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

import com.twitter.scalding.\_

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

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

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

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

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

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

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

import com.twitter.simclusters\_v2.scalding.offline\_job.SimClustersOfflineJob.\_

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

import java.util.TimeZone

/\*\*

scalding remote run --target src/scala/com/twitter/simclusters\_v2/scalding/offline\_job:simclusters\_offline\_job-adhoc \

--user cassowary \

--submitter hadoopnest2.atla.twitter.com \

--main-class com.twitter.simclusters\_v2.scalding.offline\_job.SimClustersOfflineJobAdhocApp -- \

--date 2019-08-10 --batch\_hours 24 \

--output\_dir /user/cassowary/your\_ldap/offline\_simcluster\_20190810

--model\_version 20M\_145K\_updated

\*/

object SimClustersOfflineJobAdhocApp extends TwitterExecutionApp {

import SimClustersOfflineJobUtil.\_

import com.twitter.simclusters\_v2.scalding.common.TypedRichPipe.\_

override def job: Execution[Unit] =

Execution.withId { implicit uniqueId =>

Execution.withArgs { args: Args =>

// required

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

val batchSize: Duration = Hours(args.int("batch\_hours"))

val outputDir = args("output\_dir")

val modelVersion = args.getOrElse("model\_version", ModelVersions.Model20M145KUpdated)

val scoringMethod = args.getOrElse("score", "logFav")

val tweetClusterScoreOutputPath: String = outputDir + "/tweet\_cluster\_scores"

val tweetTopKClustersOutputPath: String = outputDir + "/tweet\_top\_k\_clusters"

val clusterTopKTweetsOutputPath: String = outputDir + "/cluster\_top\_k\_tweets"

val fullInterestedInData: TypedPipe[(Long, ClustersUserIsInterestedIn)] =

args.optional("interested\_in\_path") match {

case Some(dir) =>

println("Loading InterestedIn from supplied path " + dir)

TypedPipe.from(AdhocKeyValSources.interestedInSource(dir))

case None =>

println("Loading production InterestedIn data")

readInterestedInScalaDataset(wholeDateRange)

}

val interestedInData: TypedPipe[(Long, ClustersUserIsInterestedIn)] =

fullInterestedInData.filter(\_.\_2.knownForModelVersion == modelVersion)

val debugExec = Execution.zip(

fullInterestedInData.printSummary("fullInterestedIn", numRecords = 20),

interestedInData.printSummary("interestedIn", numRecords = 20)

)

// recursive function to calculate batches one by one

def runBatch(batchDateRange: DateRange): Execution[Unit] = {

if (batchDateRange.start.timestamp > wholeDateRange.end.timestamp) {

Execution.unit // stops here

} else {

val previousScores = if (batchDateRange.start == wholeDateRange.start) {

TypedPipe.from(Nil)

} else {

TypedPipe.from(

TweetClusterScoresHourlySuffixSource(

tweetClusterScoreOutputPath,

batchDateRange - batchSize

)

)

}

val latestScores = computeAggregatedTweetClusterScores(

batchDateRange,

interestedInData,

readTimelineFavoriteData(batchDateRange),

previousScores

)

val writeLatestScoresExecution = {

Execution.zip(

latestScores.printSummary(name = "TweetEntityScores"),

latestScores

.writeExecution(

TweetClusterScoresHourlySuffixSource(

tweetClusterScoreOutputPath,

batchDateRange

)

)

)

}

val computeTweetTopKExecution = {

val tweetTopK = computeTweetTopKClusters(latestScores)

Execution.zip(

tweetTopK.printSummary(name = "TweetTopK"),

tweetTopK.writeExecution(

TweetTopKClustersHourlySuffixSource(tweetTopKClustersOutputPath, batchDateRange)

)

)

}

val computeClusterTopKExecution = {

val clusterTopK = computeClusterTopKTweets(latestScores)

Execution.zip(

clusterTopK.printSummary(name = "ClusterTopK"),

clusterTopK.writeExecution(

ClusterTopKTweetsHourlySuffixSource(clusterTopKTweetsOutputPath, batchDateRange)

)

)

}

Execution

.zip(

writeLatestScoresExecution,

computeTweetTopKExecution,

computeClusterTopKExecution

).flatMap { \_ =>

// run next batch

runBatch(batchDateRange + batchSize)

}

}

}

// start from the first batch

Util.printCounters(

Execution.zip(

debugExec,

runBatch(

DateRange(wholeDateRange.start, wholeDateRange.start + batchSize - Millisecs(1)))

)

)

}

}

}

/\*\*

For example:

scalding remote run --target src/scala/com/twitter/simclusters\_v2/scalding/offline\_job:dump\_cluster\_topk\_job-adhoc \

--user cassowary

--main-class com.twitter.simclusters\_v2.scalding.offline\_job.DumpClusterTopKTweetsAdhoc \

--submitter hadoopnest2.atla.twitter.com -- \

--date 2019-08-03 \

--clusterTopKTweetsPath /atla/proc3/user/cassowary/processed/simclusters/cluster\_top\_k\_tweets/ \

--clusters 4446

\*/

object DumpClusterTopKTweetsAdhoc extends TwitterExecutionApp {

implicit val timeZone: TimeZone = DateOps.UTC

implicit val dateParser: DateParser = DateParser.default

import com.twitter.simclusters\_v2.scalding.common.TypedRichPipe.\_

import com.twitter.simclusters\_v2.summingbird.common.ThriftDecayedValueMonoid.\_

override def job: Execution[Unit] =

Execution.withId { implicit uniqueId =>

Execution.withArgs { args: Args =>

val date = DateRange.parse(args.list("date"))

val path = args("clusterTopKTweetsPath")

val input = TypedPipe.from(ClusterTopKTweetsHourlySuffixSource(path, date))

val clusters = args.list("clusters").map(\_.toInt).toSet

val dvm = SimClustersOfflineJobUtil.thriftDecayedValueMonoid

if (clusters.isEmpty) {

input.printSummary("Cluster top k tweets")

} else {

input

.collect {

case rec if clusters.contains(rec.clusterId) =>

val res = rec.topKTweets

.mapValues { x =>

x.score

.map { y =>

val enriched = new EnrichedThriftDecayedValue(y)(dvm)

enriched.decayToTimestamp(date.end.timestamp).value

}.getOrElse(0.0)

}.toList.sortBy(-\_.\_2)

rec.clusterId + "\t" + Util.prettyJsonMapper

.writeValueAsString(res).replaceAll("\n", " ")

}

.toIterableExecution

.map { strings => println(strings.mkString("\n")) }

}

}

}

}