package com.twitter.interaction\_graph.scio.agg\_negative

import com.google.api.services.bigquery.model.TimePartitioning

import com.spotify.scio.ScioContext

import com.spotify.scio.values.SCollection

import com.twitter.algebird.mutable.PriorityQueueMonoid

import com.twitter.beam.io.dal.DAL

import com.twitter.beam.io.fs.multiformat.PathLayout

import com.twitter.beam.io.fs.multiformat.WriteOptions

import com.twitter.conversions.DurationOps.\_

import com.twitter.dal.client.dataset.SnapshotDALDataset

import com.twitter.interaction\_graph.scio.common.ConversionUtil.hasNegativeFeatures

import com.twitter.interaction\_graph.scio.common.ConversionUtil.toRealGraphEdgeFeatures

import com.twitter.interaction\_graph.scio.common.FeatureGeneratorUtil.getEdgeFeature

import com.twitter.interaction\_graph.scio.common.GraphUtil

import com.twitter.interaction\_graph.scio.common.InteractionGraphRawInput

import com.twitter.interaction\_graph.thriftscala.Edge

import com.twitter.interaction\_graph.thriftscala.FeatureName

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

import com.twitter.scio\_internal.job.ScioBeamJob

import com.twitter.scrooge.ThriftStruct

import com.twitter.socialgraph.hadoop.SocialgraphUnfollowsScalaDataset

import com.twitter.tcdc.bqblaster.beam.syntax.\_

import com.twitter.tcdc.bqblaster.core.avro.TypedProjection

import com.twitter.tcdc.bqblaster.core.transform.RootTransform

import com.twitter.timelines.real\_graph.thriftscala.RealGraphFeaturesTest

import com.twitter.timelines.real\_graph.v1.thriftscala.{RealGraphFeatures => RealGraphFeaturesV1}

import com.twitter.user\_session\_store.thriftscala.UserSession

import flockdb\_tools.datasets.flock.FlockBlocksEdgesScalaDataset

import flockdb\_tools.datasets.flock.FlockMutesEdgesScalaDataset

import flockdb\_tools.datasets.flock.FlockReportAsAbuseEdgesScalaDataset

import flockdb\_tools.datasets.flock.FlockReportAsSpamEdgesScalaDataset

import java.time.Instant

import org.apache.beam.sdk.io.gcp.bigquery.BigQueryIO

object InteractionGraphNegativeJob extends ScioBeamJob[InteractionGraphNegativeOption] {

val maxDestinationIds = 500 // p99 is about 500

def getFeatureCounts(e: Edge): Int = e.features.size

val negativeEdgeOrdering = Ordering.by[Edge, Int](getFeatureCounts)

val negativeEdgeReverseOrdering = negativeEdgeOrdering.reverse

implicit val pqMonoid: PriorityQueueMonoid[Edge] =

new PriorityQueueMonoid[Edge](maxDestinationIds)(negativeEdgeOrdering)

override protected def configurePipeline(

sc: ScioContext,

opts: InteractionGraphNegativeOption

): Unit = {

val endTs = opts.interval.getEndMillis

// read input datasets

val blocks: SCollection[InteractionGraphRawInput] =

GraphUtil.getFlockFeatures(

readSnapshot(FlockBlocksEdgesScalaDataset, sc),

FeatureName.NumBlocks,

endTs)

val mutes: SCollection[InteractionGraphRawInput] =

GraphUtil.getFlockFeatures(

readSnapshot(FlockMutesEdgesScalaDataset, sc),

FeatureName.NumMutes,

endTs)

val abuseReports: SCollection[InteractionGraphRawInput] =

GraphUtil.getFlockFeatures(

readSnapshot(FlockReportAsAbuseEdgesScalaDataset, sc),

FeatureName.NumReportAsAbuses,

endTs)

val spamReports: SCollection[InteractionGraphRawInput] =

GraphUtil.getFlockFeatures(

readSnapshot(FlockReportAsSpamEdgesScalaDataset, sc),

FeatureName.NumReportAsSpams,

endTs)

// we only keep unfollows in the past 90 days due to the huge size of this dataset,

// and to prevent permanent "shadow-banning" in the event of accidental unfollows.

// we treat unfollows as less critical than above 4 negative signals, since it deals more with

// interest than health typically, which might change over time.

val unfollows: SCollection[InteractionGraphRawInput] =

GraphUtil

.getSocialGraphFeatures(

readSnapshot(SocialgraphUnfollowsScalaDataset, sc),

FeatureName.NumUnfollows,

endTs)

.filter(\_.age < 90)

// group all features by (src, dest)

val allEdgeFeatures: SCollection[Edge] =

getEdgeFeature(SCollection.unionAll(Seq(blocks, mutes, abuseReports, spamReports, unfollows)))

val negativeFeatures: SCollection[KeyVal[Long, UserSession]] =

allEdgeFeatures

.keyBy(\_.sourceId)

.topByKey(maxDestinationIds)(Ordering.by(\_.features.size))

.map {

case (srcId, pqEdges) =>

val topKNeg =

pqEdges.toSeq.flatMap(toRealGraphEdgeFeatures(hasNegativeFeatures))

KeyVal(

srcId,

UserSession(

userId = Some(srcId),

realGraphFeaturesTest =

Some(RealGraphFeaturesTest.V1(RealGraphFeaturesV1(topKNeg)))))

}

// save to GCS (via DAL)

negativeFeatures.saveAsCustomOutput(

"Write Negative Edge Label",

DAL.writeVersionedKeyVal(

dataset = RealGraphNegativeFeaturesScalaDataset,

pathLayout = PathLayout.VersionedPath(opts.getOutputPath),

instant = Instant.ofEpochMilli(opts.interval.getEndMillis),

writeOption = WriteOptions(numOfShards = Some(3000))

)

)

// save to BQ

val ingestionDate = opts.getDate().value.getStart.toDate

val bqDataset = opts.getBqDataset

val bqFieldsTransform = RootTransform

.Builder()

.withPrependedFields("dateHour" -> TypedProjection.fromConstant(ingestionDate))

val timePartitioning = new TimePartitioning()

.setType("DAY").setField("dateHour").setExpirationMs(21.days.inMilliseconds)

val bqWriter = BigQueryIO

.write[Edge]

.to(s"${bqDataset}.interaction\_graph\_agg\_negative\_edge\_snapshot")

.withExtendedErrorInfo()

.withTimePartitioning(timePartitioning)

.withLoadJobProjectId("twttr-recos-ml-prod")

.withThriftSupport(bqFieldsTransform.build(), AvroConverter.Legacy)

.withCreateDisposition(BigQueryIO.Write.CreateDisposition.CREATE\_IF\_NEEDED)

.withWriteDisposition(

BigQueryIO.Write.WriteDisposition.WRITE\_TRUNCATE

) // we only want the latest snapshot

allEdgeFeatures

.saveAsCustomOutput(

s"Save Recommendations to BQ interaction\_graph\_agg\_negative\_edge\_snapshot",

bqWriter

)

}

def readSnapshot[T <: ThriftStruct](

dataset: SnapshotDALDataset[T],

sc: ScioContext

): SCollection[T] = {

sc.customInput(

s"Reading most recent snaphost ${dataset.role.name}.${dataset.logicalName}",

DAL.readMostRecentSnapshotNoOlderThan[T](dataset, 7.days)

)

}

}