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

import com.twitter.bijection.Injection

import com.twitter.ml.api.\_

import com.twitter.ml.api.Feature

import com.twitter.ml.api.util.SRichDataRecord

import com.twitter.scalding.typed.TypedPipe

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

import scala.collection.JavaConverters.\_

case class SparseJoinConfig(

aggregates: DataSetPipe,

sparseKey: Feature.SparseBinary,

mergePolicies: SparseBinaryMergePolicy\*)

object SparseBinaryMultipleAggregateJoin {

type CommonMap = (String, ((Feature.SparseBinary, String), DataRecord))

def apply(

source: DataSetPipe,

commonKey: Feature[\_],

joinConfigs: Set[SparseJoinConfig],

rightJoin: Boolean = false,

isSketchJoin: Boolean = false,

numSketchJoinReducers: Int = 0

): DataSetPipe = {

val emptyPipe: TypedPipe[CommonMap] = TypedPipe.empty

val aggregateMaps: Set[TypedPipe[CommonMap]] = joinConfigs.map { joinConfig =>

joinConfig.aggregates.records.map { record =>

val sparseKeyValue =

SRichDataRecord(record).getFeatureValue(sparseFeature(joinConfig.sparseKey)).toString

val commonKeyValue = SRichDataRecord(record).getFeatureValue(commonKey).toString

(commonKeyValue, ((joinConfig.sparseKey, sparseKeyValue), record))

}

}

val commonKeyToAggregateMap = aggregateMaps

.foldLeft(emptyPipe) {

case (union: TypedPipe[CommonMap], next: TypedPipe[CommonMap]) =>

union ++ next

}

.group

.toList

.map {

case (commonKeyValue, aggregateTuples) =>

(commonKeyValue, aggregateTuples.toMap)

}

val commonKeyToRecordMap = source.records

.map { record =>

val commonKeyValue = SRichDataRecord(record).getFeatureValue(commonKey).toString

(commonKeyValue, record)

}

// rightJoin is not supported by Sketched, so rightJoin will be ignored if isSketchJoin is set

implicit val string2Byte = (value: String) => Injection[String, Array[Byte]](value)

val intermediateRecords = if (isSketchJoin) {

commonKeyToRecordMap.group

.sketch(numSketchJoinReducers)

.leftJoin(commonKeyToAggregateMap)

.toTypedPipe

} else if (rightJoin) {

commonKeyToAggregateMap

.rightJoin(commonKeyToRecordMap)

.mapValues(\_.swap)

.toTypedPipe

} else {

commonKeyToRecordMap.leftJoin(commonKeyToAggregateMap).toTypedPipe

}

val joinedRecords = intermediateRecords

.map {

case (commonKeyValue, (inputRecord, aggregateTupleMapOpt)) =>

aggregateTupleMapOpt.foreach { aggregateTupleMap =>

joinConfigs.foreach { joinConfig =>

val sparseKeyValues = Option(

SRichDataRecord(inputRecord)

.getFeatureValue(joinConfig.sparseKey)

).map(\_.asScala.toList)

.getOrElse(List.empty[String])

val aggregateRecords = sparseKeyValues.flatMap { sparseKeyValue =>

aggregateTupleMap.get((joinConfig.sparseKey, sparseKeyValue))

}

joinConfig.mergePolicies.foreach { mergePolicy =>

mergePolicy.mergeRecord(

inputRecord,

aggregateRecords,

joinConfig.aggregates.featureContext

)

}

}

}

inputRecord

}

val joinedFeatureContext = joinConfigs

.foldLeft(source.featureContext) {

case (left, joinConfig) =>

joinConfig.mergePolicies.foldLeft(left) {

case (soFar, mergePolicy) =>

mergePolicy.mergeContext(soFar, joinConfig.aggregates.featureContext)

}

}

DataSetPipe(joinedRecords, joinedFeatureContext)

}

}