package com.twitter.simclusters\_v2.scio.common

import com.spotify.scio.ScioContext

import com.spotify.scio.values.SCollection

import com.twitter.beam.io.dal.DAL

import com.twitter.common.util.Clock

import com.twitter.common\_header.thriftscala.CommonHeader

import com.twitter.common\_header.thriftscala.IdType

import com.twitter.common\_header.thriftscala.VersionedCommonHeader

import com.twitter.frigate.data\_pipeline.magicrecs.magicrecs\_notifications\_lite.thriftscala.MagicRecsNotificationLite

import com.twitter.frigate.data\_pipeline.scalding.magicrecs.magicrecs\_notification\_lite.MagicrecsNotificationLite1DayLagScalaDataset

import com.twitter.iesource.thriftscala.InteractionEvent

import com.twitter.iesource.thriftscala.InteractionTargetType

import com.twitter.interests\_ds.jobs.interests\_service.UserTopicRelationSnapshotScalaDataset

import com.twitter.interests.thriftscala.InterestRelationType

import com.twitter.interests.thriftscala.UserInterestsRelationSnapshot

import com.twitter.penguin.scalding.datasets.PenguinUserLanguagesScalaDataset

import com.twitter.search.adaptive.scribing.thriftscala.AdaptiveSearchScribeLog

import com.twitter.simclusters\_v2.hdfs\_sources.UserUserFavGraphScalaDataset

import com.twitter.simclusters\_v2.scalding.embedding.common.ExternalDataSources.ValidFlockEdgeStateId

import com.twitter.simclusters\_v2.scalding.embedding.common.ExternalDataSources.getStandardLanguageCode

import com.twitter.twadoop.user.gen.thriftscala.CombinedUser

import flockdb\_tools.datasets.flock.FlockBlocksEdgesScalaDataset

import flockdb\_tools.datasets.flock.FlockFollowsEdgesScalaDataset

import flockdb\_tools.datasets.flock.FlockReportAsAbuseEdgesScalaDataset

import flockdb\_tools.datasets.flock.FlockReportAsSpamEdgesScalaDataset

import org.joda.time.Interval

import com.twitter.simclusters\_v2.thriftscala.EdgeWithDecayedWeights

import com.twitter.usersource.snapshot.combined.UsersourceScalaDataset

import com.twitter.usersource.snapshot.flat.UsersourceFlatScalaDataset

import com.twitter.util.Duration

import twadoop\_config.configuration.log\_categories.group.search.AdaptiveSearchScalaDataset

object ExternalDataSources {

def userSource(

noOlderThan: Duration = Duration.fromDays(7)

)(

implicit sc: ScioContext

): SCollection[CombinedUser] = {

sc.customInput(

"ReadUserSource",

DAL

.readMostRecentSnapshotNoOlderThan(

UsersourceScalaDataset,

noOlderThan,

Clock.SYSTEM\_CLOCK,

DAL.Environment.Prod

)

)

}

def userCountrySource(

noOlderThan: Duration = Duration.fromDays(7)

)(

implicit sc: ScioContext

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

sc.customInput(

"ReadUserCountrySource",

DAL

.readMostRecentSnapshotNoOlderThan(

UsersourceFlatScalaDataset,

noOlderThan,

Clock.SYSTEM\_CLOCK,

DAL.Environment.Prod,

)

).flatMap { flatUser =>

for {

userId <- flatUser.id

country <- flatUser.accountCountryCode

} yield {

(userId, country.toUpperCase)

}

}.distinct

}

def userUserFavSource(

noOlderThan: Duration = Duration.fromDays(14)

)(

implicit sc: ScioContext

): SCollection[EdgeWithDecayedWeights] = {

sc.customInput(

"ReadUserUserFavSource",

DAL

.readMostRecentSnapshotNoOlderThan(

UserUserFavGraphScalaDataset,

noOlderThan,

Clock.SYSTEM\_CLOCK,

DAL.Environment.Prod

)

)

}

def inferredUserConsumedLanguageSource(

noOlderThan: Duration = Duration.fromDays(7)

)(

implicit sc: ScioContext

): SCollection[(Long, Seq[(String, Double)])] = {

sc.customInput(

"ReadInferredUserConsumedLanguageSource",

DAL

.readMostRecentSnapshotNoOlderThan(

PenguinUserLanguagesScalaDataset,

noOlderThan,

Clock.SYSTEM\_CLOCK,

DAL.Environment.Prod

)

).map { kv =>

val consumed = kv.value.consumed

.collect {

case scoredString if scoredString.weight > 0.001 => //throw away 5% outliers

(getStandardLanguageCode(scoredString.item), scoredString.weight)

}.collect {

case (Some(language), score) => (language, score)

}

(kv.key, consumed)

}

}

def flockBlockSource(

noOlderThan: Duration = Duration.fromDays(7)

)(

implicit sc: ScioContext

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

sc.customInput(

"ReadFlockBlock",

DAL.readMostRecentSnapshotNoOlderThan(

FlockBlocksEdgesScalaDataset,

noOlderThan,

Clock.SYSTEM\_CLOCK,

DAL.Environment.Prod))

.collect {

case edge if edge.state == ValidFlockEdgeStateId =>

(edge.sourceId, edge.destinationId)

}

}

def flockFollowSource(

noOlderThan: Duration = Duration.fromDays(7)

)(

implicit sc: ScioContext

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

sc.customInput(

"ReadFlockFollow",

DAL

.readMostRecentSnapshotNoOlderThan(

FlockFollowsEdgesScalaDataset,

noOlderThan,

Clock.SYSTEM\_CLOCK,

DAL.Environment.Prod))

.collect {

case edge if edge.state == ValidFlockEdgeStateId =>

(edge.sourceId, edge.destinationId)

}

}

def flockReportAsAbuseSource(

noOlderThan: Duration = Duration.fromDays(7)

)(

implicit sc: ScioContext

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

sc.customInput(

"ReadFlockReportAsAbuseJava",

DAL

.readMostRecentSnapshotNoOlderThan(

FlockReportAsAbuseEdgesScalaDataset,

noOlderThan,

Clock.SYSTEM\_CLOCK,

DAL.Environment.Prod)

)

.collect {

case edge if edge.state == ValidFlockEdgeStateId =>

(edge.sourceId, edge.destinationId)

}

}

def flockReportAsSpamSource(

noOlderThan: Duration = Duration.fromDays(7)

)(

implicit sc: ScioContext

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

sc.customInput(

"ReadFlockReportAsSpam",

DAL

.readMostRecentSnapshotNoOlderThan(

FlockReportAsSpamEdgesScalaDataset,

noOlderThan,

Clock.SYSTEM\_CLOCK,

DAL.Environment.Prod))

.collect {

case edge if edge.state == ValidFlockEdgeStateId =>

(edge.sourceId, edge.destinationId)

}

}

def ieSourceTweetEngagementsSource(

interval: Interval

)(

implicit sc: ScioContext

): SCollection[InteractionEvent] = {

sc.customInput(

"ReadIeSourceTweetEngagementsSource",

DAL

.read(

com.twitter.iesource.processing.events.batch.ServerEngagementsScalaDataset,

interval,

DAL.Environment.Prod,

)

).filter { event =>

// filter out logged out users because their favorites are less reliable

event.engagingUserId > 0L && event.targetType == InteractionTargetType.Tweet

}

}

def topicFollowGraphSource(

noOlderThan: Duration = Duration.fromDays(7)

)(

implicit sc: ScioContext

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

// The implementation here is slightly different than the topicFollowGraphSource function in

// src/scala/com/twitter/simclusters\_v2/scalding/embedding/common/ExternalDataSources.scala

// We don't do an additional hashJoin on uttFollowableEntitiesSource.

sc.customInput(

"ReadTopicFollowGraphSource",

DAL

.readMostRecentSnapshotNoOlderThan(

UserTopicRelationSnapshotScalaDataset,

noOlderThan,

Clock.SYSTEM\_CLOCK,

DAL.Environment.Prod

)

).collect {

case userInterestsRelationSnapshot: UserInterestsRelationSnapshot

if userInterestsRelationSnapshot.interestType == "UTT" &&

userInterestsRelationSnapshot.relation == InterestRelationType.Followed =>

(userInterestsRelationSnapshot.interestId, userInterestsRelationSnapshot.userId)

}

}

def magicRecsNotficationOpenOrClickEventsSource(

interval: Interval

)(

implicit sc: ScioContext

): SCollection[MagicRecsNotificationLite] = {

sc.customInput(

"ReadMagicRecsNotficationOpenOrClickEventsSource",

DAL

.read(MagicrecsNotificationLite1DayLagScalaDataset, interval, DAL.Environment.Prod))

.filter { entry =>

// keep entries with a valid userId and tweetId, opened or clicked timestamp defined

val userIdExists = entry.targetUserId.isDefined

val tweetIdExists = entry.tweetId.isDefined

val openOrClickExists =

entry.openTimestampMs.isDefined || entry.ntabClickTimestampMs.isDefined

userIdExists && tweetIdExists && openOrClickExists

}

}

def adaptiveSearchScribeLogsSource(

interval: Interval

)(

implicit sc: ScioContext

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

sc.customInput(

"ReadAdaptiveSearchScribeLogsSource",

DAL

.read(AdaptiveSearchScalaDataset, interval, DAL.Environment.Prod))

.flatMap({ scribeLog: AdaptiveSearchScribeLog =>

for {

userId <- userIdFromBlenderAdaptiveScribeLog(scribeLog)

// filter out logged out search queries

if userId != 0

queryString <- scribeLog.requestLog.flatMap(\_.request).flatMap(\_.rawQuery)

} yield {

(userId, Set(queryString))

}

})

// if a user searches for the same query multiple times, there could be duplicates.

// De-dup them to get the distinct queries searched by a user

.sumByKey

.flatMap {

case (userId, distinctQuerySet) =>

distinctQuerySet.map { query =>

(userId, query)

}

}

}

private def userIdFromBlenderAdaptiveScribeLog(

blenderAdaptiveLog: AdaptiveSearchScribeLog

): Option[Long] = {

blenderAdaptiveLog.versionedCommonHeader match {

case VersionedCommonHeader.CommonHeader(CommonHeader.ServerHeader(serverHeader)) =>

serverHeader.requestInfo match {

case Some(requestInfo) => requestInfo.ids.get(IdType.UserId).map(\_.toLong)

case \_ => None

}

case \_ => None

}

}

}