package com.twitter.simclusters\_v2.scio.bq\_generation

package ftr\_tweet

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

import com.spotify.scio.ScioContext

import com.spotify.scio.coders.Coder

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

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

import com.twitter.beam.job.DateRangeOptions

import com.twitter.conversions.DurationOps.richDurationFromInt

import com.twitter.dal.client.dataset.KeyValDALDataset

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

import com.twitter.scio\_internal.coders.ThriftStructLazyBinaryScroogeCoder

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

import com.twitter.scrooge.ThriftStruct

import com.twitter.simclusters\_v2.scio.bq\_generation.common.BQTableDetails

import com.twitter.simclusters\_v2.scio.bq\_generation.common.BQGenerationUtil.getInterestedIn2020SQL

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

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

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

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

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

import java.time.Instant

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

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

import org.apache.avro.generic.GenericData

import scala.collection.mutable.ListBuffer

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

import org.apache.beam.sdk.transforms.SerializableFunction

import org.apache.avro.generic.GenericRecord

import com.twitter.wtf.beam.bq\_embedding\_export.BQQueryUtils

trait FTRJob extends ScioBeamJob[DateRangeOptions] {

// Configs to set for different type of embeddings and jobs

val isAdhoc: Boolean

val outputTable: BQTableDetails

val keyValDatasetOutputPath: String

val tweetRecommentationsSnapshotDataset: KeyValDALDataset[KeyVal[Long, CandidateTweetsList]]

val scoreKey: String

val scoreColumn: String

// Base configs

val projectId = "twttr-recos-ml-prod"

val environment: DAL.Env = if (isAdhoc) DAL.Environment.Dev else DAL.Environment.Prod

override implicit def scroogeCoder[T <: ThriftStruct: Manifest]: Coder[T] =

ThriftStructLazyBinaryScroogeCoder.scroogeCoder

override def configurePipeline(sc: ScioContext, opts: DateRangeOptions): Unit = {

// The time when the job is scheduled

val queryTimestamp = opts.interval.getEnd

// Parse tweetId candidates column

def parseTweetIdColumn(

genericRecord: GenericRecord,

columnName: String

): List[CandidateTweet] = {

val tweetIds: GenericData.Array[GenericRecord] =

genericRecord.get(columnName).asInstanceOf[GenericData.Array[GenericRecord]]

val results: ListBuffer[CandidateTweet] = new ListBuffer[CandidateTweet]()

tweetIds.forEach((sc: GenericRecord) => {

results += CandidateTweet(

tweetId = sc.get("tweetId").toString.toLong,

score = Some(sc.get("cosineSimilarityScore").toString.toDouble)

)

})

results.toList

}

//Function that parses the GenericRecord results we read from BQ

val parseUserToTweetRecommendationsFunc =

new SerializableFunction[SchemaAndRecord, UserToTweetRecommendations] {

override def apply(record: SchemaAndRecord): UserToTweetRecommendations = {

val genericRecord: GenericRecord = record.getRecord

UserToTweetRecommendations(

userId = genericRecord.get("userId").toString.toLong,

tweetCandidates = parseTweetIdColumn(genericRecord, "tweets"),

)

}

}

val tweetEmbeddingTemplateVariables =

Map(

"START\_TIME" -> queryTimestamp.minusDays(1).toString(),

"END\_TIME" -> queryTimestamp.toString(),

"TWEET\_SAMPLE\_RATE" -> Config.TweetSampleRate.toString,

"ENG\_SAMPLE\_RATE" -> Config.EngSampleRate.toString,

"MIN\_TWEET\_FAVS" -> Config.MinTweetFavs.toString,

"MIN\_TWEET\_IMPS" -> Config.MinTweetImps.toString,

"MAX\_TWEET\_FTR" -> Config.MaxTweetFTR.toString,

"MAX\_USER\_LOG\_N\_IMPS" -> Config.MaxUserLogNImps.toString,

"MAX\_USER\_LOG\_N\_FAVS" -> Config.MaxUserLogNFavs.toString,

"MAX\_USER\_FTR" -> Config.MaxUserFTR.toString,

"TWEET\_EMBEDDING\_LENGTH" -> Config.SimClustersTweetEmbeddingsGenerationEmbeddingLength.toString,

"HALFLIFE" -> Config.SimClustersTweetEmbeddingsGenerationHalfLife.toString,

"SCORE\_COLUMN" -> scoreColumn,

"SCORE\_KEY" -> scoreKey,

)

val tweetEmbeddingSql = BQQueryUtils.getBQQueryFromSqlFile(

"/com/twitter/simclusters\_v2/scio/bq\_generation/ftr\_tweet/sql/ftr\_tweet\_embeddings.sql",

tweetEmbeddingTemplateVariables)

val consumerEmbeddingSql = getInterestedIn2020SQL(queryTimestamp, 14)

val tweetRecommendationsTemplateVariables =

Map(

"CONSUMER\_EMBEDDINGS\_SQL" -> consumerEmbeddingSql,

"TWEET\_EMBEDDINGS\_SQL" -> tweetEmbeddingSql,

"TOP\_N\_CLUSTER\_PER\_SOURCE\_EMBEDDING" -> Config.SimClustersANNTopNClustersPerSourceEmbedding.toString,

"TOP\_M\_TWEETS\_PER\_CLUSTER" -> Config.SimClustersANNTopMTweetsPerCluster.toString,

"TOP\_K\_TWEETS\_PER\_USER\_REQUEST" -> Config.SimClustersANNTopKTweetsPerUserRequest.toString,

)

val tweetRecommendationsSql = BQQueryUtils.getBQQueryFromSqlFile(

"/com/twitter/simclusters\_v2/scio/bq\_generation/sql/tweets\_ann.sql",

tweetRecommendationsTemplateVariables)

val tweetRecommendations = sc.customInput(

s"SimClusters FTR BQ ANN",

BigQueryIO

.read(parseUserToTweetRecommendationsFunc)

.fromQuery(tweetRecommendationsSql)

.usingStandardSql()

)

//Setup BQ writer

val ingestionTime = opts.getDate().value.getEnd.toDate

val bqFieldsTransform = RootTransform

.Builder()

.withPrependedFields("ingestionTime" -> TypedProjection.fromConstant(ingestionTime))

val timePartitioning = new TimePartitioning()

.setType("HOUR").setField("ingestionTime").setExpirationMs(3.days.inMilliseconds)

val bqWriter = BigQueryIO

.write[CandidateTweets]

.to(outputTable.toString)

.withExtendedErrorInfo()

.withTimePartitioning(timePartitioning)

.withLoadJobProjectId(projectId)

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

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

.withWriteDisposition(BigQueryIO.Write.WriteDisposition.WRITE\_APPEND)

// Save Tweet ANN results to BQ

tweetRecommendations

.map { userToTweetRecommendations =>

{

CandidateTweets(

targetUserId = userToTweetRecommendations.userId,

recommendedTweets = userToTweetRecommendations.tweetCandidates)

}

}

.saveAsCustomOutput(s"WriteToBQTable - $outputTable", bqWriter)

val RootMHPath: String = Config.FTRRootMHPath

val AdhocRootPath = Config.FTRAdhocpath

// Save Tweet ANN results as KeyValSnapshotDataset

tweetRecommendations

.map { userToTweetRecommendations =>

KeyVal(

userToTweetRecommendations.userId,

CandidateTweetsList(userToTweetRecommendations.tweetCandidates))

}.saveAsCustomOutput(

name = "WriteFtrTweetRecommendationsToKeyValDataset",

DAL.writeVersionedKeyVal(

tweetRecommentationsSnapshotDataset,

PathLayout.VersionedPath(prefix =

((if (!isAdhoc)

RootMHPath

else

AdhocRootPath)

+ keyValDatasetOutputPath)),

instant = Instant.ofEpochMilli(opts.interval.getEndMillis - 1L),

environmentOverride = environment,

)

)

}

}

object FTRAdhocJob extends FTRJob {

override val isAdhoc = true

override val outputTable: BQTableDetails =

BQTableDetails("twttr-recos-ml-prod", "simclusters", "offline\_tweet\_recommendations\_ftr\_adhoc")

override val keyValDatasetOutputPath = Config.IIKFFTRAdhocANNOutputPath

override val tweetRecommentationsSnapshotDataset: KeyValDALDataset[

KeyVal[Long, CandidateTweetsList]

] =

OfflineTweetRecommendationsFtrAdhocScalaDataset

override val scoreColumn = "ftrat5\_decayed\_pop\_bias\_1000\_rank\_decay\_1\_1\_embedding"

override val scoreKey = "ftrat5\_decayed\_pop\_bias\_1000\_rank\_decay\_1\_1"

}

object IIKF2020DecayedSumBatchJobProd extends FTRJob {

override val isAdhoc = false

override val outputTable: BQTableDetails = BQTableDetails(

"twttr-bq-cassowary-prod",

"user",

"offline\_tweet\_recommendations\_decayed\_sum"

)

override val keyValDatasetOutputPath = Config.IIKFDecayedSumANNOutputPath

override val tweetRecommentationsSnapshotDataset: KeyValDALDataset[

KeyVal[Long, CandidateTweetsList]

] =

OfflineTweetRecommendationsDecayedSumScalaDataset

override val scoreColumn = "dec\_sum\_logfavScoreClusterNormalizedOnly\_embedding"

override val scoreKey = "dec\_sum\_logfavScoreClusterNormalizedOnly"

}

object IIKF2020FTRAt5Pop1000batchJobProd extends FTRJob {

override val isAdhoc = false

override val outputTable: BQTableDetails = BQTableDetails(

"twttr-bq-cassowary-prod",

"user",

"offline\_tweet\_recommendations\_ftrat5\_pop\_biased\_1000")

override val keyValDatasetOutputPath = Config.IIKFFTRAt5Pop1000ANNOutputPath

override val tweetRecommentationsSnapshotDataset: KeyValDALDataset[

KeyVal[Long, CandidateTweetsList]

] =

OfflineTweetRecommendationsFtrat5PopBiased1000ScalaDataset

override val scoreColumn = "ftrat5\_decayed\_pop\_bias\_1000\_rank\_decay\_1\_1\_embedding"

override val scoreKey = "ftrat5\_decayed\_pop\_bias\_1000\_rank\_decay\_1\_1"

}

object IIKF2020FTRAt5Pop10000batchJobProd extends FTRJob {

override val isAdhoc = false

override val outputTable: BQTableDetails = BQTableDetails(

"twttr-bq-cassowary-prod",

"user",

"offline\_tweet\_recommendations\_ftrat5\_pop\_biased\_10000")

override val keyValDatasetOutputPath = Config.IIKFFTRAt5Pop10000ANNOutputPath

override val tweetRecommentationsSnapshotDataset: KeyValDALDataset[

KeyVal[Long, CandidateTweetsList]

] =

OfflineTweetRecommendationsFtrat5PopBiased10000ScalaDataset

override val scoreColumn = "ftrat5\_decayed\_pop\_bias\_10000\_rank\_decay\_1\_1\_embedding"

override val scoreKey = "ftrat5\_decayed\_pop\_bias\_10000\_rank\_decay\_1\_1"

}

case class UserToTweetRecommendations(

userId: Long,

tweetCandidates: List[CandidateTweet])