package com.twitter.interaction\_graph.scio.agg\_direct\_interactions

import com.spotify.scio.ScioMetrics

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.scio.common.UserUtil.DUMMY\_USER\_ID

import com.twitter.interaction\_graph.thriftscala.Edge

import com.twitter.interaction\_graph.thriftscala.FeatureName

import com.twitter.interaction\_graph.thriftscala.Vertex

import com.twitter.timelineservice.thriftscala.ContextualizedFavoriteEvent

import com.twitter.timelineservice.thriftscala.FavoriteEventUnion.Favorite

import com.twitter.tweetsource.common.thriftscala.UnhydratedFlatTweet

import com.twitter.tweetypie.thriftscala.TweetMediaTagEvent

object InteractionGraphAggDirectInteractionsUtil {

val DefaultFeatureValue = 1L

def favouriteFeatures(

rawFavorites: SCollection[ContextualizedFavoriteEvent]

): SCollection[(FeatureKey, Long)] = {

rawFavorites

.withName("fav features")

.flatMap { event =>

event.event match {

case Favorite(e) if e.userId != e.tweetUserId =>

ScioMetrics.counter("process", "fav").inc()

Some(

FeatureKey(e.userId, e.tweetUserId, FeatureName.NumFavorites) -> DefaultFeatureValue)

case \_ => None

}

}

}

def mentionFeatures(

tweetSource: SCollection[UnhydratedFlatTweet]

): SCollection[(FeatureKey, Long)] = {

tweetSource

.withName("mention features")

.flatMap {

case s if s.shareSourceTweetId.isEmpty => // only for non-retweets

s.atMentionedUserIds

.map { users =>

users.toSet.map { uid: Long =>

ScioMetrics.counter("process", "mention").inc()

FeatureKey(s.userId, uid, FeatureName.NumMentions) -> DefaultFeatureValue

}.toSeq

}

.getOrElse(Nil)

case \_ =>

Nil

}

}

def photoTagFeatures(

rawPhotoTags: SCollection[TweetMediaTagEvent]

): SCollection[(FeatureKey, Long)] = {

rawPhotoTags

.withName("photo tag features")

.flatMap { p =>

p.taggedUserIds.map { (p.userId, \_) }

}

.collect {

case (src, dst) if src != dst =>

ScioMetrics.counter("process", "photo tag").inc()

FeatureKey(src, dst, FeatureName.NumPhotoTags) -> DefaultFeatureValue

}

}

def retweetFeatures(

tweetSource: SCollection[UnhydratedFlatTweet]

): SCollection[(FeatureKey, Long)] = {

tweetSource

.withName("retweet features")

.collect {

case s if s.shareSourceUserId.exists(\_ != s.userId) =>

ScioMetrics.counter("process", "share tweet").inc()

FeatureKey(

s.userId,

s.shareSourceUserId.get,

FeatureName.NumRetweets) -> DefaultFeatureValue

}

}

def quotedTweetFeatures(

tweetSource: SCollection[UnhydratedFlatTweet]

): SCollection[(FeatureKey, Long)] = {

tweetSource

.withName("quoted tweet features")

.collect {

case t if t.quotedTweetUserId.isDefined =>

ScioMetrics.counter("process", "quote tweet").inc()

FeatureKey(

t.userId,

t.quotedTweetUserId.get,

FeatureName.NumTweetQuotes) -> DefaultFeatureValue

}

}

def replyTweetFeatures(

tweetSource: SCollection[UnhydratedFlatTweet]

): SCollection[(FeatureKey, Long)] = {

tweetSource

.withName("reply tweet features")

.collect {

case t if t.inReplyToUserId.isDefined =>

ScioMetrics.counter("process", "reply tweet").inc()

FeatureKey(t.userId, t.inReplyToUserId.get, FeatureName.NumReplies) -> DefaultFeatureValue

}

}

// we create edges to a dummy user id since creating a tweet has no destination id

def createTweetFeatures(

tweetSource: SCollection[UnhydratedFlatTweet]

): SCollection[(FeatureKey, Long)] = {

tweetSource.withName("create tweet features").map { tweet =>

ScioMetrics.counter("process", "create tweet").inc()

FeatureKey(tweet.userId, DUMMY\_USER\_ID, FeatureName.NumCreateTweets) -> DefaultFeatureValue

}

}

def process(

rawFavorites: SCollection[ContextualizedFavoriteEvent],

tweetSource: SCollection[UnhydratedFlatTweet],

rawPhotoTags: SCollection[TweetMediaTagEvent],

safeUsers: SCollection[Long]

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

val favouriteInput = favouriteFeatures(rawFavorites)

val mentionInput = mentionFeatures(tweetSource)

val photoTagInput = photoTagFeatures(rawPhotoTags)

val retweetInput = retweetFeatures(tweetSource)

val quotedTweetInput = quotedTweetFeatures(tweetSource)

val replyInput = replyTweetFeatures(tweetSource)

val createTweetInput = createTweetFeatures(tweetSource)

val allInput = SCollection.unionAll(

Seq(

favouriteInput,

mentionInput,

photoTagInput,

retweetInput,

quotedTweetInput,

replyInput,

createTweetInput

))

val filteredFeatureInput = allInput

.keyBy(\_.\_1.src)

.intersectByKey(safeUsers) // filter for safe users

.values

.collect {

case (FeatureKey(src, dst, feature), featureValue) if src != dst =>

FeatureKey(src, dst, feature) -> featureValue

}

.sumByKey

.map {

case (FeatureKey(src, dst, feature), featureValue) =>

val age = 1

InteractionGraphRawInput(src, dst, feature, age, featureValue)

}

FeatureGeneratorUtil.getFeatures(filteredFeatureInput)

}

}