package com.twitter.interaction\_graph.scio.agg\_client\_event\_logs

import com.spotify.scio.values.SCollection

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

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

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.interaction\_graph.thriftscala.Vertex

import com.twitter.wtf.scalding.client\_event\_processing.thriftscala.InteractionDetails

import com.twitter.wtf.scalding.client\_event\_processing.thriftscala.InteractionType

import com.twitter.wtf.scalding.client\_event\_processing.thriftscala.UserInteraction

object InteractionGraphClientEventLogsUtil {

val DefaultAge = 1

val DefaultFeatureValue = 1.0

def process(

userInteractions: SCollection[UserInteraction],

safeUsers: SCollection[Long]

)(

implicit jobCounters: InteractionGraphClientEventLogsCountersTrait

): (SCollection[Vertex], SCollection[Edge]) = {

val unfilteredFeatureInput = userInteractions

.flatMap {

case UserInteraction(

userId,

\_,

interactionType,

InteractionDetails.ProfileClickDetails(profileClick))

if interactionType == InteractionType.ProfileClicks && userId != profileClick.profileId =>

jobCounters.profileViewFeaturesInc()

Seq(

FeatureKey(

userId,

profileClick.profileId,

FeatureName.NumProfileViews) -> DefaultFeatureValue

)

case UserInteraction(

userId,

\_,

interactionType,

InteractionDetails.TweetClickDetails(tweetClick))

if interactionType == InteractionType.TweetClicks &&

Some(userId) != tweetClick.authorId =>

(

for {

authorId <- tweetClick.authorId

} yield {

jobCounters.tweetClickFeaturesInc()

FeatureKey(userId, authorId, FeatureName.NumTweetClicks) -> DefaultFeatureValue

}

).toSeq

case UserInteraction(

userId,

\_,

interactionType,

InteractionDetails.LinkClickDetails(linkClick))

if interactionType == InteractionType.LinkClicks &&

Some(userId) != linkClick.authorId =>

(

for {

authorId <- linkClick.authorId

} yield {

jobCounters.linkOpenFeaturesInc()

FeatureKey(userId, authorId, FeatureName.NumLinkClicks) -> DefaultFeatureValue

}

).toSeq

case UserInteraction(

userId,

\_,

interactionType,

InteractionDetails.TweetImpressionDetails(tweetImpression))

if interactionType == InteractionType.TweetImpressions &&

Some(userId) != tweetImpression.authorId =>

(

for {

authorId <- tweetImpression.authorId

dwellTime <- tweetImpression.dwellTimeInSec

} yield {

jobCounters.tweetImpressionFeaturesInc()

Seq(

FeatureKey(

userId,

authorId,

FeatureName.NumInspectedStatuses) -> DefaultFeatureValue,

FeatureKey(userId, authorId, FeatureName.TotalDwellTime) -> dwellTime.toDouble

)

}

).getOrElse(Nil)

case \_ =>

jobCounters.catchAllInc()

Nil

}

.sumByKey

.collect {

case (FeatureKey(srcId, destId, featureName), featureValue) =>

InteractionGraphRawInput(

src = srcId,

dst = destId,

name = featureName,

age = 1,

featureValue = featureValue

)

}

val filteredFeatureInput = filterForSafeUsers(unfilteredFeatureInput, safeUsers)

// Calculate the Features

FeatureGeneratorUtil.getFeatures(filteredFeatureInput)

}

private def filterForSafeUsers(

featureInput: SCollection[InteractionGraphRawInput],

safeUsers: SCollection[Long]

): SCollection[InteractionGraphRawInput] = {

featureInput

.keyBy(\_.src)

.withName("Filter out unsafe users")

.intersectByKey(safeUsers)

.values // Fetch only InteractionGraphRawInput

.keyBy(\_.dst)

.withName("Filter out unsafe authors")

.intersectByKey(safeUsers)

.values // Fetch only InteractionGraphRawInput

}

}