package com.twitter.simclusters\_v2.scalding.offline\_tweets

import com.twitter.algebird.Aggregator.size

import com.twitter.finagle.mtls.authentication.ServiceIdentifier

import com.twitter.scalding.typed.TypedPipe

import com.twitter.scalding.Args

import com.twitter.scalding.DateOps

import com.twitter.scalding.DateParser

import com.twitter.scalding.DateRange

import com.twitter.scalding.Days

import com.twitter.scalding.Duration

import com.twitter.scalding.Execution

import com.twitter.scalding.Hours

import com.twitter.scalding.RichDate

import com.twitter.scalding.TypedTsv

import com.twitter.scalding.UniqueID

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

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

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

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

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

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

import com.twitter.simclusters\_v2.hdfs\_sources.OfflineClusterTopMediaTweets20M145K2020ScalaDataset

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

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

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

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

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

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

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

import com.twitter.snowflake.id.SnowflakeId

import com.twitter.tweetsource.common.thriftscala.MediaType

import com.twitter.tweetsource.common.thriftscala.UnhydratedFlatTweet

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

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

import java.util.TimeZone

import java.text.SimpleDateFormat

object ClusterTopTweetsJob {

def serviceIdentifier(zone: String, env: String): ServiceIdentifier = ServiceIdentifier(

role = "cassowary",

service = "offline\_cluster\_top\_media\_tweets\_20M\_145K\_2020",

environment = env,

zone = zone

)

private def isMediaTweet(tweet: UnhydratedFlatTweet): Boolean = {

tweet.media.exists { mediaSeq =>

mediaSeq.exists { e =>

e.mediaType.contains(MediaType.Video)

}

}

}

private val dateFormatter = new SimpleDateFormat("yyyy-MM-dd")

def getClusterTopMediaTweets(

persistentEmbeddingPipe: TypedPipe[((TweetId, Timestamp), PersistentSimClustersEmbedding)],

tweetSourcePipe: TypedPipe[UnhydratedFlatTweet],

maxTweetsPerClusterPerPartition: Int

): TypedPipe[(DayPartitionedClusterId, Seq[(TweetId, Double)])] = {

val mediaTweetsPipe = tweetSourcePipe.collect {

case tweet if isMediaTweet(tweet) => (tweet.tweetId, ())

}

val tweetEmbeddingsPipe: TypedPipe[(TweetId, (Int, Double))] = {

persistentEmbeddingPipe.collect {

case ((tweetId, timestamp), persistentEmbedding)

if timestamp == 1L => // 1L is the longest L2 embedding

persistentEmbedding.embedding.embedding.map { clusterWithScore =>

(tweetId, (clusterWithScore.clusterId, clusterWithScore.score))

}

}.flatten

}

mediaTweetsPipe

.join(tweetEmbeddingsPipe)

.withReducers(2000)

.map {

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

val dayPartition = dateFormatter.format(SnowflakeId(tweetId).time.inMilliseconds)

((clusterId, dayPartition), Seq((tweetId, score)))

}

.sumByKey

.mapValues(\_.sortBy(-\_.\_2).take(maxTweetsPerClusterPerPartition))

.map { case ((cid, partition), values) => (DayPartitionedClusterId(cid, partition), values) }

}

// Convert to Manhattan compatible format

def toKeyVal(

clusterTopTweets: TypedPipe[(DayPartitionedClusterId, Seq[(TweetId, Double)])],

): TypedPipe[KeyVal[DayPartitionedClusterId, TweetsWithScore]] = {

clusterTopTweets.map {

case (key, tweetsWithScores) =>

val thrift = tweetsWithScores.map { t => TweetWithScore(t.\_1, t.\_2) }

KeyVal(key, TweetsWithScore(thrift))

}

}

}

/\*\*

\* Scheduled job. Runs every couple of hours (check the .yaml for exact cron schedule).

\* Reads 21 days of tweets, and the most recent persistent tweet embeddings from a Manhattan dump.

\* It outputs a clusterId-> List[tweetId] index.

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

offline\_cluster\_top\_media\_tweets\_20M\_145K\_2020 src/scala/com/twitter/simclusters\_v2/capesos\_config/atla\_proc3.yaml

\*/

object ClusterTopMediaTweets20M145K2020BatchJob extends ScheduledExecutionApp {

override def firstTime: RichDate = RichDate("2021-08-29")

override def batchIncrement: Duration = Hours(3)

override def runOnDateRange(

args: Args

)(

implicit dateRange: DateRange,

timeZone: TimeZone,

uniqueID: UniqueID

): Execution[Unit] = {

// max public tweet has 21 days. read 1 day fewer go give some buffer

val lookbackDateRange = dateRange.prepend(Days(21))

val tweetSource: TypedPipe[UnhydratedFlatTweet] =

ExternalDataSources.flatTweetsSource(lookbackDateRange)

val persistentEmbeddingPipe: TypedPipe[

((TweetId, Timestamp), PersistentSimClustersEmbedding)

] =

TypedPipe.from(

new LogFavBasedPersistentTweetEmbeddingMhExportSource(

range = lookbackDateRange,

serviceIdentifier = ClusterTopTweetsJob.serviceIdentifier(args("zone"), args("env"))

))

val maxTweetsPerClusterPerPartition = 1200

val dailyClusterTopTweets = ClusterTopTweetsJob.getClusterTopMediaTweets(

persistentEmbeddingPipe,

tweetSource,

maxTweetsPerClusterPerPartition

)

val keyValPipe: TypedPipe[KeyVal[DayPartitionedClusterId, TweetsWithScore]] =

ClusterTopTweetsJob.toKeyVal(dailyClusterTopTweets)

keyValPipe

.writeDALVersionedKeyValExecution(

OfflineClusterTopMediaTweets20M145K2020ScalaDataset,

D.Suffix(DataPaths.OfflineClusterTopMediaTweets2020DatasetPath)

)

}

}

/\*\*

Adhoc debugging job. Uses Entity Embeddings dataset to infer user interests

./bazel bundle src/scala/com/twitter/simclusters\_v2/scalding/offline\_tweets/ &&\

scalding remote run \

--main-class com.twitter.simclusters\_v2.scalding.offline\_tweets.AdhocClusterTopMediaTweetsJob \

--target src/scala/com/twitter/simclusters\_v2/scalding/offline\_tweets/:offline\_cluster\_top\_media\_tweets\_20M\_145K\_2020-adhoc \

--user cassowary \

-- --output\_dir /scratch\_user/cassowary/your\_ldap --date 2021-08-30 --zone atla --env prod --email your\_ldap@twitter.com

\*/

object AdhocClusterTopMediaTweetsJob extends AdhocExecutionApp {

/\*\*

\* Run some stat analysis on the results, such as the number of tweets in a cluster, tweet score

\* distributions, etc.

\*

\* Ideally works on 1 day data only. If multiple days data are passed in, it'll aggregate over

\* multiple days anyway

\*/

def analyzeClusterResults(

clusterTopTweets: TypedPipe[(DayPartitionedClusterId, Seq[(TweetId, Double)])]

): Execution[String] = {

val tweetSizeExec = Util.printSummaryOfNumericColumn(

clusterTopTweets.map { case (\_, tweets) => tweets.size },

columnName = Some("Tweet size distribution of clusters")

)

val scoreDistExec = Util.printSummaryOfNumericColumn(

clusterTopTweets.flatMap(\_.\_2.map(\_.\_2)),

columnName = Some("Score distribution of the tweets")

)

val numClustersExec =

clusterTopTweets.map(\_.\_1.\_1).distinct.aggregate(size).getOrElseExecution(0L)

val numTweetsExec =

clusterTopTweets.flatMap(\_.\_2.map(\_.\_1)).distinct.aggregate(size).getOrElseExecution(0L)

Execution.zip(tweetSizeExec, scoreDistExec, numClustersExec, numTweetsExec).map {

case (tweetSizeDist, scoreDist, numClusters, numTweets) =>

s"""

|Number of unique tweets = $numTweets

|Number of clusters = $numClusters

|------------------------

|$tweetSizeDist

|------------------------

|$scoreDist

|""".stripMargin

}

}

override def runOnDateRange(

args: Args

)(

implicit dateRange: DateRange,

timeZone: TimeZone,

uniqueID: UniqueID

): Execution[Unit] = {

val startTime = System.currentTimeMillis()

Execution.withArgs { args =>

Execution.getMode.flatMap { implicit mode =>

implicit val dateRange: DateRange =

DateRange.parse(args.list("date"))(DateOps.UTC, DateParser.default)

val outputDir = args("output\_dir")

val maxTweetsPerCluster = 100

// max public tweet has 21 days. read 1 day fewer go give some buffer

val lookbackDateRange = dateRange.prepend(Days(21))

val tweetSource: TypedPipe[UnhydratedFlatTweet] =

ExternalDataSources.flatTweetsSource(lookbackDateRange)

val persistentEmbeddingPipe: TypedPipe[

((TweetId, Timestamp), PersistentSimClustersEmbedding)

] =

TypedPipe.from(

new LogFavBasedPersistentTweetEmbeddingMhExportSource(

range = lookbackDateRange,

serviceIdentifier = ClusterTopTweetsJob.serviceIdentifier(args("zone"), args("env"))

))

val results = ClusterTopTweetsJob.getClusterTopMediaTweets(

persistentEmbeddingPipe,

tweetSource,

maxTweetsPerCluster

)

analyzeClusterResults(TypedPipe.empty)

.flatMap { distributions =>

val timeTakenMin = (System.currentTimeMillis() - startTime) / 60000

val text =

s"""

| AdhocClusterTopMediaTweetsJob finished on: $dateRange.

| Time taken: $timeTakenMin minutes.

| maxTweetsPerCluster: $maxTweetsPerCluster.

| output\_dir: $outputDir

|

| $distributions

""".stripMargin

Util.sendEmail(text, "AdhocClusterTopMediaTweetsJob finished.", args("email"))

results

.writeExecution(TypedTsv(outputDir))

}

}

}

}

}