package com.twitter.simclusters\_v2.scalding.inferred\_entities

import com.twitter.scalding.{DateRange, Days, TypedPipe}

import com.twitter.scalding\_internal.dalv2.DAL

import com.twitter.scalding\_internal.dalv2.remote\_access.{ExplicitLocation, ProcAtla}

import com.twitter.scalding\_internal.multiformat.format.keyval.KeyVal

import com.twitter.simclusters\_v2.common.{ModelVersions, SemanticCoreEntityId, UserId}

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

SimclustersInferredEntitiesFromKnownForScalaDataset,

SimclustersV2InterestedIn20M145KUpdatedScalaDataset,

SimclustersV2InterestedInScalaDataset,

SimclustersV2KnownFor20M145KDec11ScalaDataset,

SimclustersV2KnownFor20M145KUpdatedScalaDataset,

UserUserNormalizedGraphScalaDataset

}

import com.twitter.simclusters\_v2.scalding.KnownForSources

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

EntitySource,

SimClusterWithScore,

SimClustersSource,

TopSimClustersWithScore,

UserAndNeighbors

}

import java.util.TimeZone

/\*\*

\* Convenience functions to read data from prod.

\*/

object ProdSources {

// Returns the Dec11 KnownFor from production

def getDec11KnownFor(implicit tz: TimeZone): TypedPipe[(UserId, Seq[SimClusterWithScore])] =

KnownForSources

.readDALDataset(

SimclustersV2KnownFor20M145KDec11ScalaDataset,

Days(30),

ModelVersions.Model20M145KDec11)

.map {

case (userId, clustersArray) =>

val clusters = clustersArray.map {

case (clusterId, score) => SimClusterWithScore(clusterId, score)

}.toSeq

(userId, clusters)

}

// Returns the Updated KnownFor from production

def getUpdatedKnownFor(implicit tz: TimeZone): TypedPipe[(UserId, Seq[SimClusterWithScore])] =

KnownForSources

.readDALDataset(

SimclustersV2KnownFor20M145KUpdatedScalaDataset,

Days(30),

ModelVersions.Model20M145KUpdated

)

.map {

case (userId, clustersArray) =>

val clusters = clustersArray.map {

case (clusterId, score) => SimClusterWithScore(clusterId, score)

}.toSeq

(userId, clusters)

}

def getInferredEntitiesFromKnownFor(

inferredFromCluster: SimClustersSource,

inferredFromEntity: EntitySource,

dateRange: DateRange

): TypedPipe[(UserId, Seq[(SemanticCoreEntityId, Double)])] = {

DAL

.readMostRecentSnapshot(SimclustersInferredEntitiesFromKnownForScalaDataset, dateRange)

.withRemoteReadPolicy(ExplicitLocation(ProcAtla))

.toTypedPipe

.map {

case KeyVal(userId, entities) =>

val validEntities =

entities.entities

.collect {

case entity

if entity.entitySource.contains(inferredFromEntity) &&

entity.simclusterSource.contains(inferredFromCluster) =>

(entity.entityId, entity.score)

}

.groupBy(\_.\_1)

.map { case (entityId, scores) => (entityId, scores.map(\_.\_2).max) }

.toSeq

(userId, validEntities)

}

}

def getUserUserEngagementGraph(dateRange: DateRange): TypedPipe[UserAndNeighbors] = {

DAL

.readMostRecentSnapshot(UserUserNormalizedGraphScalaDataset, dateRange)

.withRemoteReadPolicy(ExplicitLocation(ProcAtla))

.toTypedPipe

}

}