package com.twitter.simclusters\_v2.scalding

package multi\_type\_graph.assemble\_multi\_type\_graph

import com.twitter.dal.client.dataset.{KeyValDALDataset, SnapshotDALDataset}

import com.twitter.scalding.{Execution, \_}

import com.twitter.scalding\_internal.dalv2.DALWrite.{D, \_}

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

import com.twitter.simclusters\_v2.scalding.common.TypedRichPipe.typedPipeToRichPipe

import com.twitter.simclusters\_v2.scalding.common.Util

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

LeftNode,

Noun,

NounWithFrequency,

NounWithFrequencyList,

RightNodeType,

RightNodeTypeStruct,

RightNodeWithEdgeWeight,

RightNodeWithEdgeWeightList,

MultiTypeGraphEdge

}

import com.twitter.wtf.scalding.jobs.common.DateRangeExecutionApp

import java.util.TimeZone

/\*\*

\* In this file, we assemble the multi\_type\_graph user-entity engagement signals

\*

\* It works as follows and the following datasets are produced as a result:

\*

\* 1. FullGraph (fullMultiTypeGraphSnapshotDataset) : reads datasets from multiple sources and generates

\* a bipartite graph with LeftNode -> RightNode edges, capturing a user's engagement with varied entity types

\*

\* 2. TruncatedGraph (truncatedMultiTypeGraphKeyValDataset): a truncated version of the FullGraph

\* where we only store the topK most frequently occurring RightNodes in the bipartite graph LeftNode -> RightNode

\*

\* 3. TopKNouns (topKRightNounsKeyValDataset): this stores the topK most frequent Nouns for each engagement type

\* Please note that this dataset is currently only being used for the debugger to find which nodes we consider as the

\* most frequently occurring, in FullGraph

\*/

trait AssembleMultiTypeGraphBaseApp extends DateRangeExecutionApp {

val truncatedMultiTypeGraphKeyValDataset: KeyValDALDataset[

KeyVal[LeftNode, RightNodeWithEdgeWeightList]

]

val topKRightNounsKeyValDataset: KeyValDALDataset[

KeyVal[RightNodeTypeStruct, NounWithFrequencyList]

]

val fullMultiTypeGraphSnapshotDataset: SnapshotDALDataset[MultiTypeGraphEdge]

val isAdhoc: Boolean

val truncatedMultiTypeGraphMHOutputPath: String

val topKRightNounsMHOutputPath: String

val fullMultiTypeGraphThriftOutputPath: String

override def runOnDateRange(

args: Args

)(

implicit dateRange: DateRange,

timeZone: TimeZone,

uniqueID: UniqueID

): Execution[Unit] = {

import Config.\_

import AssembleMultiTypeGraph.\_

val numKeysInTruncatedGraph = Stat("num\_keys\_truncated\_mts")

val numKeysInTopKNounsGraph = Stat("num\_keys\_topk\_nouns\_mts")

val fullGraph: TypedPipe[(LeftNode, RightNodeWithEdgeWeight)] =

getFullGraph().count("num\_entries\_full\_graph")

val topKRightNodes: TypedPipe[(RightNodeType, Seq[(Noun, Double)])] =

getTopKRightNounsWithFrequencies(

fullGraph,

TopKConfig,

GlobalDefaultMinFrequencyOfRightNodeType)

val truncatedGraph: TypedPipe[(LeftNode, RightNodeWithEdgeWeight)] =

getTruncatedGraph(fullGraph, topKRightNodes).count("num\_entries\_truncated\_graph")

// key transformations - truncated graph, keyed by LeftNode

val truncatedGraphKeyedBySrc: TypedPipe[(LeftNode, RightNodeWithEdgeWeightList)] =

truncatedGraph

.map {

case (LeftNode.UserId(userId), rightNodeWithWeight) =>

userId -> List(rightNodeWithWeight)

}

.sumByKey

.map {

case (userId, rightNodeWithWeightList) =>

(LeftNode.UserId(userId), RightNodeWithEdgeWeightList(rightNodeWithWeightList))

}

// key transformation - topK nouns, keyed by the RightNodeNounType

val topKNounsKeyedByType: TypedPipe[(RightNodeTypeStruct, NounWithFrequencyList)] =

topKRightNodes

.map {

case (rightNodeType, rightNounsWithScoresList) =>

val nounsListWithFrequency: Seq[NounWithFrequency] = rightNounsWithScoresList

.map {

case (noun, aggregatedFrequency) =>

NounWithFrequency(noun, aggregatedFrequency)

}

(RightNodeTypeStruct(rightNodeType), NounWithFrequencyList(nounsListWithFrequency))

}

//WriteExecs - truncated graph

val truncatedGraphTsvExec: Execution[Unit] =

truncatedGraphKeyedBySrc.writeExecution(

TypedTsv[(LeftNode, RightNodeWithEdgeWeightList)](AdhocRootPrefix + "truncated\_graph\_tsv"))

val truncatedGraphDALExec: Execution[Unit] = truncatedGraphKeyedBySrc

.map {

case (leftNode, rightNodeWithWeightList) =>

numKeysInTruncatedGraph.inc()

KeyVal(leftNode, rightNodeWithWeightList)

}

.writeDALVersionedKeyValExecution(

truncatedMultiTypeGraphKeyValDataset,

D.Suffix(

(if (!isAdhoc)

RootPath

else

AdhocRootPrefix)

+ truncatedMultiTypeGraphMHOutputPath),

ExplicitEndTime(dateRange.`end`)

)

//WriteExec - topK rightnouns

val topKNounsTsvExec: Execution[Unit] =

topKNounsKeyedByType.writeExecution(

TypedTsv[(RightNodeTypeStruct, NounWithFrequencyList)](

AdhocRootPrefix + "top\_k\_right\_nouns\_tsv"))

// writing topKNouns MH dataset for debugger

val topKNounsDALExec: Execution[Unit] = topKNounsKeyedByType

.map {

case (engagementType, rightList) =>

val rightListMH =

NounWithFrequencyList(rightList.nounWithFrequencyList.take(TopKRightNounsForMHDump))

numKeysInTopKNounsGraph.inc()

KeyVal(engagementType, rightListMH)

}

.writeDALVersionedKeyValExecution(

topKRightNounsKeyValDataset,

D.Suffix(

(if (!isAdhoc)

RootPath

else

AdhocRootPrefix)

+ topKRightNounsMHOutputPath),

ExplicitEndTime(dateRange.`end`)

)

//WriteExec - fullGraph

val fullGraphDALExec: Execution[Unit] = fullGraph

.map {

case (leftNode, rightNodeWithWeight) =>

MultiTypeGraphEdge(leftNode, rightNodeWithWeight)

}.writeDALSnapshotExecution(

fullMultiTypeGraphSnapshotDataset,

D.Daily,

D.Suffix(

(if (!isAdhoc)

RootThriftPath

else

AdhocRootPrefix)

+ fullMultiTypeGraphThriftOutputPath),

D.Parquet,

dateRange.`end`

)

if (isAdhoc) {

Util.printCounters(

Execution

.zip(

truncatedGraphTsvExec,

topKNounsTsvExec,

truncatedGraphDALExec,

topKNounsDALExec,

fullGraphDALExec).unit)

} else {

Util.printCounters(

Execution.zip(truncatedGraphDALExec, topKNounsDALExec, fullGraphDALExec).unit)

}

}

}