package com.twitter.simclusters\_v2.scalding.common.matrix

import com.twitter.algebird.{ArrayMonoid, BloomFilterMonoid, Monoid, Semigroup}

import com.twitter.algebird.Semigroup.\_

import com.twitter.bijection.Injection

import com.twitter.scalding.{TypedPipe, ValuePipe}

/\*\*

\* A class that represents a row-indexed dense matrix, backed by a TypedPipe[(R, Array[Double])].

\* For each row of the TypedPipe, we save an array of values.

\* Only use this class when the number of columns is small (say, <100K).

\*

\* @param pipe underlying pipe

\* @param rowOrd ordering function for row type

\* @param rowInj injection function for the row type

\* @tparam R Type for rows

\*/

case class DenseRowMatrix[R](

pipe: TypedPipe[(R, Array[Double])],

)(

implicit val rowOrd: Ordering[R],

val rowInj: Injection[R, Array[Byte]]) {

lazy val semigroupArrayV: Semigroup[Array[Double]] = new ArrayMonoid[Double]()

// convert to a SparseMatrix

lazy val toSparseMatrix: SparseMatrix[R, Int, Double] = {

this.toSparseRowMatrix.toSparseMatrix

}

// convert to a SparseRowMatrix

lazy val toSparseRowMatrix: SparseRowMatrix[R, Int, Double] = {

SparseRowMatrix(

this.pipe.map {

case (i, values) =>

(i, values.zipWithIndex.collect { case (value, j) if value != 0.0 => (j, value) }.toMap)

},

isSkinnyMatrix = true)

}

// convert to a TypedPipe

lazy val toTypedPipe: TypedPipe[(R, Array[Double])] = {

this.pipe

}

// filter the matrix based on a subset of rows

def filterRows(rows: TypedPipe[R]): DenseRowMatrix[R] = {

DenseRowMatrix(this.pipe.join(rows.asKeys).mapValues(\_.\_1))

}

// get the l2 norms for all rows. this does not trigger a shuffle.

lazy val rowL2Norms: TypedPipe[(R, Double)] = {

this.pipe.map {

case (row, values) =>

row -> math.sqrt(values.map(a => a \* a).sum)

}

}

// normalize the matrix to make sure each row has unit norm

lazy val rowL2Normalize: DenseRowMatrix[R] = {

DenseRowMatrix(this.pipe.map {

case (row, values) =>

val norm = math.sqrt(values.map(v => v \* v).sum)

if (norm == 0.0) {

row -> values

} else {

row -> values.map(v => v / norm)

}

})

}

}