package com.twitter.timelines.data\_processing.ml\_util.aggregation\_framework.scalding

import com.twitter.gizmoduck.snapshot.DeletedUserScalaDataset

import com.twitter.ml.api.DataRecord

import com.twitter.ml.api.Feature

import com.twitter.scalding.typed.TypedPipe

import com.twitter.scalding.DateOps

import com.twitter.scalding.DateRange

import com.twitter.scalding.Days

import com.twitter.scalding.RichDate

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

import com.twitter.scalding\_internal.dalv2.remote\_access.AllowCrossClusterSameDC

import com.twitter.scalding\_internal.job.RequiredBinaryComparators.ordSer

import com.twitter.scalding\_internal.pruner.Pruner

import com.twitter.timelines.data\_processing.ml\_util.aggregation\_framework.AggregationKey

import com.twitter.timelines.data\_processing.ml\_util.aggregation\_framework.TypedAggregateGroup

import com.twitter.scalding.serialization.macros.impl.ordered\_serialization.runtime\_helpers.MacroEqualityOrderedSerialization

import java.{util => ju}

object DeletedUserSeqPruner extends Pruner[Seq[Long]] {

implicit val tz: ju.TimeZone = DateOps.UTC

implicit val userIdSequenceOrdering: MacroEqualityOrderedSerialization[Seq[Long]] =

ordSer[Seq[Long]]

private[scalding] def pruneDeletedUsers[T](

input: TypedPipe[T],

extractor: T => Seq[Long],

deletedUsers: TypedPipe[Long]

): TypedPipe[T] = {

val userIdsAndValues = input.map { t: T =>

val userIds: Seq[Long] = extractor(t)

(userIds, t)

}

// Find all valid sequences of userids in the input pipe

// that contain at least one deleted user. This is efficient

// as long as the number of deleted users is small.

val userSequencesWithDeletedUsers = userIdsAndValues

.flatMap { case (userIds, \_) => userIds.map((\_, userIds)) }

.leftJoin(deletedUsers.asKeys)

.collect { case (\_, (userIds, Some(\_))) => userIds }

.distinct

userIdsAndValues

.leftJoin(userSequencesWithDeletedUsers.asKeys)

.collect { case (\_, (t, None)) => t }

}

override def prune[T](

input: TypedPipe[T],

put: (T, Seq[Long]) => Option[T],

get: T => Seq[Long],

writeTime: RichDate

): TypedPipe[T] = {

lazy val deletedUsers = DAL

.readMostRecentSnapshot(DeletedUserScalaDataset, DateRange(writeTime - Days(7), writeTime))

.withRemoteReadPolicy(AllowCrossClusterSameDC)

.toTypedPipe

.map(\_.userId)

pruneDeletedUsers(input, get, deletedUsers)

}

}

object AggregationKeyPruner {

/\*\*

\* Makes a pruner that prunes aggregate records where any of the

\* "userIdFeatures" set in the aggregation key correspond to a

\* user who has deleted their account. Here, "userIdFeatures" is

\* intended as a catch-all term for all features corresponding to

\* a Twitter user in the input data record -- the feature itself

\* could represent an authorId, retweeterId, engagerId, etc.

\*/

def mkDeletedUsersPruner(

userIdFeatures: Seq[Feature[\_]]

): Pruner[(AggregationKey, DataRecord)] = {

val userIdFeatureIds = userIdFeatures.map(TypedAggregateGroup.getDenseFeatureId)

def getter(tupled: (AggregationKey, DataRecord)): Seq[Long] = {

tupled match {

case (aggregationKey, \_) =>

userIdFeatureIds.flatMap { id =>

aggregationKey.discreteFeaturesById

.get(id)

.orElse(aggregationKey.textFeaturesById.get(id).map(\_.toLong))

}

}

}

// Setting putter to always return None here. The put function is not used within pruneDeletedUsers, this function is just needed for xmap api.

def putter: ((AggregationKey, DataRecord), Seq[Long]) => Option[(AggregationKey, DataRecord)] =

(t, seq) => None

DeletedUserSeqPruner.xmap(putter, getter)

}

}