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

import com.twitter.ml.api.analytics.DataSetAnalyticsPlugin

import com.twitter.ml.api.matcher.FeatureMatcher

import com.twitter.ml.api.util.FDsl

import com.twitter.ml.api.DailySuffixFeatureSource

import com.twitter.ml.api.DataRecord

import com.twitter.ml.api.DataSetPipe

import com.twitter.ml.api.FeatureStats

import com.twitter.ml.api.IMatcher

import com.twitter.scalding.typed.TypedPipe

import com.twitter.scalding.Execution

import com.twitter.scalding.TypedJson

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

import com.twitter.timelines.data\_processing.util.execution.UTCDateRangeFromArgs

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

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

import com.twitter.timelines.prediction.features.recap.RecapFeatures

import scala.collection.JavaConverters.\_

/\*\*

\* Compute counts and fractions for all labels in a Recap data source.

\*

\* Arguments:

\* --input recap data source (containing all labels)

\* --output path to output JSON file containing stats

\*/

object EarlybirdStatsJob extends TwitterExecutionApp with UTCDateRangeFromArgs {

import DataSetAnalyticsPlugin.\_

import FDsl.\_

import RecapFeatures.IS\_EARLYBIRD\_UNIFIED\_ENGAGEMENT

lazy val constants: EarlybirdTrainingConfiguration = new EarlybirdTrainingRecapConfiguration

private[this] def addGlobalEngagementLabel(record: DataRecord) = {

if (constants.LabelInfos.exists { labelInfo => record.hasFeature(labelInfo.feature) }) {

record.setFeatureValue(IS\_EARLYBIRD\_UNIFIED\_ENGAGEMENT, true)

}

record

}

private[this] def labelFeatureMatcher: IMatcher = {

val allLabels =

(IS\_EARLYBIRD\_UNIFIED\_ENGAGEMENT :: constants.LabelInfos.map(\_.feature)).map(\_.getFeatureName)

FeatureMatcher.names(allLabels.asJava)

}

private[this] def computeStats(data: DataSetPipe): TypedPipe[FeatureStats] = {

data

.viaRecords { \_.map(addGlobalEngagementLabel) }

.project(labelFeatureMatcher)

.collectFeatureStats()

}

override def job: Execution[Unit] = {

for {

args <- Execution.getArgs

dateRange <- dateRangeEx

data = DailySuffixFeatureSource(args("input"))(dateRange).read

\_ <- computeStats(data).writeExecution(TypedJson(args("output")))

} yield ()

}

}