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

import com.twitter.scalding.DateRange

import com.twitter.scalding.Days

import com.twitter.scalding.typed.TypedPipe

import com.twitter.simclusters\_v2.common.ClusterId

import com.twitter.simclusters\_v2.common.ModelVersions

import com.twitter.simclusters\_v2.common.UserId

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

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

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

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

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

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

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

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

import java.util.TimeZone

/\*\*

\* Opt-out compliance for SimClusters means offering users an option to opt out of clusters that

\* have inferred legible meanings. This file sets some of the data sources & thresholds from which

\* the inferred entities are considered legible. One should always refer to the sources & constants

\* here for SimClusters' inferred entity compliance work

\*/

object InferredEntities {

val MHRootPath: String =

"/user/cassowary/manhattan\_sequence\_files/simclusters\_v2\_inferred\_entities"

// Convenience objects for defining cluster sources

val InterestedIn2020 =

SimClustersSource(ClusterType.InterestedIn, ModelVersion.Model20m145k2020)

val Dec11KnownFor = SimClustersSource(ClusterType.KnownFor, ModelVersion.Model20m145kDec11)

val UpdatedKnownFor = SimClustersSource(ClusterType.KnownFor, ModelVersion.Model20m145kUpdated)

val KnownFor2020 = SimClustersSource(ClusterType.KnownFor, ModelVersion.Model20m145k2020)

/\*\*

\* This is the threshold at which we consider a simcluster "legible" through an entity

\*/

val MinLegibleEntityScore = 0.6

/\*\*

\* Query for the entity embeddings that are used for SimClusters compliance. We will use these

\* entity embeddings for a cluster to allow a user to opt out of a cluster

\*/

def getLegibleEntityEmbeddings(

dateRange: DateRange,

timeZone: TimeZone

): TypedPipe[(ClusterId, Seq[SemanticCoreEntityWithScore])] = {

val entityEmbeddings = EntityEmbeddingsSources

.getReverseIndexedSemanticCoreEntityEmbeddingsSource(

EmbeddingType.FavBasedSematicCoreEntity,

ModelVersions.Model20M145K2020, // only support the latest 2020 model

dateRange.embiggen(Days(7)(timeZone)) // read 7 days before & after to give buffer

)

filterEntityEmbeddingsByScore(entityEmbeddings, MinLegibleEntityScore)

}

// Return entities whose score are above threshold

def filterEntityEmbeddingsByScore(

entityEmbeddings: TypedPipe[(ClusterId, Seq[SemanticCoreEntityWithScore])],

minEntityScore: Double

): TypedPipe[(ClusterId, Seq[SemanticCoreEntityWithScore])] = {

entityEmbeddings.flatMap {

case (clusterId, entities) =>

val validEntities = entities.filter { entity => entity.score >= minEntityScore }

if (validEntities.nonEmpty) {

Some((clusterId, validEntities))

} else {

None

}

}

}

/\*\*

\* Given inferred entities from different sources, combine the results into job's output format

\*/

def combineResults(

results: TypedPipe[(UserId, Seq[InferredEntity])]\*

): TypedPipe[(UserId, SimClustersInferredEntities)] = {

results

.reduceLeft(\_ ++ \_)

.sumByKey

.map {

case (userId, inferredEntities) =>

(userId, SimClustersInferredEntities(inferredEntities))

}

}

}