package com.twitter.interaction\_graph.scio.common

import com.spotify.scio.ScioMetrics

import com.spotify.scio.values.SCollection

import com.twitter.socialgraph.presto.thriftscala.{Edge => SocialGraphEdge}

import com.twitter.flockdb.tools.datasets.flock.thriftscala.FlockEdge

import com.twitter.interaction\_graph.scio.common.FeatureGroups.HEALTH\_FEATURE\_LIST

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

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

import java.time.Instant

import java.time.temporal.ChronoUnit

object GraphUtil {

/\*\*

\* Convert FlockEdge into common InteractionGraphRawInput class.

\* updatedAt field in socialgraph.unfollows is in seconds.

\*/

def getFlockFeatures(

edges: SCollection[FlockEdge],

featureName: FeatureName,

currentTimeMillis: Long

): SCollection[InteractionGraphRawInput] = {

edges

.withName(s"${featureName.toString} - Converting flock edge to interaction graph input")

.map { edge =>

val age = ChronoUnit.DAYS.between(

Instant.ofEpochMilli(edge.updatedAt \* 1000L), // updatedAt is in seconds

Instant.ofEpochMilli(currentTimeMillis)

)

InteractionGraphRawInput(

edge.sourceId,

edge.destinationId,

featureName,

age.max(0).toInt,

1.0)

}

}

/\*\*

\* Convert com.twitter.socialgraph.presto.thriftscala.Edge (from unfollows) into common InteractionGraphRawInput class.

\* updatedAt field in socialgraph.unfollows is in seconds.

\*/

def getSocialGraphFeatures(

edges: SCollection[SocialGraphEdge],

featureName: FeatureName,

currentTimeMillis: Long

): SCollection[InteractionGraphRawInput] = {

edges

.withName(s"${featureName.toString} - Converting flock edge to interaction graph input")

.map { edge =>

val age = ChronoUnit.DAYS.between(

Instant.ofEpochMilli(edge.updatedAt \* 1000L), // updatedAt is in seconds

Instant.ofEpochMilli(currentTimeMillis)

)

InteractionGraphRawInput(

edge.sourceId,

edge.destinationId,

featureName,

age.max(0).toInt,

1.0)

}

}

def isFollow(edge: Edge): Boolean = {

val result = edge.features

.find(\_.name == FeatureName.NumFollows)

.exists(\_.tss.mean == 1.0)

result

}

def filterExtremes(edge: Edge): Boolean = {

if (edge.weight.exists(\_.isNaN)) {

ScioMetrics.counter("filter extremes", "nan").inc()

false

} else if (edge.weight.contains(Double.MaxValue)) {

ScioMetrics.counter("filter extremes", "max value").inc()

false

} else if (edge.weight.contains(Double.PositiveInfinity)) {

ScioMetrics.counter("filter extremes", "+ve inf").inc()

false

} else if (edge.weight.exists(\_ < 0.0)) {

ScioMetrics.counter("filter extremes", "negative").inc()

false

} else {

true

}

}

def filterNegative(edge: Edge): Boolean = {

!edge.features.find(ef => HEALTH\_FEATURE\_LIST.contains(ef.name)).exists(\_.tss.mean > 0.0)

}

}