}")

// separate tweets from different display locations for now

val tweetType = args("timeline\_tweet") match {

case "rectweet" => DisplayLocation.TimelinesRectweet

case "recap" => DisplayLocation.TimelinesRecap

case e =>

throw new IllegalArgumentException(s"$e isn't a valid timeline display location")

}

val sampleRate = args.double("sample\_rate", 1.0)

val validRefPipe = getProdTimelineReference(tweetType, refTweetDateRange, sampleRate)

val targetUserPipe = validRefPipe.map { \_.targetUserId }

// Read a fixed-path in atla if provided, otherwise read prod data from atla for date range

val userInterestInPipe = args.optional("user\_interested\_in\_dir") match {

case Some(fixedPath) =>

println(s"user\_interested\_in\_dir is provided at: $fixedPath. Reading fixed path data.")

TypedPipe.from(AdhocKeyValSources.interestedInSource(fixedPath))

case \_ =>

println(s"user\_interested\_in\_dir isn't provided. Reading prod data.")

interestedInProdSource(candTweetDateRange)

}

// Offline simulation of this dataset

val clusterTopKDir = args("cluster\_top\_k\_dir")

println(s"cluster\_top\_k\_dir is defined at: $clusterTopKDir")

val clusterTopKPipe = TypedPipe.from(

ClusterTopKTweetsHourlySuffixSource(clusterTopKDir, candTweetDateRange)

)

// Configs for offline simcluster tweet recommendation

val maxTweetRecs = args.int("max\_cand\_tweets", 30000000)

val minTweetScoreThreshold = args.double("min\_tweet\_score", 0.0)

val offlineRecConfig = OfflineRecConfig(

maxTweetRecs,

maxTweetResults,

maxClustersToQuery,

minTweetScoreThreshold,

ClusterRanker.RankByNormalizedFavScore

)

println("SimClusters offline config: " + offlineRecConfig)

getValidCandidate(

targetUserPipe,

userInterestInPipe,

clusterTopKPipe,

offlineRecConfig,

candTweetDateRange

).flatMap { validCandPipe =>

val outputDir = args("output\_dir")

EvaluationMetricHelper.runAllEvaluations(validRefPipe, validCandPipe).map { results =>

toEmailAddressOpt.foreach { address =>

Util.sendEmail(

results,

"Results from tweet evaluation test bed " + testRunName.getOrElse(""),

address)

}

TypedPipe.from(Seq((results, ""))).writeExecution(TypedTsv[(String, String)](outputDir))

}

}

}

}

}

/\*\*

\* Given a pipe of raw timelines reference engagement data, collect the engagements that took

\* place during the given date range, then sample these engagements

\*/

private def getProdTimelineReference(

displayLocation: DisplayLocation,

batchDateRange: DateRange,

sampleRate: Double

)(

implicit tz: TimeZone

): TypedPipe[ReferenceTweets] = {

// Snapshot data timestamps itself with the last possible time of the day. +1 day to cover it

val snapshotRange = DateRange(batchDateRange.start, batchDateRange.start + Days(1))

val timelinesRefPipe = DAL

.readMostRecentSnapshot(TweetEvaluationTimelinesReferenceSetScalaDataset, snapshotRange)

.withRemoteReadPolicy(ExplicitLocation(ProcAtla))

.toTypedPipe

timelinesRefPipe

.flatMap { refTweets =>

val tweets = refTweets.impressedTweets

.filter { refTweet =>

refTweet.timestamp >= batchDateRange.start.timestamp &&

refTweet.timestamp <= batchDateRange.end.timestamp &&

refTweet.displayLocation == displayLocation

}

if (tweets.nonEmpty) {

Some(ReferenceTweets(refTweets.targetUserId, tweets))

} else {

None

}

}

.sample(sampleRate)

}

/\*\*

\* Given a list of target users, simulate SimCluster's online serving logic offline for these

\* users, then convert them into [[CandidateTweets]]

\*/

private def getValidCandidate(

targetUserPipe: TypedPipe[Long],

userIsInterestedInPipe: TypedPipe[(Long, ClustersUserIsInterestedIn)],

clusterTopKTweetsPipe: TypedPipe[ClusterTopKTweetsWithScores],

offlineConfig: OfflineRecConfig,

batchDateRange: DateRange

)(

implicit uniqueID: UniqueID

): Execution[TypedPipe[CandidateTweets]] = {

OfflineTweetRecommendation

.getTopTweets(offlineConfig, targetUserPipe, userIsInterestedInPipe, clusterTopKTweetsPipe)

.map(\_.map {

case (userId, scoredTweets) =>

val tweets = scoredTweets.map { tweet =>

CandidateTweet(tweet.tweetId, Some(tweet.score), Some(batchDateRange.start.timestamp))

}

CandidateTweets(userId, tweets)

})

}

/\*\*

\* Read interested in key-val store from atla-proc from the given date range

\*/

private def interestedInProdSource(

dateRange: DateRange

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

implicit val timeZone: TimeZone = DateOps.UTC

DAL

.readMostRecentSnapshot(SimclustersV2InterestedInScalaDataset, dateRange.embiggen(Weeks(1)))

.withRemoteReadPolicy(ExplicitLocation(ProcAtla))

.toTypedPipe

.map {

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

}

}

}