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

package tweets\_ann

import com.spotify.scio.ScioContext

import com.spotify.scio.values.SCollection

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

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

import org.apache.avro.generic.GenericData

import org.apache.avro.generic.GenericRecord

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

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

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

import org.joda.time.DateTime

import scala.collection.mutable.ListBuffer

object TweetsANNFromBQ {

// Default ANN config variables

val topNClustersPerSourceEmbedding = Config.SimClustersANNTopNClustersPerSourceEmbedding

val topMTweetsPerCluster = Config.SimClustersANNTopMTweetsPerCluster

val topKTweetsPerUserRequest = Config.SimClustersANNTopKTweetsPerUserRequest

// SQL file paths

val tweetsANNSQLPath =

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

val tweetsEmbeddingGenerationSQLPath =

s"/com/twitter/simclusters\_v2/scio/bq\_generation/sql/tweet\_embeddings\_generation.sql"

// 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"),

)

}

}

// 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("logCosineSimilarityScore").toString.toDouble)

)

})

results.toList

}

def getTweetEmbeddingsSQL(

queryDate: DateTime,

consumerEmbeddingsSQL: String,

tweetEmbeddingsSQLPath: String,

tweetEmbeddingsHalfLife: Int,

tweetEmbeddingsLength: Int

): String = {

// We read one day of fav events to construct our tweet embeddings

val templateVariables =

Map(

"CONSUMER\_EMBEDDINGS\_SQL" -> consumerEmbeddingsSQL,

"QUERY\_DATE" -> queryDate.toString(),

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

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

"MIN\_SCORE\_THRESHOLD" -> 0.0.toString,

"HALF\_LIFE" -> tweetEmbeddingsHalfLife.toString,

"TWEET\_EMBEDDING\_LENGTH" -> tweetEmbeddingsLength.toString,

"NO\_OLDER\_TWEETS\_THAN\_DATE" -> queryDate.minusDays(1).toString(),

)

BQQueryUtils.getBQQueryFromSqlFile(tweetEmbeddingsSQLPath, templateVariables)

}

def getTweetRecommendationsBQ(

sc: ScioContext,

queryTimestamp: DateTime,

consumerEmbeddingsSQL: String,

tweetEmbeddingsHalfLife: Int,

tweetEmbeddingsLength: Int

): SCollection[UserToTweetRecommendations] = {

// Get the tweet embeddings SQL string based on the provided consumerEmbeddingsSQL

val tweetEmbeddingsSQL =

getTweetEmbeddingsSQL(

queryTimestamp,

consumerEmbeddingsSQL,

tweetsEmbeddingGenerationSQLPath,

tweetEmbeddingsHalfLife,

tweetEmbeddingsLength

)

// Define template variables which we would like to be replaced in the corresponding sql file

val templateVariables =

Map(

"CONSUMER\_EMBEDDINGS\_SQL" -> consumerEmbeddingsSQL,

"TWEET\_EMBEDDINGS\_SQL" -> tweetEmbeddingsSQL,

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

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

"TOP\_K\_TWEETS\_PER\_USER\_REQUEST" -> topKTweetsPerUserRequest.toString

)

val query = BQQueryUtils.getBQQueryFromSqlFile(tweetsANNSQLPath, templateVariables)

// Run SimClusters ANN on BQ and parse the results

sc.customInput(

s"SimClusters BQ ANN",

BigQueryIO

.read(parseUserToTweetRecommendationsFunc)

.fromQuery(query)

.usingStandardSql()

)

}

case class UserToTweetRecommendations(

userId: Long,

tweetCandidates: List[CandidateTweet])

}