package com.twitter.timelines.data\_processing.ad\_hoc.earlybird\_ranking.training\_data\_generation

import com.twitter.ml.api.HourlySuffixFeatureSource

import com.twitter.ml.api.IRecord

import com.twitter.scalding.Args

import com.twitter.scalding.DateRange

import com.twitter.scalding.Days

import com.twitter.scalding.Execution

import com.twitter.scalding.ExecutionUtil

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

import com.twitter.timelines.data\_processing.ad\_hoc.earlybird\_ranking.common.EarlybirdTrainingRecapConfiguration

import com.twitter.timelines.data\_processing.ad\_hoc.earlybird\_ranking.common.EarlybirdTrainingRectweetConfiguration

import com.twitter.timelines.data\_processing.ad\_hoc.recap.offline\_execution.OfflineAdhocExecution

import com.twitter.timelines.data\_processing.ad\_hoc.recap.offline\_execution.OfflineAnalyticsBatchExecution

import com.twitter.timelines.data\_processing.ad\_hoc.recap.offline\_execution.OfflineExecution

import scala.util.Random

import com.twitter.scalding\_internal.dalv2.dataset.DALWrite.\_

import com.twitter.timelines.prediction.features.common.TimelinesSharedFeatures

import timelines.data\_processing.ad\_hoc.earlybird\_ranking.training\_data\_generation.\_

/\*\*

\* Generates data for training an Earlybird-friendly model.

\* Produces a single "global" engagement, and samples data accordingly.

\* Also converts features from Earlybird to their original Earlybird

\* feature names so they can be used as is in EB.

\*

\* Arguments:

\* --input path to raw Recap training data (all labels)

\* --output path to write sampled Earlybird-friendly training data

\* --seed (optional) for random number generator (in sampling)

\* --parallelism (default: 1) number of days to generate data for in parallel

\* [splits long date range into single days]

\*/

trait GenerateEarlybirdTrainingData { \_: OfflineExecution =>

def isEligibleForEarlybirdScoring(record: IRecord): Boolean = {

// The rationale behind this logic is available in TQ-9678.

record.getFeatureValue(TimelinesSharedFeatures.EARLYBIRD\_SCORE) <= 100.0

}

override def executionFromParams(args: Args)(implicit dateRange: DateRange): Execution[Unit] = {

val seedOpt = args.optional("seed").map(\_.toLong)

val parallelism = args.int("parallelism", 1)

val rectweet = args.boolean("rectweet")

ExecutionUtil

.runDateRangeWithParallelism(Days(1), parallelism) { splitRange =>

val data = HourlySuffixFeatureSource(args("input"))(splitRange).read

.filter(isEligibleForEarlybirdScoring \_)

lazy val rng = seedOpt.map(new Random(\_)).getOrElse(new Random())

val (constants, sink) =

if (rectweet)

(new EarlybirdTrainingRectweetConfiguration, EarlybirdRectweetDataRecordsJavaDataset)

else (new EarlybirdTrainingRecapConfiguration, EarlybirdRecapDataRecordsJavaDataset)

val earlybirdSampler =

new EarlybirdExampleSampler(

random = rng,

labelInfos = constants.LabelInfos,

negativeInfo = constants.NegativeInfo

)

val outputPath = args("output")

earlybirdSampler

.weightAndSample(data)

.transform(constants.EarlybirdFeatureRenamer)

// shuffle row-wise in order to get rid of clustered replies

// also keep number of part files small

.viaRecords { record =>

record

.groupRandomly(partitions = 500)

.sortBy { \_ => rng.nextDouble() }

.values

}

.writeDALExecution(

sink,

D.Daily,

D.Suffix(outputPath),

D.EBLzo()

)(splitRange)

}(dateRange).unit

}

}

object EarlybirdTrainingDataAdHocJob

extends OfflineAdhocExecution

with GenerateEarlybirdTrainingData

object EarlybirdTrainingDataProdJob

extends OfflineAnalyticsBatchExecution

with GenerateEarlybirdTrainingData